Germán Harvey Alférez Salinas
PhD Thesis

Achieving Autonomic Web Service Compositions with Models at Runtime

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dr. Vicente Pelechano Ferragud
Thesis Advisor

ACHIEVING AUTONOMIC WEB SERVICE COMPOSITIONS WITH MODELS AT RUNTIME

GERMÁN HARVEY ALFÉREZ SALINAS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

December 2013

ACHIEVING AUTONOMIC WEB SERVICE COMPOSITIONS WITH MODELS AT RUNTIME

GERMÁN HARVEY ALFÉREZ SALINAS

Thesis Advisor:
Vicente Pelechano Ferragud

Centro de Investigación en Métodos de Producción de Software
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia, Spain
Phone +34 963877007 (Ext. 83533)
Fax +34 963877359
www.pros.upv.es

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at the Universitat Politècnica de València

Members of the Thesis Committee:

Dr. Oscar Pastor López,
Universitat Politècnica de València

Dr. Antonio Ruiz Cortés,
Universidad de Sevilla

Dr. Xavier Franch Gutiérrez,
Universitat Politècnica de Catalunya

Dedicated to my beloved wife Doris and son Johan.
You are heavenly gifts, my joy, and inspiration.

"[I] do not cease to give thanks for you, making mention of you in my prayers."

— Paul the Apostle

AGRADECIMIENTOS

Alguien dijo que realizar una tesis doctoral es una tarea solitaria que no se puede llevar a cabo solo. ¡Cuánta verdad encierran estas palabras!

Al finalizar este proceso, en primer lugar le doy infinitas gracias a Dios por haberme dado la fortaleza, los recursos, y la inteligencia para culminar mis estudios doctorales. ¡Tu amor, Señor, llega hasta los cielos; tu fidelidad alcanza las nubes!

Expreso mi más sincero agradecimiento a mi director de tesis, el Dr. Vicente Pelechano, por haberme brindado su apoyo al 100% durante mis estudios. No sólo admiro tus soluciones brillantes a problemas no triviales, sino tu interés en mi investigación y en mi progreso personal y académico. Además, tu disponibilidad de 24/7 es algo que admiro y que sin duda ha sido un factor de éxito en esta investigación.

Le agradezco a los miembros del equipo de investigación en la Universitat Politècnica de València por su apoyo y las valiosas ideas que enriquecieron esta tesis. Definitivamente, esta universidad cuenta con investigadores de gran calidad humana y científica.

I am grateful to Dr. Camille Salinesi, Dr. Raúl Mazo, and Dr. Daniel Diaz at the Université Paris-1 Panthéon-Sorbonne. Their feedback and great ideas nurtured the verification dimension of this thesis. It was a pleasure to create synergy and new knowledge with you.

Expreso una profunda gratitud a mi amada esposa Doris y a mi hijito Johan. Chiquis, gracias por haber emprendido esta aventura en búsqueda de nuevo conocimiento a mi lado. Muchas gracias por compartir tantas alegrías y por tu apoyo en momentos difíciles. Tu amor y aliento me impulsaron a diario. Johan, cuando puedas leer estas palabras, quiero que sepas que llegaste para darme fuerzas en un momento crucial. Mi deseo es que llegues a amar el estudio y las ciencias tanto o mucho más que yo.

No puedo dejar pasar esta oportunidad para agradecer a mis padres, a mi hermano Mauricio, y a la familia de mi esposa por sus oraciones

y apoyo en este proceso. A pesar de la distancia, nunca nos sentimos solos. Asimismo, le agradezco a cada uno de nuestros amigos y amigas en Valencia. Vuestro apoyo en todos los sentidos ha sido una alegría para nosotros. En especial, le agradezco a la querida familia Gavín por haber sido nuestra familia en España.

Finalmente, la realización de este doctorado no hubiera sido posible sin el apoyo financiero de la Universidad de Montemorelos. Gracias por esta oportunidad. Estoy seguro que la educación que he recibido ha afianzado mi visión para emprender y mi pasión para servir.

¡Muchas gracias!

ABSTRACT

Over the last years, Web services have become increasingly popular. It is because they allow businesses to share data and business process logic through a programmatic interface across networks. In order to reach the full potential of Web services, they can be combined to achieve specific functionalities.

Web services run in complex contexts where arising events may compromise the quality of the system (e.g. a sudden security attack). As a result, it is desirable to count on mechanisms to adapt Web service compositions (or simply called *service compositions*) according to problematic events in the context. Since critical systems may require prompt responses, manual adaptations are unfeasible in large and intricate service compositions. Thus, it is suitable to have autonomic mechanisms to guide their self-adaptation. One way to achieve this is by implementing variability constructs at the language level. However, this approach may become tedious, difficult to manage, and error-prone as the number of configurations for the service composition grows.

The goal of this thesis is to provide a tool-supported framework to guide autonomic adjustments of context-aware service compositions using models at runtime. This framework spans over design time and runtime to face arising known and unknown context events (i.e., foreseen and unforeseen at design time) in the closed and open worlds, respectively.

At design time, we propose to create the models that guide autonomic changes. In order to reach optimum adaptations, a variability model and its possible configurations are verified at design time. At runtime, when problematic events arise in the context, the variability model is leveraged for guiding autonomic changes of the service composition. The activation and deactivation of features in the variability model result in changes in a composition model that abstracts the underlying service composition. Changes in the composition model are reflected into the service composition by adding or removing fragments of Web Services Business Process Execution Language (WS-BPEL) code, which are deployed at runtime.

Under the closed-world assumption, the possible context events are fully known at design time. These events will eventually trigger the *dynamic adaptation* of the service composition. Nevertheless, it is difficult to foresee all the possible situations arising in uncertain contexts where service compositions run. Therefore, the proposed framework also covers the *dynamic evolution* of service compositions to deal with unexpected events in the open world. If dynamic adaptations are not enough to solve uncertainty, the supporting models self-evolve according to abstract tactics, which preserve expected requirements.

The proposal has been validated with a case study and simulations. The answers to several research questions demonstrate the feasibility of models at runtime to guide dynamic adjustments of autonomic service compositions in the closed and open worlds.

RESUMEN

En los últimos años, los servicios Web se han popularizado. Ésto se debe a que ellos permiten que negocios intercambien datos y lógica de procesos de negocio a través de una interfaz programática sobre la red. Para alcanzar el mayor potencial de los servicios Web, éstos se pueden combinar para alcanzar funcionalidades específicas.

Los servicios Web se ejecutan en contextos complejos en donde eventos que surgen pueden comprometer la calidad del sistema (por ejemplo, un ataque repentino de seguridad). Como resultado, es deseable contar con mecanismos para adaptar composiciones de servicios Web (o simplemente *composiciones de servicios*) de acuerdo a eventos problemáticos en el contexto. Ya que sistemas críticos pueden requerir respuestas rápidas, las adaptaciones manuales no son factibles en composiciones de servicios grandes e intrincadas. Por lo tanto, es adecuado contar con mecanismos autonómicos para guiar auto-adaptaciones. Una forma de alcanzar ésto es mediante la implementación de constructores de variabilidad a nivel del lenguaje de programación. Sin embargo, esta aproximación puede resultar tediosa, difícil de manejar, y propensa a errores a medida que el número de configuraciones de la composición de servicios crece.

El objetivo de esta tesis es proveer un marco, apoyado por herramientas, para guiar ajustes autonómicos de composiciones de servicios sensibles al contexto usando modelos en tiempo de ejecución. Este marco cubre el tiempo de diseño y el tiempo de ejecución para enfrentar eventos conocidos y desconocidos en el contexto (ésto es, previstos y no previstos en tiempo de diseño) en el mundo cerrado y abierto respectivamente.

En tiempo de diseño, proponemos la creación de los modelos que guían cambios autonómicos. Con el fin de alcanzar adaptaciones óptimas, un modelo de variabilidad y sus posibles configuraciones son verificadas en tiempo de diseño. En tiempo de ejecución, el modelo de variabilidad se utiliza para guiar los cambios autonómicos de la composición de servicios cuando eventos problemáticos ocurren en el contexto. La activación y desactivación de características en el modelo de variabilidad resultan en cambios en un modelo de composición que abstrae la composición de servicios subyacente. Los cambios en el modelo de composición son reflejados en la composición de servicios al añadir o remover fragmentos de código en el Lenguaje de Ejecución de Procesos de Negocio con Servicios Web (WS-BPEL).

Bajo la asunción del mundo cerrado, los posibles eventos del contexto se conocen totalmente en tiempo de diseño. Estos eventos eventualmente dispararán la adaptación dinámica de la composición de servicios. Sin embargo, es difícil prever todas las posibles situaciones que ocurren en contextos inciertos en donde las composiciones de servicios se ejecutan. Por lo tanto, el marco propuesto también cubre la evolución dinámica de composiciones de servicios para enfrentar eventos inesperados en el mundo abierto. Si las adaptaciones dinámicas no son suficientes para resolver la incertidumbre, entonces los modelos auto-evolucionan de acuerdo a tácticas abstractas que preservan los requisitos esperados.

La propuesta ha sido evaluada a través de un caso de estudio y simulaciones. Las respuestas a varias preguntas de investigación demuestran la factibilidad de usar modelos en tiempo de ejecución para guiar ajustes dinámicos en composiciones de servicios autonómicas en el mundo abierto y cerrado.

RESUM

En els darrers anys, els serveis Web s'han popularitzat. Açò es deu al fet que permeten que les empreses intercanvien dades i lògica de processos de negoci a través d'una interfície programàtica sobre la xarxa. Per tal d'aconseguir el major potencial dels servicis Web, aquestos es poden combinar per a aconseguir funcionalitats específiques.

Els serveis Web s'executen en contextos complexos on certs esdeveniments que sorgeixen poden comprometre la qualitat del sistema (per exemple, un atac sobtat de seguretat). Com a resultat, és desitjable comptar amb mecanismes que permeten adaptar les composicions de serveis Web (o simplement composicions de serveis) d'acord amb esdeveniments problemàtics en el contexte. A causa que sistemes crítics poden requerir respostes ràpides, les adaptacions manuals no són factibles en composicions de serveis grans i intricades. Per tant, és adient comptar amb mecanismes autonòmics per tal de guiar les autoadaptacions. Una forma d'aconseguir aquest objectiu és mitjançant la implementació de constructors de variabilitat a nivell del llenguatge de programació. Malgrat açò, aquesta aproximació pot resultar tediosa, difícil de gestionar, i propensa a errors a mesura que el nombre de configuracions de la composició de serveis creix.

L'objectiu d'aquesta tesi és proveir un marc, suportat amb un conjunt d'eines, que guie els ajustos autonòmics en les composicions de serveis sensibles al contexte mitjançant models en temps d'execució. Aquest marc ha de donar suport tant en temps de disseny com en temps d'execució per afrontar esdeveniments coneguts i desconeguts en el contexte (és a dir, esdeveniments prevists i imprevists en temps de disseny), tant en el món tancat com en l'obert respectivament.

En temps de disseny, proposem la creació dels models que guien els canvis autonòmics. Per tal d'aconseguir adaptacions òptimes, es verifiquen en temps de disseny un model de variabilitat i les seues possibles configuracions són verificades en temps de disseny. En temps d'execució, el model de variabilitat s'empra per a guiar els canvis autonòmics de la composició de serveis quan esdeveniments problemàtics succeeixen en el contexte. L'activació i desactivació de característiques en el model de variabilitat es tradueix en canvis en un model de composició que abstrau la composició de serveis subjacent.

Els canvis en el model de composició és reflexen en la composició de serveis afegint o el.liminant fragments de codi en el Llenguatge d'Execució de Processos de Negoci amb Serveis Web (WS-BPEL).

Sota l'assumpció d'un món tancat, tots els possibles esdeveniments del contexte es coneixen totalment en temps de disseny. Aquestos esdeveniments poden desencadenar l'adaptació dinàmica de la composició de serveis. No obstant açò, és difícil preveure totes les possibles situacions que esdevenen en contextos incerts on s'executen les composicions de serveis. Per tant, el marc proposat també dona suport a l'evolució dinàmica de composicions de serveis per tal d'afrontar esdeveniments inesperats en el món obert. Si les adaptacions dinàmiques no són suficients per a resoldre la incertesa, llavors els models autoevolucionen d'acord amb tàctiques abstractes que preserven els requeriments esperats.

La proposta ha sigut avaluada a través d'un cas d'estudi i mitjançant simulacions. Les respostes a diverses preguntes de recerca demostren la factibilitat d'emprar models en temps d'execució per a guiar els ajustos dinàmics en composicions de serveis autonòmiques, tant en el món obert com en el tancat.

Complex
Adaptation
Quality
Response
World
Runtime
Functionalities
Models
Validation
Weaving
Dynamic
Known
Naturalistic
Deployment
Compositions
Guide
Evolution
Tools
Preserve
Framework
Open
Variability
Uncertainty
Services
Design
Infrastructure
Requirements
Unknown
Support
Unexpected
Observation
Deactivation
Simulations
Tactics
Events
Computing
Critical
Mechanisms
Adjustments
Web
Business
Fragments
Awareness
Features
Closed
Abstractions
Prompt
Context.
WS-BPEL
Autonomic
Changes
Verification
Activation

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AOP	Aspect-Oriented Programming
AC	Autonomic Computing
AI	Artificial Intelligence
AMW	ATLAS Model Weaver
API	Application Programming Interface
ATL	ATL Transformation Language
BDD	Binary Decision Diagrams
BP	Business Process
BPEL4WS	Business Process Execution Language for Web Services
BPM	Business Process Management
BPMN	Business Process Model and Notation
BVR	Base-Variation-Resolution
CORE	Computing Research and Education Association of Australasia
CP	Constraint Programming
CPU	Central Processing Unit
CRI	Centre de Recherche en Informatique
CSP	Constraint Satisfaction Problem
CVL	Common Variability Language

DAML-S	DARPA Agent Markup Language for Services
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
DSPL	Dynamic Software Product Line
DSPLE	Dynamic Software Product Line Engineering
ebXML	Electronic Business using eXtensible Markup Language
ECA	Event Condition Action
ECOWS	European Conference on Web Services
EMF	Eclipse Modeling Framework
EPF	Eclipse Process Framework
FAMA-FW	Feature Model Analyzer Framework
FEFEM	Forms Editors Framework for Ecore Models
FMF	Forms Modeling Framework
GMF	Graphical Modeling Framework
GQM	Goal/Question/Metric
GRL	Goal-oriented Requirements Language
HTTP	Hypertext Transfer Protocol
Hz	hertz
IBM	International Business Machines Corporation
ICWS	International Conference on Web Services
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
JCIS	Jornadas de Ciencia e Ingeniería de Servicios
JCR	Journal Citation Reports

JSS	Journal of Systems and Software
JVM	Java Virtual Machine
KAOS	Knowledge Acquisition in Automated Specification of Software
MAPE-K	Monitor, Analyze, Plan, Execute and Knowledge
MB	megabytes
MBA	Master of Business Administration
MBE	Model-based Engineering
Mbps	Megabits Per Second
MDA	Model-driven Architecture
MDD	Model-driven Development
MDE	Model-driven Engineering
MODELS	International Conference on Model Driven Engineering Languages and Systems
MoRE	Model-based Reconfiguration Engine
MoRE-WS	Model-based Reconfiguration Engine for Web Services
MOSKitt	Modeling Software Kit
MOSKitt4SPL	Modeling Software Kit for Software Product Lines
ms	milliseconds
NASA	National Aeronautics and Space Administration
NFR	Non-Functional Requirement
nm	nanometers
NP	Non-deterministic Polynomial-time
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OSGi	Open Services Gateway Initiative

OWL	Web Ontology Language
PC	Personal Computer
PLE	Product Line Engineering
PROS	Research Center on Software Production Methods
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
SAS	Self-Adaptive Software
SAT	Satisfiability
SCA	Service Component Architecture
SEI	Software Engineering Institute
SLA	Service-Level Agreement
SMS	Short Message Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOPL	Service-Oriented Product Line
SPARQL	SPARQL Protocol and RDF Query Language
SPEM	Software and Systems Process Engineering Meta-Model
SPL	Software Product Line
SPLC	International Software Product Line Conference
SPLE	Software Product Line Engineering
StAX	Streaming API for XML
STP	SOA Tools Platform
UDDI	Universal Description, Discovery and Integration

ULS	Ultra-Large-Scale
UML	Unified Modeling Language
VARY	VARiability for You
W3C	World Wide Web consortium
WAR	Web application ARchive
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Services Description Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XPATH	XML Path Language

Part I

PRELIMINARIES

Bears hibernate during winter to survive.

Photo credit: Louise Staalsen, Flickr, 2013.

In nature, adaptation contributes to the survival of individuals to cope with the weather, enemies, or any hazards. For example, bears hibernate in the winter and possums play dead to avoid predators. As organisms live in intricate, changing environments, Web service compositions are executed in complex, heterogeneous and highly-intertwined computing infrastructures in which a diversity of events may arise. Thus, it is desirable to translate the ideas of adaptation in the natural world to service compositions in order to solve these situations. In this part, we introduce a model-driven approach to achieve autonomic Web service compositions that can self-adjust according to situations in the computing infrastructure. Part I is structured as follows. Chapter 1 presents the introduction of this thesis. Chapter 2 presents the background. Chapter 3 presents the state of the art. Chapter 4 presents the overview of our approach.

1

INTRODUCTION

"The last thing one discovers in composing a work is what to put first." — Blaise Pascal

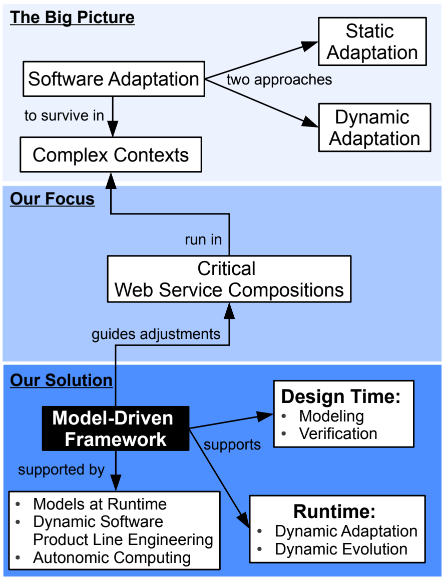


Figure 1.1: Scope of Chapter 1.

In nature, adaptation involves the adjustment of an organ to the quality or intensity of stimulation, or even the modification of an organism or any of its parts to make it more suitable to an environment. Therefore, adaptation contributes to the survival of individuals to cope with the weather, enemies, or any hazards. For example, some species of chameleons adjust their colors for camouflage in accordance with the vision of specific predator species (Stuart-Fox et al., 2008). In many cases, adaptations are carried out automatically; otherwise, we would be constantly busy consciously adapting to the environment. For example, our pupils get wider in the dark but narrower in light.

As organisms live in intricate, changing environments, software is executed in complex, heterogeneous and highly-intertwined computing infrastructures in which a diversity of events may arise. For example, security threats, network problems, performance reduction in one of the servers, etc. Thus, it is desirable to translate the ideas of adaptation in the natural world to software in order to solve these situations. In this way, software can be adapted according to situations in the computing infrastructure to continue offering the required functionality.

Software adaptation can be seen as the ability for humans to reconfigure the software and then restart it, or the ability of the software to reconfigure itself during execution (Akkawi et al., 2007). The first case can be referred to as *static adaptation* and the second one as *dynamic adaptation* (see Figure 1.1). It is possible to carry out static adaptations in cases where the system can be shut down in order to make the required manual adaptations. However, there are critical systems that cannot be stopped to implement the adaptations, e.g. software that run power grids and software for global banking. In such cases, software needs to dynamically adapt its behavior at runtime in response to changing conditions in its supporting computing infrastructure (McKinley et al., 2004; Cetina et al., 2009; Alférez and Pelechano, 2011a). *Dynamic adaptation* of software behavior refers to the act of changing the behavior of some part of a software system as it executes, without stopping or restarting it (Keeney, 2004).

Critical systems need to dynamically adapt at runtime.

In order to carry out dynamic adaptations, we argue that software needs to take the following key issues into account:

- `Context Awareness`: For the purpose of supporting dynamic adaptations, software should be aware of changes in its context. The *context* is any information that can be used to characterize the situation of an entity (Dey, 2001). Context-aware systems are concerned with the acquisition of context, the abstraction and understanding of context, and application behavior based on the recognized context (Schmidt, 2002).

- `Adaptation Policies`: *Adaptation policies* change the behavior of the system during execution (Morin et al., 2008). They state in a declarative manner the actions required to adapt the running system to a configuration that better fits its current context.

- `A Supporting Infrastructure`: It is unthinkable to depend on manual adaptations because of the inherent intricacy of today's systems and the desired prompt responses. Furthermore, critical systems cannot be stopped in order to carry out the necessary adaptations. Thus, a computing infrastructure should provide support for dynamic adaptations to face context events (Alférez and Pelechano, 2011a; Cetina et al., 2009).

A good example of systems that require dynamic adaptations are the ones based on Web service compositions (or simply called *service compositions*). Web services have evolved as a standardized and technology-agnostic interoperable way of integrating processes and applications (Little, 2003). Basically, a Web service is a special software component that is searched, bound, and executed at runtime and allows systems to interact through standard Internet protocols (Koning et al., 2009). In order to reach the full potential of Web services, they can be combined or composed to achieve specific functionalities: if the implementation of a Web service's business logic involves the invocation of other Web services, it is called a *composite service*. The process of assembling a composite service is called *service composition*.

Several exceptional situations may arise in the complex, heterogeneous, and changing contexts where Web service operations run.

Web services run in a context (e.g. their operating computing infrastructure). In an ideal scenario, Web service operations would do their job smoothly. However, several exceptional situations may arise in the complex, heterogeneous, and changing contexts where they run. For instance, a Web service operation may have greatly increased its execution time or may have become unavailable. Cases like these make evident the need for dynamic adaptations in critical systems that are based on service compositions. These adaptations may be triggered in order to do the following at runtime: keep certain contracts known as Service-Level Agreement (SLA)s, offer extra functionality depending on the context, protect the system, or make the system more usable. Implementing dynamic adaptations with variability constructs at the language level can become complex and error-prone, specially in large systems (Fleurey and Solberg, 2009). Moreover, predefined adaptation actions for fully foreseen context events are not enough in the unpredictable open world. In the open world, service compositions have to react to continuous and unanticipated changes in uncertain contexts.

This thesis provides a framework based on models at runtime to guide dynamic adjustments of context-aware service compositions.

In order to tackle the aforementioned situations, this thesis provides a framework based on models at runtime to guide dynamic adjustments of context-aware service compositions. A semantically rich variability model supports the dynamic adaptation of service compositions. When a problematic event arises in the context, this model is leveraged for decision-making. The activation and deactivation of features in the variability model results in changes in a composition model that abstracts the underlying service composition. These changes are reflected into the service composition by adding or removing fragments of Web Services Business Process Execution Language (WS-BPEL) code, which can be deployed at runtime. WS-BPEL is a standard language for specifying Business Process (BP) behavior based on Web Services (OASIS, 2007). In order to reach optimum adaptations, the variability model and its possible configurations are verified at design time using Constraint Programming (CP). If model adaptations are not enough to solve uncertainty at runtime, we provide a solution to guide the evolution of the supporting models to preserve high-level requirements.

The rest of this chapter is organized as follows: Section 1.1 explains the purpose of this work. Section 1.2 details the problems that the present thesis solves. Section 1.3 introduces the research challenges associated with this thesis. Section 1.4 describes the proposed approach. Section 1.5 describes the research methodology that was used in this thesis. Section 1.6 explains the context in which this thesis has been performed. Finally, Section 1.7 gives an overview of the structure of this thesis.

1.1 MOTIVATION

Increasingly, software needs to adapt at runtime in response to arising context events (Cetina et al., 2009; McKinley et al., 2004). In fact, adaptability is emerging as a necessary underlying capability of highly-dynamic context-aware systems (Hong et al., 2009). This dissertation focuses on systems that are based on service compositions, which are composed of Web service operations that run on dynamic, heterogeneous, and complex contexts.

When facing problematic context events, it is unthinkable to manually reconfigure critical service compositions due to their intricacy and required prompt response. Therefore, Autonomic Computing (AC) (Horn, 2001) is an approach that can help to solve this issue with service compositions that change without the need for human intervention. In this way, tasks such as self-configuration, self-optimization, self-healing, and self-protection can be automated. For example, service operations can be added, removed, replaced, optimized, protected, or healed due to security attacks, failures in specific Web services, slow processing in one of the servers, etc.

It is unthinkable to manually reconfigure critical service compositions.

Related work on dynamic adaptation of service compositions has traditionally tended to focus on two groups: 1) the first group supports dynamic adaptations with variability constructs at the language level (Colombo et al., 2006; Charfi and Mezini, 2007; Koning et al., 2009; Baresi and Guinea, 2011). The drawback of this approach is that it can hinder reasoning about adaptations with complex and error-prone scripts (Fleurey and Solberg, 2009); and 2) the second group focuses on low-level implementation mechanisms for self-adaptation (e.g. brokers) (Zeng et al., 2004; Berbner et al., 2006; Ezenwoye and Sadjadi, 2007; Ardagna et al., 2007; Moser et al., 2008; Mosincat and Binder, 2008; Canfora et al., 2008; Karastoyanova and Leymann, 2009; Cardellini et al., 2010). However, most of the research works in this group lack support for analyzing the inherent variability of dynamic adaptation at design time. In order to fill this gap, several authors have proposed BP-variability-modeling approaches, which can be applied to the case of service compositions (Puhlmann et al., 2005; Rosemann and Van der Aalst, 2007; Razavian and Khosravi, 2008; Gottschalk et al., 2008; Hadaytullah et al., 2009; Sun et al., 2010; Nguyen et al., 2011). Nevertheless, the proposed variability models are only used at design time. We argue that the knowledge in variability models could be leveraged at runtime to guide adaptations and hide the complexity of the adaptation space. Moreover, it is important to notice that the aforementioned research works lack the verification of possible service composition configurations caused by dynamic adjustments. We argue that it is necessary to ensure that new service composition configurations are not invalid in a given contextual situation.

Several research works have recently proposed to use models at runtime as a feasible way to guide dynamic adaptations in domains as diverse as service compositions (Bosloper et al., 2005; Menasce et al., 2011; Franch et al., 2011; Hallsteinsen et al., 2012; Calinescu et al., 2011; Cardellini et al., 2012), mobile devices (Morin et al., 2008), and home automation (Cetina et al., 2009). *Models at runtime* can be defined as causally connected self-representations of the associated system that emphasize the structure, behavior, or goals of the system from a problem space perspective (Blair et al., 2009). In response to changes in the context, the system itself can query these models to determine the necessary modifications in the underlying architecture. Therefore, instead of programming complex scripts to describe adaptation actions to change the behavior of the system during execution, easy-to-understand and technology-independent models can be used to express dynamic adaptations.

Models at runtime can guide dynamic adaptations.

Nevertheless, most current model-driven approaches for dynamic adaptation still tend to be based on the *closed-world* assumption, in which the boundary between system and environment is known ahead

Dynamic adaptation vs. dynamic evolution.

and unchanging (Baresi et al., 2006). Under this assumption, models at runtime can remain stable for a long time. However, in the unpredictable *open world*, software should react to continuous and unanticipated changes in complex and uncertain contexts (Baresi et al., 2006).

Models at runtime should be able to evolve in the open world.

In order to manage uncertainty in the open world, the supporting models should be able to evolve at runtime for a better functioning and system "survival". We define *dynamic evolution* as the process of moving the software to a new version, which cannot be supported by predefined dynamic adaptations, in order to manage unknown context events at runtime (Alférez and Pelechano, 2012a). We refer to *unknown context events* as those arising situations in the context that have not been foreseen at design time. Uncertainty is caused by how the service composition should deal with these unknown context events.

There is a small number of approaches that manage uncertainty using models at runtime. Some of them focus on analyzing the collected context information using an ontology (Ranganathan et al., 2004) and others on modeling uncertainty at the requirements and design phases for latter use at runtime (Welsh et al., 2011; Cheng et al., 2009b; Goldsby and Cheng, 2008). Even though these approaches are interesting, the models that are created at design time to deal with uncertainty do not evolve at runtime. Therefore, the capacity of reaction to face new unknown context events decreases because the initial models are unable to support them. Moreover, this situation affects the feasibility of models at runtime as a means to guide the dynamic evolution of critical service compositions that cannot be stopped to modify the supporting models.

1.2 PROBLEM STATEMENT

The autonomic adjustment of service compositions is not a closed research topic. The above discussion indicates that some problems still need to be considered. First, it is necessary to describe and manage dynamic adaptations in a higher abstraction level. Second, it is necessary to count on tools that guide autonomic changes on services compositions according to the abstractions created at design time. Last but not least, it is necessary to manage unknown context events in the open world in order to preserve expected requirements. We formulate the main research question this thesis aims to answer as follows:

- `Research Question 1`: How to design the autonomic behavior of context-aware service compositions by means of easy-to-understand models?
- `Research Question 2`: How to implement a tool-supported software engineering approach for the development of context-aware service compositions from design time to execution?
- `Research Question 3`: How to dynamically evolve service compositions to protect expected requirements when facing unknown context events in the open world?

We believe that a research related to the questions above can contribute to help researchers and practitioners to move towards a sound and seamless engineering support for context-aware autonomic service compositions.

1.3 RESEARCH CHALLENGES

There are several challenges associated with the objectives of this thesis, which are clustered according to the research questions in Section 1.2:

- Regarding `research question 1`, there is one challenge:

  Although current research works have paved the way towards the dynamic adaptation of service compositions, most solutions have tended to implement dynamic adaptations with variability constructs at the language level (Colombo et al., 2006; Koning et al., 2009; Baresi and Guinea, 2011; Sonntag and Karastoyanova, 2011) (e.g. by extending WS-BPEL code). However, this approach can become complex and error-prone, specially in large systems (Fleurey and Solberg, 2009). Therefore, there is a need to manage adjustments at a higher abstraction level. Specifically, there is a need for easy-to-understand and highly-abstract models that represent the context, the dynamic configurations of the service composition, and the service composition itself. These abstraction can be used at runtime to guide dynamic adjustments that move the service composition to new configurations.

- Regarding `research question 2`, there three challenges to be considered:

1. There is a need for tools that support the tasks that need to be carried out at design time to prepare the service composition for dynamic adjustments. Otherwise, this activity can become time-consuming and error-prone. Moreover, it is unthinkable to depend on manual adjustments of service compositions because of the inherent intricacy of today's systems and the desired prompt responses. Furthermore, critical systems cannot be stopped in order to carry out the necessary adaptations. Thus, there is a need for a computing infrastructure that provides support for dynamic adjustments of service compositions.
2. Adjusting the service composition according to changes in the context is not enough. It is necessary to ensure that new configurations of the service composition are not invalid in a given situation. There is a need to verify possible configurations at design time to avoid invalid configurations during execution.
3. In order to develop autonomic service compositions, it is neither enough to focus on the creation of variability models at design time nor to implement low-level implementation mechanisms for self-adaptation to be used during execution. On one hand, the first approach does not leverage variability models at runtime to guide adaptations. On the other hand, the second approach lacks support for analyzing the inherent variability of dynamic adaptation at design time. Therefore, there is a need for a software engineering approach to develop autonomic service compositions, which spans from design time to runtime.

- Regarding `research question 3`, current research works have focused on the dynamic adaptation of service compositions in the closed world. In the closed world, the boundary between system and context is known ahead and unchanging (Baresi et al., 2006). In this scenario, a set of adaptation actions is predefined for fully foreseen context events (Bosloper et al., 2005; Menasce et al., 2011; Alférez and Pelechano, 2011a). However, in the unpredictable *open world*, service compositions should react to continuous and unanticipated changes in uncertain contexts.

  Recently, the community of models at runtime has shown an increasing interest to use models during execution to face uncertainty in the open world (Aßmann et al., 2011). However, just a small group of works, which are not focused on service compositions, deal with uncertainty by means of models at runtime (Welsh et al., 2011; Cheng et al., 2009b; Goldsby and Cheng, 2008). Moreover, the models that are proposed by these approaches do not evolve at runtime. Therefore, the capacity of reaction to face unknown context events decreases because the initial models are unable to support them.

There is a need to move towards the dynamic evolution of service compositions. When a service composition faces unknown context events during execution, the expected requirements can be protected by new versions of the service composition, which cannot be supported by predefined dynamic adaptations.

1.4 THE APPROACH

The mentioned research challenges in Section 1.3 call for a well-defined approach to achieve autonomic service compositions. To this end, we propose the following strategy. First, the service composition is modeled at design time. Then, we introduce mechanisms to express where and how service compositions can be adjusted to face arising context events. These mechanisms are expressed as easy-to-understand and as highly-abstract as possible. At runtime, we provide a computing infrastructure that detects changes in the context and enables dynamic adjustments. In case of arising unknown context events, the computing infrastructure guides the model-driven dynamic evolution of the service composition to preserve expected requirements.

We propose easy-to-understand and highly-abstract mechanisms to achieve autonomic service compositions.

In general terms, we argue that a service composition can be viewed as the assembly of pieces to deliver functionality; those pieces can be Web services or composite services offered by different providers. Specifically, we propose to abstract service compositions as a set of Software Product Line (SPL) *features* (logical units of behavior specified by a set of functional and non-functional requirements (Bosch, 2000)) in a variability model.

Service compositions are abstracted as a set of SPL features.

In the advent of problematic context events, functional pieces can be added, removed, replaced, split or merged from a service composition at runtime, hence delivering a new service composition configuration. If service compositions are abstracted as a set of SPL features, then adaptation policies can be described in terms of the activation or deactivation of features. This is in line with Dynamic Software Product

Line Engineering (DSPLE), which goes a step further from SPL with the investigation of development issues for reusable and dynamically reconfigurable core assets. In a Dynamic Software Product Line (DSPL), variation points can be bound at runtime, initially when the system is launched to adapt to the current context, as well as during operation to adapt to changes in the context (Hallsteinsen et al., 2008). This fact is of particular interest to reach autonomic service compositions.

The contributions of this thesis.

To make the aforementioned strategy a reality, we propose a **tool-supported framework to guide autonomic adjustments of context-aware service compositions in the closed and open worlds using models at runtime**. Specifically, this framework provides the following contributions:

1. The first contribution is **a set of models to support the dynamic adaptation and the dynamic evolution of service compositions**. In order to support dynamic adaptations, we propose models to abstract the following: 1) the underlying service composition; 2) the variability of the service composition; and 3) the context in which the service composition runs. In order to support dynamic evolutions, we propose models to abstract the following: 1) the corrective actions to deal with uncertainty; and 2) the requirements to be preserved at runtime despite arising unknown context events.

2. The second contribution is **a tool-supported software engineering approach for the development of context-aware service compositions from design time to execution**. The proposed framework covers the tools to create models and other supporting artifacts at design time, and the computing infrastructure to guide dynamic adjustments at runtime.

   At design time, the framework provides software process models that specify the tools for creating the models to guide autonomic changes. Prior execution, we also propose that the variability model and its possible configurations are verified to ensure safe recompositions. To this end, we collaborated with the Centre de Recherche en Informatique (CRI), Université Paris-1 Panthéon-Sorbonne, France[1].

   At runtime, a computing infrastructure implements the components of the reference model for autonomic control loops proposed by the International Business Machines Corporation (IBM)

1 http://www.univ-paris1.fr/centres-de-recherche/cri

(IBM, 2006). This model is sometimes called the Monitor, Analyze, Plan, Execute and Knowledge (MAPE-K) loop. The proposed infrastructure detects problematic events that arise in the context and carries out the necessary adjustments on the service composition.

The infrastructure leverages a variability model to make decisions when a problematic event is detected. The activation and deactivation of features in the variability model result in changes in a composition model that abstracts the underlying service composition. Thus, adaptation policies are described in terms of the activation or deactivation of features. Flexible service composition updates are possible through a DSPL, which binds variation points at runtime to adapt to the current context as well as to context changes (Hallsteinsen et al., 2008).

Changes in the composition model are reflected into the service composition by adding or removing fragments of WS-BPEL code, which are deployed at runtime. Instead of extending the functionality of the WS-BPEL engine, our approach is transparent to the engine (i.e., it is unchanged). Therefore, our approach could be used with several WS-BPEL engines.

3. The third contribution is **an approach based on models at runtime to dynamically evolve service compositions to protect expected requirements when facing unknown context events in the open world**. To this end, we extend the dynamic adaptation of service compositions with a dynamic evolution layer. This extension answers the following questions: 1) which corrective actions can trigger the dynamic evolution of the service composition to preserve the expected requirements when unknown problematic context events are faced?; 2) which requirements can be affected by unknown context events?; and 3) how can the service composition self-evolve to manage arising unknown context events? Answers to these questions are offered through easy-to-understand and highly-abstract models at runtime.

The proposal has been validated through a case study and simulations. We answered ten research questions that encourage the use of models at runtime as a feasible way to guide dynamic adjustments of autonomic service compositions.

1.5 RESEARCH METHODOLOGY

In order to perform the work of this thesis, we have carried out a research project following the design methodology for performing research in information systems as described by (March and Smith, 1995) and (Vaishnavi and Kuechler, 2004). Design research involves the analysis of the use and performance of designed artifacts to understand, explain and, very frequently, to improve on the behavior of aspects of Information Systems (Vaishnavi and Kuechler, 2004). The design cycle consists of 5 process steps: 1) awareness of the problem, 2) solution suggestion, 3) development, 4) evaluation, and 5) conclusion. The design cycle is an iterative process; knowledge produced in the process by constructing and evaluating new artifacts is used as input for a better awareness of the problem. Figure 1.2 presents the research methodology followed in this thesis.

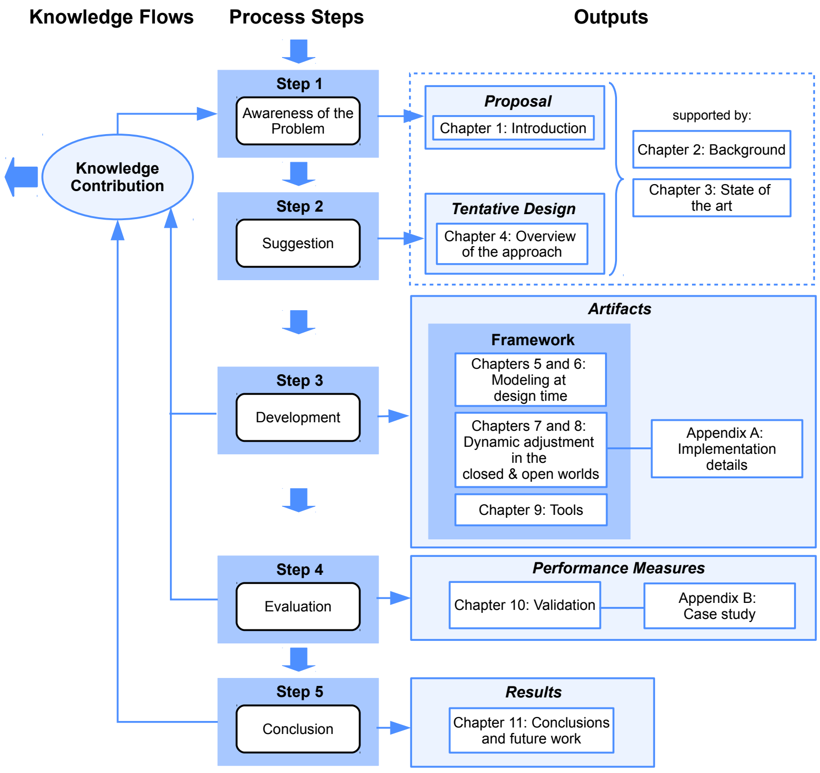


Figure 1.2: Research methodology followed in this thesis.

Following the cycle defined in the design research methodology, we started with the awareness of the problem. The output of this step is in this chapter, which identifies the problem to be solved and states it clearly.

Next, we performed the second step. This step suggests a solution to the problem. The outputs of the first and second steps are intimately connected (see the dotted line around them). The tentative design was refined in several iterations until we achieved a mature overview of the approach, which is described in Chapter 4.

In order to understand the problem and suggest a solution, we supported the outputs of the first two steps with the following: 1) we studied the background of this work, which focuses on autonomic service compositions. Chapter 2 presents this background; and 2) we analyzed the state of the art with the most relevant approaches for achieving autonomic service compositions. Chapter 3 presents the state of the art, which is organized in a taxonomy.

Once the solution of the problem was suggested, in the third step we developed a tool-supported framework to guide autonomic adjustments of context-aware service compositions. Several chapters describe this model-driven framework, which covers the closed and open worlds. First, Chapter 5 and Chapter 6 present the models and other artifacts that are created at design time to support adjustments at runtime. Afterwards, Chapter 7 and Chapter 8 describe the mechanisms to support the model-driven dynamic adaptation and dynamic evolution of service compositions in the closed and open worlds, respectively. Appendix A shows implementation details of these mechanisms. In turn, Chapter 9 presents the tools that support our framework. The key tool at runtime is our MODEL-BASED RECONFIGURATOR, which uses models at runtime to guide autonomic adjustments.

In the fourth step, we performed the evaluation of our solution. To this end, we carried out several experiments. Chapter 10 describes the experimental results. The experiments were carried out mostly on a case study. Appendix B gives technical details of this case study.

In the last step, we analyzed the results of our research work in order to obtain several conclusions. Moreover, we studied areas for further research. Chapter 11 presents the output of this step.

1.6 THESIS CONTEXT

This thesis was developed in the context of the Research Center on Software Production Methods (PROS)[2], Universitat Politècnica de València[3]. The work that has made the development of this thesis possible is in the context of: 1) the sponsorship of Universidad de Montemorelos[4], Mexico; and 2) the EVERYWARE government project: construcción de software adaptativo para la integración de personas, servicios y cosas usando modelos en tiempo de ejecución. CICYT project referenced as TIN2010-18011.

1.7 OUTLINE

Each part is introduced with examples taken from nature.

Researchers and practitioners in AC have learned from nature how to make systems more autonomous (Horn, 2001). In nature, adaptations help organisms to survive in their ecological niche or habitat. This document is further organized as follows. In order to make this document as lifelike as possible, each part is introduced with examples taken from nature:

- `Part I`: Preliminaries:
  - Chapter 2 presents the related background knowledge of this thesis. This chapter covers the enablers that form the foundation of autonomic service compositions.
  - Chapter 3 presents the state of the art. This chapter shows the analysis of the most relevant approaches that have been proposed to achieve autonomic service compositions.
  - Chapter 4 presents the overview of the approach. This chapter introduces the proposed model-driven framework to develop autonomic service compositions. In addition, this chapter introduces a case study that is used to exemplify our approach.
- `Part II`: Paving the Way for the Dynamic Adjustment of Service Compositions:
  - Chapter 5 presents the models and other artifacts that are created at design time for the dynamic adaptation of service compositions in the closed world. This chapter also

2 http://www.pros.upv.es
3 https://www.upv.es
4 http://www.um.edu.mx

describes the verification of the possible reconfigurations to avoid problematic situations during execution. The activities that are carried out at design time to design dynamic adaptations are supported by a software process model.

 - Chapter 6 presents a model-driven approach to prepare for the dynamic evolution of service compositions in the open world. The activities that are carried out at design time to design dynamic evolutions are supported by a software process model.

- `Part III`: Dynamic Adjustment of Service Compositions:
  - Chapter 7 presents the computing infrastructure that deals with dynamic adaptations of service compositions. This infrastructure is based on the components of the MAPE-K loop. In our case, the knowledge that is managed in this loop is based on models at runtime.
  - Chapter 8 presents the mechanisms for the dynamic evolution of service compositions in the open world. The models that are created at design time are used to manage external uncertainty when facing unknown context events at runtime.
- `Part IV`: Applicability and Provability:
  - Chapter 9 presents the tools that support our model-driven approach for the dynamic adjustment of service compositions. The tools for design time are presented in the context of two method contents, which support the software process models for dynamic adaptation and dynamic evolution (described in `Chapter 5` and `Chapter 6`, respectively). This chapter also gives technical details about our Model-Based Reconfigurator, which leverages models at runtime for decision making.
  - Chapter 10 presents the validation of our approach.
- `Part V`: Closing Remarks:
  - Chapter 11 presents the conclusions and discusses future research directions in the area of model-driven dynamic adaptation and dynamic evolution of autonomic service compositions.

- Part VI: Appendices:
  - Appendix A describes the implementation details of the operations that leverage models at runtime to guide dynamic adjustments.
  - Appendix B describes a case study that shows how our approach supports the dynamic adaptation and the dynamic evolution of service compositions.

2

BACKGROUND

"The whole of science is nothing more than a refinement of everyday thinking." — Albert Einstein

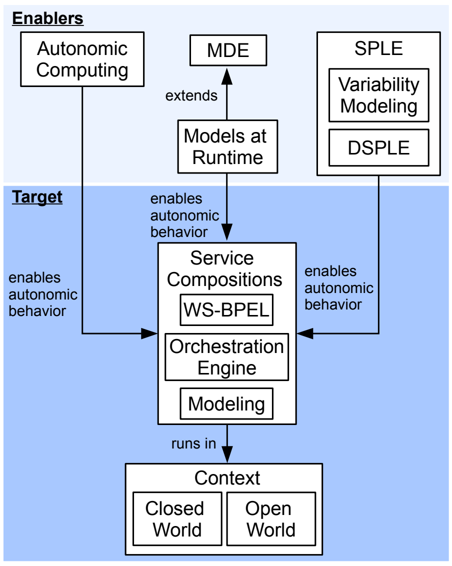


Figure 2.1: Scope of Chapter 2.

This chapter lays the foundations to understand the subjects related to this dissertation. It is composed of two main sections (see Figure 2.1).

Fist, Section 2.1 presents the "`Enablers`", which are the foundation for autonomic service compositions in our approach. These disciplines are as follows: 1) AC is an initiative proposed by IBM in 2001. Its main goal is to develop computer systems with self-management capabilities; 2) Model-driven Engineering (MDE) is a paradigm to build software models that can be then transformed into code. Its main goal is to translate abstract software specifications into a fully functional software product. To date, research on MDE has mainly focused on the use of models during software development. Research on `models at runtime` seeks to extend the applicability of models and abstractions to the runtime environment (Aßmann et al., 2011); and 3) Software Product Line Engineering (SPLE) intends to produce a set of products that share a common set of assets in an specific domain. With SPLE, it is possible to adapt a product according to the customer needs while production costs and time-to-market are decreased. Its main goal is to develop families of systems. This section also discusses two SPLE-related enablers for our approach. First, variability modeling is about abstracting the capability to change or customize a system. Second, DSPLE goes a step further from SPL with the investigation of development issues for reusable and dynamically reconfigurable core assets.

Second, Section 2.2 presents the "`Target`" on which our proposal is applied: service compositions. This section describes the following three aspects in the area of service compositions: 1) WS-BPEL, which is the de facto language to orchestrate service compositions; 2) the orchestration engine that orchestrates the service composition at runtime; and 3) service composition modeling. Finally, since service compositions run in a context, this section describes the concept of context, which is related to the closed and open worlds.

2.1 ENABLERS OF OUR APPROACH

This section presents the enablers that form the basis for our approach: AC, MDE and SPLE. The description of these enablers goes deeper in specific key areas, such as models at runtime, variability modeling, and DSPLE, which enable self-adjustments during execution.

2.1.1 *Autonomic Computing*

Today's systems evidence an explosion of information and technology integration. In fact, there is a continuous evolution towards Ultra-Large-Scale (ULS) systems (Feiler et al., 2006). In addition to its increasing complexity, software systems must become more versatile, flexible, resilient, dependable, energy-efficient, recoverable, customizable, configurable, and self-optimizing by adapting to changes that may occur in their operational contexts. As a result, self-adaptation has become an important research topic (Cheng et al., 2009a; Lemos et al., 2013).

Self-adaptation has become an important research topic.

A commonly used definition of Self-Adaptive Software (SAS) (or Self-Adaptive System) is provided by the Defense Advanced Research Projects Agency (DARPA) (Laddaga, 1998): "Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible." A similar definition is given in (Oreizy et al., 1999): "Self-adaptive software modifies its own behaviour in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation." The key point in SAS is that its life cycle should not be stopped after its development and initial setup. This cycle should be continued in an appropriate form after installation to evaluate the system and respond to changes all the time (Salehie and Tahvildari, 2009).

In 2001, IBM introduced the AC initiative, with the aim of developing SAS (Horn, 2001; Kephart and Chess, 2003; Ganek and Corbi, 2003). The AC initiative is inspired by the human body's autonomic nervous system (IBM, 2006):

The AC initiative is inspired by the human body's autonomic nervous system.

> "Autonomic Computing helps to address complexity by using technology to manage technology. The term autonomic is derived from human biology. The autonomic nervous system monitors your heartbeat, checks your blood sugar level and keeps your body temperature close to 98.6°F without any conscious effort on your part. In much the same way, self-managing autonomic capabilities anticipate IT system requirements and resolve problems with minimal human intervention... However, there is an important distinction between autonomic activity in the human body and autonomic activities in IT systems. Many of the decisions made by autonomic capabilities in the body are involuntary. In contrast, self-managing autonomic capabilities in computer systems perform tasks that IT professionals choose to delegate to the technology according to policies."

Self- Properties*

Upon launching the AC initiative, IBM defined four general adaptivity properties a system should have to constitute self-management (often known as *self-* properties*): self-configuring, self-healing, self-optimizing, and self-protecting. These properties have been defined according to biological self-adaptation mechanisms. For example, the human body has similar properties to adapt itself to changes in its context (e.g. changing environment temperature) or self (an injury in one of the organs) (Kephart and Chess, 2003). These properties are described as follows:

The major self-properties of AC.

- `Self-configuring` is the capability of dynamically adapting to changing environments by installing, updating, integrating, and composing/decomposing software entities (IBM, 2006; Salehie and Tahvildari, 2009). An autonomic system configures itself according to high-level goals (i.e., by specifying what is desired), not necessarily how to accomplish it (Cetina, 2010).

- `Self-healing` is the capability of discovering, diagnosing and reacting to disruptions (IBM, 2006). It can also anticipate potential problems, and accordingly take proper actions to prevent a failure. This property is related to self-diagnosing, which refers to diagnosing errors, faults, and failures, and to self-repairing, which focuses on recovery from them (Salehie and Tahvildari, 2009). As a result of the healing process the system should not be further harmed (e.g. by introducing new bugs or the loss of vital system settings) (Cetina, 2010).

- `Self-optimizing` is the capability to monitor and tune resources automatically to meet end-user or business needs (IBM, 2006). Some important concerns related to this property are response time, throughput, utilization, and workload (Salehie and Tahvildari, 2009).

- `Self-protecting` is the capability to anticipate, detect, identify and protect against threats. Self-protecting components can detect hostile behaviors as they occur and take corrective actions to make themselves less vulnerable (IBM, 2006).

Figure 2.2 illustrates a hierarchy of self-* properties in three levels according to (Salehie and Tahvildari, 2009). The aforementioned core self-properties are categorized into the `Major Level`. The `General Level` contains global properties of SAS. A subset of these properties consists of self-managing, self-governing, self-maintenance, self-control, and self-evaluating. Another subset at this level is self-organizing, which emphasizes decentralization and emergent functionalities. The `Primitive Level` contains the underlying primitive properties. Self-awareness means that the system is aware of its self states and behaviors. This property is based on self-monitoring which reflects what is monitored. Context-Awareness means that the system is aware of its context, which is its operational environment.

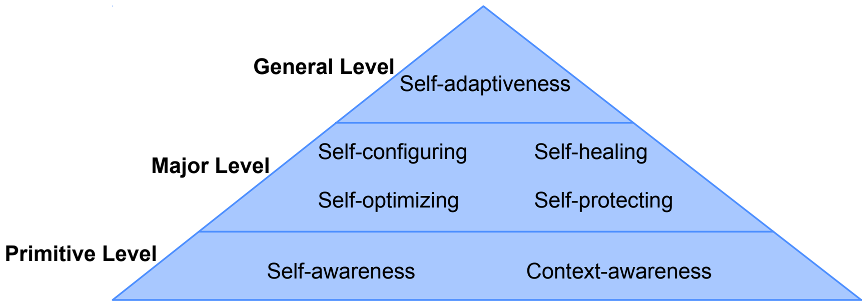


Figure 2.2: Hierarchy of the self-* properties (Salehie and Tahvildari, 2009).

The MAPE-K Loop

According to (Salehie and Tahvildari, 2009), adaptation approaches can be organized into the following categories with respect to the separation of the adaptation mechanism and application logic:

- The `internal approach` interweaves application and adaptation specifications. It is based on programming language features, such as conditional expressions, parametrization, and exceptions (Oreizy et al., 1999; Floch et al., 2006). Since application and adaptation specifications are mixed, this approach can lead to poor maintainability and scalability. It can be used for handling local adaptations (e.g. exception handling). However, adaptation often needs global information about the system and correlating events happening in itself or in the context.

- The `external approach` uses an external subsystem (or engine) containing adaptation processes to control the adaptation of the software system. Therefore, it separates adaptation concerns from other functional concerns of the system. The external subsystem implements the adaptation logic, mostly with the aid of middleware (Kon et al., 2002; Floch et al., 2006), a policy engine (Bigus et al., 2002), or other application-independent mechanisms. According to (Amoui Kalareh, 2012), the major benefits of external approaches are: 1) adaptation requirements are separated from functional requirements; 2) separation of adaptation logic and application business logic; 3) increase of reusability, scalability, and maintainability; 4) support of dynamic adaptation; and 5) applicability of techniques such as Artificial Intelligence (AI) and control theory.

We base our solution on the external autonomic control loop proposed by IBM.

Due to the limitations of the internal approach, we base our solution on the external autonomic control loop proposed by IBM, which is sometimes called the MAPE-K loop (IBM, 2006) (see Figure 2.3). The `Autonomic Manager` continuously runs the monitor, analyze, plan, and execute processes (using shared knowledge) in a closed loop form. The `knowledge` can come from sources as diverse as human experts (such as in static policy based systems (Bougaev, 2005)) to logs that accumulated data from probes charting the day-to-day operation of a system to observe its behavior, which is used to train predictive models (Shivam et al., 2006). The loop is completed by connecting to the adaptable `Managed System` through `Sensors` and `Actuators`. Here is a summary of each process:

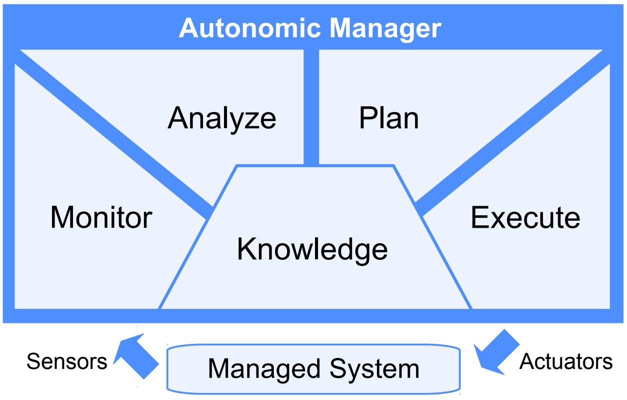


Figure 2.3: MAPE-K reference model for autonomic control loops.

- The `Monitor Process` collects and correlates data and converts them to behavioral patterns and symptoms. The software or hardware components used to perform monitoring are called `Sensors` (e.g. sensors for network latency and bandwidth measure the performance of a network). In (Salehie and Tahvildari, 2009), the authors describe examples of different kinds of sensors that use the following techniques: logging, monitoring and events information models, management protocols and standards, profiling, management frameworks, Aspect-Oriented Programming (AOP), and signal monitoring.

- The `Analyze Process` detects the symptoms provided by the `Monitor Process` and the system's history to detect when a change (response) needs to be applied. It also helps to identify where the source of a transition to a new state (deviation from desired states or goals) is.
- The `Plan Process` determines what needs to be changed, and how to change it to achieve the best outcome.
- The `Execute Process` applies the adaptation actions on the `Managed System` using `Actuators`. In (Salehie and Tahvildari, 2009), the authors describe examples of different kinds of `Actuators` that use the following techniques: design, architectural, and autonomic patterns, middleware-based effectors, metaobject protocol, dynamic aspect weaving, and function pointers.

2.1.2 *Model-driven Engineering*

Abstractions are widely applied in science and technology, where they are usually referred to as *modeling*. A *model* can be informally defined as a simplified or partial representation of the reality, defined to accomplish a task or to reach an agreement on a topic (Brambilla et al., 2012). In MDE, a *model* is an abstraction or reduced representation of a system that is built for specific purposes (e.g. technology-independent software models describe applications using concepts that abstract over the underlying computing technologies) (Blair et al., 2009). Modeling is an effective strategy for managing system complexity (Ludewig, 2003), for example by hiding unnecessary details.

A model is an abstraction or reduced representation of a system that is built for specific purposes.

Nowadays, there are several model-driven acronyms that may be confusing. In (Brambilla et al., 2012), these concepts are demystified (see Figure 2.4). First, Model-driven Development (MDD) is a development paradigm that uses models as the primary artifact of the development process. Usually, in MDD, the implementation is (semi)automatically generated from the models. Model-driven Architecture (MDA) is the Object Management Group (OMG) particular vision of MDD and thus relies on the use of OMG standards. As a result, MDA is a subset of MDD. MDE is a superset of MDD because MDE goes beyond of the pure development activities and covers other model-based tasks of a complete software engineering process, such as the model-based evolution of the system. Finally, Model-based Engineering (MBE) is a softer version of MDE in which models do not drive the process as in MDE.

Demystifying MDA, MDD, MDE, and MBE.

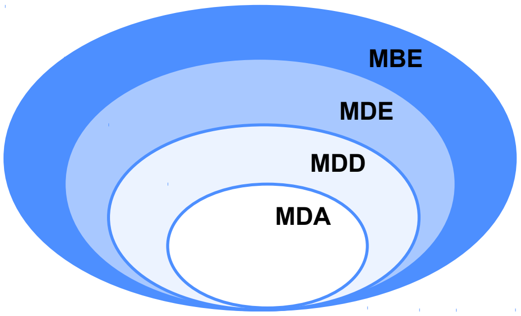


Figure 2.4: Relationships between the different model-driven acronyms (Brambilla et al., 2012).

MDE can be defined as a *methodology* for applying the advantages of modeling to engineering activities (Brambilla et al., 2012). Specifically, MDE combines *process* and *analysis* with architecture (Kent, 2002). MDE intents to raise the level of abstraction in software specification and increase automation in program development. The idea promoted by MDE is to use models at different levels of abstraction for developing systems. The main advantage of this approach is that models can be expressed using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain. This makes the models easier to specify, understand, and maintain (Selic, 2003).

It is expected that MDE techniques will have an increasing adoption.

MDE practices have proven to increase efficiency and effectiveness in software development (Acerbis et al., 2007). In fact, it is expected that MDE techniques will have an increasing adoption, specially in scenarios where the complexity of Service-Oriented Architecture (SOA)s and cloud-based architectures jointly applied with Business Process Management (BPM) demand more abstract approaches than mere coding (Brambilla et al., 2012). Moreover, in the context of SAS, the construction of precise and accurate models of software has proven to be essential to support various aspects of adaptation (Cheng et al., 2009a; Andersson et al., 2009).

2.1.3 *Models at Runtime*

A challenging problem in SAS concerns the complexity that arises from the wealth of information that can be associated with runtime phenomena. A promising approach to manage complexity in runtime contexts is to develop adaptation mechanisms that leverage software models, referred to as `models at runtime` (sometimes called `models@run.time`):

> "A model@run.time is a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective." (Blair et al., 2009)

`Models at runtime` extend the applicability of models produced in MDE approaches to execution time (Aßmann et al., 2011). Since `models at runtime` provide up-to-date and exact information about the runtime system, they can provide a richer semantic base for runtime decision-making in order to achieve system adaptation. Thus, the system itself can query the models at runtime to make adaptation decisions, to choose the adaptation strategy, and to control and to steer the adaptation process.

Development models vs. executable models vs. models at runtime.

For the sake of comprehension, it is important to describe the differences among `development models`, `executable models`, and `models at runtime`. When the models are produced in an MDE process, they are named `development models`. These models are in an abstraction level above the code level (France and Rumpe, 2007) (e.g. use case models). If `development models` are fully expressive to be automatically executed, they are considered `executable models`. `Executable models` can be executed by means of translating them into the system code that will be executed (Mellor and Balcer, 2002) or by using an interpreter/engine that directly executes what is specified (Jurič et al., 2004). When the models are used at runtime, they are considered `models at runtime`. In contrast to `development models`, `runtime models` are used to reason about the operating environment and runtime behavior for some purpose (e.g. to determine an appropriate form of adaptation. Therefore, these models must capture abstractions of runtime phenomena) (Aßmann et al., 2011).

Reflection vs. models at runtime.

In (Blair et al., 2009), the authors state the similarities and differences between `reflection` and `models at runtime`. On one hand, both of them are concerned with defining representations of the underlying system that are both self-representations and causally connected. *Causal connection* means that if the system changes, the representations of the system (the models) should also change, and vice versa (Blair et al., 2009). The causal connection of the self-representation of the system and the system itself is an important requirement for SAS for two reasons: 1) the model provides up-to-date and exact information about the system to drive subsequent adaptation decisions; and 2) if models are causally connected, then adaptations can be made at the model level rather than at the system level.

On the other hand, `reflection` seeks models that are intrinsically related to the computation model and hence tend to be based on the solution space and often rather low level. `Models at runtime` seek models at a much higher level of abstraction and, particularly, causally connected models related to the problem space. Another distinction is that `models at runtime` should be intrinsically tied to the models produced as artifacts from the MDE process. In brief, the idea of `models at runtime` is built on `reflection` but seeks to move from the solution space up to the problem space.

2.1.4 *Software Product Line Engineering*

In the early 20th Century, mass production was popularized by Henry Ford. In 1969, M. Douglas McIlroy coined the term *mass-production development*. He showed the benefits of component-based systems, such as greater reuse (McIlroy, 1969). In 1976, David L. Parnas introduced the notion of *software program families* (Parnas, 1976). In the early 1990s, Kio Kang et al. proposed the use of features to drive mass production (Kang et al., 1990). Shortly, the first conferences appeared turning SPL into a new body of research.

> "A SPL is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" (Clements and Northrop, 2001).

SPLE encompasses the creation and management of products' families for a particular domain. *Domain* is an application area or, more formally, a set of systems that share design decisions, based on a repeatable process, and concerned primarily with the reuse of higher level life-cycle artifacts (Frakes and Isoda, 1994). A newer definition, defines domain as a specialized body of knowledge, an area of expertise, or a collection of related functionality (Northrop, 2002). Most companies build software within a few domains, repeatedly building system variants within those domains. This approach can be used to improve the quality and productivity of the development process (Frakes and Kang, 2005). In SPLE each *product* in the family is derived from a shared set of core assets, following a set of prescribed rules (Clements and Northrop, 2001). A *core asset* can be any relevant artifact in the software process, such as code units (packages, classes, methods, etc.) or models, documentation, configuration files, etc.

Nowadays, companies have a competitive advantage when they understand their *domains* and implement *systematic reuse* (Frakes and Isoda, 1994). *Systematic reuse* is a paradigm switch from building single systems to creating families of related systems. The systematic reuse of core assets in mass-production environments provided by SPL is quickly arising as a viable software development paradigm (SEI, n.d.). In fact, SPLE has yielded notable improvements in time-to-market, product quality, productivity, cost schedule, customer satisfaction, and mass customization (Clements and Northrop, 2001).

Companies have a competitive advantage when they understand their domains and implement systematic reuse.

According to the Software Engineering Institute (SEI), substantial production economies can be achieved when the systems in a SPL are developed from a common set of assets in a prescribed way (SEI, n.d.). This contrasts to software being developed separately, from scratch or in an arbitrary fashion (as in early small-grained reuse efforts (Bergey et al., 2000)). Product Line Engineering (PLE) seeks to achieve *economies of scope*, which are conceptually similar to *economies of scale:*

Substantial production economies can be achieved with SPLs.

> "Economies of scale arise when multiple identical instances of a single design are produced collectively, rather than individually. Economies of scope arise when multiple similar but distinct designs and prototypes are produced collectively, rather than individually." (Greenfield and Short, 2003).

Economies of scope imply *mass customization*, which can be defined as "producing goods and services to meet individual customers' needs with near mass production efficiency" (Tseng and Jiao, 2001).

Products in a SPL are characterized by their features, which are useful to express product functionalities or properties concisely (Pohl et al., 2005). A *feature* can be defined as "a logical unit of behavior specified by a set of functional and non-functional requirements" (Bosch, 2000). Features may be common to all products or vary between products. The terms *commonality* and *variability* are often used to denote the common and variable features within a SPL, respectively.

A *commonality* is a quality or functionality that all the applications in a product family share. As a consequence, commonalities are the elements with the highest reuse potential because an implementation of a commonality is used in all family members.

In contrast, *variability* represents a capability to change or customize a system (Geyer and Becker, 2002). In SPLs, variability is made explicit through variation points. A *variation point* is an identifier of one or more locations in a software asset at which the variation will occur (Brownsword et al., 1996).

SPLE consists of three main processes: `Domain Engineering` (also known as Core Asset Development), `Application Engineering` (also known as Product Development) and `Management`. During `Domain Engineering`, the variability of a SPL is defined and common and variable domain artifacts are developed. During `Application Engineering`, individual products are developed by selecting and configuring shared artifacts and, where necessary, adding product-specific extensions (Bosch et al., 2002). During `Management`, organizational issues are handled (i.e., by giving resources, coordinating, and supervising domain and application engineering activities) (Clements and Northrop, 2001).

Variability Modeling

Variability models can be used to describe SPLs. During the past decade, a number of methods and techniques for describing SPLs have been defined (Bayer et al., 2006). Nowadays, new variability modeling languages, such as the Common Variability Language (CVL) (Haugen et al., 2008), are emerging. In this research, feature modeling was chosen for variability modeling and analysis because it can offer coarse-grained variability management and it has good tool support for variability reasoning (Mazo, 2011; Benavides et al., 2005).

In this research, feature modeling was chosen for variability modeling and analysis.

In a feature model, features are hierarchically linked in a tree-like structure through variability relationships such as optional, mandatory, and alternative (Kang et al., 1990). The categories of parental relationships between a feature and its subfeatures can be: 1) `And`: all subfeatures must be selected. 2) `Alternative`: only one subfeature can be selected; 3) `Or`: one or more subfeatures can be selected; 4) `Mandatory`: required features; and 5) `Optional`: optional features. In addition to the parental relationships between features, cross-tree constraints are allowed. The most common are: 1) `Requires`: the selection of a feature in a product implies the selection of another feature; and 2) `Excludes`: features cannot be part of the same product. In a feature model, there is only one root feature from which all the other features depend on. Primitive features are considered as leaves and compound features are the interior nodes. An instance of a feature model is visually represented by means of a *feature diagram*. Figure 2.5 shows the common notations of a feature model and an example.

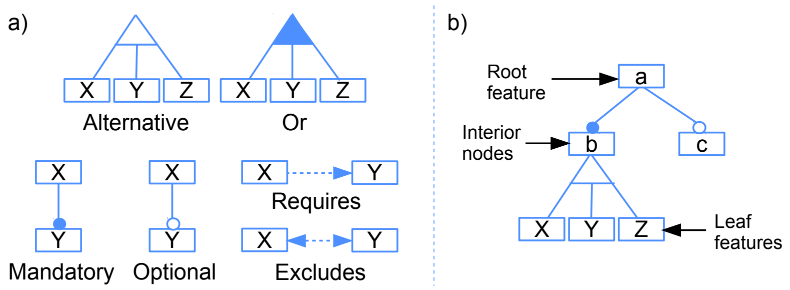


Figure 2.5: Feature model: a) notations and b) example.

An individual product of the SPL is described by a selection of features that obeys the rules of the feature model. Such a collection of selected features is termed a *feature configuration* (Asikainen et al., 2006). It is important not to confuse feature models with part-of hierarchies or decompositions of software modules. *Features may or may not correspond to concrete software modules* (Czarnecki et al., 2004).

Dynamic Software Product Line Engineering

Dynamic variations in user requirements and system environments become more frequent. As a result, there is an increasing need for systems to be able to self-adapt to changes in user needs and evolving resource constraints. To address this need, researchers have investigated DSPLs.

Given the success of SPLs, DSPLs offer a promising strategy to deal with the design and implementation of software changes that need to be handled at runtime. DSPLs extend the concept of conventional SPLs by enabling software-variant generation at runtime (Bencomo et al., 2012). Specifically, "DSPLs bind variation points at runtime, initially when software is launched to adapt to the current environment, as well as during operation to adapt to changes in the environment." (Hallsteinsen et al., 2008).

DSPLs bind variation points at runtime.

Although DSPLs build on the central ideas of SPLs, there are also differences. For example, SPLE typically binds variation points before delivery of the software (i.e., the adaptation of their products is known and occurs during product line development (Bencomo et al., 2012)). In contrast, DSPLE typically is not concerned with pre-runtime variation points. Also, the primary goal of DSPLs is to adapt to variations in individual needs and situations rather than market forces (Hallsteinsen et al., 2008).

According to (Hallsteinsen et al., 2008), a DSPL has many, if not all, of the following properties: 1) dynamic variability: configuration and binding at runtime; 2) changes binding several times during its lifetime; 3) variation points change during runtime: variation point addition; 4) it deals with unexpected changes (in some limited way); 5) it deals with changes by users, such as functional or quality requirements; 6) context awareness (optional) and situation awareness; 7) autonomic or self-adaptive properties (optional); 8) automatic decision making (optional); and 9) individual context situation instead of a "market".

Fields such as AC, domotics, and sensor networks have introduced a niche in which DSPLs can play a key role in supporting runtime changes (Bosch and Capilla, 2012).

2.2 TARGET OF OUR RESEARCH

This section introduces the target of our research: service compositions. Three areas are described in this field: service composition modeling, WS-BPEL, and the role of the WS-BPEL orchestration engine to execute BPs following the WS-BPEL standard. Since service compositions run in a context, this section describes the context concept in the closed and open worlds.

2.2.1 *Service Compositions*

In order to describe the concept of Web service compositions (or simply, service compositions), first it is necessary to understand the role of Web services in SOA. During the last years, software engineering has put an important focus on the SOA paradigm (Papazoglou et al., 2007). The Organization for the Advancement of Structured Information Standards (OASIS) consortium defines SOA as: "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with, and use capabilities to produce desired effects consistent with measurable preconditions and expectations" (OASIS, 2006).

SOA propagates the use of services as a means to create decoupled, distributed, composite applications in heterogeneous environments. A *service* can be defined as: "An autonomous applications made available in a computer network using standardized interface description and message exchange" (Hummer et al., 2011).

In theory, the basic SOA model consists of the following three actors that communicate in a loosely coupled way as shown in Figure 2.6 (this model is often referred to as *the SOA triangle*) (Michlmayr et al., 2010) :

1. `Service providers` implement services and make them available at a certain location (endpoint) in the network.
2. `Service registries` store information about services. `Service providers` can publish their services in such registries.
3. `Service consumers` discover services by querying a service registry, bind to the obtained service references and execute the services' operations.

A `service contract` "embodies the information covering the profiting demand of customers, the expectation of users and the implementation capability of IT technology on target services which will be deployed in a business process and validated by economical analysis"(Duan, 2012).

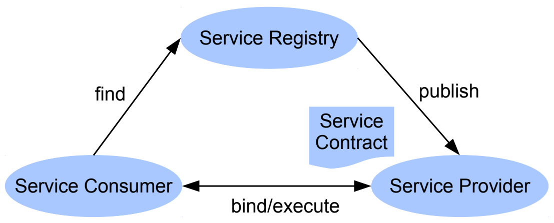


Figure 2.6: SOA model (Michlmayr et al., 2010).

Due to platform-independent service descriptions, it is possible to implement flexible applications with respect to manageability and adaptivity. For example, services can easily be exchanged during execution and `service consumers` can switch to alternative services seamlessly, which increases organizational agility (Michlmayr et al., 2010).

Web Services represent the most common way of implementing SOAs.

Web Services represent the most common way of implementing SOAs, building on the standards Simple Object Access Protocol (SOAP) (W3C, 2007) for communication, Web Services Description Language (WSDL) (OASIS, 2001) for service interface descriptions, and Universal Description, Discovery and Integration (UDDI) (OASIS, 2005) for registries (Michlmayr et al., 2010)[1]. Web services are a key breakthrough to support the openness, heterogeneity, and flexibility of modern information systems (Baresi et al., 2007). They have evolved as a standardized and technology-agnostic interoperable way of integrating processes and applications (Little, 2003).

The most accurate definition of Web services is provided by the World Wide Web consortium (W3C): "A Web Service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols" (W3C, 2002).

According to (Alonso et al., 2004), the aforementioned definition hints at how Web services should work. The definition states that Web services should be "services" similar to those in conventional middlewares. They should be described and advertised so that it is possible to write clients that bind and interact with them. Therefore, it is possible to say that a Web service is a special software component (Koning et al., 2009). The W3C also states that the XML is a key part of the solution. Specifically, XML is used as a common language to render all exchanged artifacts, and a new interaction paradigm, which may support the dynamic configuration of BPs (Baresi et al., 2007). Although the role of service contracts is not evident in this definition, Web service providers are usually bound by a contract called SLA to provide a certain level of Quality of Service (QoS) (Koning et al., 2009).

In order to reach the full potential of Web services, they can be combined to achieve specific functionalities. If the implementation of a Web service business logic involves the invocation of other Web services, it is called a *composite service* (e.g. a stock-trading Web service

1 It has been argued that currently the SOA triangle is actually broken (Michlmayr et al., 2007), since the binding between consumer and service provider often happens at design-time and service registries are rarely used in practice. According to (Michlmayr et al., 2010), this is largely due to the limited success of service registry standards such as UDDI (OASIS, 2005) and the Electronic Business using eXtensible Markup Language (ebXML) (OASIS, 2012).

could invoke a payment service, which could then invoke an authentication service (Menasce, 2004)). The process of assembling a composite service is called *service composition* (Chakraborty et al., 2002). Service composition accelerates application development and enables service reuse (Milanovic and Malek, 2004; Khadka and Sapkota, 2010). There have been several proposals for services composition, for instance WS-BPEL, Semantic Web, Web Components, Algebraic Process Composition, Petri Nets, Model Checking, and Finite-State Machines (Milanovic and Malek, 2004).

The process of assembling a composite service is called service composition.

WS-BPEL

Orchestration and choreography are the two different viewpoints to represent the behavior of service compositions. These two approaches can be defined as follows (Peltz, 2003a,b):

- `Orchestration`: Refers to an executable BP that may interact with both internal and external Web services. It describes how Web services can interact at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional process. The process is always controlled from the perspective of one of the business parties.

- `Choreography`: Each party involved in the process describes the part they play in the interaction. It tracks the sequence of messages that may involve multiple parties and multiple sources. It is associated with the public message exchanges that occur between multiple Web services.

Orchestration differs from choreography in that it describes a process flow between services, controlled by a single party. Orchestration is imperative and contains enough information to enable execution of BPs by an `orchestration engine`. More collaborative and descriptive in nature, choreography tracks the sequence of messages involving multiple parties, where no one party "owns" the conversation (see Figure 2.7).

This research focuses on autonomic adjustment of service compositions from the orchestration point of view. From the set of available service composition options, we have chosen WS-BPEL (OASIS, 2007) because it is the de facto standard language for orchestrating composite Web services.

This research focuses on the orchestration point of view using WS-BPEL.

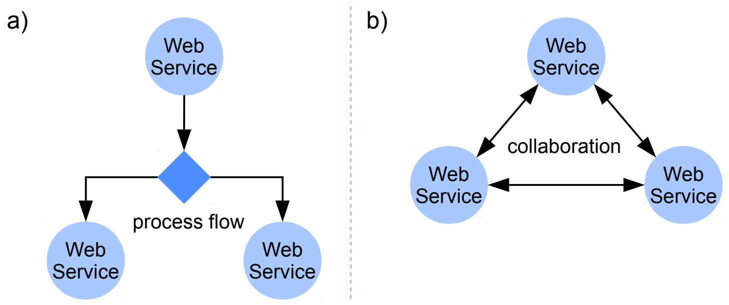


Figure 2.7: a) Orchestration and b) choreography (Peltz, 2003a).

WS-BPEL combines two different approaches to specify BPs, namely a programmatic (or block-oriented) approach (stemming from XLANG[2]) as well as a graph-oriented approach (stemming from WSFL[3]) (Leymann, 2011). The combination of these two approaches is of utmost importance: both approaches had been supported by products at the time the work on WS-BPEL began.

The WS-BPEL language provides a high-level XML-based notation and semantics for specifying BPs behavior based on Web services. In WS-BPEL, the composition result is called a *process*, participating services are *partners*, and message exchange or intermediate result transformation is called an *activity*. Therefore, a process consists of a set of activities. A process interacts with external partner services through a WSDL interface (OASIS, 2001). WS-BPEL orchestrates services by specifying the order in which it is meaningful to call a collection of services, and assigns responsibilities for each of the services to partners.WS-BPEL does not come with a standard graphical representation.

WS-BPEL does not come with a standard graphical representation.

To define a process, it is possible to use the following elements (Milanovic and Malek, 2004): 1) a WS-BPEL source file (`.bpel`), which describes activities. In this work, it is called the `composition schema`; 2) a process interface (`.wsdl`), which describes ports of a composite service; and 3) an optional deployment descriptor (`.xml`), which contains the partner services physical locations (a partner service's implementation and location can be changed without modifying the source file).

WS-BPEL has several element groups, but the basic ones are the following (Milanovic and Malek, 2004):

- process initiation: `<process>`

2 http://msdn.microsoft.com/en-us/library/aa577463.aspx
3 http://www.ibm.com/developerworks/library/ws-ref4

- definition of services participating in the composition: `<partnerLink>`
- synchronous and asynchronous calls: `<invoke>`, `<invoke>...<receive>`
- intermediate variables and results manipulation: `<variable>`, `<assign>`, `<copy>`
- error handling: `<scope>`, `<faultHandlers>`
- sequential and parallel execution: `<sequence>`, `<flow>`
- logic control: `<switch>`

Listing 2.1 illustrates a simplified example of a WS-BPEL process. A WS-BPEL process consists of activities (such as `<invoke>` and `<assign>`), which are basically execution steps, and activity containers (such as `<sequence>`), which provide information about the execution of the contained activities. Listing 2.1 also shows how WS-BPEL supports message exchanges between activities. Messages are basically treated as complex variables. The `<receive>`, `<invoke>`, and `<reply>` activities are all message-related statements. There are also control-flow activities such as if-statements and while-statements and variable-related activities such as the `<assign>` activity in the example.

Listing 2.1: A simplified example of a WS-BPEL process definition (Koning et al., 2009).

```
<process name="loanApprovalProcess">
 <sequence>
  <receive partnerLink="customer"
        operation="request" .../>

  <invoke partnerLink="assessor"
    operation="check"
    inputVariable="request"
    outputVariable="risk"/>

  <assign>
   <copy>
    <from>
     <expression>'yes'</expression>
    </from>
    <to variable="approval"
     part="accept"/>
   </copy>
  </assign>

  <reply partnerLink="customer"
        operation="request"
    variable="approval"/>
 </sequence>
</process>
```

Orchestration Engine

WS-BPEL is supported by commercial vendors and also the open-source community who have contributed with good quality WS-BPEL editors and efficient orchestration engines, such as Microsoft BizTalk[4], Oracle BPEL Process Manager[5], and ActiveVOS[6].

Apache ODE provides a mature support for hot-deployment.

In this work, Apache ODE[7] was chosen for orchestrating the service composition because it provides a mature support for hot-deploying the composition schema. Apache ODE executes BPs written following the WS-BPEL standard. It talks to Web services, sending and receiving messages, handling data manipulation and error recovery as described by the composition schema.

Apache ODE supports both long and short living process executions to orchestrate all the services that are part of the application. Based on the IBM's BP categorization (IBM, 2005),

- A `long-running` BP "executes over an extended period of time, and is much more flexible and resilient than a microflow. Interruptible BPs and asynchronous BPs are examples of long running processes". Examples of this sort of BPs are the book loan and return service offered by libraries. In general, these processes involve not only the coordination of different systems but also the coordination of different people behaving with different roles.

- A `short-running` BP "is contained within a single transaction. This is ideal for situations where the user is expecting an immediate response". Good examples of processes of this type are the checkout process, which is usually found in online stores, or the booking service, usually found in online travel agencies.

4 http://www.microsoft.com/en-us/biztalk
5 http://www.oracle.com/technetwork/middleware/bpel/overview
6 http://www.activevos.com
7 http://ode.apache.org

Apache ODE has the following features: 1) side-by-side support for both WS-BPEL 2.0 and the legacy Business Process Execution Language for Web Services (BPEL4WS) 1.1 vendor specification; 2) supports two communication layers: one based on Axis2[8] (Web Services Hypertext Transfer Protocol (HTTP) transport) and another one based on the JBI[9] standard (using ServiceMix[10]); 3) support for the HTTP WSDL binding, allowing invocation of Representational State Transfer (REST)-style Web services; 4) possibility to map process variables externally to a database table; 5) high level Application Programming Interface (API) to the engine that allows to integrate the core with virtually any communication layer; 5) process hot-deployment; 6) compiled approach to WS-BPEL that provides detailed analysis and validation at the command line or at deployment; and 7) management interface for processes, instances, and messages.

In Apache ODE, processes are deployed in a `deployment bundle`. It is a compressed file or a directory containing Apache ODE's deployment descriptor (`deploy.xml`), the composition schema (the WS-BPEL file), and all the other artifacts necessary for the WS-BPEL to run (e.g. WSDLs). When a deployment bundle is redeployed in Apache ODE, here is what happens: 1) a new version is attributed to the bundle by incrementing the version number of the last deployment; 2) Apache ODE checks whether the same bundle has been deployed before, all processes in those older bundles are retired; 3) the processes in the bundle are deployed in the engine using the same version number as the bundle itself; and 4) new executions of all newly deployed processes are ready to be started. The version is a single, sequentially incremented number. All deployed bundles share the same sequence.

Whenever a WS-BPEL process is started in Apache ODE, a new instance of that process is created. An *instance* can be defined as the instantiation of the composition schema. An instance completes when its last activity completes, a terminate activity runs, or a fault occurs that is not handled by the process.

Service Composition Modeling

According to (Leymann, 2011), the most important aspect of WS-BPEL is that it is supported by most BPM middleware vendors. This results in the portability of skills and in the interoperability of process models

8 http://axis.apache.org/axis2/java/core
9 http://jcp.org/en/jsr/detail?id=208
10 http://servicemix.apache.org

between tools. Former attempts in this area did not achieve this at such a broad scale (Leymann et al., 2010). Therefore, WS-BPEL is an important cornerstone of BPM technology.

WS-BPEL lacks visual representation of BPs.

Nevertheless, WS-BPEL left the visual representation dimension of BP models completely out of scope. Although some WS-BPEL tools offer graphical editing tools for defining WS-BPEL processes, they follow the syntax of WS-BPEL without elevating the level of abstraction to make them usable during analysis and design (Ouyang et al., 2006). This visual dimension became the domain of Business Process Model and Notation (BPMN) 1.x (White, 2004), which now fills this gap (White, 2005)[11].

BPMN 1.x enables non-IT experts to communicate and mutually understand their models by providing a visual modeling language for BPs. With BPMN 1.x, it is possible to visualize concepts such as human tasks, executable scripts, and automated decisions in a vendor-neutral standardized way. Therefore, BPMN 1.x became quickly popular within the business analyst audience.

The use of WS-BPEL engines for BPMN execution is an obvious choice since there is a widespread support of WS-BPEL by middleware vendors. In order to support the execution of BPMN process models in WS-BPEL engines, the transformation of BPMN process models to WS-BPEL process models is required. However, the metamodels underlying BPMN and WS-BPEL are quite different. Therefore, the transformation is not straightforward and sometimes requires complex mappings resulting in a WS-BPEL process model that looks quite different from the original BPMN process model (Leymann, 2011). There are several publications related to this transformation (Ouyang et al., 2006, 2007; White, 2004, 2005; Biermann and Ermel, 2009; Doux et al., 2009). They provide partial solutions as the translation of BPMN to WS-BPEL is far from trivial (Recker and Mendling, 2006; Ouyang et al., 2007; Schumm et al., 2009).

The translation of BPMN to WS-BPEL is far from trivial.

The second version of BPMN (2.0) (OMG, 2011), includes execution semantics and a common exchange format. This means that BPMN 2.0 process definition models are not only exchangeable between graphical editors, but those models can also be executed as-is on any BPMN 2.0 compliant engine.

The current landscape is as follows (Leymann, 2011). On one hand, only a subset of BPMN 2.0 is "isomorphic" to WS-BPEL. On the other hand, BPMN 2.0 is a process modeling language with an operational

11 Unified Modeling Language (UML)-based approaches have been also proposed (as referred in (Zhao et al., 2009)). However they have not reached the same impact as BPMN at the industrial level.

semantics imprinted by WS-BPEL (i.e., BPMN 2.0 builds on the success of WS-BPEL). Thus, it is possible to build an engine that directly supports BPMN 2.0 (without the intermediate step of generating WS-BPEL). There is an emerging group of BPMN 2.0 engines, such as ActiveVOS[12], jBPM[13], Activiti[14] (it extends jBPM), and Oracle BPEL Process Manager[15]. However, due to the recent launch of BPMN 2.0, this group of tools is not as wide and mature as the group of available WS-BPEL engines. Table 2.1 summarizes some basic modeling elements that can be described with the BPMN notation (OMG, 2011).

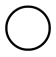
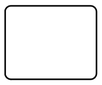
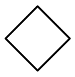
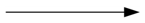
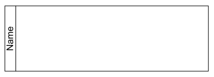
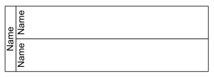
Element	Description	Notation
Event	An Event is something that "happens" during the course of a Process or a Choreography. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.	
Activity	An Activity is a generic term for work that a company performs in a Process. An Activity can be atomic or compound: Subprocess or Task.	
Gateway	A Gateway is used to control the divergence and convergence of Sequence Flows in a Process and in a Choreography.	
Sequence Flow	A Sequence Flow is used to show the order that Activities will be performed in a Process and in a Choreography.	
Pool	A Pool is the graphical representation of a Participant in a Collaboration. It also acts as a "swimlane" and a graphical container for partitioning a set of Activities from other Pools, usually in the context of business-to-business situations.	
Lane	A Lane is a subpartition within a Process, sometimes within a Pool, and will extend the entire length of the Process, either vertically or horizontally. Lanes are used to organize and categorize Activities.	

Table 2.1: Basic BPMN model elements (OMG, 2011).

12 http://www.activevos.com
13 http://www.jboss.org/jbpm
14 http://www.activiti.org
15 http://www.oracle.com/technetwork/middleware/bpel/overview

2.2.2 *Context*

Applications and services should be aware of their contexts and automatically adapt to their changing contexts in order to provide adequate services to users (Hong et al., 2009). Context is very important, since it provides information about the present status of people, places, things and devices in the environment (Korpipaa et al., 2003; Kwon, 2004). Specifically, "*context* is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." (Dey, 2001).

Context is any information that can be used to characterize the situation of an entity.

Context-awareness means that one is able to use context information (Hong et al., 2009). A system is *context-aware* if it can extract, interpret and use context information and adapt its functionality to the current context of use (Byun and Cheverst, 2004). One goal of context-aware systems is to acquire and utilize information in the context to provide services that are appropriate to the particular people, place, time, event, etc. (Hong et al., 2009).

The SAS community is concerned with the increasing complexity of the context (Cheng et al., 2009a). This complexity is caused by systems that are moving from the closed world into the open world (e.g. ubiquitous and pervasive computing). Regarding this issue, Baresi et al. state the following (Baresi et al., 2006):

> "Traditional software development is based on the closed-world assumption that the boundary between system and environment is known and unchanging. However, this assumption no longer works within today's unpredictable open-world settings, which demand techniques that let software react to changes by self-organizing its structure and self-adapting its behavior."

Table 2.2 summarizes the main differences between the closed world and the open world according to three criteria.

A distinguishing factor of SAS is the degree of anticipation with regard to the adaptation process (Piechnick et al., 2012). According to this factor, there are two kinds of SAS, which run in the closed and open worlds, respectively:

- **`SAS with anticipated adaptation:`** This kind of SAS defines all possible context situations, application variants, and their rela-

Criteria	Closed World	Open World
Stable contexts vs. dynamic contexts	Traditionally, developers handle environment changes by eliciting change requests, modifying the software design and implementation, verifying that the resulting product meets the changed requirements, and, finally, redeploying the application. This approach assumes that the external world changes slowly and that software can remain stable for long periods.	Closed-world assumptions do not hold in an increasing number of cases, especially in ubiquitous and pervasive computing settings, where the world is intrinsically open. Applications cover a wide range of areas, such as automotive applications and home automation on the embedded-systems level.
Anticipated changes vs. unanticipated changes	It assumes a closed external world, meaning that requirements leading to a specification for the software system's interaction with the external world can capture all phenomena of interest.	In an open world, the environment changes continuously. Software must adapt and react to changes dynamically, even if they are unanticipated.
Closed systems vs. open systems	The software itself is closed since it is composed of parts that do not change while it is executing.	The world is open to new components that context changes could make dynamically available. Systems can discover and bind such components dynamically to the application while it is executing.

Table 2.2: Closed world vs. open world. Adapted from (Baresi et al., 2006).

tionship at design time, whereby the system variation is performed at runtime according to predefined rules. Also, an adaptation to situations that have not been considered during the development process is impossible. Therefore, we can say that this kind of SAS works under the `closed-world assumption`.

- **`SAS that support unanticipated adaptation:`** In this case, the adaptation mechanisms must be open and extensible during execution. Therefore, we can say that this kind of SAS works under the `open-world assumption`.

Uncertainty in the Open World

Uncertainty is a topic frequently studied in several areas.

A key aspect in the open world is *uncertainty*. Uncertainty is a topic frequently studied in several areas:

- In **`economics`**, uncertainty is usually studied in the context of risk, expected utility, and rational decision-making (Laffont, 1989). In economics, uncertainty typically describes a process or situation with a stochastic or probabilistic component. The concept of uncertainty is interpreted as not being certain about the occurrence or outcome of an event or decision.

- In **`classical physics`**, uncertainty is considered as an artifact introduced by imperfect measurements (Taylor and Kuyatt, 1994). Every measurement of a physical entity or phenomenon is inherently uncertain either by limitations in the measuring equipment or by the application of ad hoc experimental techniques.

- In **`quantum mechanics`**, the *uncertainty principle* (Heisenberg, 1927) is any of a variety of mathematical inequalities asserting a fundamental limit to the precision with which certain pairs of physical properties of a particle known as complementary variables, such as position x and momentum p, can be known simultaneously. This principle implies that the act of measuring somehow disturbs other properties of the subject being measured.

- In **`psychology`**, uncertainty often refers to the ability of human beings to make judgments in the presence of incomplete information (Tversky and Kahneman, 1974).

- In **`software engineering`**, self-adaptation is subject to uncertainty (Cheng et al., 2009a). Uncertainty can be observed in every facet of adaptation and at varying degrees (Esfahani et al., 2011). It is because system's user, adaptation logic, and business logic are loosely coupled, introducing several sources of uncertainty (Cheng and Garlan, 2007). In fact, "uncertainty is inherent and inevitable in software development processes and products" (Ziv et al., 1997)[16].

Uncertainty in SAS can be defined as follows:

> "Uncertainty is a system state of incomplete or inconsistent knowledge such that it is not possible for a SAS

16 This is known as the "Uncertainty Principle in Software Engineering."

> to know which of two or more alternative environmental or system configurations hold at a specific point. This uncertainty can occur due to missing or ambiguous requirements, false assumptions, unpredictable entities or phenomena in the execution environment, and unresolvable conditions caused by incomplete and inconsistent information obtained by potentially imprecise, inaccurate, and unreliable sensors in its monitoring infrastructure." (Ramirez et al., 2012).

According to (Esfahani et al., 2011), there are two kinds of uncertainty: 1) **`external uncertainty`** arises from the context or domain in which the software is deployed; and 2) **`internal uncertainty`** is rooted in the difficulty of determining the impact of adaptation on the system's quality objectives, e.g. determining the impact of replacing a software component on the system's responsiveness.

Finally, it is important to mention that "allowing uncertainty levels when developing SAS requires a trade-off between flexibility and assurance such that the critical high-level goals of the application are always met." (Cheng et al., 2009a).

2.3 CONCLUSIONS

In this chapter, we have presented the foundations for this thesis. This chapter is divided into two main sections. First, Section 2.1 presented the enablers of our approach: AC, MDE, models at runtime, SPLE, variability modeling, and DSPLE. Second, Section 2.2 presented the target on which our proposal is applied: service compositions. In this area, we presented WS-BPEL, orchestration engines, and service composition modeling. Since service compositions run in a context, this section also introduced context-related concepts and made the distinction between the closed and open worlds.

3 STATE OF THE ART

"Research is to see what everybody else has seen,
and to think what nobody else has thought."

— Albert Szent-Gyorgyi

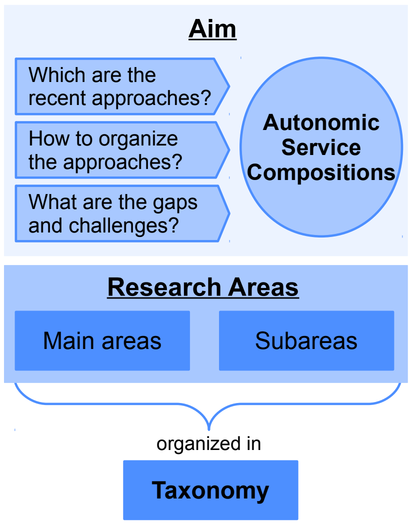

Aim
Which are the recent approaches?
How to organize the approaches?
What are the gaps and challenges?
Autonomic Service Compositions
Research Areas
Main areas
Subareas
organized in
Taxonomy

Figure 3.1: Scope of Chapter 3.

SOA represents an architectural model that aims to improve the agility and cost-effectiveness of a company. Enterprises can plug in new services or upgrade existing ones to address new business requirements thanks to the loosely coupled nature of SOA. Web services are the most common realization of SOA. They run in heterogeneous and complex environments (e.g. they could run on distributed servers with variable processing and memory capabilities, experience network failures during communication, and be under attack). As a result, it is appropriate to have adaptation mechanisms to reconfigure them according to context changes (e.g. access to a Web service running on a slow server could be replaced by accessing another Web services on a faster server to improve the QoS). However, it is impractical to assign manual reconfiguration tasks to systems engineers because of the intricacy of today's service compositions and the large amount of human effort required.

This chapter carries out a literature review in order to examine studies proposing approaches to achieve autonomic service compositions (see Figure 3.1). This review follows the systematic method proposed by (Kitchenham, 2004; Webster and Watson, 2002). The aim of this review is to answer the following questions:

- **Question 1 (Q1):** Which approaches have been proposed in recent years to achieve autonomic service compositions?
- **Question 2 (Q2):** How can the approaches for autonomic service compositions be coherently organized according to desirable properties of SAS in the context of service orchestrations?
- **Question 3 (Q3):** What are the gaps and challenges to be faced in the future?

Q1 gives the structure to this chapter. **Q2** drives the analysis during the literature review presented in this chapter. **Q3** is answered at the end of this chapter.

As recommended by (Webster and Watson, 2002), we used both manual and automated methods to make a selection of candidate papers in leading journals and relevant conferences. The following exclusion criteria was carried out: 1) the paper is not electronically available on the Web; 2) the paper is not presented entirely in the English language; 3) the data register identified after applying search strings does not actually refer to a scientific paper, but to some non-peer reviewed publication, such as: technical reports, books and book chapters, proceedings' prefaces, and journal's editorials; 4) the paper presents some type of review, such as a survey or some secondary study, and not the outcomes of some specific research work (i.e., a primary study).

The remainder of this chapter is structured as follows. Section 3.1 introduces the research areas that are covered in this chapter. Section 3.2 describes a taxonomy that is used to classify research works in the main research areas. Section 3.3 describes related work in the main areas. Section 3.4 describes related work in the supporting subareas. Section Section 3.5 presents the conclusions.

3.1 RESEARCH AREAS

This section describes the research areas that are studied in this chapter. These areas are covered because of their relevance in the state of the art of autonomic service compositions, and their closed relationship with our work. Figure 3.2 summarizes them. The top of Figure 3.2 shows four main research areas that provide direct solutions to achieve autonomic service compositions: variability constructs at the language level, broker, models at runtime, and DSPL. The bottom of figure 3.2 shows two research subareas: variability modeling and uncertainty management in the open world. Although these subareas are not directly related to dynamic adjustments of service compositions, we include them because they support key aspects of our solution.

This chapter covers four main research areas and two research subareas.

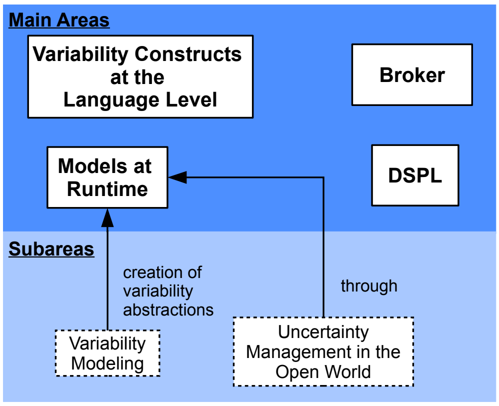

Main Areas
Variability Constructs at the Language Level
Broker
Models at Runtime
DSPL
Subareas
creation of variability abstractions
through
Variability Modeling
Uncertainty Management in the Open World

Figure 3.2: Research areas covered in this chapter.

The main research areas covered in this work are described as follows. The research subareas are described in the context of the main areas:

1. **Autonomic service compositions through variability constructs at the language level:** WS-BPEL has become the de facto language to orchestrate service compositions. However, there are two main drawbacks when using WS-BPEL in enterprise systems:

   a) WS-BPEL has an inherently static nature. Basically, if a process definition is deployed into the orchestration engine, it cannot be changed dynamically. Every information is hard-wired after a process is deployed. For instance, references to other service operations (called partner links) used in the WS-BPEL process cannot be changed and exchanged without editing and redeploying the process; it implies a downtime of the whole system. Although it is possible for the process to bind to partner links at runtime, the process definition would contain a tremendous amount of code that is not related to the BP (Moser et al., 2008; Mosincat and Binder, 2008).

b) WS-BPEL does not provide any means for monitoring the context. The monitoring of the context is a necessary foundation to decide whether a dynamic adjustment should be performed. To the best of our knowledge, available orchestration engines lack this ability.

Therefore, research works in this area are particularly focused on extending WS-BPEL to guide dynamic adjustments according to collected context data. Intrusive and non-intrusive strategies have been proposed to specify these extensions. On one hand, an intrusive strategy has been to define variation points, variants, and configurations for a process inside the service composition specification (i.e., the composition schema). On the other hand, non-intrusive strategies have been proposed by means of aspect orientation, Event Condition Action (ECA) rules, and language-based monitoring and recovery strategies.

2. **Autonomic service compositions through a broker:** A system works as a broker (intermediary) that is in charge of selecting or binding partner service operations at runtime. To this end, the broker intercepts service messages. Then, it exchanges service operations by invoking the most adequate set of service operations (e.g. according to QoS attributes) to accomplish a task.

3. **Autonomic service compositions through models at runtime:** Models are used to guide dynamic adjustments at a highly-abstract level. Since our work uses models as a means to describe variability and to manage uncertainty in the open world, we also studied research works in the following two subareas:

   a) **Variability modeling:** In order to guide autonomic service compositions with models at runtime, first it is necessary to model variability for later use during execution.

Variability modeling has been widely studied in BPM. In fact, BPM and SOA, when used together, can help to achieve the strategic alignment between business related areas and information technology (dos Santos Rocha and Fantinato, 2013). Therefore, we study approaches that are related to variability modeling in BPM, whose principles can be applied to SOA.

b) **Model-driven techniques to face uncertainty in the open world:** A hot topic in the models at runtime research community is to use highly-abstract models to face uncertainty in the open world (Aßmann et al., 2011). Uncertainty can be produced, for example, by unexpected context events.

4. **Autonomic service compositions through DSPLs:** In DSPLs, variation points are bound initially when software is launched to adapt to the current context, as well as during operation to adapt to changes in the context. Thanks to the flexibility offered by DSPLs, DSPL-based solutions have been proposed for SOA.

3.2 TAXONOMY TO CLASSIFY RESEARCH WORKS

In this section, we propose a taxonomy to facilitate the analysis of the research works in the *main areas* of the state of the art (i.e., autonomic service compositions through variability constructs at the language level, brokers, models at runtime, and DSPLs). This taxonomy has a set of dimensions that describe several expected facets of autonomic service compositions (see Table 3.1). We have grouped the identified key dimensions into five groups: 1) the dimensions associated with causes of self-adaptation – the "**what**"; 2) the dimensions associated with the mechanisms to achieve self-adaptability – the "**how**"; 3) the dimension about the frequency, duration, and anticipation of changes – the "**when**"; 4) the dimensions related to the object of change – the "**where**"; and 5) the dimensions related to the **maturity** of the approach.

We propose a taxonomy to facilitate the analysis of the research works in the main areas of the state of the art.

Several dimensions were taken from previous taxonomies for SAS (Buckley et al., 2005; Andersson et al., 2009) because they are general enough to be applied to autonomic service compositions. However, autonomic service compositions have certain particular dimensions, which require to extend these generic taxonomies. Specifically, we have added the following dimensions: transparency, anticipation, generality,

Dimension	**Degree**	**Definition**
WHAT change is the cause for adaptation		
Source	External (context) or internal	Where is the source of change?
Type	Functional or non-functional	What is the nature of change?
HOW the service composition faces changes		
Autonomy	Automated, partially automated, manual	What is the degree of outside intervention during adaptation?
Organization	Centralized to decentralized	Is the adaptation done by a single component or distributed among several components?
Scope	Local to global	Is the adaptation localized or involves the entire system?
Triggering	Event-trigger or time-trigger	Is the change that triggers adaptation associated with an event or a time slot?
Safety	Verified or unverified configurations	Are the possible configurations verified?
Transparency	The orchestration engine needs to change or not (transparent)	Does the orchestration engine have to change?
WHEN changes are carried out		
Frequency	Continuously, periodically, at arbitrary intervals	What is the frequency of change?
Adaptation time	Short, medium, long	How long does the adaptation last?
Anticipation	Foreseen or unforeseen context events	Are context events foreseen at design time?
WHERE changes are carried out		
Generality	SOA in a generalized manner or Web services	Is the solution related to SOA in a generalized manner?
Change level	Abstract, language, SOA message	Where are changes carried out?
MATURITY of the approach		
Applicability scenarios	None, prototypes, industrial orchestration engines	Has the approach been applied on matured scenarios?
Demonstrability	None, tool, running examples	Is there any available demonstration?
Life-cycle support	Analysis to runtime	Which software life-cycle phases are covered?
Evaluation	None, example, simulation, experiments in industry, case study	How was the evaluation carried out?

Table 3.1: Taxonomy to classify research works on autonomic service compositions.

change level, and the dimensions that are related to the maturity of the approaches for autonomic service compositions.

The following subsections describe the taxonomic dimensions according to the groups in which they have been cataloged.

3.2.1 *What Change is the Cause of Adaptation*

Changes are the cause of self-adaptation. The dimensions in this group are as follows:

SOURCE: This dimension identifies the origin of the change, which can be either *external* to the service composition (i.e., its context) or *internal* to the service composition (e.g. an operation in the service composition is deleted).

TYPE: This dimension refers to the nature of change. It can be *functional* (the purpose of the service composition has changed and services delivered need to reflect this change) or *non-functional* (service composition performance and reliability need to be improved).

3.2.2 *How the Service Composition Faces Changes*

The dimensions in this group capture the service composition reaction towards change (they are related to the adaptation process itself). The dimensions associated with this group are as follows:

AUTONOMY: This dimension identifies the degree of outside intervention during adaptation. The range of this dimension goes from *automated* to *manual*. In the automated case, there is no external influence to the service composition guiding how the service composition should adapt at runtime. On the other hand, a service composition can have a degree of self-adaptability when externally assisted.

ORGANIZATION: This dimension captures whether adaptation is performed by a single component (*centralized*), or distributed among several components (*decentralized*).

SCOPE: This dimension identifies whether adaptation is localized or involves the entire service composition. The scope of adaptation can range from *local* to *global*. If adaptation affects the entire service composition then more meticulous analysis is required to commit the adaptation.

TRIGGERING: This dimension identifies whether the change that initiates adaptation is *event-trigger* or *time-trigger*. Although it is difficult to control how and when changes occur, it is possible to control how and when adaptations should react to certain changes (Andersson et al., 2009).

SAFETY: This dimension states whether or not the adapted configurations of the service composition are safe (i.e., error-free). In other words, this dimension checks whether or not possible configurations have been verified before they are applied to the running service composition.

TRANSPARENCY: This dimension states whether or not the orchestration engine has to change in order to apply the self-adaptive nature on the service composition. If the orchestration engine does not have to change, then the dynamic adaptation approach is considered *transparent*. The non-transparent option is expensive and makes it difficult to transport the solution to other orchestration engines. Moreover, most industrial orchestration engines do not allow third-parties to modify them.

3.2.3 *When Changes are Carried Out*

The dimensions in this group capture time-related aspects of self-adaptation in service compositions. The dimensions associated with this group are as follows:

FREQUENCY: This dimension states whether the frequency of changes on the service composition is performed *continuously*, *periodically*, or *at arbitrary intervals*.

ADAPTATION TIME: This dimension refers to the period of time in which the service composition carries out the self-adaptation.

ANTICIPATION: This dimension captures whether the possible context events, which will eventually cause changes in the service composition, are foreseen at design time. In the closed world, possible context events can be fully foreseen at design time. However, there are unknown or unforeseen context events in the open world that cannot be anticipated.

3.2.4 *Where Changes are Carried Out*

Dynamic adaptation solutions may be proposed for generalized SOAs or for service compositions that are composed of Web services. Also, dynamic adaptations may require changes at different abstraction levels of the service composition. The dimensions associated with this group are as follows:

GENERALITY: This dimension states whether the solution is applied on a generalized SOA or on service compositions that are composed of Web service operations.

CHANGE LEVEL: This dimension refers to the level in which changes are carried out on the service composition. Changes can be carried out in abstractions that represent the service composition (e.g. in models at runtime). In turn, these changes are replicated into the underlying service composition. Also, changes can be carry out at the language level (i.e., the source code). Another level is to carry out changes at the SOA messaging level (i.e., a message can be intercepted and modified to invoke an alternative service operation). Some of these approaches are more expensive than others. For instance, changes at the language level can be intrusive (i.e., the code has to be extended with additional variability-related instructions). On the other hand, approaches that are based on brokers can select sets of service operations without having to modify the underlying code of the service composition.

3.2.5 *Maturity of the Approach*

The dimensions in this group state the maturity of the approach. We consider that an approach is mature when it 1) runs on industrial orchestration engines, 2) can be demonstrated with available tools or running examples, 3) offers support to systems analysts throughout the software life cycle, and 4) has been evaluated through adequate mechanisms. The dimensions associated with this group are as follows:

APPLICABILITY SCENARIOS: This dimension identifies the orchestration engine in which the adaptation mechanism has been applied. An approach that uses an industrial orchestration engine is more mature than the one that uses a prototypical engine.

DEMONSTRABILITY: This dimension states how the adaptation approach has been (or can be) demonstrated through downloadable tools or running examples.

LIFE CYCLE: This dimension states whether the adaptation approach spans through the development life cycle.

EVALUATION: This dimension identifies the kind of evaluation that was carried out on the adaptation approach. The quality of the evaluation increases according to the formality of the evaluation process. The most mature evaluations are the ones that are carried out through industrial experiments, simulations, or case studies.

3.3 RELATED WORK IN THE MAIN RESEARCH AREAS

We identified 21 primary works.

This section describes related work in the main areas of the state of the art (see Figure 3.3). During the literature review process, we identified **21 primary works** in the four main state-of-the-art areas on autonomic service compositions (described in Section 3.1). Table 3.2 shows the research works that have been proposed in these areas during a nine-year period[1].

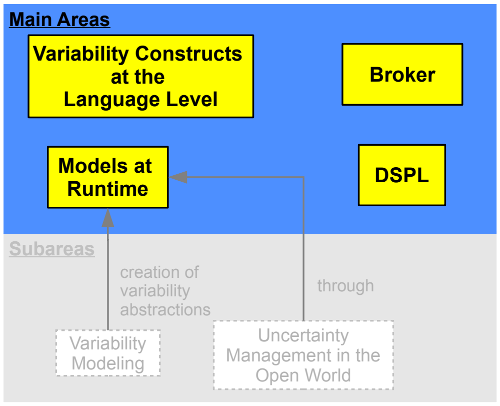

Main Areas
Variability Constructs at the Language Level
Broker
Models at Runtime
DSPL
Subareas
creation of variability abstractions
through
Variability Modeling
Uncertainty Management in the Open World

Figure 3.3: The focus of this section.

1 We did not study earlier research works, which may be considered obsolete.

Year	Variability Constructs At the Language Level	Broker	Models at Runtime	DSPL
2012			MUSIC and MOSES	
2011	Self-Supervising WS-BPEL Processes		SASSY, MAESoS, and QoSMOS	
2010		Percentile-Based SLAs		SOPL
2009	VxBPEL	BPEL'n'Aspects		CAPucine
2008		VieDAME, Transparent Runtime Adaptability, and QoS-Aware Binding and Rebinding		
2007	AO4BPEL	Robust-BPEL2 and Paws		
2006	SCENE	WSQoSX		
2005			DySOA	
2004		AgFlow		

Table 3.2: Compendium of primary research works on autonomic service compositions from 2004 to 2012.

Table 3.2 shows that from the set of four main areas, brokers have been the most widely used mechanism to reach autonomic service compositions (42.86%). Moreover, variability constructs at the language level and brokers can be considered as the two "traditional" ways to achieve autonomic service compositions (research works in these areas span over 2004 to 2011). Nevertheless, in recent years, there has been an increasing interest on using models at runtime and DSPLs to achieve autonomic service compositions (as evidenced in the research works from 2009 to 2012).

The following subsections summarize research works that reach autonomic service compositions by different means. First, Subsection 3.3.1 presents relevant approaches that use variability constructs at the language level. Subsection 3.3.2 presents related work that propose brokers. Subsection 3.3.3 presents research works that use models at runtime. Although, several general approaches use models at runtime to guide dynamic adjustments of SAS, we only describe research works in which models at runtime have been used to guide dynamic adjustments of service compositions. Finally, Subsection 3.3.4 describes approaches that use DSPLs. Although DSPLs have been successfully applied to several disciplines, such as home automation, mobile devices, software-intensive embedded system families, and in general approaches for SAS, we describe approaches that have applied DSPLs to SOA.

3.3.1 *Variability Constructs at the Language Level*

In this section, we present four relevant primary works that achieve autonomic service compositions through variability constructs at the language level. Table 3.3 summarizes these approaches according to the taxonomy in Section 3.2.

SCENE

SCENE (Colombo et al., 2006) extends WS-BPEL with ECA rules that define consequences for conditions to guide the execution of binding and rebinding self-reconfiguration operations. This approach does not pose any restriction to the structure of the WS-BPEL code. Rules are used to associate a WS-BPEL workflow with the declaration of the policy to be used during (re)configuration. These rules contain what is needed to dynamically select or change services. Rules are aware of the state of the process. Rules can either be defined at design time or later before the execution of the system. Moreover, several sets of rules can coexist and be activated depending on the preferences of the system users. A composition written in the proposed language is executed by SCENE.

AO4BPEL

In (Charfi and Mezini, 2007), the authors identify two problems on WS-BPEL. First, they argue that WS-BPEL lacks means to modularize crosscutting concerns (i.e., concerns that cut across the modular structure of WS-BPEL processes, e.g. logging, persistence, and security). Second, they observe that WS-BPEL does not provide appropriate support for dynamic adaptation of service compositions. Therefore, they propose an aspect-oriented approach to service compositions and present the design and implementation of AO4BPEL, an aspect-oriented extension to WS-BPEL.

The application's behavior is adapted at runtime with dynamic weaving (i.e., the dynamic integration of aspects with the process). Required dynamic changes are crosscutting concerns because they arise in many places across different processes. These crosscutting concerns can be non-functional and functional (i.e., related to the process business logic). The dynamic weaver of AO4BPEL implements the dynamic weaving. It modifies the interpretation flow by checking if any aspect matches the current join point activity. If there is a match, the respective advice activity is executed. This strategy offers native support for

Dimension	**SCENE (2006)**	**AO4BPEL (2007)**	**VxBPEL (2009)**	**Self-Supervising WS-BPEL Processes (2011)**
WHAT change is the cause for adaptation				
Source	N/S	N/S	N/A	External and Internal
Type	Functional and non-functional	Functional and non-functional	N/A	Functional and non-functional
HOW the service composition faces changes				
Autonomy	Automated	Automated	Manual	Automated
Organization	Decentral.	Centralized	N/A	Decentral.
Scope	Local	Local	Local	Local and global (backward)
Triggering	Event-trigger	Event-trigger	N/A	Time- and event-trigger
Safety	Unverified	Unverified	Unverified	Unverified
Transparency	Transparent	Transparent	No transparent	Transparent
WHEN changes are carried out				
Frequency	N/S	At arbitrary intervals	N/A	Periodically by supervision parameters
Adaptation time	N/S	Short	N/A	Long
Anticipation	Foreseen	Foreseen	N/A	Foreseen
WHERE changes are carried out				
Generality	Web services	Web services	Web services	Web services
Change level	Language	Language	Language	Language
MATURITY of the approach				
Applicability scenarios	PXE	BPWS4J	ActiveBPEL	ActiveBPEL
Demonstrability	N/S	N/S	N/S	N/S
Life-cycle support	Implement. and runtime	Implement. and runtime	Design and runtime	Design and runtime
Evaluation	Examples	Examples	Example	Example and industrial experiments

Table 3.3: Comparison of the approaches that support autonomic service compositions through variability constructs at the language level (N/A means No Applicable, N/S means Not Specified).

AOP concepts (aspects, pointcuts, and advices) as first-class entities in the orchestration engine, on a par with processes and activities.

VxBPEL

VxBPEL (Koning et al., 2009) is a language that adapts WS-BPEL to capture variability in service-centric systems. VxBPEL allows to define variation points, variants, and configurations for a process. For example, in order to indicate that a part of a WS-BPEL process is a variation point, it is enclosed by a `<VariationPoint>` element. VxBPEL addresses the adaptive composition of Web services by providing the variability constructs in the language level, and treats the changes as first-class entities. The specifications of adaptive services composition in VxBPEL integrate main business logic and adaptation of process elements. VxBPEL supports the following types of variability: service replacement, service parameters change, and system composition change. The execution of adaptation is supported at compile time and at runtime since the extensions to the orchestration engine are used to interpret the variability constructs.

Although the authors do not propose an automated mechanism for dynamic adjustments, this approach is listed because it offers an interesting solution for variability specification.

Self-Supervising WS-BPEL Processes

In (Baresi and Guinea, 2011), the authors propose the idea of *self-supervising* WS-BPEL *processes*: special-purpose service compositions that assess their behavior and react through user-defined rules. Supervision consists of monitoring and recovery. *Monitoring* is about checking the system's execution to see whether everything is proceeding as planned, while *recovery* attempts to fix any anomalies.

Monitoring directives are expressed in the Web Service Constraint Language, which is a special-purpose language that mixes typical propositional logic constructs with XML-based technology. The Web Service Constraint Language concentrates on both functional and non-functional properties and expresses general dependability properties such as safety, integrity, availability, and reliability.

Recovery strategies follow the ECA paradigm and are stated in the Web Service Recovery Language. The *event* is the discovery of a runtime anomaly. The *condition* is expressed in the Web Service Constraint Language, and allows to choose among alternative recovery options which are defined by picking and mixing atomic recovery actions from an extensible library of predefined actions.

Supervision is treated as a cross-cutting concern that is only blended at runtime. At runtime, the business logic and the supervision directives are intertwined by means of AOP.

3.3.2 *Brokers*

In this section, we present nine relevant primary works that achieve autonomic service compositions through brokers. Table 3.4 and Table 3.5 summarize these approaches according to the taxonomy in Section 3.2.

AgFlow

AgFlow (Zeng et al., 2004) is a middleware platform that enables the quality-driven composition of Web services. In AgFlow, the QoS of Web services is evaluated by means of a multi-dimensional QoS model (which captures non-functional properties). The selection of component services is performed to optimize the QoS of the composite service executions. Furthermore, the AgFlow adapts to changes that occur during the execution of a composite service. To this end, it revises the execution plan to optimize the QoS given a set of user requirements and a set of candidate component services.

AgFlow offers two alternative QoS driven service selection approaches for composite service execution: one based on local optimization (task-level) and the other on global allocation of tasks to services using integer programming. An adaptive execution engine reacts to changes occurring during the execution of a composite service by re-planning the execution in order to ensure that the QoS is optimal.

WSQoSX

In (Berbner et al., 2006), the authors present an extension of the proxy architecture WSQoSX with a heuristic based approach to solve the QoS-aware service composition problem. They introduce three heuristics. The first heuristic consists of two steps: 1) a Mixed Integer Programming formulation of the composition problem is generated and its linear programming relaxation is solved; and 2) a backtracking algorithm is used to create a valid solution to the original, non-relaxed problem.

Dimension	**AgFlow (2004)**	**WSQoSX (2006)**	**Robust-BPEL2 (2007)**	**PAWS (2007)**	**VieDAME (2008)**
WHAT change is the cause for adaptation					
Source	External	External	External	External	External
Type	Non-functional	Non-functional	Non-functional	Functional and non-functional	Non-functional
HOW the service composition faces changes					
Autonomy	Automated	Automated	Automated	Automated	Automated
Organization	Centralized	Decentral.	Centralized	Decentral.	Decentral.
Scope	Local and global	Local	Local	Local	Local
Triggering	Event-trigger	Event-trigger	Event-trigger	Event- and time-trigger	Event-trigger
Safety	Unverified	Unverified	Unverified	Unverified	Unverified
Transparency	N/S	Transparent	Transparent	No transparent	Transparent
WHEN changes are carried out					
Frequency	N/S	At arbitrary intervals	N/S	Periodically	N/S
Adaptation time	Long	Short	Short	Long with several proxies	Short
Anticipation	Foreseen	Foreseen	Foreseen	Foreseen	Foreseen
WHERE changes are carried out					
Generality	Web services	Web services	Web services	Web services	Web services
Change level	Service message	Service message	Service message	Service message	Service message
MATURITY of the approach					
Applicability scenarios	N/S	N/S	N/S	ActiveBPEL	ActiveBPEL
Demonstrability	N/S	N/S	N/S	N/S	N/S
Life-cycle support	Design time and runtime	Runtime	Runtime	Design time and runtime	Runtime
Evaluation	Examples	Simulations and examples	Example	Simulations	Example

Table 3.4: Comparison of the approaches that support autonomic service compositions through a broker (N/A means No Applicable, N/S means Not Specified).

Dimension	**Transparent Runtime Adaptability (2008)**	**QoS-Aware Binding and Re-binding (2008)**	**PBPEL'n' Aspects (2009)**	**Percentile-Based SLAs (2010)**
WHAT change is the cause for adaptation				
Source	External	External	External	External and internal
Type	Non-functional	Non-functional	Functional	Functional and non-functional
HOW the service composition faces changes				
Autonomy	Automated	Automated	Automated	Automated
Organization	Decentral.	Decentral.	Decentral.	Decentral.
Scope	Local	Local	Local	Local
Triggering	Event-trigger	Event-trigger	Event-trigger	Event-trigger
Safety	Unverified	Unverified	Unverified	Unverified
Transparency	N/S	Transparent	Transparent	N/S
WHEN changes are carried out				
Frequency	N/S	At arbitrary intervals	At arbitrary intervals	Periodically
Adaptation time	Short	N/S	N/S	Short
Anticipation	Foreseen	Foreseen	Foreseen	Foreseen
WHERE changes are carried out				
Generality	Web services	Web services	Web services	Web services
Change level	Service message	Service message	Service message	Service message
MATURITY of the approach				
Applicability scenarios	ActiveBPEL	ActiveBPEL	ActiveBPEL	N/S
Demonstrability	N/S	N/S	N/S	N/S
Life-cycle support	Runtime	Design time and runtime	Implementation and runtime	Runtime
Evaluation	Examples	Examples	N/S	Example

Table 3.5: Comparison of the approaches that support autonomic service compositions through a broker (N/A means No Applicable, N/S means Not Specified).

The two other heuristics, are meta-strategies to improve the results of the first heuristic.

RobustBPEL2

RobustBPEL2 (Ezenwoye and Sadjadi, 2007) is a framework to adapt WS-BPEL processes by redirecting their interactions with partner services to a dynamic proxy. Upon failure or delay of any of the partner services of the WS-BPEL process, an equivalent service can be discovered dynamically to serve as a substitute. The adaptation preserves the original behavior of the BP and does not tangle the code that provides autonomic behavior with that of the BP. This transparency is achieved by using a dynamic proxy that encapsulates the autonomic behavior (adaptive code). The authors propose to incorporate some generic hooks at sensitive joinpoints in the original WS-BPEL process. Adaptive code is introduced in these joinpoints at runtime.

PAWS

The PAWS framework (Ardagna et al., 2007; Ardagna and Pernici, 2007) facilitates flexible and adaptive execution of managed Web-service-based BPs. This proxy-based framework integrates several service-adaptation modules and couples design-time and runtime mechanisms for process specification and global framework execution. At design time, PAWS identifies a set of candidate services for each process task (retrieved from an extended UDDI repository), negotiates QoS, specifies quality constraints, and then identifies mapping rules for invoking services with different interfaces. At runtime, PAWS supports adaptation during process execution: it selects the best set of partner services to execute the process, reacts to service failures, and preserves execution when a context change occurs with a set of simple recovery actions (retry, substitute, and compensate). A self-healing module uses a mediator to perform recovery actions, either to retry/redo process task execution or to substitute a faulty candidate service. Monitoring, discovery, or mediation are treated externally in proxies that are placed between the process and the partner services. Unfortunately, the extensive use of proxies brings a high performance overhead.

VieDAME

VieDAME (Moser et al., 2008) is a system that allows to monitor WS-BPEL processes according to QoS attributes and to replace existing partner services based on pluggable replacement strategies. The chosen replacement services can be syntactically or semantically equivalent to the WS-BPEL interface. In case of an interface mismatches, a set of transformers can be specified to handle theses mismatches on a SOAP message level. Services can be automatically replaced at runtime without downtime of the overall system. ViEDAME uses aspect orientation to intercept SOAP messages and allow services to be exchanged during runtime with little performance penalty costs. This approach does not adapt the process control flow itself, which is, for example, necessary when a service has to be replaced by two other services which fulfill the same functional requirements but require adaptations in the control flow to achieve the same behavior.

Transparent Runtime Adaptability

In (Mosincat and Binder, 2008), the authors present an infrastructure to handle dynamic (re)binding of stateful and stateless services for every process instance. While a stateless service can be replaced upon each invocation, a service replacing a stateful service must first be brought into the right state. The proposed infrastructure separates the process business logic from service binding and failure recovery strategies. Also, it is transparent both to the process developer and to the orchestration engine.

The infrastructure consists of two major parts: the Bind System manages service bindings for process instances, and the Transformation Tool enhances processes for automated failure recovery, interacting with the Bind System. In addition, the Bind System enables dynamic changes of service selection and fault-handling strategies without requiring any redeployment of existing processes.

QoS-Aware Binding and Re-binding

In (Canfora et al., 2008), the authors propose a QoS-aware binding approach for service compositions based on Genetic Algorithms. QoS-aware dynamic binding of composite services provides the capability of binding each service invocation in a composition to a service chosen among a set of functionally equivalent ones to achieve a QoS goal (e.g. to minimize the response time while limiting the price under a maximum value). The main advantage in the use of Genetic Algorithms is the possibility to apply the approach in presence of arbitrary, nonlinear QoS aggregation formulae, whereas traditional approaches, such as linear integer programming, require linearization. Re-bindings (i.e., to renew the bindings between abstract and concrete services) are triggered as soon as possible to predict that the actual service QoS will likely deviate from the initial estimates. Then, the slice, i.e., the part of the service workflow that still has to be executed, is determined and re-bound.

BPEL'n'Aspects

The BPEL'n'Aspects approach (Karastoyanova and Leymann, 2009; Sonntag and Karastoyanova, 2011) utilizes the AOP paradigm and existing Web service and WS-BPEL infrastructures for improving the flexibility of WS-BPEL processes, and in particular the adaptation of process logic. The BPEL'n'Aspects approach weaves aspects into WS-BPEL process models and instances. Orchestration engine events are used to signal points of interest where aspects can be weaved in. This decouples weaving of aspects from specific engine implementations. BPEL'n'Aspects relies on existing technologies and standards. WS-Policy is used to specify aspects. The association of aspects with WS-BPEL processes is done with WS-PolicyAttachment. This mechanism allows attaching aspects to processes at runtime without changing the target process definition. In other words, the original service composition descriptions (and thus the used orchestration engine) do not need to be modified in order to achieve adaptations.

AOP terms map to BPEL'n'Aspects terms as follows: joinpoints are WS-BPEL activities or transition conditions. Advices are Web services specified by an endpoint reference and an operation. Pointcuts identify concrete process artifacts where aspects are to be weaved in. An advice can be weaved in before, instead, or after a given joinpoint. Aspects are packages that connect engine events (= pointcut + advice type) with Web services (advice). Input and output data of an advice can be fetched from and written to process variables with the help of input and output transformation operations.

Percentile-Based SLAs

In (Cardellini et al., 2010), the authors present a brokering service for the adaptive management of composite services. The goal of this broker is to dynamically adapt the composite service configuration in order to fulfill the SLAs negotiated with different classes of requestors. It considers SLAs that specify upper bounds on the percentile of the service response time, which are expected to better capture user perceived QoS. The adaptive composite service management is based on a service selection scheme that minimizes the service broker cost while guaranteeing the negotiated QoS to the different service classes. The optimal service selection is determined by means of a linear programming problem. The selection is driven by the goal of maximizing some broker utility goal. The search for a new solution is triggered by the occurrence of events that could make no longer valid a previously calculated solution, e.g. the arrival or departure of a user or a change in the set of providers. The results only apply to the service selection scenario and only consider a subset of the workflows' structured activities.

3.3.3 *Models at Runtime*

Several approaches propose to use models at runtime to guide dynamic adjustments of SAS. One of the most relevant works in this area is the Rainbow framework (Garlan et al., 2004). Rainbow uses an abstract architectural model to monitor an executing system's runtime properties. Also, it evaluates the model for constraint violation, and, if a problem occurs, performs global- and module-level adaptations on the running system.

In a more recent work, Morin et al. (Morin et al., 2009a) use software models at runtime and at design time to support the dynamic adaptation of SAS. Their approach relies on four metamodels supported by design tools, to assist in modeling SAS: 1) `DSPL`, which is a feature model that describes the system's variability; 2) `context`, which specifies the system's context; 3) `reasoning`, which describes selection of DSPL's features according to the context; and 4) `architecture`, which describes component-based architectures. Models conforming to these four metamodels are the main data manipulated by a runtime infrastructure responsible for dynamically adapting component-based applications at runtime.

In another recent work, Amoui et al. (Amoui et al., 2012) present a generic model-centric approach for realizing fine-grained dynamic adaptation in software systems by managing and interpreting graph-based models of software at runtime. This approach uses TGraphs and its accompanying technologies, as the enabling technology, for modeling and manipulating runtime models.

The aforementioned works present general model-driven dynamic adaptation approaches for SAS. In this section, we focus on research works that have proposed model-driven solutions for dynamic adaptations in SOA. Table 3.6 describes these approaches according to the taxonomy in Section 3.2.

DySOA

DySOA (Bosloper et al., 2005; Siljee et al., 2005) is a dynamic SOA that extends service-centric applications to make them self-adaptive. DySOA provides a framework for monitoring the application system, evaluating acquired monitoring data against the QoS requirements, and adapting the application at runtime. The purpose of DySOA is to assist the service application system in maintaining its QoS. At design time, an application developer designs a system that is targeted to fulfill the requirements. At runtime, the application system is able to self-adapt when necessary in order to keep delivering the QoS requirements. DySOA splits the self-adaptation process from monitoring to reconfiguration into several steps. The different concerns are then addressed in different components and models within each step.

DySOA proposes the following set of models: 1) a *context model* is used to associate monitoring data with context situations; 2) a *composition model* contains the current configuration of the application and QoS metrics; 3) a *variability model* abstracts the runtime variability of the self-adaptive service system. The variants of a variation point are, for example, several services that provide the same functionality but with different QoS characteristics, or several composition fragments: sets of services organized in different process flows (e.g. WS-BPEL activities). A verifier checks the correctness of new configurations. Examples of checks include variability constraints and deadlock detection.

The separation of models for the different aspects that are covered by DySOA allows better communication between different stakeholders (e.g. service providers or service users) and independent evolution of the aspects. Furthermore, in order to adapt at runtime, the specific models are available during execution. DySOA is described at a very high abstraction level and implementation details are not provided.

SASSY

SASSY (Menasce et al., 2011) is a model-driven framework targeted at dynamic settings in which a system's requirements might change.

Dimension	**DySOA (2005)**	**SASSY (2011)**	**MAESoS (2011)**	**QoSMOS (2011)**	**MUSIC (2012)**	**MOSES (2012)**
WHAT change is the cause for adaptation						
Source	External and internal	External	N/S	External	External and internal	External
Type	Non-functional	Functional and non-functional	Non-functional	Non-functional	Non-functional	Non-functional
HOW the service composition faces changes						
Autonomy	Automated	Automated	Manual or automated	Automated	Automated	Automated
Organization	Decentral.	N/S	Centralized	Centralized	Decentral.	Centralized
Scope	N/S	Local	Local	Local	Local	Local
Triggering	N/S	Event-trigger	Event-trigger	Event-trigger	Event-trigger	Event-trigger
Safety	Verified	Verified	Unverified	Verified	Verified	Unverified
Transparency	N/S	N/S	N/S	Transparent	N/S	N/S
WHEN changes are carried out						
Frequency	N/S	At arbitrary intervals	At arbitrary intervals	At arbitrary intervals	At arbitrary intervals	At arbitrary intervals
Adaptation time	N/S	N/S	N/S	Long in large workflows	Short	Short
Anticipation	Foreseen	Foreseen	Foreseen	Foreseen	Unforeseen	Foreseen
WHERE changes are carried out						
Generality	SOA	SOA	Web services	Web services	SOA	SOA
Change level	Abstract (set of models) and service message	Abstract (architectural model) and architecture	Abstract (set of models) and in the service composition	Abstract (Markov model) and service message	Abstract (set of models) and in the architecture	Abstract (behavioral model) and service message
MATURITY of the approach						
Applicability scenarios	N/S	Apache ServiceMix, XTEAM	N/S	N/S	OSGi	N/S
Demonstrability	N/S	N/S	N/S	N/S	Yes (code and demonstrations)	N/S
Life-cycle support	Design time and runtime	Design time and runtime	Design time and runtime	Runtime	Design time and runtime	Runtime
Evaluation	Example	Example	Example	Simulations and examples	Case studies	Simulations

Table 3.6: Comparison of the approaches that support autonomic service compositions through models at runtime (N/A means No Applicable, N/S means Not Specified).

Throughout the system's life cycle, SASSY maintains a near-optimal architecture for satisfying functional and QoS requirements. In order to generate the software architecture, third-party software engineers develop services and register them in a service directory so that SASSY can discover them. Software architects develop QoS architectural patterns, which are patterns of service composition (such as replication for fault tolerance). A software performance engineer associates these patterns with parameterized QoS analytic models that determine how a particular pattern influences several QoS metrics of interest. In addition, the architects develop software adaptation patterns to dynamically adapt an execute system from its current architecture to another at runtime.

Domain experts specify service activity schemas, which express system requirements. The modeling constructs are defined in a domain ontology that distinguishes different concepts and elements to facilitate service discovery. SASSY uses the service activity schemas requirements to automatically generate a base system service architecture composed of structural and behavioral views. These views consist of components (associated with service providers) and connectors. SASSY uses the system service architecture at runtime as an up-to-date representation of the running software system. SASSY derives an optimized architecture from the base architecture by selecting the most suitable service providers and by applying QoS architectural patterns. It determines this optimized architecture with the help of QoS analytic models and optimization techniques aimed at finding near-optimal choices that maximize system utility. SASSY produces a running system by instantiating the optimized architecture through service binding and deploying the coordination logic.

SASSY's autonomic capabilities follow the MAPE-K loop. Its monitoring component gathers QoS metric values and passes them to the analyzer, which aggregates the data and computes the system's utility. If the utility falls below a stakeholder-specified threshold, the system sends a request to the architecture planner to automatically determine a near-optimal architecture and a corresponding set of service providers. The self-adaptation component executes the changes to the running system through the adaptation patterns. SASSY handles changing requirements through modifications at the service activity schema level that trigger the generation of the revised architecture and the runtime adaptation to the new architecture.

MAESoS

MAESoS (Franch et al., 2011) is a model-driven approach that covers issues ranging from stakeholder goals to low-level aspects of system composition and monitoring. It addresses design-time and runtime aspects and covers monitoring as well as adaptation. MAESoS puts together the following models: 1) goal models define goals for managing different aspects of the system specification process; 2) quality models define a hierarchy of quality attributes; and 3) variability models describe the common and variable features of a set of software systems in a particular domain.

At the requirements layer, stakeholders' needs are represented as goals that involve actors. At the architecture layer, the goals are mapped to architectural concepts and real world elements. In turn, the deployment layer includes services and monitors that deploy the corresponding definitions obtained in the previous layer. Each service may be monitored by several monitors.

A monitoring system puts together all monitors in a single infrastructure which continuously collects runtime information from the service-based system. This allows to compute measures to detect possible violations of requirements by services. The variability model is then used to automatically identify alternatives that can be presented to an engineer to mediate negotiation with stakeholders. In other cases, the adaptation of the system may be performed automatically.

MUSIC

MUSIC (Hallsteinsen et al., 2012) is a framework for applications that operate in ubiquitous and dynamic computing environments and adapt to context changes. MUSIC is a successor of the earlier project MADAM (Geihs et al., 2009). MUSIC supports several adaptation mechanisms and offers a model-driven application development approach supported by a middleware that facilitates the dynamic and automatic adaptation of applications and services based on a clear separation of business logic, context awareness, and adaptation concerns.

MUSIC's middleware implements a control loop which complies with the MAPE-K loop. It monitors the relevant context sensors, and when significant changes are detected, it triggers a planning process to decide if adaptation is necessary. When this is the case, the planning process finds a new configuration that fits the current context better than the one that is currently running, and triggers the adaptation of the running application. To this end, the middleware relies on an annotated QoS-aware *architecture model* of the application, which specifies its adaptation capabilities and its dependencies on context information. The planning process evaluates the utility of alternative configurations, selects the most suitable one for the current context, and adapts the application accordingly.

MUSIC's middleware is divided into the *context middleware* and the *adaptation middleware*. The context middleware encapsulates the diversity of context information and maintains a *context model*, storing and providing uniform access both to the current state and history. The adaptation middleware groups middleware components depending on an *adaptation model*, which is a runtime representation of the annotated architecture model provided by the application developer.

QoSMOS

QoSMOS (Calinescu et al., 2011) is a tool-supported framework for the development of adaptive service-based systems, which combines existing techniques and tools for: 1) formal specification of QoS requirements with probabilistic temporal logics and the ProProST specification system (Grunske, 2008); 2) model-based QoS evaluation (through analytic solving of Markov Models) with probabilistic verification techniques provided by the PRISM model checker (Kwiatkowska et al., 2004); 3) monitoring and Bayesian-based parameter adaptation of the QoS models exploiting KAMI (Epifani et al., 2009); and 4) planning and execution of system adaptation based on GPAC (Calinescu, 2009). QoSMOS can be used to develop service-based systems that achieve their QoS requirements through dynamically adapting to changes in the system state, context, and workload. QoSMOS service-based systems translate high-level QoS requirements specified by their administrators into probabilistic temporal logic formulae, which are then formally and automatically analyzed to identify and enforce optimal system configurations. The QoSMOS self-adaptation mechanism can handle reliability and performance-related QoS requirements. QoSMOS autonomic architecture is based on the MAPE-K loop.

The QoSMOS framework supports the practical realization of adaptive service-based systems architectures by means of two complementary mechanisms. The first mechanism consists of selecting the services that compose a QoSMOS service-based system dynamically. Given a set of functionally equivalent services for each component of a service-based system, QoSMOS selects those services whose reliability, performance, and cost guarantee the realization of the QoS requirements for the system. QoSMOS dynamically adapts its selection of services to runtime changes in both the service characteristics and the system QoS requirements. The second adaptation mechanism consists of adjusting the resources (e.g. the Central Processing Unit (CPU)) allocated to individual services within a service-based system dynamically.

MOSES

MOSES (Cardellini et al., 2012) is a methodology and a software tool to support QoS-driven adaptation of a service-oriented system. It is focused on composite services that need to sustain a traffic of requests generated by several users. Within this scenario, MOSES determines the most suitable configuration of the system for a given context by solving a Linear Programming problem derived from a model of the composite service and a model of its context. The adopted model allows MOSES to integrate in a unified framework both the selection of the set of concrete services to be used in the composition and (possibly) the selection of the coordination pattern for multiple functionally equivalent services.

MOSES receives from a broker administrator the description of the composite service in some suitable workflow orchestration language (e.g. WS-BPEL), and builds a *behavioral model* of the composite service. Once created, the behavioral model is saved to make it accessible to the other system components. An optimization engine implements the Plan component of the MAPE-K loop to solve an optimization problem, which is based on the behavioral model. This model is kept up to date by the monitoring activity carried out by MOSES. The solution of the optimization problem determines the adaptation policy in a given context, which is passed to an adaptation manager for its actual implementation. If changes in the context are relevant, the behavioral model is modified at runtime and a new adaptation policy is calculated.

3.3.4 *DSPLs*

DSPLE has been applied to several disciplines, such as home automation (Istoan et al., 2009; Cetina et al., 2009), mobile devices (Hallsteinsen et al., 2006; White et al., 2007), software-intensive embedded system families (Bosch and Capilla, 2012), and in general approaches for SAS (Morin et al., 2009a). In this section we present two approaches that have used DSPLs in SOA (see Table 3.7).

Dimension	**CAPucine (2009)**	**SOPL (2010)**
WHAT change is the cause for adaptation		
Source	External	External
Type	Non-functional	Non-functional
HOW the service composition faces changes		
Autonomy	Automated	Automated
Organization	Decentral.	N/S
Scope	Local	Local
Triggering	Event-trigger	N/S
Safety	Unverified	Unverified
Transparency	N/S	N/S
WHEN changes are carried out		
Frequency	At arbitrary intervals	N/S
Adaptation time	N/S	N/S
Anticipation	Foreseen	Foreseen
WHERE changes are carried out		
Generality	SOA	SOA
Change level	Bind/unbind components	Bricks (components)
MATURITY of the approach		
Applicability scenarios	N/S	N/S
Demonstrability	N/S	N/S
Life-cycle support	Design time and runtime	Design time and runtime
Evaluation	Example	Example

Table 3.7: Comparison of the approaches that support autonomic service compositions through DSPLs (N/A means No Applicable, N/S means Not Specified).

CAPucine

CAPucine (Parra et al., 2009) is a context-aware DSPL for building service-oriented applications and adapting them at runtime according to their context. CAPucine is based on two different processes for product derivation. The first process uses assets that represent features of the product family. The assets, represented as models, are composed and transformed to generate a product. The second process relates to dynamic adaptation. This process introduces context-aware assets that operate at runtime. These context-aware assets contain the following three kinds of data: the context, the place where the assets must be applied, and the change that must be performed.

In CAPucine, for every selected feature in a feature model, there is an associated asset that corresponds to a partial model of the product itself. Afterwards, CAPucine composes the selected partial models to have one integrated model that represents the product. The next step is to transform this model to enrich it with concepts of the platform, and the implementation language. This is done by performing a series of model-to-model transformations towards the platform and the implementation domains. Finally, the product is built by generating the code from the target domains. The code generation produce the context-aware assets that can be integrated at runtime.

The CAPucine platform is able to suspend and resume the execution of the system, and modify its structure by performing different operations such as deploy, add, bind or delete components by means of a Service Component Architecture (SCA) platform. This enables dynamic adaptation for each context-aware asset.

SOPL

In (Lee and Kotonya, 2010), the authors propose a Service-Oriented Product Line (SOPL) development approach. SOPL is a DSPL application domain that is built on services and a SOA. This approach is composed of a set of activities. First, the feature analysis activity identifies externally visible characteristics of the products in a SPL in terms of features and organizes them into a feature model. Then, the service analysis activity provides a mechanism for mapping the feature model onto services. This activity classifies features into one of two categories: workflow services and dynamic services. Workflow services define service transactions (behaviors), whereas dynamic services are used to execute workflow services at runtime. The next activities specify and develop the workflow services, as well as the identified dynamic services and their QoS levels.

The C2 architecture style was extended to include workflow and dynamic service bricks. In this approach, there is an explicit mapping relation between features and architectural components (bricks) so that selecting the features for a product generates a corresponding product configuration. The runtime system interacts with service providers through an automated negotiation broker. This broker incorporates a consumer strategy with pluggable QoS negotiation, a QoS ontology, and

SLA evaluation, as well as a provider-rating system to ensure service acceptability.

3.4 RELATED WORK IN THE RESEARCH SUBAREAS

This section presents related work in the two subareas of the state of the art, namely variability modeling and uncertainty management in the open world (see Figure 3.4). Although these areas are not directly focused on dynamic adjustments of service compositions, we describe relevant research works in these areas since our solution is based on them.

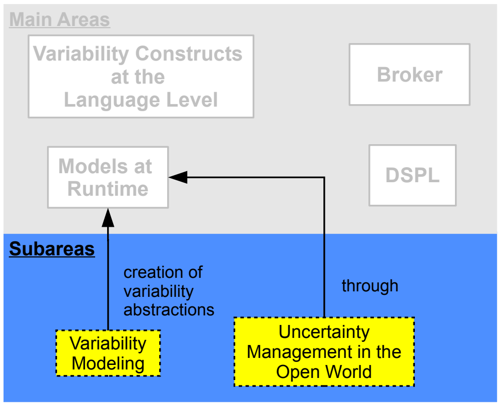

Main Areas
Variability Constructs at the Language Level
Broker
Models at Runtime
DSPL
Subareas
creation of variability abstractions
through
Variability Modeling
Uncertainty Management in the Open World

Figure 3.4: The focus of this section.

First, Subsection 3.4.1 presents relevant approaches for variability modeling. The produced variability abstractions can eventually be leveraged at runtime to guide dynamic adjustments in the service composition. Then, Subsection 3.4.2 describes research works that deal with the inherent uncertainty in the open world. Since our work is fundamentally model-driven, this section focuses on model-driven related work for uncertainty management in context-aware systems.

3.4.1 *Variability Modeling*

There are several approaches that deal with modeling variability in service compositions that support BPs (Nguyen et al., 2011; Sun et al., 2010;

Hadaytullah et al., 2009; Razavian and Khosravi, 2008). A comprehensive set of research works that use SPL principles for variability modeling can be found on (dos Santos Rocha and Fantinato, 2013). In this section we describe three relevant research works. PESOA (Puhlmann et al., 2005) abstracts the BP in an unique model with a set of annotations that identify variable behavior. C-EPC (Rosemann and Van der Aalst, 2007) is a language extension to configure reference BP models that formalize recommended practices for specific domains. A single BP model contains configurable elements, alternatives that depend on the context of use, and context conditions. In (Gottschalk et al., 2008), the authors propose an approach to identify configurable elements of a workflow language (such as WS-BPEL) with opportunities for predefining alternative model versions within a single workflow model.

3.4.2 *Uncertainty Management in the Open World*

This section describes relevant model-driven works that deal with uncertainty of context-aware systems in the open world. In (Ranganathan et al., 2004), authors present a pervasive infrastructure to reason about uncertainty using learning based on Bayesian networks and rules written in probabilistic logic. They use ontologies to reason about uncertain context information. However, they do not offer the mechanisms to adapt the underlying system. There is an interesting research trend towards the analysis of uncertainty using *goal models* at the requirements phase. For instance, in (Welsh et al., 2011), authors propose to attach claims to softgoal contribution links to record the rationale for a choice of goal realization strategy when there is uncertainty about the optimum choice. Nevertheless, it may be difficult to find out the whole set of claims in large and complex systems. In (Cheng et al., 2009b), authors introduce a goal-based modeling approach to develop the requirements for a dynamic adaptive system, while explicitly factoring uncertainty into the process and resulting requirements. Another trend focuses on the generation of models at design time that represent possible target systems suitable for different environmental conditions. For example, in (Goldsby and Cheng, 2008), authors propose a digital evolution-based approach to generate these models at design time. Finally, in (Cheng and Garlan, 2007), authors describe three sources of uncertainty and explain how they address those in the Rainbow Project: the identification of a system problem, the selection of an adaptation strategy, and the determination whether a strategy effected changes on the system successfully. Even though Rainbow includes an architecture model to adapt the system at runtime, uncertainty management techniques are not model-driven.

3.5 CONCLUSIONS

In this chapter, we have presented a report of the state of the art on autonomic service compositions. To this end, we organized this chapter in research areas and subareas (as described in Section 3.1). These areas and subareas were chosen because of their relevance in the state of the art of autonomic service compositions, and their closed relationship with this thesis. In Section 3.3, we carried out the systematic review in the main research areas that have been used to guide autonomic service compositions, namely variability constructs at the language level, brokers, models at runtime, and DSPLs. In order to facilitate the analysis of the works in these main areas, in Section 3.2 we proposed a taxonomy for autonomic service compositions. Finally, in Section 3.4 we presented related work in the following supporting subareas: variability modeling and uncertainty management in the open world.

Table 3.8 presents a summary of the research works in the four main areas that have been presented in this chapter. This table shows that adaptations are caused by external factors in the 61.9% of works, and the nature of change is related to Non-Functional Requirement (NFR)s in the 90.5% of works. Also, the vast majority of the presented works is fully automated (95.2%) and localizes the necessary adaptations – has local scope (95.2%). Moreover, 57.1% of works has a decentralized architecture for adaptation, and the 85.71% of works trigger adaptations to face events (they are event-triggered). There are several works that do not provide information about the frequency of changes (33.3%). Nevertheless, the 47.61% of works show a high tendency to carry out adaptations at arbitrary intervals. We cannot give any conclusion about the adaptation time of the presented solutions because the 42.8% of approaches do not provide this information. The majority of research works deal with autonomic adjustments of service compositions that are implemented with Web services (71.4%).

Dimension	Summary of Research Works
WHAT change is the cause for adaptation	
Source	13 external, 4 external and internal, 3 N/S, and 1 N/A
Type	13 non-functional, 1 functional, 6 functional and non-functional, and 1 N/A
HOW the service composition faces changes	
Autonomy	19 automated, 1 manual, and 1 automated or manual
Organization	12 decentralized, 6 centralized, 2 N/S, and 1 N/A
Scope	18 local, 2 local and global, and 1 N/S
Triggering	16 event-trigger, 2 event- and time-trigger, 2 N/S, and 1 N/A
Safety	4 verified and 17 unverified
Transparency	9 transparent, 2 no transparent, and 10 N/S
WHEN changes are carried out	
Frequency	10 at arbitrary intervals, 3 periodically, 7 N/S, and 1 N/A
Adaptation time	8 short, 4 long, 8 N/S, and 1 N/A
Anticipation	19 foreseen, 1 unforeseen, and 1 N/A
WHERE changes are carried out	
Generality	6 SOA and 15 Web services
Change level	4 at the language level, 9 at the service message level, 6 at the abstract level and in underlying mechanisms (e.g. service message), and 2 approaches bind/unbind components
MATURITY of the approach	
Applicability scenarios	1 PXE, 1 BPWS4J, 7 ActiveBPEL, 1 OSGi, 1 Apache ServiceMix and XTEAM, and 10 N/S
Demonstrability	1 has available code and demonstrations, and 20 N/S
Life-cycle support	11 cover design time and runtime, 3 cover implementation and runtime, and 7 cover runtime
Evaluation	14 use examples, 2 use simulations, 1 uses case studies, 1 uses an example and industrial experiments, 2 use simulations and examples, and 1 N/S

Table 3.8: Summary of the approaches that support autonomic service compositions (N/A means No Applicable, N/S means Not Specified).

According to the maturity-related dimensions, the 66.6% of the studied approaches provide support for autonomic service composition development throughout the life cycle (from design time to runtime or from implementation to runtime). Therefore, we can conclude that the most widely-accepted way to develop autonomic service compositions is when there is guidance during the life cycle. Nevertheless, we cannot conclude the maturity level of the presented approaches because of lack of information: 1) the 47.61% of studied approaches do not specify the enterprise orchestration engines that were used to execute the adjusted service composition; and 2) the 95.2% of works do not provide demonstrability means (such as source code or video demonstration). However, we cannot conclude it is a sign of immaturity. Instead, it may be caused by other facts, such as protection of intellectual property until the project has finished. Finally, although the 66.6% of works provide evaluation results based on small examples, there is a significant percentage of works that use strong evaluation mechanisms (such as simulations or industrial case studies).

The information in this table, together with the literature review, allows to answer the following question: **Q3**, *What are the gaps and challenges to be faced in the future*? (as defined in the introduction of this chapter). Our analysis is as follows:

- **Need for verification:** The vast majority of related work does not provide means to verify autonomic adjustments (80.9%). As a result, there is a need for verifying autonomic adjustments to achieve safe configurations at runtime. Otherwise, the benefits of AC will be diminished with erroneous adjustments.

- **Need for transparency:** The 57.2% of studied research works do not offer a transparent solution (they require changes in the orchestration engine) or do not provide any information about transparency. Thus, we argue that there is a need for transparent solutions to guide dynamic adjustments. A transparent solution can result in higher flexibility to adopt AC in service compositions.

- **Need for facing unanticipated context events:** The vast majority of research works foresee the possible context events at design time (90.5%). In other words, they can anticipate the possible adaptations for these events a priori. This result indicates the need for mechanisms to allow the service composition to face unknown (unanticipated) context events in the open world.

- **Need for abstract mechanisms to guide dynamic adjustments:** On one hand, the use of variability constructs at the language level can hinder reasoning about adaptation with complex and error-prone scripts (Fleurey and Solberg, 2009). On the other hand, brokers have been widely used in the area of QoS-aware Web service composition. A QoS-aware Web service composition can be defined as "the selection of Web Services maximizing the QoS of the overall Web Service composition, taking into account preferences and constraints defined by the user. For this, a utility function maximizing the overall QoS subject to QoS constraints is introduced" (Berbner et al., 2006). However, finding the solution of such a problem leads to an optimization problem that is Non-deterministic Polynomial-time (NP)-hard (Canfora et al., 2008). Moreover, several broker-related approaches are focused on implementation mechanisms that are used at runtime (55.5%), not on guiding systems analysts during the development life cycle. This fact can hinder the implementation of these approaches.

Despite the flexibility that models at runtime can offer to autonomic service compositions, only the 28.6% of the studied research works use highly-abstract mechanisms to guide dynamic adjustments. This result implies that the knowledge in models that are created at design time to reason about the problem and solution domains is not exploited during execution. A better solution can be to leverage these models at runtime to reason about necessary self-adjustments.

Also, we believe that using models at runtime together with DSPLs (which have not been widely exploded for autonomic service compositions, as evidenced in Section 3.3.4), can offer an abstract and flexible solution to dynamically adjust service compositions.

In connection with the research subareas, we conclude the following according to the studied related work:

- Variability modeling approaches are limited to design time. In other words, models are not leveraged at runtime to guide dynamic adaptations. Also, they integrate all possible process variants in a single model. It results in large and difficult-to-understand models.

- The analyzed model-driven works that deal with uncertainty of context-aware systems in the open world offer interesting solutions. They model uncertainty at the requirements and design phases for latter use at runtime. However, the models do not evolve at runtime to face arising problematic unknown context events. Therefore, the capacity of reaction to face new unknown context events decreases because the initial models are unable to support them. Moreover, this situation affects the feasibility of models at runtime as a means to guide the dynamic evolution of critical systems that cannot be stopped to modify the supporting models.

4 OVERVIEW OF THE APPROACH

"We are just in the kindergarten of uncovering things; there is no downcurve in science."

— Charles F. Kettering

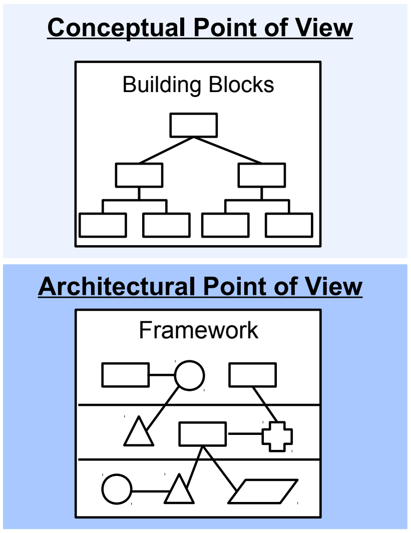


Figure 4.1: This chapter presents an overview of our approach from two points of view.

In order to achieve autonomic behavior for service compositions, we argue that the models that are produced as artifacts from MDE methodologies can be leveraged during execution to drive autonomic behavior. In MDE, a model is an abstraction or reduced representation of a system that is built for specific purposes (e.g. to abstract the underlying computing infrastructure). We share this view of what constitutes a model and explore the use of models at runtime to drive the autonomic behavior of service compositions.

Our decision to use models at runtime to achieve autonomic service compositions originated from two reasons. First, if models reflect the service composition and its operational context, then they can provide up-to-date and exact information to drive subsequent adaptation decisions. Secondly, if models are causally connected, then adaptations can be made first at the modeling level rather than at the service composition level.

This thesis provides a model-driven framework to guide the dynamic adjustment of context-aware autonomic service compositions. At design time, we propose the creation of models that guide autonomic changes. Service operations are expressed as SPL features in a variability model to guide dynamic adaptations. In order to reach optimum reconfigurations at runtime, the variability model and its possible configurations are verified at design time.

At runtime, in order to face problematic context events, the variability model is leveraged for decision-making. The activation and deactivation of features in the variability model result in changes in a composition model that abstracts the service composition. Changes in the composition model are reflected into the service composition by adding or removing fragments of WS-BPEL code, which are deployed at runtime. Our framework also covers the dynamic evolution of service compositions to deal with unknown context events in the open world. When facing uncertainty, the supporting models self-evolve according to abstract tactics that preserve requirements.

Our approach is introduced from two related points of view.

Our approach is introduced from two related points of view (see Figure 4.1): First, we introduce our approach in terms of building blocks from a *conceptual point of view*. Then, we describe our framework to achieve autonomic service compositions from an *architectural point of view*.

The remainder of this chapter is structured as follows. Section 4.1 introduces a case study that illustrates the need for autonomic service compositions. This case study is used throughout this dissertation to exemplify several key aspects of our approach. Section 4.2 presents the main conceptual building blocks of this thesis. Section 4.3 introduces our framework to achieve autonomic service compositions. Section 4.4 presents the conclusions.

4.1 CASE STUDY

To illustrate the need for autonomic service compositions, we introduce a composite service that supports online book shopping at Orange Country Bookstore. The example is specified with BPMN in Figure 4.2. BPMN tasks express Web service operations (e.g. UPS Shipping service); and BPMN subprocesses express composite service operations (e.g. Barnes & Noble Books composite service). Appendix B shows the technical details of this case study.

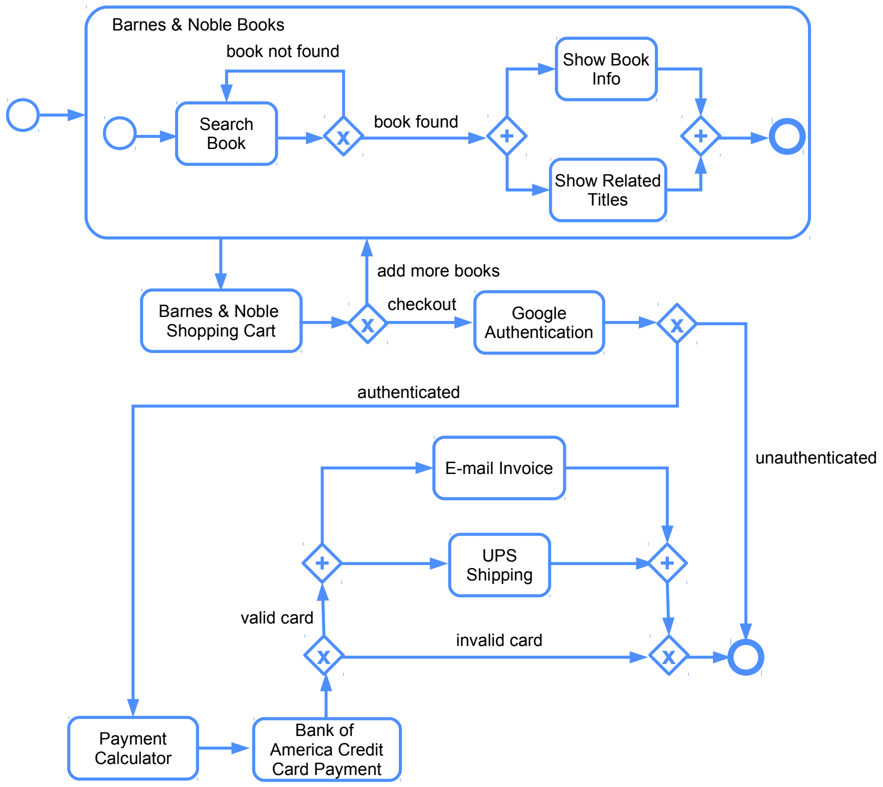


Figure 4.2: A BPMN model that represents a composite service for online book shopping.

The BP starts when a customer looks for a book on the website of Orange Country Bookstore. The first thing the customer wants to do is identify the books to purchase. The searching operation is provided by the Search Book Web service, which is part of the Barnes & Noble Books composite service. When a book is found, then the book information is returned to the customer by the Show Book Info Web service while at the same time the information for other related books is listed by the Show Related Titles Web service. If no book is found, then the customer must refine the search, e.g. using supplementary or different search criteria, or undertake another search. In the next step, the customer adds books into the shopping cart through the Barnes & Noble Shopping Cart Web service. The process can start over again until the customer is satisfied with his or her selection. When the customer is ready to checkout, he or she has to be authenticated by the Google Authentication Web service. The process continues if the costumer is registered. Otherwise, the process terminates. Then, the in-house Payment Calculator Web service calculates the total amount to be paid. The payment is done through the Bank of America Credit Card Payment Web service. Finally, if the credit card information is valid, the in-house E-mail Invoice Web service sends an e-mail to the customer with the invoice while the UPS Shipping Web service is invoked to deliver the book. Otherwise, the process terminates. The technical details of this case study are described in Appendix B.

The technical details of this case study are described in Appendix B.

Different context events may arise in this heterogeneous infrastructure, which call for adaptations. For example, any third-party Web service operation may fail or perform below required SLAs. The drivers to carry out dynamic adaptations in this example are as follows: 1) since this service composition supports a critical short-running BP, it is impossible to shut down the system to make adaptations. Autonomous reconfigurations could reduce the burden of the staff while improving the reaction to deal with arising context events; and 2) as a business differentiator, the online-book-shopping process requires high availability and high performance. Availability deals with the readiness for correct service in a specific time (Cotroneo et al., 2002). Performance can be measured by observing the execution time that a Web service takes to execute a job (response time plus execution time) (Ameller and Franch, 2008). Service operations that violate these quality attributes trigger service recompositions.

Implementing scripts with predefined adaptation actions to guide dynamic adaptations can make it difficult to reason about the service composition as it grows. Moreover, if there are no predefined adaptation actions for a particular context situation, then no adaptation is carried out. These situations help us to identify the following challenges for autonomic context-aware service compositions:

- The actions to guide the service composition to a better configuration should be sufficiently expressive and easy-to-understand in order to facilitate the development of the logic behind autonomic management.

- A software engineering approach should support the development of autonomic service compositions from design to execution.
- Autonomic service compositions should be able to count on corrective actions that trigger the dynamic evolution of the service composition to preserve the expected requirements when facing unknown context events.

4.2 MAIN BUILDING BLOCKS

Figure 4.3 presents the main building blocks of the proposed approach from a conceptual point of view. Each block is denoted by a feature. The approach is divided into two main building blocks: `Design Block` and `Runtime Block`. At the `Design Block`, we propose the creation of a set of models that are used to support dynamic adjustments of the service composition. At the `Runtime Block`, the models created at design time are queried in response to context events to reconfigure the service composition. The building blocks of our approach are described in the following subsections.

Our approach is divided into two main conceptual blocks: Design and Runtime.

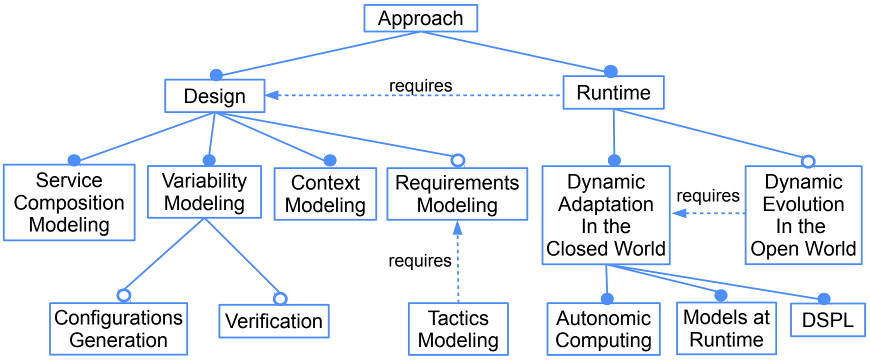


Figure 4.3: Main building blocks of the approach from a conceptual point of view.

4.2.1 *Design-Related Building Blocks*

The following blocks support the creation of abstractions at design time to guide autonomic service compositions during execution:

- **`Service Composition Modeling:`** In our approach, dynamic adjustments are carried out first at the modeling level and then injected into the running service composition. Therefore, it is necessary to count on an abstraction of the underlying service composition. This model has to be causally connected to the service composition in order to reflect the changes in this model into running service operations.

- **`Variability Modeling:`** Variability models are used to describe the variants in which the service composition can self-adapt. We argue that in response to changes in the context, the system itself can query a variability model to determine the necessary modifications in the service composition. Feature modeling was chosen for variability modeling because it has good tool support for variability reasoning. There are two optional subblocks related to variability modeling:

  – **`Configurations Generation:`** The *configuration* of a service-based system is the set of all active features in its variability model at a particular moment. At runtime, the system queries the adaptation space (with all the possible configurations of the variability model) in order to adapt from one configuration to another. Although it is possible to define the adaptation space manually, it can become a cumbersome operation in large systems. Therefore, we propose a tool to automatize the generation of variability model configurations.

  – **`Verification:`** Even though it is possible to use unverified variability models at runtime to guide dynamic adaptations, it is a very error-prone task. We argue that a best practice for SAS is to ensure that system configurations are not invalid in a given contextual situation. Therefore, we propose to verify the variability model and its possible configurations prior execution to ensure safe service recompositions.

- **`Context Modeling:`** In order to carry out dynamic adjustments in the service composition to face arising context events, first it is necessary to count on an abstraction of the context. This abstraction can be used at runtime to reason about the current contextual situation. We propose an ontology-based context model that leverages Semantic Web technology. In addition, we define context conditions as queries to the context model. The fulfillment of these context conditions triggers the reconfiguration of the service composition.

- **`Requirements Modeling:`** According to the Institute of Electrical and Electronics Engineers (IEEE), *requirement* can be defined as: "1) A condition or capability needed by a user to solve a problem or achieve an objective. 2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. 3) A documented representation of a condition or capability as in 1 or 2" (IEEE, 1990). A requirements model describes the requirements that the service composition must preserve at runtime. These requirements have to be fulfilled despite arising unknown context events. In this work, we are particularly interested in keeping NFRs at runtime (e.g. performance and availability). Since the requirements model is leveraged during dynamic evolutions (in the open world), it is considered an optional block.
- **`Tactics Modeling:`** In our approach, *tactics* are last-resort surviving actions to be used when the service composition does not have predefined adaptation actions to deal with arising problematic context events in the open world. Writing complex scripts to specify tactics can be cumbersome. Therefore, we propose the creation of highly-abstract tactic models that can be used to express the tactical functionality to be triggered on the underlying service composition to preserve affected requirements (one requirement can be preserved by many tactics). There is a *requires* relationship between the `Tactics Modeling Block` and the `Requirements Modeling Block` because tactics exist to preserve requirements.

4.2.2 *Runtime-Related Building Blocks*

The following blocks support autonomic service compositions during execution:

- **`Dynamic Adaptation in the Closed World:`** This block is focused on the closed-world assumption, in which all context events are foreseen at design time. Predefined actions (in terms of the activation or deactivation of features in the variability model) guide adaptations in the service composition according to known (or foreseen at design time) context events. The following subblocks are the underpinnings for the dynamic adaptation of service compositions:
  - **`Autonomic Computing:`** AC has evolved as a discipline that covers the broad spectrum of computing in domains as diverse as mobile devices (White et al., 2007) and home-automation (Cetina et al., 2009; Morin et al., 2009b), thereby demonstrating its feasibility and value by automating tasks such as installation (Bhola et al., 2006), healing (Zenmyo et al., 2006), and updating (Fenson and Howard, 2004). In order to support the dynamic adaptation of service composition through AC, we propose a computing infrastructure that implements the components of *IBM*'s MAPE-K loop (Horn, 2001).
  - **`Models at Runtime:`** In our approach, the set of models that are created under the `Design Building Block` (in Section 4.2.1) are used at runtime to automatically determine how the service composition should be adjusted.
  - **`DSPL:`** DSPLE goes a step further from SPL with the investigation of development issues for reusable and dynamically reconfigurable core assets. When features are activated or deactivated at runtime due to changes in the context, a DSPL architecture supports the dynamic service recomposition.
- **`Dynamic Evolution in the Open World:`** Predefined adaptation actions for known context events in the closed world are not enough in the open world where several unknown context events can arise (e.g. sudden security attacks). Despite the recognized need for handling unexpected events in SAS (Cheng et al., 2009a; Calinescu et al., 2012), the dynamic evolution in the open world of service compositions is still an open and challenging research topic. In the open world, our approach tries to reduce the impact of unknown context events on expected requirements (described in a requirements model) with a group of tactics (described in tactic models). Therefore, the open world can be seen as:

Open world = ($\sum$ unknown context events that can be handled by tactics) $\cup$ ($\sum$ unhandled unknown context events).

The `Dynamic Evolution in the Open World Block` can be considered as an extension of the `Dynamic Adaptation in the Closed World Block`. Therefore, the former requires (depends on) the latter at runtime.

4.3 A FRAMEWORK FOR AUTONOMIC SERVICE COMPOSITIONS

We propose the following strategy to offer a solution for the dynamic adjustment of service compositions. First, the service composition is modeled at design time. Then, we introduce mechanisms to express where and how service compositions can be adapted or evolved to face arising context events. These mechanisms are expressed as easy-to-understand and as highly-abstract as possible. At runtime, we provide an infrastructure that detects changes in the context and enables dynamic adjustments.

In order to make this strategy a reality, we propose a framework that states the models, tools, and artifacts to support dynamic adjustment of service compositions from design time to runtime. This framework is depicted from an architectural point of view in Figure 4.4. This framework consists of three phases: `Design`, `Dynamic Adaptation`, and `Dynamic Evolution`.

Our framework supports the dynamic adjustment of service compositions from design time to runtime.

According to the main conceptual building blocks in Section 4.2: 1) the `Design Phase` implements the `Design Block`; 2) the `Dynamic Adaptation Phase` implements the `Dynamic Adaptation in the Closed World Block`; and 3) the `Dynamic Evolution Phase` implements the `Dynamic Evolution in the Open World Block`. These phases are described in the following subsections.

4.3.1 *Design Phase*

In order to support dynamic adaptations in our framework, it is necessary to count on abstractions that represent the context, the dynamic configurations of the service composition, and the service composition itself. Also, it is necessary to create the adaptation policies that move the service composition to new configurations.

Specifically, the `Design Phase` covers the creation of the following models (see the top of Figure 4.4). A *composition model* describes the

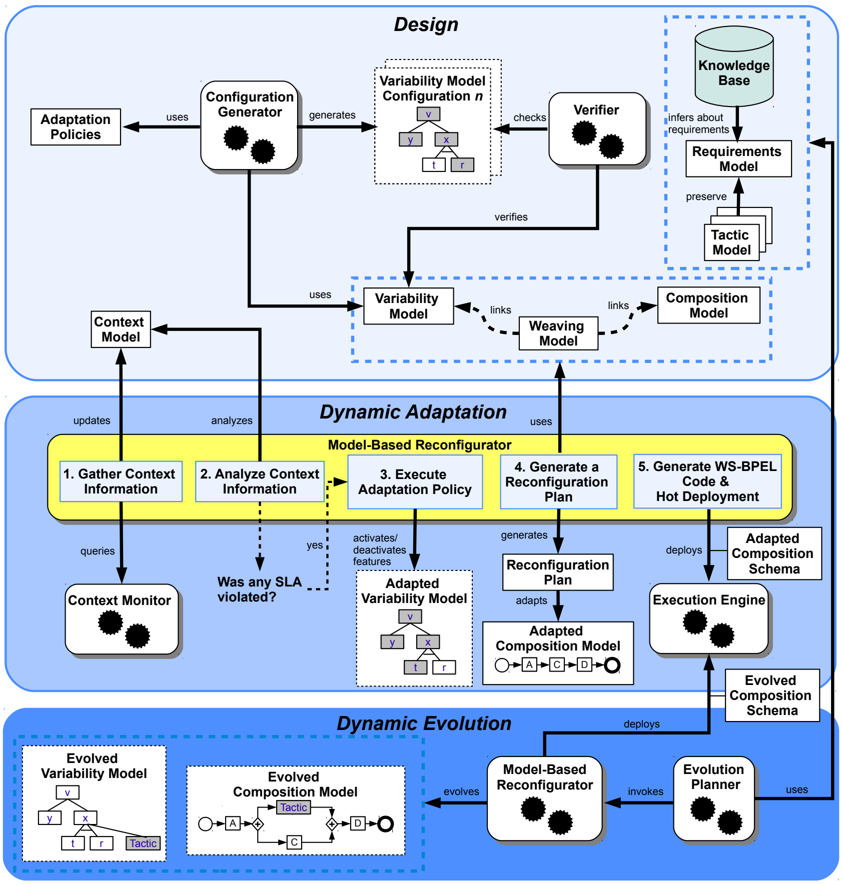

Design
Knowledge Base
infers about requirements
Requirements Model
preserve
Tactic Model
Adaptation Policies
uses
Configuration Generator
generates
Variability Model Configuration *n*
checks
Verifier
verifies
uses
Variability Model
links
Weaving Model
links
Composition Model
Context Model
Dynamic Adaptation
updates
analyzes
uses
Model-Based Reconfigurator
1. Gather Context Information
2. Analyze Context Information
3. Execute Adaptation Policy
4. Generate a Reconfiguration Plan
5. Generate WS-BPEL Code & Hot Deployment
queries
yes
Was any SLA violated?
activates/ deactivates features
generates
deploys
Adapted Composition Schema
Context Monitor
Adapted Variability Model
Reconfiguration Plan
adapts
Adapted Composition Model
Execution Engine
Evolved Composition Schema
Dynamic Evolution
deploys
Evolved Variability Model
Evolved Composition Model
Tactic
evolves
Model-Based Reconfigurator
invokes
Evolution Planner
uses

Figure 4.4: A framework for the dynamic adjustment of service compositions.

service composition. A *variability model* describes the dynamic configurations of the service composition in terms of activation or deactivation of features. Thus, the knowledge that is captured by this model is the basis for *adaptation policies*. Since the initial composition model may lack support for variability, we propose to extend this model with variation points where variants can be injected at runtime. The variability-related information to carry out this extension is based on the variability model. We also propose the creation of two additional supporting models. First, a *context model* formalizes collected context knowledge. Second, since changes in the variability model guide adaptations in the service composition, which is abstracted in the composition model, we propose a *weaving model* to connect these two models.

Two tools provide variability reasoning at design time: 1) the Configuration Generator uses the variability model and the set of adaptation policies to automatically generate the *adaptation space* with all the possible configurations of the variability model; and 2) the Verifier uses CP to verify the variability model and check that the generated configurations respect the constraints imposed by the variability model. Verification of the variability model entails finding undesirable properties, such as contradictory information or the impossibility to offer a valid configuration for a particular context. If there are errors in the variability model, they will inevitably spread to an undefined number of configurations, which can drastically diminish the quality and outcome of the entire adaptation.

The dynamic adaptation of context-aware service compositions is possible by adjusting models at runtime through predefined adaptation actions. This approach can work fine under the closed-world assumption. However, predefined adaptation actions are not enough in the open world where several unforeseen context events can arise. These unknown events create uncertainty in the way the system should face them. Therefore, we propose to manage problematic unknown context events through the dynamic evolution of the service composition.

The corrective actions to deal with uncertainty are expressed as abstract *tactic models*. *Tactics* are last-resort surviving actions or strategies to preserve the requirements that can be negatively impacted by unknown context events. Therefore, tactics trigger the dynamic evolution of the service composition to preserve requirements at runtime. Requirements are represented in an abstract way in a *requirements model*. At runtime, an AI mechanism looks for the requirements that can be affected by an unknown context event. Therefore, it is also necessary to define in this phase a set of *rule premises* to evaluate arising context facts against them at runtime. These rules are kept in a *knowledge base*.

4.3.2 *Dynamic Adaptation Phase*

In the `Dynamic Adaptation Phase`, the models and adaptation policies that are created in the `Design Phase` are used to guide the self-adaptation of the service composition (see the middle of Figure 4.4). This phase is framed in the closed-world assumption, in which possible context events, and the necessary adaptations for those events, are fully known at design time.

The proposed infrastructure carries out the following steps to support dynamic adaptations. First, the Model-Based Reconfigurator queries the context information that is collected by the Context Monitor and updates the context model accordingly. Then, the Model-Based Reconfigurator determines if any SLA has been violated in the context according to the information in the context model. If any SLA has been violated, the Model-Based Reconfigurator executes an adaptation policy that indicates the activation or deactivation of features in the variability model (i.e., to move to a new configuration). Then, the set of active features in the new configuration of the variability model is used to generate a *reconfiguration plan*, which is used to modify the elements in the composition model accordingly. Modifications in the composition model are reflected into the service composition by adding or removing fragments of WS-BPEL code from a WS-BPEL template. The adapted composition schema is hot deployed on the Execution Engine. In turn, the Execution Engine uses the adapted WS-BPEL composition schema to orchestrate the service composition.

4.3.3 *Dynamic Evolution Phase*

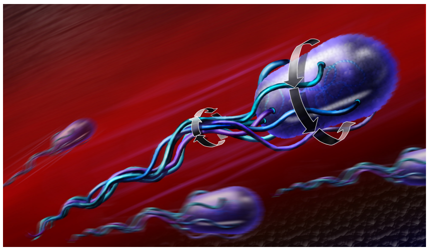
In the `Dynamic Evolution Phase`, the knowledge in models is used to guide the dynamic evolution of the service composition (see the bottom of Figure 4.4). Therefore, this phase is focused on the open world. To this end, the Evolution Planner queries the information that is collected by the Context Monitor to find out if a requirement in the requirements model can be negatively impacted by an unknown context event. To this end, the Evolution Planner uses the set of inference rules in the knowledge base to realize the requirements that can be affected. Since we are interested in managing uncertainty that arises from the context in which the service composition is deployed, our approach is related to *external uncertainty* (Esfahani et al., 2011). In order to preserve affected requirements, the Evolution Planner chooses surviving tactics. According to the chosen tactics, the Model-Based Reconfigurator evolves the variability and composition models by means of tactic models. Finally, the Model-Based Reconfigurator uses these evolved models to evolve the WS-BPEL composition schema, which in turn is hot deployed on the Execution Engine.

4.4 CONCLUSIONS

In this chapter, we have presented an overview of our approach to guide the dynamic adjustment of context-aware autonomic service compositions. In order to illustrate the need for autonomic service compositions, we introduced a case study in Section 4.1. Then, our approach was introduced from two points of view. First, it was described in terms of building blocks from a *conceptual point of view* in Section 4.2. Afterwards, we described a framework to achieve autonomic service compositions from an *architectural point of view* in Section 4.3. This framework spans over design time and runtime.

Part II

PAVING THE WAY FOR THE DYNAMIC ADJUSTMENT OF SERVICE COMPOSITIONS



How Escherichia coli move.

Image credit: Nicolle Rager Fuller, National Science Foundation.

Biologist Howard Berg at Harvard describes the bacterial flagellar motor as a "nanotechnological marvel" (Berg, 2003). This motor has no more than 50 nanometers (nm) in diameter and spins clockwise or counterclockwise at speeds on the order of 100 hertz (Hz). Flagella receive feedback from the context and can self-adjust their course in response to external stimuli. The flagellar motor has numerous proteins involved in its function. If just one of these proteins is taken away, either the flagellum is not produced, or it does not function at all.

As the bacterial flagellum exhibits remarkable intelligent design, we argue that SAS have to be designed first in order to accomplish a particular task at runtime.

In this part, we present a solution to design autonomic service compositions before execution. Part II is structured as follows. Chapter 5 describes a set of models that are created at design time to support dynamic adaptation in the closed world. Chapter 6 describes a set of models that can be leveraged at runtime to face unknown context events in the open world.

5 MODEL-DRIVEN DESIGN FOR DYNAMIC ADAPTATION

"Before anything else, preparation is the key to success."

— Alexander Graham Bell

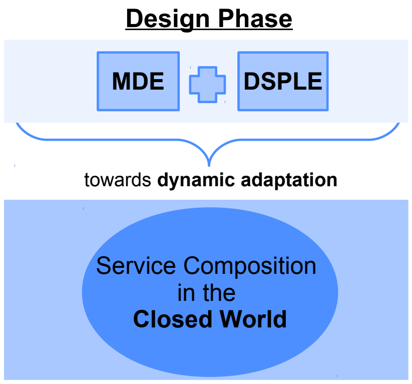

Design Phase
MDE
DSPLE
towards dynamic adaptation
Service Composition in the Closed World

Figure 5.1: Scope of Chapter 5.

The importance of software design has been discussed for quite a long time (Miller, 1989). A good software design is even more important when the created models are used to guide dynamic adjustments of the service composition. Therefore, in this chapter we introduce the models and other artifacts that are created in the `Design Phase` to support the dynamic adaptation of service compositions in the closed world (see Figure 5.1).

In order to facilitate the implementation of our approach, the creation of models is organized in a software process model, which combines DSPLE and MDE. In traditional SPLE, software products are developed by selecting and configuring shared core assets at design time. In this chapter, we extend this concept with DSPLE, which can be used to bind variation points at runtime to adapt the service composition according to changes in the context.

In order to carry out adaptations in a higher level of abstraction, our DSPLE approach is based on models that are leveraged at runtime for decision making. To this end, we propose to base the software process model on MDE in order to create these models at design time. Among the set of models, a *variability model* describes the *variants* (representations of variability objects within domain artifacts (Pohl et al., 2005)) in which a service composition can change at runtime. Thus, the variability model is appointed to solve the need for expressive and easy-to-understand adaptation policies. Since adaptations at the modeling level need to be replicated into the service composition at runtime, we also propose the creation of WS-BPEL *code fragments*, which represent variants that can be bound in variation points at runtime.

Autonomic adaptations can be used to support critical operations (e.g. in means of transportation or banking). Therefore, it is essential that resulting configurations are verified before they are applied on the running system. Otherwise, the benefits of self-adaptation may be harmed by erroneous undesirable results. In other words, it is necessary to have confidence on adaptations. Therefore, in this chapter we also offer a solution to carry out the verifications of the possible reconfigurations at design time.

The remainder of this chapter is structured as follows. Section 5.1 introduces our approach to achieve autonomic service compositions by combining MDE and DSPLE. Section 5.2 describes a process model to engineer the dynamic adaptation of autonomic service compositions at design time with our MDE-DSPLE approach. Section 5.3 presents the conclusions.

5.1 ACHIEVING AUTONOMIC SERVICE COMPOSITIONS WITH MDE AND DSPLE

SPLE has proven to support systematic reuse across the set of similar products that software companies offer (Clements and Northrop, 2001). In SPLs (or systems families), products are derived by selecting the features that are part of a product and removing those that are not part of it. Therefore, SPLE is about exploiting commonalities among a set of systems in a particular domain while managing the variabilities among them in order to improve time to market, achieve systematic reuse goals, and improve product quality. Commonalities are the elements with the highest reuse potential and variabilities represent capabilities to change or customize a system (Geyer and Becker, 2002).

According to (BigLever Software, Inc., n.d.), SPLs can be described in terms of the following concepts, as illustrated in Figure 5.2:

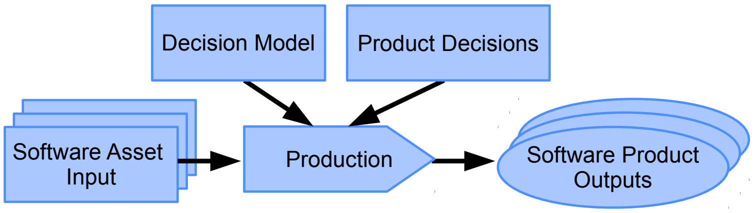


Figure 5.2: Basic SPL concepts (BigLever Software, Inc., n.d.).

- **`Software asset inputs:`** They are a collection of software assets (such as requirements, source code components, test cases, architecture, and documentation) that can be configured and composed in different ways to create all of the products in a SPL. To accommodate variation among the products, some of the assets may be optional and some of the assets may have internal variation points that can be configured in different ways to provide different behavior.

- **`Decision model and product decisions:`** The *decision model* describes optional and variable features for the products in the SPL. Each product in the SPL is uniquely defined by its *product decisions* (choices for each of the optional and variable features in the decision model).

- **`Production mechanism and process:`** They are the means for composing and configuring products from the software asset inputs. Product decisions are used during production to determine which software asset inputs to use and how to configure the variation points within those assets.

- **`Software product outputs:`** They are the collection of all products that can be produced for the SPL. The scope of the SPL is determined by the set of software product outputs that can be produced from the software assets and the decision model.

As described in Chapter 2, given the success of SPLs to produce families of products, DSPLs offer a promising strategy to manage variability at runtime. In order to reach autonomic service compositions with DSPLE, the aforementioned SPLE-related concepts have to be contextualized for DSPLE. To this end, we propose to support the different aspects of the DSPL from a highly-abstract point of view with MDE.

Our approach uses MDE to represent the different aspects of the DSPL from a highly-abstract point of view.

MDE aims to capture every important aspect of the software through appropriate models. Compared to implementation code, models capture the intentions of the stakeholders more directly, avoid accidental implementation details, and are more amenable to analysis. In MDE, models are not just auxiliary documentation artifacts. Instead, they are source artifacts and can be used for automated analysis. Figure 5.3 depicts the models that support the DSPL for autonomic service compositions.

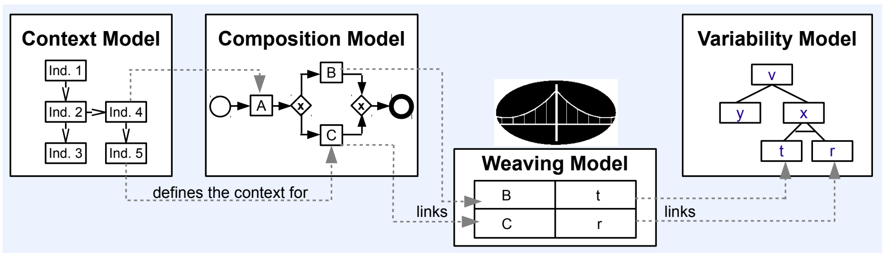


Figure 5.3: Models that support the DSPL for autonomic service compositions.

First, we propose the following *software asset inputs*: 1) a *composition model* abstracts the underlying service composition; and 2) a *context model* abstracts the collected contextual data of the service operations, which are abstracted in the composition model.

In our approach, the *decision model* is implemented as a *variability model*. The variability model describes the variants in which a service composition can change at runtime. Thus, this model is appointed to solve the need for expressive and easy-to-understand adaptation policies.

Since the composition model abstracts the service composition, the definition of a bridge between the elements in the variability model and the elements in the composition model could be used to support dynamic adaptations in the underlying service composition. To this end, we propose a *weaving model* as an additional *software asset input* to project the changes in variability model's features on abstract elements at the composition model. In other words, the weaving model works as a *bridge* between the elements in these models.

Adaptation policies are the *product decisions* that activate or deactivate features in the variability model to manage context events during execution. Therefore, adaptation policies depend upon information in the context model. The application of adaptation policies on the variability model produces the *adaptation space* with 1) all the possible configurations of the service composition (in terms of active and inactive features in the variability model) and 2) the transitions among configurations. In order to avoid problematic adaptations in critical service compositions, we argue that the variability model and its possible configurations, which are described in the adaptation space, to be verified before runtime.

At runtime, a *production mechanism* leverages the aforementioned elements to guide the dynamic adaptation of the service composition. Therefore, the *software product outputs* are the different configurations of the WS-BPEL composition schema.

5.2 A PROCESS TO DESIGN DYNAMIC ADAPTATIONS

In order to engineer autonomic service compositions at design time, we use software process modeling to help to implement our MDE-DSPLE approach in industrial scenarios. In general, a *process* can be defined as a series of actions or operations conducing to an end. In software, a process is constituted by engineering activities to develop and evolve a software product (Osterweil, 1987). A *process model* specifies how a process is carried out (Andersson et al., 2013). Specifically, a process model defines a partially ordered set of what is done, when, where, and by whom (Curtis et al., 1992). The materialization of processes in models enables communication, coordination, and human understanding (Andersson et al., 2013; Curtis et al., 1992; Osterweil, 1987).

We use software process modeling to help to implement our MDE-DSPLE approach in industrial scenarios.

The UML Activity diagram in Figure 5.4 shows the proposed process model to design dynamic adaptations with our MDE-DSPLE approach. This process is composed of a sequence of activities. Chapter 9 describes the elements that support this process model (e.g. tools). Each one of these activities is described in the following sections.

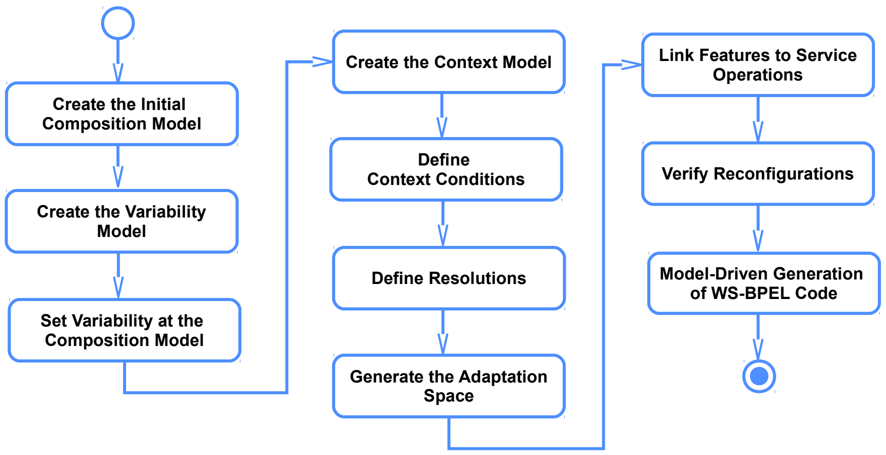

Create the Initial Composition Model
Create the Variability Model
Set Variability at the Composition Model
Create the Context Model
Define Context Conditions
Define Resolutions
Generate the Adaptation Space
Link Features to Service Operations
Verify Reconfigurations
Model-Driven Generation of WS-BPEL Code

Figure 5.4: A process model to design dynamic adaptations with our MDE-DSPLE approach.

5.2.1 *Create the Initial Composition Model*

In our MDE-DSPLE approach, the initial *composition model* is an input for the production of configurations of the service composition. This model abstracts the underlying current service composition "as is" (without variability notations). This model illustrates the following: 1) the service operations, which are involved in the service composition; and 2) the sequence flows among service operations, which state the order in which the operations are performed. The composition model is causally connected to the underlying service composition (i.e., if this model changes, then the service composition changes and vice versa).

Different modeling notations can be used to express the sequences and dependencies among Web services and composite services (e.g. UML Activity Diagram or BPMN). In this work, a BPMN model was chosen to represent the elements in the service composition because BPMN is a user-friendly notation that is suitable to express sequences and dependencies among Web services and composite services (Ayora et al., 2012).

We propose the following rules to abstract a service composition in the composition model: 1) atomic Web service operations are abstracted as BPMN tasks (since they represent atomic activities); 2) composite service operations are abstracted as BPMN subprocesses (since they represent compound activities); and 3) the workflow that is followed by the service composition is abstracted by BPMN gateways and sequence flows that connect BPMN tasks and subprocesses.

Example:

Figure 4.2 shows the initial composition model of our case study. This model abstracts the underlying service composition.

5.2.2 *Create the Variability Model*

Even though the initial composition model represents the underlying service composition, it lacks semantics for variability. Therefore, it is necessary to count on feasible, semantically-rich, and coarse-grained variability representations of service compositions.

We propose to create a variability model to describe the variants in which a service composition can self-adjust. These variants may provide better QoS, offer new services that did not make sense in the previous context, or discard some other services (Morin et al., 2009a; Cetina et al., 2009). Different techniques can be used to create a variability model that is aligned with the elements in a composition model (Chen et al., 2005; Bae and Kang, 2007; Montero et al., 2008; Alférez and Pelechano, 2011b).

The variability model describes the variants in which a service composition can self-adjust.

Our approach requires a variability modeling technique to implement the variability model, such as feature modeling (Kang et al., 1990), the CVL (Haugen et al., 2008), or any domain-specific language to express variability. In this work, feature modeling was chosen for variability modeling and analysis because it can offer coarse-grained variability management of service compositions and it has good tool support for variability reasoning[1]. Specifically, features represent the functionalities of the Web-service-based system in a coarse-grained fashion (i.e., one feature can abstract η service operations, which support a particular functionality). Therefore, dynamic adaptations are carried out to keep the features of the system at runtime when context changes are faced.

Feature modeling was chosen for variability modeling and analysis.

We propose to appoint certain features in a feature model as variants that may be used to solve context events and preserve the functionality of the service composition at runtime. The feature model also has variation points that express decisions leading to different variants at runtime. Since only one variant can be chosen at a time in a particular

1 Since a *feature model* "implements" the *variability model*, these two concepts are interchangeable in this document.

variation point, there is an alternative relationship between a variation point and its variants.

The current configuration expresses the set of "active" features at a particular time.

The *current configuration* concept expresses the set of features in the feature model with "active" state at a particular time. Thus, the current configuration indicates the functionalities that are provided by a composite service at a specific moment. Before execution, we propose to define the initial configuration (i.e., the configuration that is used when the system starts). This configuration depends on BP requirements. For example, the functionality provided by Bank of America is the preferred one for credit card payment. Therefore, the system starts with this functionality.

Example:

Figure 5.5 shows the feature model for our case study. For instance, the `UPS Shipping`, the `FedEX Express`, and the `DHL Delivery` features are variants that can be used during execution to accomplish the shipment functionality in the `Shipment` variation point. The features in blue express the current configuration when the service composition starts to run.

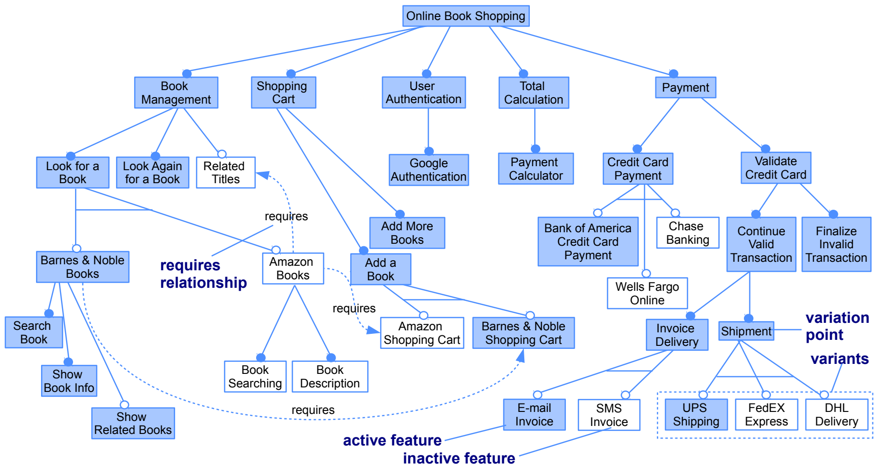


Figure 5.5: Feature model for our case study.

5.2.3 *Set Variability at the Composition Model*

The initial composition model, introduced in Section 5.2.1, does not have semantics for variability. Nevertheless, this model should be able to change at runtime when facing arising context events. Also, a service composition dictates an ordered main workflow that has to be preserved in the composition model after adaptations have taken place. Nevertheless, the feature model does not provide this sequence-related information. Therefore, in order to solve these situations, we propose the creation of the following models:

The base composition model and the set of variant models extend the initial composition model with semantics for variability.

1. A *base composition model* extends the initial composition model with semantics for variability and preserves the main workflow during adaptations. This model defines a set of commonalities (i.e., elements that are shared by all the configurations of the service composition) and variation points. Variant behavior is injected at runtime on the base composition model through *variant models*. Our approach does not require the creation of a new kind of notation to describe the variability points in the base composition model. In our case, variation points in a BPMN base composition model are expressed as BPMN subprocesses.

2. A set of *variant models* to be bound into the variation points of the base composition model during execution. The prerequisite to bind variant models into the base composition model is that both of them conform to the same metamodel. For instance, the base composition model and the set of variant models of our case study are specified in BPMN. The elements in variant models are encapsulated into a BPMN subprocess.

The base model and the set of variant models were inspired by the Base-Variation-Resolution (BVR) approach (Bayer et al., 2006). The idea behind BVR is to define orthogonal variation models that apply to a single base model. On one hand, a *Base Model* is a model in any language (textual or graphical). On the other hand, a *Variation Model* is a collection of variation elements, which represent something with variable nature.

The reasoning about what is common and variable in the base composition model is supported by the knowledge in the variability model. On one hand, a variation point in the variability model indicates that a variation point for a particular functionality has to be specified in the base composition model. On the other hand, each variant in the variability model indicates that a particular variant model has to be created.

Example:

Figure 5.6 shows the base composition model and two variant models in our case study[2].

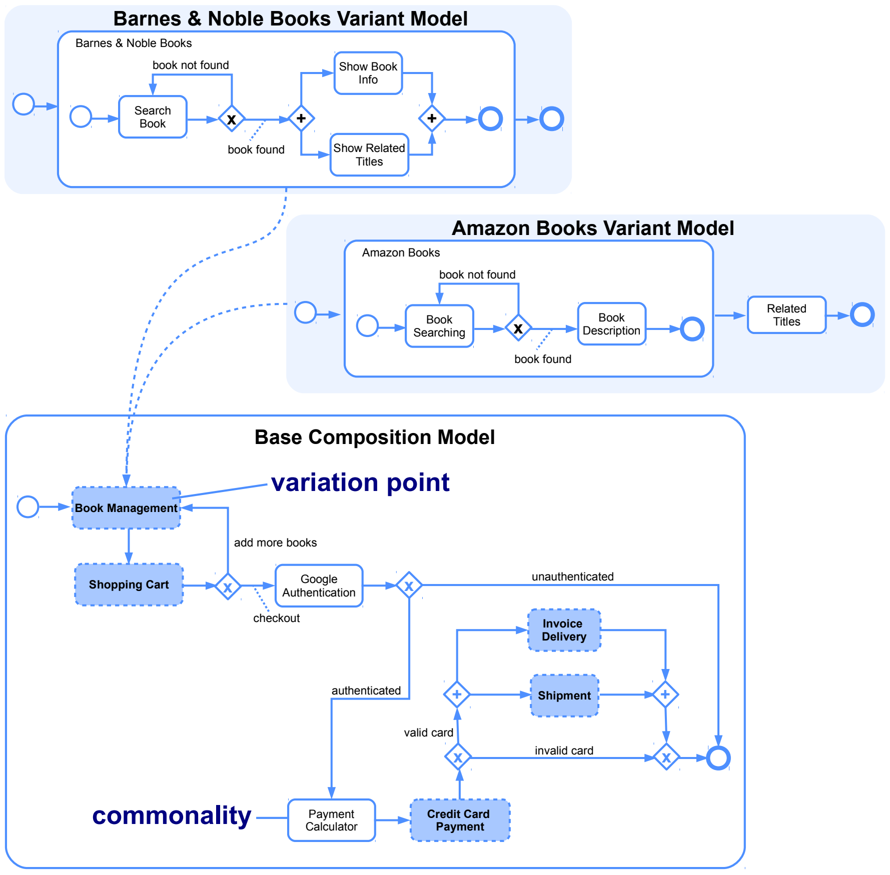


Figure 5.6: Base composition model and two variant models in our case study.

2 For readability reasons, variation points are surrounded by a dashed line and are in blue. Nevertheless, they are normal BPMN subprocesses.

At runtime, commonalities and the main workflow remain constant in the base composition model (e.g. the workflow indicates that in every adaptation the user authentication is carried out first, then the payment). However, the variation points can be bound with different variant models. The base composition model and the set of variant models conform to the BPMN metamodel. The variability model in Figure 5.5 states that the `Look for a Book` variation point has two variants: the `Barnes & Noble Books` and the `Amazon Books` variants. Therefore, this information was used to create two variant models, one for each variant.

5.2.4 *Create the Context Model*

In order to solve the need for expressing the context in a way that supports formal reasoning of its current status and possible arising situations, we propose an ontology-based *context model* that leverages Semantic Web technology. Specifically, we make use of the Web Ontology Language (OWL) (Dean and Schreiber, 2004) to support the formal analysis of the contextual information that is captured at runtime. OWL extends the expressivity of the Resource Description Framework (RDF) (W3C, 2004) by adding an additional layer of semantics on top of RDF. With RDF, the contextual knowledge can be decomposed into small pieces, with some rules about the meaning (or semantics) of those pieces.

We propose an ontology-based context model that leverages Semantic Web technology.

We propose the following organization in the context model. First, there is a built-in most general class named *Thing* that is the class of all individuals and is a superclass of all OWL classes. The *Thing* superclass has two classes, namely *CompositeWebService* and *WebService*, which represent composite services and Web services, respectively. *Classes* are interpreted as sets of individuals. *Individuals* represent specific composite service and Web service operations. Individuals keep runtime information for service operations represented in the base composition model and in variant models to collect contextual information no matter what the current configuration of the service composition is. Each individual in the context model has a set of *datatype properties* (i.e., relations between instances of classes and RDF literals or XML schema datatypes). These datatype properties are used to keep track of context information.

Since dynamic adaptations are triggered when a particular SLA is at risk, datatype properties keep information about quality attributes.

In this work, the quality attributes to be monitored are based on the quality model for services in (Ameller and Franch, 2008). This quality model is based on the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 9126-1 standard. This standard is defined by means of general software characteristics, which are further refined into subcharacteristics. In turn, subcharacteristics can be decomposed into attributes. The values of measurable software attributes can be computed using some metric.

The context model observes the Availability and Time Behavior subcharacteristics at runtime.

Among the quality attributes in the ISO/IEC 9126-1 standard, the quality model for services in (Ameller and Franch, 2008) only chooses those attributes that can be measured using a monitoring technique, namely those related to the Availability, Time Behavior, and Accuracy subcharacteristics. In our approach, the context model observes the Availability and Time Behavior subcharacteristics at runtime (see Table 5.1). According to (Ameller and Franch, 2008), in order to monitor the accuracy of the service, it is necessary to know the concrete functionality of the service and have available concrete predefined tests to run on it. These tests are out of the scope of this work.

Technical Characteristics	Reliability	Efficiency
Technical Subcharacteristics	Availability	Time Behavior

Table 5.1: Technical characteristics and subcharacteristics that are managed by the context model.

In this work, we use the following datatype properties in the context model. Nevertheless, we believe our approach could be easily extended with other datatype properties to observe quality attributes such as execution cost and non-repudiation (Bandara et al., 2009):

- The `isAvailable` datatype property observes the availability subcharacteristic (this subcharacteristic is not decomposed into quality attributes). It indicates whether the service operation is currently available (it is a Boolean value).

- The `hasExecutionTime` datatype property observes the execution time quality attribute (response time plus execution time). It indicates the current execution time in milliseconds (ms) that a service operation takes to execute a job. Since the Time Behavior subcharacteristic is not a single measurable concept, therefore it is possible to define the Execution Time quality attribute to measure this subcharacteristic.

Example:

Figure 5.7 shows the ontology for our case study. Since our case study is concerned with the availability and the performance of the service composition, each individual of the *WebService* class has datatype properties to observe the availability and the execution time of service operations. The individuals of the *CompositeWebService* class have the same datatype properties of the *WebService* class' individuals. In this case, the value of the `isAvailable` datatype property is TRUE if all the compound service operations are available. Also, the value of the `hasExecutionTime` datatype property is the average of the execution times of the compound service operations.

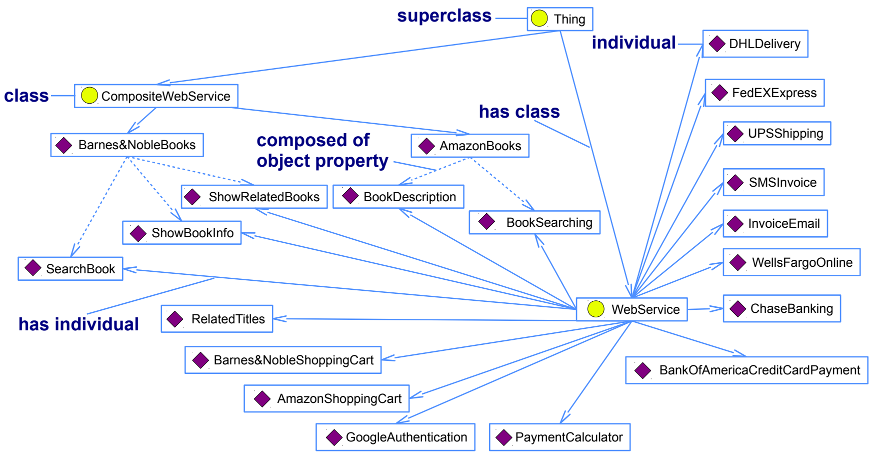


Figure 5.7: Context model for the case study.

5.2.5 *Define Context Conditions*

Adaptation policies are in charge of activating or deactivating features in the variability model at runtime to guide dynamic adjustments on the service composition. In order to define adaptation policies, first it is necessary to define the *context conditions* that may trigger adaptations.

A context condition works as an SLA.

Context conditions are extracted from the context model as Boolean expressions to solve the need for examining the compliance of certain situations in the context. A context condition works as an SLA. If a context condition is fulfilled (i.e., an SLA is violated), then an adaptation is triggered on the service composition to deal with the arising situation. Each context condition is represented as a RDF triple in the form of (subject, predicate, object). The *subject* (i.e., an ontology individual) denotes a resource (i.e., a Web service or a composite service operation), and the *predicate* expresses a relationship between the *subject* and the *object* (i.e., the value of a datatype property).

Composite context conditions increase the semantic power of context conditions.

In order to increase the semantic power of context conditions, they can be combined in *composite context conditions* (C_{comp}), which are defined as follows:

$$C_{comp} = \{(C_1 \vee C_2) \wedge (C_3 \vee C_4) \wedge \ldots \wedge C_n\}$$

Each C_n in the C_{comp} represents a specific context condition. AND and OR logical operators are used to connect each C_n.

Example:

Two context conditions in our case study are the following:

1. **`B&NUnavailable`** `= (Barnes&NobleBooks, isAvailable, false)`. It is triggered when the `Barnes & Noble Books` composite service is currently unavailable.

2. **`UPSHiExecTime`** `= (UPSShipping, executionTime, >1,500 ms)`. It is triggered when the current execution time of the `UPS Shipping` service operation is greater than 1.5 seconds.

5.2.6 *Define Resolutions*

In traditional SPLs, products are derived by selecting the features (reusable core assets or shared product family artifacts) that are part of a product and removing those that are not part of it. *Product derivation* is about the complete process of constructing a product from product family software assets (Deelstra et al., 2005).

In order to derive software products, variability models can describe commonalities (which are present in all the products), and variabilities (which are not). The resulting variability model configuration contains the set of features in the variability model that have been selected for a particular product. Once a product is derived, it retains the same configuration throughout its lifetime. In other words, although SPLE recognizes that variation points are bound at different stages of development, and possibly also at runtime, it typically binds variation points before delivery of the software (Hallsteinsen et al., 2008).

We argue that a service composition can activate or deactivate its own features at runtime by fulfilling certain context conditions. To this end, the static perspective of SPLE has to change to a dynamic one to allow service compositions to self-adapt at runtime. The solution to this matter is offered by DSPLE (Hallsteinsen et al., 2008). DSPLE goes a step further from SPLE with the investigation of development issues for reusable and dynamically reconfigurable core assets. DSPLs break down the complexity of managing dynamic reconfiguration points by modeling them explicitly in a product line approach as late variability (Van Gurp, 2000).

We propose to use the *resolution* concept to represent the set of changes in a feature model triggered by a context condition. Resolutions are the adaptation policies that express the transitions among different configurations of the service composition in terms of activation or deactivation of features. Resolutions materialize the vision of DSPLE by binding variation points with the activation of variant features at runtime.

Resolutions represent the set of changes in a feature model (i.e., they define the adaptation policies).

A resolution (R) can be expressed as a list of pairs (F, S) where each pair is made up of a feature (F) in a feature model (FM) and the state (S) of the feature. Each resolution is associated to a context condition (C). A feature's state is set to active or inactive:

$$R_C = \{(F, S) \mid F \epsilon [FM] \wedge S \epsilon \{Active, Inactive\}\}$$

In a different approach, adaptations are triggered to reach a configuration with the maximum overall utility (Esfahani et al., 2011). To this end, utility functions provide the objective function for self-optimization, mapping each possible state (e.g. the set of active service operations in a particular configuration) of an entity (e.g. a service composition) into a real scalar value (Tesauro and Kephart, 2004). For instance, the utility function of a service composition can be calculated with the quality attributes of each individual service operation in the service composition (i.e., the utility of combining service operations). The major problem with utility functions is that they can be extremely hard to define, as every aspect that influences the decision by the utility function must be quantified. On the other hand, our approach offers an easy, effective, and abstract way to define adaptation policies.

Examples:

The resolution for the `B&NUnavailable` context condition is as follows:

$R_{B\&NUnavailable}$ = *{(Barnes & Noble Books, Inactive), (Search Book, Inactive), (Show Book Info, Inactive), (Show Related Books, Inactive), (Amazon Books, Active), (Book Searching, Active), (Book Description, Active), (Related Titles, Active), (Barnes & Noble Shopping Cart, Inactive), (Amazon Shopping Cart, Active)}*.

Although the `Barnes & Noble Shopping Cart` may not be unavailable when `B&NUnavailable` occurs, $R_{B\&NUnavailable}$ deactivates this functionality and activates the `Amazon Shopping Cart` functionality in order to respect *requires* relationships in the variability model (see Figure 5.5).

An example of a resolution for a composite context condition is as follows:

$$R_{UPSHiExecTime \wedge DHLExecTimeLowerThanFedEX}$$

This resolution is triggered when both context conditions occur. In case the `UPSHiExecTime` context condition occurs, only one of two variant features can be activated in the `Shipment` variation point (`FedEX Express` or `DHL Delivery` in Figure 5.5). Therefore, the `DHLExecTimeLowerThanFedEX` context condition is used to activate the `DHL Delivery` functionality when it has lower execution time than the `FedEX Express` service operation.

5.2.7 *Generate the Adaptation Space*

The application of resolutions on the variability model can be used to generate the adaptation space.

The application of resolutions on the variability model can be used to generate the *adaptation space* of the service composition that contains all the possible variability model configurations and migration paths among configurations. In other words, the adaptation space shows the level of autonomic behavior that can be achieved by means of a variability model. The adaptation space can be abstracted as a highly-connected state machine where *states* are the possible variability model configurations and *transitions* the migration paths among configurations (Zhang and Cheng, 2006; Bencomo et al., 2008). Fully specifying this state machine lets the systems analyst do the following:

1. Simulate the service composition's dynamic variability before actually implementing the service composition.

2. Use the generated state machine at runtime to count on controlled adaptations from one configuration to another by means of transitions among configurations.

The adaptation space can be auto-generated with the CONFIGURATION GENERATOR, which is described in Chapter 9. To this end, the CONFIGURATION GENERATOR applies a set of resolutions on the variability model.

Example:

The right-hand side of Figure 5.8 shows the adaptation space that can be generated with a simple variability model with six features. The adaptation space contains twelve possible service composition configurations (CC_1 to CC_{12}). The four resolutions at the left express the transitions between different configurations in the adaptation space in a declarative manner (without the need for an exhaustive definition of each state transition). Resolutions are represented as arrows in the adaptation space. For example, R_{C1} results in thirteen transitions in the adaptation space.

5.2.8 *Link Features to Service Operations*

In our approach, the features in the variability model are dynamically activated or deactivated to reach a particular configuration of the service composition. Therefore, it is necessary to count on a connection between the elements in the variability model and low-level service operations. Since the composition model represents the operations in the service composition at any moment, the definition of a bridge between the elements in the variability model and the elements in the composition model could be used to support dynamic adaptations in the underlying service composition. Mapping features to elements in the composition model gives semantics to features about service operations and their organized connections.

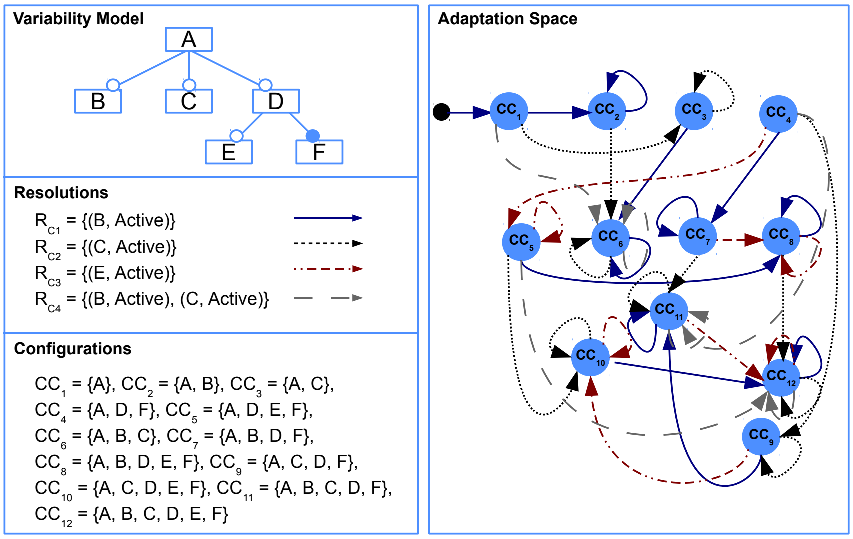

Variability Model
A
B
C
D
E
F
Resolutions
R_{C1} = {(B, Active)}
R_{C2} = {(C, Active)}
R_{C3} = {(E, Active)}
R_{C4} = {(B, Active), (C, Active)}
Configurations
CC_1 = {A}, CC_2 = {A, B}, CC_3 = {A, C},
CC_4 = {A, D, F}, CC_5 = {A, D, E, F},
CC_6 = {A, B, C}, CC_7 = {A, B, D, F},
CC_8 = {A, B, D, E, F}, CC_9 = {A, C, D, F},
CC_{10} = {A, C, D, E, F}, CC_{11} = {A, B, C, D, F},
CC_{12} = {A, B, C, D, E, F}
Adaptation Space
CC_1
CC_2
CC_3
CC_4
CC_5
CC_6
CC_7
CC_8
CC_9
CC_{10}
CC_{11}
CC_{12}

Figure 5.8: Adaptation space with a simple variability model and a set of resolutions.

The weaving model works as a bridge to link features to service operations.

In order to define this bridge, different techniques can be used, such as the template approach based on superimposed variants (Czarnecki and Antkiewicz, 2005). In this work, we propose the creation of a weaving model (Del Fabro et al., 2006) since it has good tool support. The weaving model can be considered as a *mapping model* (Czarnecki and Eisenecker, 2000) that defines the mapping relationships between a *problem space model* (i.e., the variability model) and a *solution space model* (i.e., the composition model).

Each link (or mapping) in the weaving model has the following endpoints: the first endpoint refers to features in the variability model; the second endpoint refers to variation points and commonalities in the base composition model, and variant models.

The relationship between a variant feature and a variant model is an implicit one-to-many relationship, i.e., a feature may map to a variant model, which encapsulates several elements that abstract service operations. Therefore, it is possible to say that features encapsulate variability in a coarse-grained way, i.e., one feature can encapsulate several service operations.

Example:

Figure 5.9 shows a fragment of the weaving model for the case study. It highlights the following mappings: 1) the mapping between the `Book Management` feature in the variability model and the `Book Management` variation point in the base composition model; 2) the mapping between the `Google Authentication` feature in the variability model and the `Google Authentication` commonality in the base composition model; and 3) the mapping between the `Barnes & Noble Books` feature in the variability model and the `Barnes & Noble Books variant model`, which is encapsulated in a BPMN subprocess with a set of BPMN activities (e.g. `Search Book` task), events, and sequence flows.

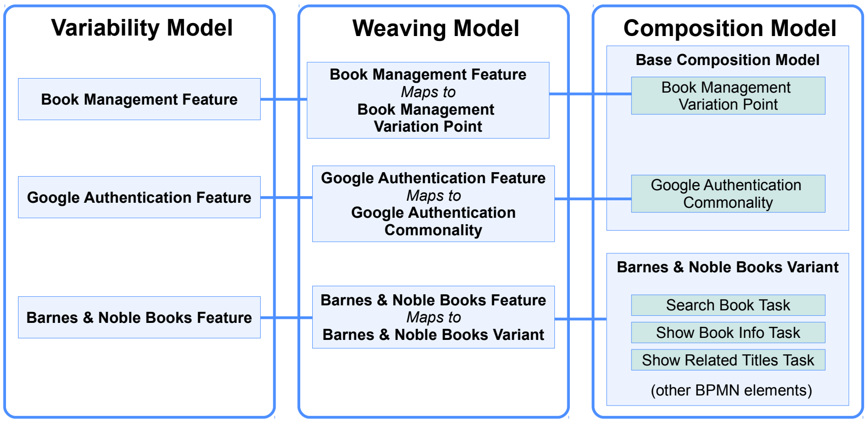

Variability Model
Book Management Feature
Google Authentication Feature
Barnes & Noble Books Feature
Weaving Model
Book Management Feature
Maps to
Book Management Variation Point
Google Authentication Feature
Maps to
Google Authentication Commonality
Barnes & Noble Books Feature
Maps to
Barnes & Noble Books Variant
Composition Model
Base Composition Model
Book Management Variation Point
Google Authentication Commonality
Barnes & Noble Books Variant
Search Book Task
Show Book Info Task
Show Related Titles Task
(other BPMN elements)

Figure 5.9: A fragment of the weaving model for our case study.

5.2.9 *Verify Reconfigurations*

In the previous sections, we have described that dynamic changes in the underlying service composition can be guided by activating or deactivating features in the variability model. Changes in the feature model are propagated to the composition model by means of a weaving model. The adapted composition model will eventually guide adaptations in the underlying service composition. Therefore, it is possible to state that if the variability model or any of its configurations has any inconsistency, then the result during execution could be wrong or even catastrophic (e.g. in critical scenarios, such as in transportation systems).

This section covers the verification at design time of the variability model and its possible configurations in order to avoid inconsistencies in the self-adjusting service composition at runtime. *Verification* can be defined as "the formal proof of program correctness" (IEEE, 1990). It answers the following question: *Are we building the product right?* Although it is not possible to guarantee the complete quality of variability models (Batory, 2005), neither to prove that a model is correct, the quality can be improved by means of a verification process (Mazo, 2011).

Since in this work variability models are implemented as feature models, we propose to use a VERIFIER to verify feature models and their related configurations. This VERIFIER is not attached to any specific tool. Therefore, we have chosen the Feature Model Analyzer Framework (FAMA-FW)[3] and GNU PROLOG[4] to implement the VERIFIER depending on the desired verification operations. Specifically, in order to verify the variability model, which is described as a feature model, and its possible configurations, the VERIFIER is composed of the following two complementary approaches:

The VERIFIER is composed of two complementary approaches.

1. **Use Generic Verification Operations:** According to (Benavides et al., 2010), the following set of generic operations can be used to verify feature models and their related configurations. The Modeling Software Kit for Software Product Lines (MOSKitt4SPL)[5] supports these operations by invoking the functionality provided by FAMA-FW.

   - **Void feature model:** This operation takes a feature model as input and returns a value saying whether such feature model is void or not. A feature model is void if it represents no configurations.

   - **Right configuration:** This operation takes a feature model and a set of features as input and returns a value that determines whether the configuration belongs to the set of configurations represented by the feature model or not.

   - **Anomalies detection:** The following operations take a feature model as input and return information about the anoma-

3 http://www.isa.us.es/fama
4 http://gprolog.univ-paris1.fr
5 http://www.pros.upv.es/m4spl

lies detected: 1) `dead features`: a feature is dead if it cannot appear in any configuration; 2) `conditionally dead features`: a feature is conditionally dead if it becomes dead under certain circumstances (e.g. when selecting another feature); 3) `false optional features`: a feature is false optional if it is included in all the configurations in the adaptation space despite not being modeled as mandatory; 4) `wrong cardinalities`: a group cardinality is wrong if it cannot be instantiated; and 5) `redundancies`: a feature model contains redundancies when some semantic information is modeled in multiple ways.

2. **`Use SAS-Oriented Verification Operations:`** The generic verification operations described above are well-established for verifying feature models and their configurations (Benavides et al., 2010). Nevertheless, we argue that this set of generic operations has to be extended in the case of SAS. Specifically, it is also necessary to verify the suitability of the feature model and its configurations in particular contextual situations.

   To this end, we propose the VERIFIER to use CP to implement SAS-oriented verification operations. CP has proven to be successful in many relevant application areas such as scheduling, planning, vehicle routing, and resource allocation (Rossi et al., 2006). Moreover, CP has been widely used to verify feature models since a feature model can be expressed as a Constraint Satisfaction Problem (CSP) (Benavides et al., 2010).

   CP is a declarative programming paradigm to solve CSPs. A CSP is defined by a set of problem variables (i.e., the unknowns), each associated with a domain of values, and a set of constraints. A *constraint* is a logical relation between several variables restricting the values these variables can simultaneously take. Solving a CSP consists in finding an assignment of variables satisfying all the constraints. A constraint program mainly states the constraints (incrementally) and asks the constraint solver to find a solution. The solver is then used as a "black-box" responsible for ensuring the consistency of the constraints.

   In order to carry out the verification, the feature model, the context model, and the constraints that relate them are transformed into a constraint program over finite domains. Details about the transformation rules and an algorithm that transforms feature models into constraint programs can be found in (Mazo et al., 2011). Details about the transformation of context models into a constraint program and their relationships with variability models are provided in (Sawyer et al., 2012). The composition model is not represented as a constraint program because its relationships with the variability model are implemented by means of the weaving model.

Once the variability model is transformed into a constraint program, the model and its configurations can be automatically verified with a solver (Salinesi et al., 2010). We take advantage of GNU Prolog to implement the verification criteria by means of algorithms that avoid expensive computations and reuse the precedent results to avoid wasting time in unnecessary operations (Salinesi and Mazo, 2012). GNU Prolog was chosen over FAMA-FW with Choco[6] to implement the verification criteria because experiments have demonstrated very extensive performance gains with GNU Prolog (Mazo, 2011).

The verification criteria in this section are the result of a joint effort with researchers at the CRI.

In the following subsections we present two criteria to verify, at design time, variability models intended to be used for dynamic adaptation of service compositions running on a particular context. The verification of the composition model and the mappings between the variability model and the composition model are out of the scope of this section (see (Gröner et al., 2011; Van der Aalst et al., 2010) for details). The verification criteria in this section are the result of a joint effort with researchers at the CRI, Panthéon Sorbonne University, France.

5.2.9.1 *Stability*

The stability operation verifies that for every combination of context variable values, there is at least one legal configuration.

The stability operation verifies that for every combination of context variable values that represent a context, there is at least one legal configuration that satisfies all the variability constraints. However, in complex systems there may be thousands of possible configurations (Khan et al., 2008), which can lead to configurations that are mutually inconsistent, contradictory, or simply unachievable. In our approach, features are represented as Boolean variables that can be satisfied or not by variability dependencies and context values. If no configuration exists that satisfies all the variability dependencies for a given context,

6 http://www.emn.fr/z-info/choco-solver

the systems analyst either needs to rethink the context model or the variability model.

Listing 5.1 shows the algorithm for this operation. This operation activates each possible combination of context variables (a context variable is activated when its value is greater than 0) and queries the solver for one solution that satisfies the extra constraints imposed by the activated context variables. No finding a solution for a particular combination of context variables means that the variability model cannot offer a valid configuration for the context at hand and a corresponding message is shown to the user.

Listing 5.1: Algorithm of the verification criterion for stability.

```
stability(VariabilityModel VM, ContextModel CM, Solver S) {
    S.load(VM);
    S.load(CM);
    C1...Cn is the list of variables in CM;
    Solution = S.getOneSolution(C1>0,...,Cn>0);
    If (Solution = FALSE) {
        Write (VM + " is NOT stable for all the possible values that can
             take the variables of "+ CM);
    } Else {
        Write (VM + " is stable for the combination of the context
             variable values of "+ CM);
    }
}
```

Example:

Let us suppose the following:

1. Current configuration: `E1 = [DHL Delivery = 1, UPS Shipping = 0, FedEX Express = 0...]`.

2. Context event: `DHLDeliveryHasExecTime = 25,000` (all the other context variables keep their current values).

According to this information, the following configuration constraint should be triggered: (DHLDeliveryHasExecTime $>$ 20,000 $\bigwedge$ UPSShippingHasExecTime $<$ FedEXExpressHasExecTime) $==>$ (DHL Delivery $=$ 0 $\bigwedge$ UPS Shipping $=$ 1). In this case, `UPS Shipping` is 1 in the variability model (if UPSShippingHasExecTime < FedEXExpressHasExecTime). In this example, the combination of context variables results in a legal configuration that satisfies the variability constraints.

5.2.9.2 *Semi-aliveness*

The semi-aliveness operation verifies whether or not the variability model is sensitive to changes in the context.

The semi-aliveness operation verifies whether or not the variability model is sensitive to changes in the context (i.e., it verifies that the variability model is able to respond to changes in the context). To this end, this operation verifies that there is a way to change from one configuration *C(E1)* to another *C(E2)*, such that *E1-E2 = [0, 0, 0, x, ... , 0]* (i.e., when one context variable has changed its value, the configuration should also change to adapt itself to the new context). *E* is a given context represented as a tuple of η context variables, where each variable takes a particular value of its domain. Thus, when a value of the tuple *E1-E2* is equal to 0, it means that the corresponding domain variables remain the same.

Listing 5.2 shows the algorithm for the semi-aliveness criterion. This operation is computationally complex because it combines the values of a collection of variables. In order to improve its performance, we divide the collection of context variables into partitions as small as possible. For instance, in the case where context variables are Boolean, our algorithm searches for $2^k * \eta$, instead of 2^η solutions, where k is the maximum number of variables into η partitions. Each partition corresponds to a collection of context variables, which do not depend from the rest of context variables.

Listing 5.2: Algorithm of the verification criterion for semi-aliveness.

```
semi-aliveness(VariabilityModel VM, ContextModel CM, Solver S) {
    S.load(VM);
    S.load(CM);
    Vector currentSolution = S.getOneSolution();
    For each Partition P in CM {
        For each combination C of context variables values into P {
            Solution newSol = S.getOneSolution (C);
            If (newSol != FALSE) {
                Vector delta = currentSolution - newSol;
                If(the sum of values of delta != 0){
                    Write ("The variability model is sensitive to the
                         Partition " + P);
                    currentSolution = newSol;
                    Exit from the current "For loop" and continue with
                         the next Partition;
                }
            } Else {
                Write("The variability model is insensitive to the
                     Partition " + P);
            }
        }
    }
}
```

Example:

Three partitions in our case study are as follows (each one corresponds to a collection of related functionalities):

- **`Credit Card Payment Partition:`** `Bank of America Credit Card Payment`, `Wells Fargo Online`, and `Chase Banking`.
- **`Invoice Delivery Partition:`** `Email Invoice` and `SMS Invoice`.
- **`Shipment Partition:`** `UPS Shipment`, `FedEX Express`, and `DHL Delivery`.

Let us suppose a current configuration when the `Barnes & Noble Books` feature is active (equals to 1) and the `Amazon Books` service is inactive (equals to 0). Thus, the current configuration is as follows: `E1 = [Barnes & Noble Books = 1, Barnes & Noble Shopping Cart = 1, Amazon Books = 0, Amazon Shopping Cart = 0, Related Titles = 0...]`.

Also, let us suppose that the system perceives that the `Barnes & Noble Books` service operation is unavailable. In this case, the `Amazon Books` feature is activated (equals to 1) and the `Barnes & Noble Books` feature is deactivated (equals to 0), as represented in the following configuration constraint: (AvailabilityBarnesAndNobleBooks = 0) ==> (Amazon Books = 1 $\bigwedge$ Barnes & Noble Books = 0).

Thus, the new configuration that is proposed by the solver is as follows: `E2 = [Barnes & Noble Books = 0, Barnes & Noble Shopping Cart = 0, Amazon Books = 1, Amazon Shopping Cart = 1, Related Titles = 1...]`. The `Amazon Shopping Cart` feature and the `Related Titles` feature are also activated thanks to the following implication instruction: `Amazon Books` ==> `Related Titles, Amazon Books ==> Amazon Shopping Cart`.

The difference between `E1` and `E2` can be calculated by taking the absolute value of each result ([1, 1, 0, 0, 0...] - [0, 0, 1, 1, 1...]). It produces the following list of results: [1, 1, 1, 1, 1...]. The sum of this list of values is different from zero. This means that our case study, in this scenario, is sensitive to the changes in the context. It is worth noting that when all possible changes in the context model are simulated, and the solution that is given by the solver is always the same, the variability model is insensitive to the context. The semi-aliveness operation also identifies this problem.

5.2.10 *Model-Driven Generation of WS-BPEL Code*

The models created at design time are intended to be used during execution to make self-adaptation decisions. To this end, these models need to be causally connected to the implementation code of the service composition. As a result, the self-adaptation of models at runtime will be reflected in the underlying code. Since this research is focused on autonomic adjustment of service compositions from the orchestration point of view, this section describes the creation of the WS-BPEL code that is going to be used during dynamic adjustments.

Among the models created at design time, the composition model is specially important when reflecting model decisions into WS-BPEL code because the composition model constantly represents the running service composition. In other words, it works as a "mirror" that reflects the currently orchestrated service operations and the workflow among them. If the composition model changes, then the running service composition changes and vice versa.

One possible solution to translate the changes at the composition model level into WS-BPEL code is to carry out model-to-text transformations at runtime from the composition model to WS-BPEL code. Although this transformation works fine at design time (Torres et al., 2012), it is unfeasible during execution because transformations can take a long time[7]. In this section, we propose another solution that can be used to reflect the changes in the composition model into WS-BPEL code at runtime. This idea was

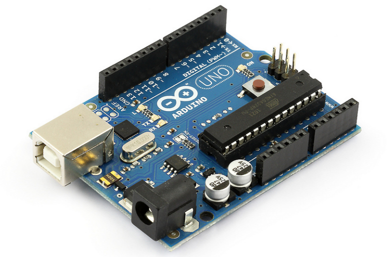


Figure 5.10: ARDUINO UNO. *Photo credit: Snootlab, Flickr, 2011.*

7 Transformations take around 95% of the total time required for the adaptation, from discovering a context condition that has been achieved to reconfiguration (Alférez and Pelechano, 2012b).

inspired by electronic boards (such as the ones created by ARDUINO[8] – see Figure 5.10). A board can be used for multiple projects by adding or removing elements from its surface. Therefore, the board is always the same, but the elements that can be connected are variable.

Our solution is composed of two elements that are created at design time:

1. A WS-BPEL *template* is a composition schema with variation points (it works as a board). The WS-BPEL template also indicates commonalities that do not vary at runtime and are common to all the versions of the service composition.

   The basic structure of the WS-BPEL template is as follows. The first part of the process always includes the partner links and variables that are used by all the commonalities and variants. Therefore, it is possible to reuse this information in every adaptation. Then, we propose to add variation points into the WS-BPEL `<sequence>` element.

   Since WS-BPEL lacks semantics for variability, variation points are stated with the WS-BPEL `<documentation>` element. At runtime, the line immediately after a documentation element is replaced by the variants' WS-BPEL code. This strategy was followed to keep the information about the variation points in every adaptation. The `<documentation>` element was chosen because its definition does not affect the sequence logic of the BP.

2. WS-BPEL *code fragments* represent variants that can be bound in the variation points of the WS-BPEL template at runtime. Each variant model maps to a WS-BPEL code fragment. WS-BPEL code fragments implement the code that invokes variant service operations (with the WS-BPEL `<invoke>` element). WS-BPEL code fragments are stored in a repository (i.e., a directory) to allow their retrieval at runtime.

In order to facilitate the creation of the WS-BPEL template, we propose a model-driven process (see Figure 5.11)[9]. In this process, we make use

8 http://www.arduino.cc

9 Although we propose a model-driven process to facilitate the creation of WS-BPEL code, it is not mandatory. Other tools can also be used to this end. For example, the ECLIPSE BPEL DESIGNER PROJECT (http://www.eclipse.org/bpel) supports the definition, authoring, editing, deploying, testing, and debugging of WS-BPEL 2.0 processes.

of the Babel project[10] (**Babel to Initial WS-BPEL** task) to translate the BPMN base composition model into WS-BPEL code.

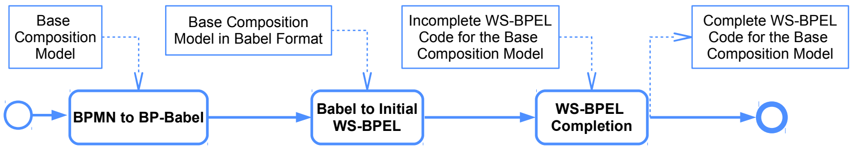

Base Composition Model
Base Composition Model in Babel Format
Incomplete WS-BPEL Code for the Base Composition Model
Complete WS-BPEL Code for the Base Composition Model
BPMN to BP-Babel
Babel to Initial WS-BPEL
WS-BPEL Completion

Figure 5.11: Process for generating the WS-BPEL template from the BPMN base composition model.

Babel is an open-source tool that is the result of integrating a set of techniques to translate models captured using a core subset of BPMN into WS-BPEL (Ouyang et al., 2009). The proposed techniques are capable of generating readable WS-BPEL code by discovering "patterns" in the BPMN models that can be mapped onto WS-BPEL block-structured constructs or acyclic graphs of control links. The latest version of Babel can be downloaded as an Eclipse plugin[11].

Before performing the **Babel to Initial WS-BPEL** task, the base composition model must be prepared according to the format accepted by the Babel tool. This is performed in the **BPMN to BP-Babel** task by means of a model-to-model transformation. This transformation is widely described in our previous work (a codirected master's thesis (De La Fuente, 2012)) and in a previous PhD thesis in our research group (Torres, 2008).

Although the Babel tool uses model-to-text transformations to generate a WS-BPEL document from a BPMN model, the generated WS-BPEL document is incomplete. For example, it lacks information about the *partner links* of services participating in the process and the *variables* used in the process. In order to solve this situation, we created a graphical tool based on Forms Modeling Framework (FMF)[12] that allows systems analysts input the missing information in the WS-BPEL document generated with Babel.

FMF is a framework whose main goal is to facilitate the development of form-based editors that manipulate models. FMF is an extension of

10 http://www.bpm.scitech.qut.edu.au/research/projects/oldprojects/babel/tools
11 http://code.google.com/p/bpmn2bpel
12 http://download.moskitt.org/moskitt/fmf/updates-1.3.8/

the Forms Editors Framework for Ecore Models (FEFEM)[13]. FMF is a part of the Modeling Software Kit (MOSKitt)[14] framework, in which the PROS has been constantly involved.

Thanks to our FMF-based configuration tool, the **`WS-BPEL Completion`** task is carried out as follows:

1. Our FMF-based configuration tool looks for patterns in the BABEL's generated WS-BPEL code in order to fill up an intermediate model that can be handled by FMF. The intermediate model conforms to the WS-BPEL metamodel, which is described in detail in (De La Fuente, 2012) and is available online[15]. Our tool uses the JDOM[16] libraries and XML Path Language (XPATH)[17] in order to do the following: 1) manipulate the WS-BPEL composition schema (a XML file); and 2) look for WS-BPEL elements such as invokes, imports, whiles, switches, variables, and namespaces to be completed.

2. The systems analyst inputs the missing information in our tool in order to complete the WS-BPEL composition schema. Specifically, the following information is introduced in FMF forms: 1) information about Web services, such as partner links, port types, input and output variables, and necessary WSDLs; 2) information about assign elements that manipulate data; 3) information to complete the behavior of whiles, switches, and timers.

   Figure 5.12 shows the graphical interface of this tool. The pattern that is used in most of the forms is the following. *Section A* shows a list with all the elements of a WS-BPEL element group (e.g. partner links). *Section B* shows a set of tabs to have access to the forms that allow to add information to elements in particular WS-BPEL element groups. *Section C* shows the attributes associated to a particular selected element in *Section A*. These attributes can be modified by the systems analyst. Since a WS-BPEL composition schema has only one `Process` element, its related form omits *Section C*.

3. Once the information has been completed in the intermediate model, our tool uses this information to automatically complete

13 http://www.moskitt.org/eng/fefem-creacion_de_formularios
14 http://www.moskitt.org/eng/moskitt
15 https://github.com/eclipse/bpel/blob/master/plugins/org.eclipse.bpel.model
16 http://www.jdom.org
17 http://www.w3.org/TR/xpath

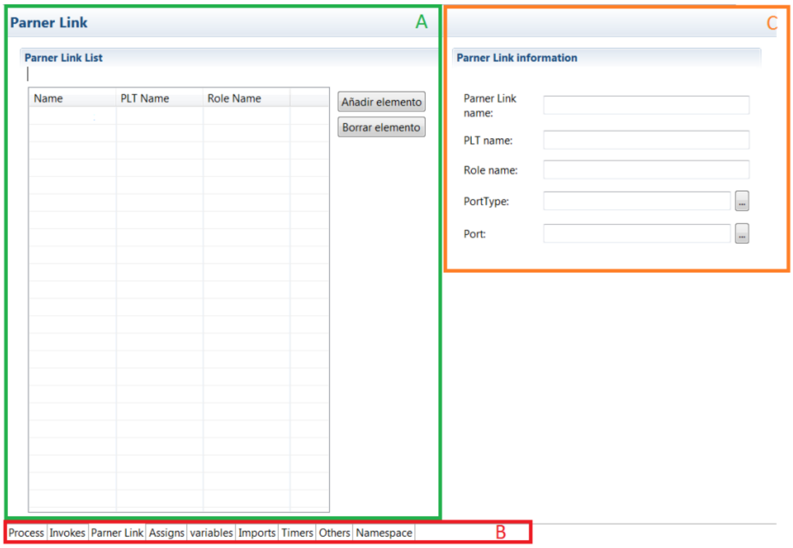


Figure 5.12: Graphical interface of our FMF-based tool (De La Fuente, 2012).

the WS-BPEL composition schema. The steps that are carried out in this operation are as follows: 1) the tool looks for each element that has been completed in the intermediate model; 2) the tool looks in the WS-BPEL file generated by BABEL for the elements that have been completed in the intermediate model; and 3) the tool completes the missing information in the WS-BPEL file generated by BABEL with the information in the intermediate model. Our tool can also use the completed intermediate model to automatically generate other artifacts to execute the process in the EXECUTION ENGINE, such as the WSDL file to access the service composition and the deployment descriptor (e.g. the deploy.xml file in APACHE ODE).

Additional details about our FMF-based configuration tool are available on (De La Fuente, 2012). Also, a video demonstration of this tool is available online[18].

Example:

The left-hand side in Figure 5.13 shows an abstract representation of the WS-BPEL template for our case study. Variation points are in blue (e.g. `Shopping Cart`) while commonalities are in blank (e.g. `Payment`

18 https://vimeo.com/58989214

Calculator). The abstract representation of the repository of WS-BPEL code fragments is at the right. Technical details about the WS-BPEL template and the WS-BPEL code fragments for our case study are described in Appendix B.

Technical details about the WS-BPEL template and the WS-BPEL code fragments for our case study are described in Appendix B.

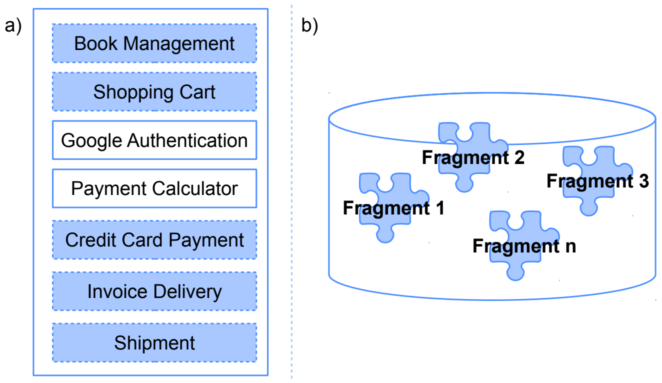


Figure 5.13: a) WS-BPEL template and b) WS-BPEL code fragments in a repository.

Listing 5.3 shows the basic structure of the WS-BPEL template for our case study. At runtime, code fragments can be bound into the variation points of the WS-BPEL template. For example, the Credit Card Payment variation point (namely VP_CreditCardPayment) can be implemented with variant WS-BPEL code fragments that invoke the Bank of America Credit Card Payment, the Wells Fargo Online, or the Chase Banking service operations.

Listing 5.3: Basic structure of the WS-BPEL template in our case study.

```
<process name="name" ... >

    <import>
        <!-- Import the client WSDLs -->
    </import>

    <partnerLinks>
        <!-- The declaration of all partner links (for all commonalities
             and variants) -->
    </partnerLinks>

    <variables>
```

```
12          <!-- The declaration of variables (for all commonalities and
                 variants) -->
13      </variables>
14
15      <sequence>
16          <!-- The definition of the WS-BPEL business process main body -->
17
18          <documentation>VP_BookManagement</documentation>
19
20          <documentation>VP_ShoppingCart</documentation>
21
22          <!-- GoogleAuthentication commonality implementation -->
23
24          <if name="isAuthenticated">
25              <condition>Authenticated user</condition>
26
27              <!-- PaymentCalculator commonality implementation -->
28
29              <documentation>VP_CreditCardPayment</documentation>
30
31              <if name="isValidCard">
32                  <condition>Valid card</condition>
33
34                  <flow>
35                      <documentation>VP_InvoiceDelivery</documentation>
36
37                      <documentation>VP_Shipment</documentation>
38                  </flow>
39
40                  <else>
41                      <reply name="isInvalidCard"... />
42                  </else>
43              </if>
44              <else>
45                  <reply name="unauthenticatedUser"... />
46              </else>
47          </if>
48      </sequence>
49  </process>
```

Listing 5.4 shows a simplified version of the WS-BPEL code fragment for the `Chase Banking` service operation. This functionality can be merged into the WS-BPEL at runtime by inserting its WS-BPEL code fragment in line 30 of Listing 5.3 (this is the line immediately after the specification of the `VP_CreditCardPayment` variation point).

Listing 5.4: WS-BPEL code fragment example.

```
<bpel:invoke name="ChaseBanking" partnerLink="ChaseBankingPL" operation="
    payment" portType="ns13:chasebanking" inputVariable="
    ChaseBankingRequest" outputVariable="ChaseBankingResponse">
</bpel:invoke>
```

5.3 CONCLUSIONS

In this chapter, we have introduced a set of models and other artifacts that are created in the `Design Phase` of our framework to support the dynamic adaptation of service compositions in the closed world. First, in Section 5.1 we introduced our approach to achieve autonomic service compositions by combining MDE and DSPLE. Then, in Section 5.2 we presented a process model to engineer the dynamic adaptation of autonomic service compositions at design time with our MDE-DSPLE approach.

In the introduced process model, we proposed to create the initial composition model first because it vividly abstracts the service operations and the workflows among them. In turn, the variability model offers a coarse-grained variability abstraction of the service composition. Afterwards, the information in the variability model is used to set variability at the composition model in terms of a base composition model and a set of variant models. Then, the context model is created in order to abstract the collected context information. Afterwards, context conditions are extracted from the context model as Boolean expressions. Subsequently, we proposed to create resolutions that represent the set of changes in a variability model triggered by context conditions. Then, the CONFIGURATION GENERATOR can be used to apply the resolutions on the variability model in order to generate the adaptation space. The adaptation space contains all the possible configurations of the variability model in terms of active/inactive features. Afterwards, we proposed to create a weaving model for mapping the features in the variability model on the elements in the composition model. Therefore, the weaving model can be used to reflect any activation/deactivation of features on the composition model at runtime. Then, the variability model and its set of configurations in the adaptation space can be verified with the VERIFIER in order to avoid inconsistencies during execution. Finally, we proposed an approach to create the WS-BPEL template and the set of WS-BPEL code fragments in order to reflect the changes in the composition model into WS-BPEL code at runtime.

The contributions presented in this chapter are based on scientific publications (Alférez and Pelechano, 2011a,b; De La Fuente, 2012; Ayora et al., 2011; Alférez et al., 2013; Pelechano et al., 2013).

6 MODELING TO FACE UNCERTAINTY IN THE OPEN WORLD

"We sail within a vast sphere, ever drifting in uncertainty, driven from end to end."

— Blaise Pascal

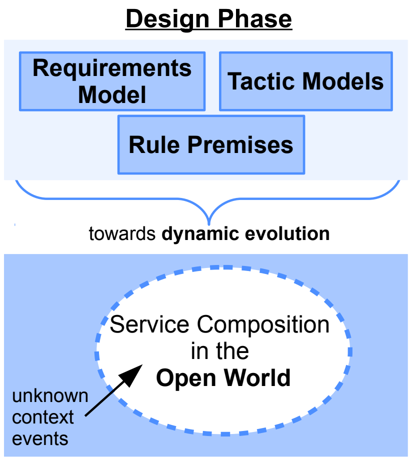


Figure 6.1: Scope of Chapter 6.

Dynamic adaptation of software behavior refers to the act of changing the behavior of some part of a software system as it executes, without stopping or restarting it (Keeney, 2004). This type of adaptation is specially important in critical systems that cannot be stopped to implement the adaptations.

Adaptations are carried out in response to changing conditions in the supporting computing infrastructure and in the surrounding physical environment. Therefore, a requirement for dynamic adaptation is context awareness. The information that is collected from the context is used as a basis for automating tasks such as installation, adaptation, or healing.

As evidenced in Chapter 3, the vast majority of current approaches for the dynamic adaptation of service compositions still tend to be based on the *closed-world* assumption. Under this assumption, a set of adaptation actions is predefined for fully foreseen context events. In the closed world, models at runtime can remain stable for a long time. However, in the unpredictable *open world*, software should react to continuous and unanticipated context events in complex and uncertain contexts (Baresi et al., 2006) (see Figure 6.1).

We argue that the following questions have to be answered for facing unknown context events in the open world. In general, we refer to *unknown context events* as those situations in the context that have not been foreseen at design time:

1. Which requirements can be affected by unknown context events?

2. Which corrective actions can trigger the dynamic evolution of the service composition to preserve the expected requirements when unknown problematic context events are faced?

These questions arise two main concerns. First, the preservation of expected requirements at runtime (despite unknown context events) implies the need to count on the representation of these requirements during execution. This fact also implies the need for a mechanism to realize which requirements can be affected by unknown context events. Second, implementing corrective actions through complex scripts can make it difficult to reason about the service composition as it grows. Therefore, the evolution actions to guide the service composition to a better configuration should be sufficiently expressive and easy-to-understand in order to facilitate the development of the logic behind autonomic management.

In order to handle the aforementioned concerns, in this chapter we propose the creation of models that can be leveraged at runtime to face problematic unknown context events in the open world. First, a *requirements model* describes the requirements that the service composition must preserve at runtime. Then, *tactic models* abstract tactics with last-resort surviving actions to preserve requirements when unknown context events are faced.

The remainder of this chapter is structured as follows. Section 6.1 explains the differences between dynamic adaptation and dynamic evolution of software. Section 6.2 summarizes the dimensions of dynamic adaptation and dynamic evolution of service compositions. These dimensions facilitate the comprehension of this chapter's focus. Section

6.3 presents an overview of our model-driven approach to prepare for uncertainty in the open world. Section 6.4 presents the process model to design dynamic evolutions. Section 6.5 describes the requirements model. Section 6.6 describes the tactic models. Section 6.7 describes rule premises that can be used at runtime to find out which requirements can be affected by unknown context events. Finally, Section 6.8 presents the conclusions.

6.1 DYNAMIC ADAPTATION AND DYNAMIC EVOLUTION OF SOFTWARE

In literature, it is common to find cases in which the terms "dynamic adaptation of software" and "dynamic evolution of software" (including other terms in between) are used interchangeably. Two examples are described as follows[1]:

> "Many commercial software systems that are deployed by large companies need to be constantly available. Halting the software system to make changes cannot be afforded. Therefore, techniques are needed to change the software while it keeps on running. This very challenging problem is known under a variety of terms, including **runtime evolution**, **runtime configuration**, **dynamic adaptation** and **dynamic upgrading**." (Mens and Demeyer, 2008).

> "**Dynamic evolution** considers the case where the changes are made or activated at run-time. Systems evolve dynamically, either by hot-swapping existing components or by integrating newly developed ones without the need for stopping the system. In most cases, the new or modified binary code has been evolved before run-time. Its integration into the running system is called runtime activation. In contrast, genuine **run-time evolution** directly modifies the executable image. This is typical in reflective software systems that can reason about, and modify, their own internal state and behaviour." (Maes, 1987).

This mixture of terms arises the following questions: Are these terms interchangeable? Or, Are there semantic differences between these terms that can affect the meaning of what really happens in the underlying software? In order to answer these questions, let us go back to the semantics of "*adaptation*" and "*evolution*". The Oxford dictionary[2] defines these two terms as follows:

1 Words in bold are ours.
2 http://oxforddictionaries.com

- **Adaptation:** The action or process of adapting or being adapted. *Adapt:* become adjusted to new conditions.
- **Evolution:** The gradual development of something.

These definitions clearly indicate that these terms are not interchangeable. In fact, they offer an important basis to specify the differences between adaptation and evolution of software.

Dynamic adaptation and dynamic evolution of software are not the same.

On one hand, *adaptation* is about adjusting to arising conditions (e.g. context events). Therefore, it is possible to say that dynamic adaptations of software are carried out to make punctual changes to face particular events (e.g. by activating or deactivating features of the system).

On the other hand, *evolution* is about gradual or continuous growth. Dynamic evolution does not imply just punctual adaptations to punctual events but a gradual structural or architectural growth into a better state. This idea goes in line with Lehman's eight laws of software evolution, which were first formulated in the early 1970s in Belady and Lehman's study on the evolution of OS/360 (Belady and Lehman, 1976). These laws essentially characterize the software evolution process as a self-stabilizing and self-regulating system, subject to continuing growth and change (Lehman, 1996; Lehman et al., 1997; Lehman and Ramil, 2001).

Figure 6.2 exemplifies a dynamic adaptation and a dynamic evolution of software. At the left, it presents the initial system configuration with active and inactive features. At the middle, it presents the adapted system configuration with features that have been activated and deactivated. At the right, it presents the evolved systems configuration with features that have been added to the system. In addition, this evolution required the adaptation (activation or deactivation) of some features.

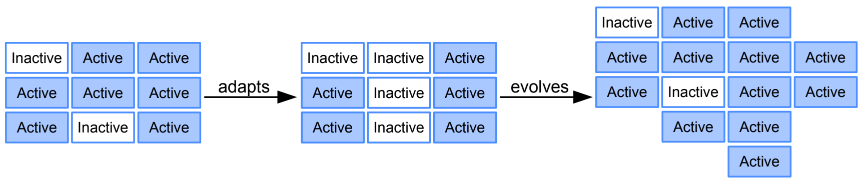


Figure 6.2: Dynamic adaptation vs. Dynamic evolution of software.

6.2 CONCEPTUAL FRAMEWORK FOR AUTONOMIC SERVICE COMPOSITIONS

In the case of autonomic service compositions, we argue that there is a need for a *conceptual framework*, which clearly describes the terms "*dynamic adaptation*" and "*dynamic evolution*" in the focal area of this thesis: service compositions. To this end, we propose to organize the conceptual framework in the following two layers (see Figure 6.3):

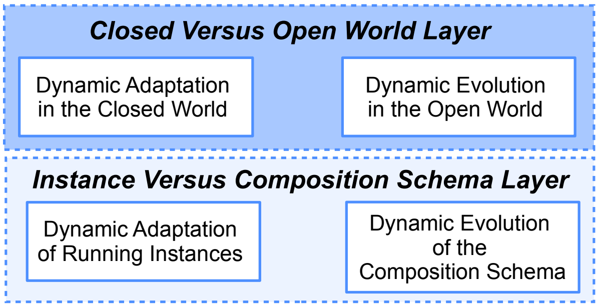

Closed Versus Open World Layer
Dynamic Adaptation in the Closed World
Dynamic Evolution in the Open World
Instance Versus Composition Schema Layer
Dynamic Adaptation of Running Instances
Dynamic Evolution of the Composition Schema

Figure 6.3: Conceptual framework for the dynamic adaptation and the dynamic evolution of service compositions.

1. **`Instance Versus Composition Schema Layer:`** This layer covers dynamic adjustments that are carried out in particular instances or in the structure of the composition schema. These two dimensions can be described as follows:

    a) **`Dynamic Adaptation of Running Instances:`** Dynamic adaptations are carried out on particular instances of the service composition that are involved in a peculiar situation (e.g. a problematic service operation). Since these situations do not affect all the running instances, then it is possible to trigger adaptations on specific instances. Nevertheless, instance adaptation is not an easy task since each instance may be running a different operation at the same time. For example, some instances are almost finishing their execution while others are just starting.

    b) **`Dynamic Evolution of the Composition Schema:`** There are situations in which the structure of the composition schema needs to be adjusted in order to reflect the applied changes into all the running instances. For example, a persistent problem in several instances indicates that a deeper solution at the structural level is required. In this dimension, all running instances have to be restarted (it is an expensive option) or migrated according to the new version of the composition schema. In any case, all future instances run according to the new version of the composition schema.

The dimensions in this layer have been widely described in literature (Weber et al., 2008; Marconi et al., 2009).

2. **Closed Versus Open World Layer:** This layer covers dynamic adjustments that are carried out on the service composition depending on whether it is deployed in the closed or open worlds. This layer contains two dimensions:
   a) **Dynamic Adaptation in the Closed World:** This dimension is focused on the closed-world assumption, in which all context events are foreseen at design time. Predefined actions guide adaptations in the service composition according to known context events (for example, through ECA rules that activate or deactivate service operations). There are several research works that focus on this dimension (as described in Chapter 3).
   b) **Dynamic Evolution in the Open World:** Predefined adaptation actions for known context events are not enough in the open world where several unknown context events can arise (e.g. sudden security attacks). In the open world, the initial service composition should be able to continuously grow or change to face these events. However, despite the recognized need for handling unexpected events in SAS (Cheng et al., 2009a; Calinescu et al., 2012), this dimension is still an open and challenging research topic. Therefore, *this chapter focuses on this dimension*.

Uncertainty is caused by how the service composition should deal with unknown context events.

In the open world, *uncertainty* is caused by how the service composition should deal with unknown context events, which have not been foreseen at design time. The relationship between known or foreseen context events (KE) at design time and unknown or unforeseen events (UE) at runtime can be formulated as follows: $KE \cap UE = \emptyset$.

In order to manage uncertainty in the open world, the supporting models should be able to evolve at runtime for a better functioning and system "survival". For example, suppose that a set of adaptation actions have been created for foreseen context events in our case study (see Section 4.1). In this case, if the `Barnes & Noble Books` composite service operation is unavailable, then other service operations can be invoked instead. Nevertheless, if there are not predefined adaptation actions for unknown context situations (e.g. any third-party Web service operation fails or performs below required SLAs), then no adaptation is carried out. As a result, the whole system quality may be harmed. We argue that despite unknown context events, the service composition has to keep offering expected requirements.

We define *dynamic evolution* of service compositions as follows:

> Dynamic evolution of service compositions is the process of moving the service composition to a new version, which cannot be supported by predefined dynamic adaptations, in order to manage unknown context events at runtime (Alférez and Pelechano, 2013b).

Definition of dynamic evolution of service compositions.

6.3 GETTING READY TO FACE UNCERTAINTY IN THE OPEN WORLD

Under the closed-world assumption, the set of possible context events, which will eventually arise during execution, is fully known at design time. Nevertheless, it is difficult to foresee all the possible contextual situations that may arise in uncertain and complex contexts. The following interesting quote states the following about this matter:

> "A DAS, especially one intertwined with physical elements, must increasingly reason about and cope with unpredictable events in its execution environment. Unfortunately, it is often infeasible for a human to exhaustively explore, anticipate, or resolve all possible system and environmental conditions that a DAS will encounter as it executes. While uncertainty can be difficult to define, its effects can hinder the adaptation capabilities of a DAS."[3] (Ramirez et al., 2012).

Therefore, there is a need for supporting the dynamic evolution of context-aware service compositions to deal with unexpected context events in the open world. Since our approach is based on models at runtime, if model adaptations are not enough to solve uncertainty, the

3 DAS stands for Dynamic Adaptive System (a.k.a SAS).

models have to evolve to preserve the expected requirements of the service composition.

In this section, we propose pieces of knowledge that can be used during execution to reach the dynamic evolution of service compositions. These pieces are defined as abstract models. Specifically, we propose to extend the initial set of models that are created when designing dynamic adaptations (described in Chapter 5) with two additional models (see Figure 6.4). First, a *requirements model* abstracts the expected requirements that the service composition must preserve at runtime. Second, the corrective actions for dealing with uncertainty are expressed in *tactic models*.

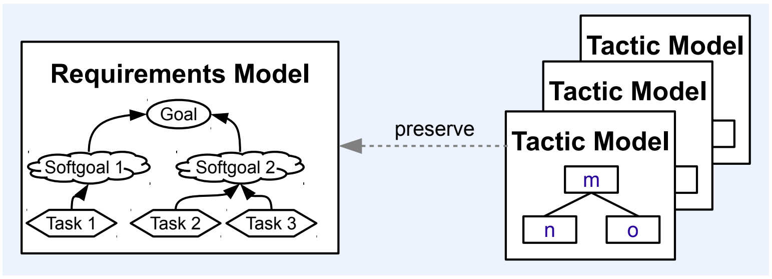


Figure 6.4: Models that are created in the `Design Phase` of our framework to prepare for dynamic evolution.

We define *tactics* follows:

Definition of tactics.

> Tactics are abstract last-resort surviving actions or strategies to preserve the requirements that can be negatively impacted by unknown context events (Alférez and Pelechano, 2012a). Therefore, tactics trigger the dynamic evolution of the service composition to preserve requirements at runtime.

In order to find the requirements that can be affected by unknown context events, we propose to evaluate these events against general *rule premises*. At runtime, these inference rules are used to extract more data from the available context data (an unknown event) until the goal of finding the affected requirements is reached. Therefore, we can say that these rules can be used to move from the *unknown* (i.e., unforeseen context events) to the *known* (i.e., requirements that can be affected by these events). We propose to create these rules at design time and to keep them in a knowledge base to be used during execution.

6.4 A PROCESS TO DESIGN DYNAMIC EVOLUTIONS

We propose to organize the models and other supporting artifacts introduced in this chapter in a process model in order to facilitate their implementation in industrial scenarios. Figure 6.5 shows the proposed process model to design the dynamic evolution of service compositions. This process model is composed of a sequence of activities to be followed. Chapter 9 describes the elements that support this process model (e.g. tools).

In the process model of Figure 6.5, the requirements model is created first to count on an abstraction that expresses the needs and constraints placed on a service composition. Then, tactic models are created in order to preserve the expected requirements during execution. Tactic models are expressed in two levels of abstraction. First, tactic models are expressed as feature models. These tactic models can be merged into the variability model (which is described as a feature model). The evolved variability model indicates the functionalities that have to be present in the service composition to preserve the expected requirements during execution. Second, tactic models are expressed as composition models. This level of abstraction is necessary in order to express the workflow among service operations in tactics. The creation and refinement of tactic models in these two levels of abstraction can be carried out in parallel.

The merging and activation of tactic models in the variability model trigger changes in the underlying composition model. Therefore, the next activity in Figure 6.5 is to create a feature model to link the elements in these two models. Afterwards, we propose to 1) implement the service operations (atomic and composite service operations) for tactics; and 2) the fragments of WS-BPEL code to invoke these service operations. Finally, we propose to create the rule premises to find the requirements that can be affected by unknown context events.

6.5 ABSTRACTING REQUIREMENTS IN THE REQUIREMENTS MODEL

Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world

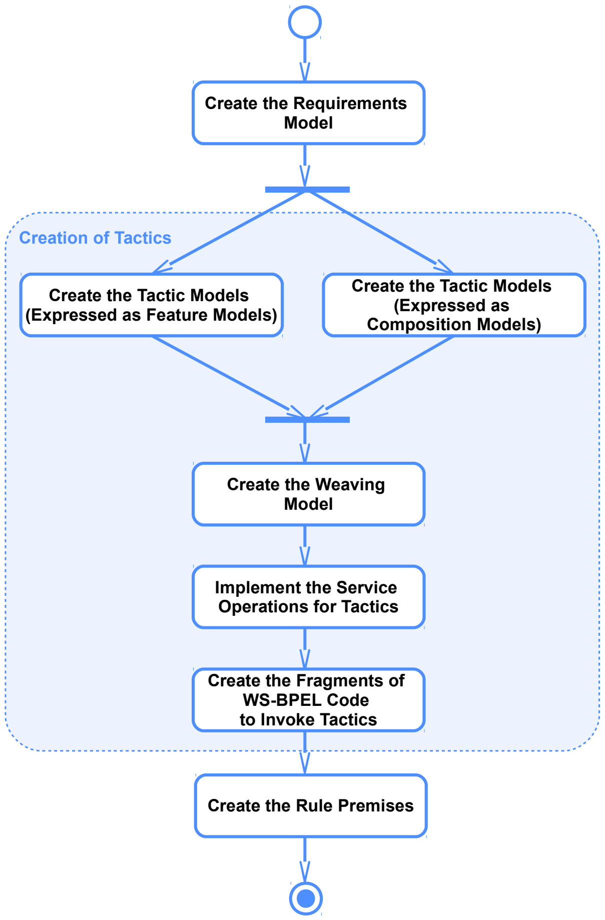

Create the Requirements Model
Creation of Tactics
Create the Tactic Models (Expressed as Feature Models)
Create the Tactic Models (Expressed as Composition Models)
Create the Weaving Model
Implement the Service Operations for Tactics
Create the Fragments of WS-BPEL Code to Invoke Tactics
Create the Rule Premises

Figure 6.5: A process model to design dynamic evolutions.

problem (Kotonya and Sommerville, 1998). The *requirements model* describes requirements in an abstract way. This model is leveraged at runtime to count on the representation of the requirements that the service composition must preserve at runtime despite arising unknown context events. There are several notations for requirements modeling, including UML Use Case diagrams and Goal Modeling (i*, the Goal-oriented Requirements Language (GRL), and the Knowledge Acquisition in Automated Specification of Software (KAOS) (Horkoff and Yu, 2011)).

In this thesis, we are particularly interested in keeping NFRs at runtime. Therefore, the GRL (Liu and Yu, 2004) has been used for requirements modeling because it is focused on NFRs (GRL is a variant of *i** (Yu, 2009)). This model has *softgoals* that in our case describe the NFRs to be kept by the service compositions in order to reach the top-level goal. This model also contains *tasks* that in our case specify particular surviving tactics to reach the softgoals. Since tasks represent core assets to keep the QoS of the system, they have a positive contribution to softgoals. Based on the temporal relationships of the tasks, they can be annotated as sequential, parallel or exclusive (Yu et al., 2008). There is a one-to-many relationship between a requirement (expressed as a softgoal in our case) and tactics that can preserve it. A large set of available generic tactics can be found in related work (Bass et al., 2012).

Conflicts may exist between tactics and softgoals. In order to solve conflicts, we define *claims* or *beliefs*. In our case, a claim or belief is a design assumption or condition about the problem that systems analysts hold to be true. Contributions of tactics on claims follow an ordinal scale, ranging from complete denial (--) to complete satisfaction (++). These contributions express the level of softgoals satisfaction when applying tactics. If a claim states that a tactic operationalization has a negative impact on a softgoal, then another tactic can be tried.

Example:

Figure 6.6 depicts a fragment of the goal model, which implements the requirements model, for our case study (see Section 4.1). The main goal of the elements in this model is to keep alive the service composition. In order to reach this goal, three softgoals are defined: `High Security`, `High Performance`, and `24/7 Availability` softgoals. Tasks specify the surviving tactics to reach the softgoals, e.g. the `Deception` task[4]. Also, this model indicates that the `Active Redundancy` and the `Shadow Operation` tactics can run in parallel during execution (i.e., they can be chosen together).

At runtime, if an unknown context event affects the `High Security` softgoal, the `Deception` tactic is chosen first because it has a more positive contribution on this softgoal. If this tactic does not solve the problem, then the `Temporal Separation` tactic can be triggered.

In a more complex scenario, if an unknown context event affects the `High Security` and the `High Performance` softgoals at the same time, `C1` is checked to decide the best tactic to choose from. `C1` is checked

4 *Deception* (or *honeypot*) ensures survivability by inducing enemy behaviors that may be exploited.

because it contributes to these two softgoals (see the two contribution links that go out from this claim to the two involved softgoals). In this case, the `Temporal Separation` and the `Introduce Concurrency` tactics are chosen together because they have the most positive contributions on these softgoals. It is interesting to notice that `C1` tries to avoid the use of the `Deception` tactic when an unknown context event affects the `High Security` and the `High Performance` softgoals at the same time. In this case, the `Deception` tactic may have a negative impact on the `High Performance` softgoal because more computational resources are required. This example also demonstrates that claims try to benefit all the affected softgoals as much as possible: the two selected tactics positively contribute to both the `High Security` and the `High Performance` softgoals.

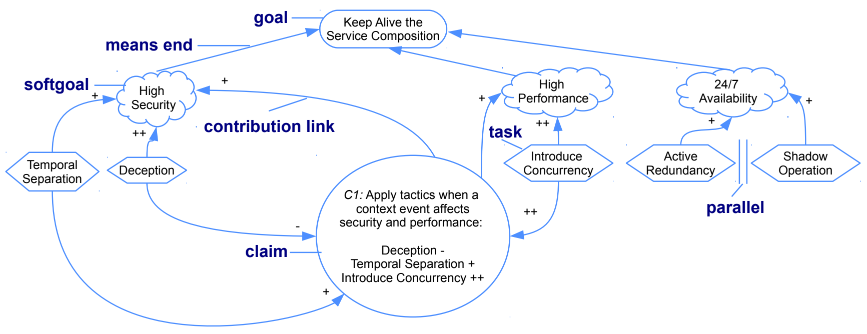


Figure 6.6: Fragment of the requirements model for the case study.

6.6 PRESERVING THE REQUIREMENTS WITH TACTIC MODELS

Tactics are the main components to face uncertainty in the open world.

Highly-abstract tactics are the main components to face uncertainty in service compositions that run in the open world. Specifically, *tactic models* express the tactical functionality to be triggered on the service composition to preserve requirements.

The use of tactics is common in sports, war, or even in daily matters to accomplish an end. Two examples of tactics are as follows:

1. A Master of Business Administration (MBA) student has an important goal: to finish his studies as soon as possible to find a job. However, his father had an unexpected accident that required an

expensive surgery. Therefore, the student had to use his savings for school to help with his father's surgery expenses. Now, he has to choose between two tactics to reach his goal. In a course, he learned about the principles for creating a company. Hence, the first tactic is to start up a company while he continues his MBA. The advantage of this tactic is that he can start a flourishing company before graduation. The second tactic is to stop studying to save money while working as a waiter until he has enough money to continue studying. However, this tactic can slow down the process to reach his goal. His best option is the first tactic, which he chooses to implement. In this example, there are different key elements: a *goal* to be reached, an arising *unknown or unforeseen situation* that can affect the goal, and a set of *tactics* that can be used to reach the goal.

2. The most important goal during a battle is to win. However, unknown or unforeseen events, such as surprise assaults, may arise. These events may negatively affect the expected goal. Therefore, it is necessary to choose among a set of tactics to reach the goal (e.g. to escape vs. to do a frontal attack). It is important to notice that in the battlefield, tactics are known beforehand, but soldiers do not know to which specific arising unknown context events they will be applied (see Figure 6.7).

Figure 6.7: The battlefield is full of arising unknown context events. *Photo credit: UK Ministry of Defense, Flickr, 2013.*

In the case of service compositions, we propose that one requirement can be preserved by many tactics. Tactic models are expressed as broadly as possible in order to be reused in different projects.

Tactics, compensation mechanisms, and exception handling.

There is a main difference between tactics and compensations mechanisms. On one hand, compensation mechanisms try to reverse actions performed in a transaction when failures are faced. On the other hand, tactics try to preserve expected requirements that may be affected by arising problematic unknown context events. Also, tactics are not the same as traditional exception handling. On one hand, exception handling is provided by specialized programming language constructs or hardware mechanisms; it is focused on implementation. On the other hand, tactics are expressed in a higher abstraction level to reason about their impact on the service composition, both at design time and at runtime.

Our approach manages known unknowns.

During dynamic evolutions, requirements (described in the requirements model) and tactics (described in tactic models) need to be known beforehand. Otherwise, it will be impossible to face uncertainty in a controlled way. However, they are not attached to any context event or specific reconfiguration actions (as dynamic adaptations do (Alférez and Pelechano, 2011a)). Therefore, our approach manages *known unknowns*: tactics are known beforehand, but we do not know to which specific arising unknown context events they will be applied. The idea of "known unknowns" has been described by different authors and practitioners in several disciplines (Modica and Rustichini, 1994; Raymond B., 1984; Epstein, 1984; Maluf et al., 2005). For instance, Donald Rumsfeld, briefly described this concept while serving as United States Secretary of Defense:

> "There are *known knowns*; there are things we know that we know. There are *known unknowns*; that is to say, there are things that we now know we don't know. But there are also *unknown unknowns* – there are things we do not know we don't know."[5]

Tactic models can be expressed as feature models.

Since tactics inject new variant functionalities into the service composition to preserve requirements, tactic models can be expressed as feature models that can be merged into the variability model at runtime. In this way, the evolved variability model includes the tactical functionality. The only merging prerequisite is that the variability model and the tactics models conform to the same metamodel; in this case,

5 http://www.defense.gov/transcripts/transcript.aspx?transcriptid=2636

the feature model metamodel. Each tactic model can have *restrictions* to indicate whether or not the tactic can run in parallel with other current active system functionalities. If not, it is necessary to activate or deactivate current functionalities to apply the tactic.

During execution, all the features in merged tactic models are activated to indicate that all the means for the preservation of requirements are at hand. Nevertheless, features in the merged tactic model lack information about the workflow that service operations have to follow. Therefore, we propose to count also on tactic models that reflect these workflows. To these end, for each tactic model expressed as a feature model, there is a tactic model expressed as a composition model. A tactic's composition model can abstract 1) just one atomic Web service operation or 2) a set of service operations in a composite service. Since tactics' composition models are used at runtime to evolve the composition model, both of them have to conform to the same metamodel.

For each tactic model expressed as a feature model, there is a tactic model expressed as a composition model.

At runtime, the evolved variability model has to be synchronized with the evolved composition model to accurately reflect the current situation of the underlying service composition. To this end, we propose the creation of a weaving model to reflect the activation of features in tactic models, which are expressed as feature models, on composition models, which abstract the workflow among service operations in tactics. Each link (or mapping) in the weaving model has the following endpoints: the first endpoint refers to features in all the tactic models, which are expressed as feature models; the second endpoint refers to elements in all the tactic models, which are expressed as composition models.

In the case of composite services that implement tactics, it is possible to generate the WS-BPEL code from a BPMN composition model with our model-driven approach described in Section 5.2.10. Each tactic model (expressed as a composition model) maps to a specific WS-BPEL code fragment, which invokes the tactical functionality. Tactical-related WS-BPEL code fragments are stored in the repository of WS-BPEL code fragments.

Each tactic model (expressed as a composition model) maps to a specific WS-BPEL code fragment.

Example:

Figure 6.8 shows the deception tactic model (expressed as a feature model and a composition model) and a fragment of the weaving model that links their elements. The root feature in the deception tactic, which is expressed as a feature model, indicates the tactic's functionality, and leaf features express coarse-grained functionalities to fulfill the tactic. A restriction states that this tactic can run in parallel with the current active features.

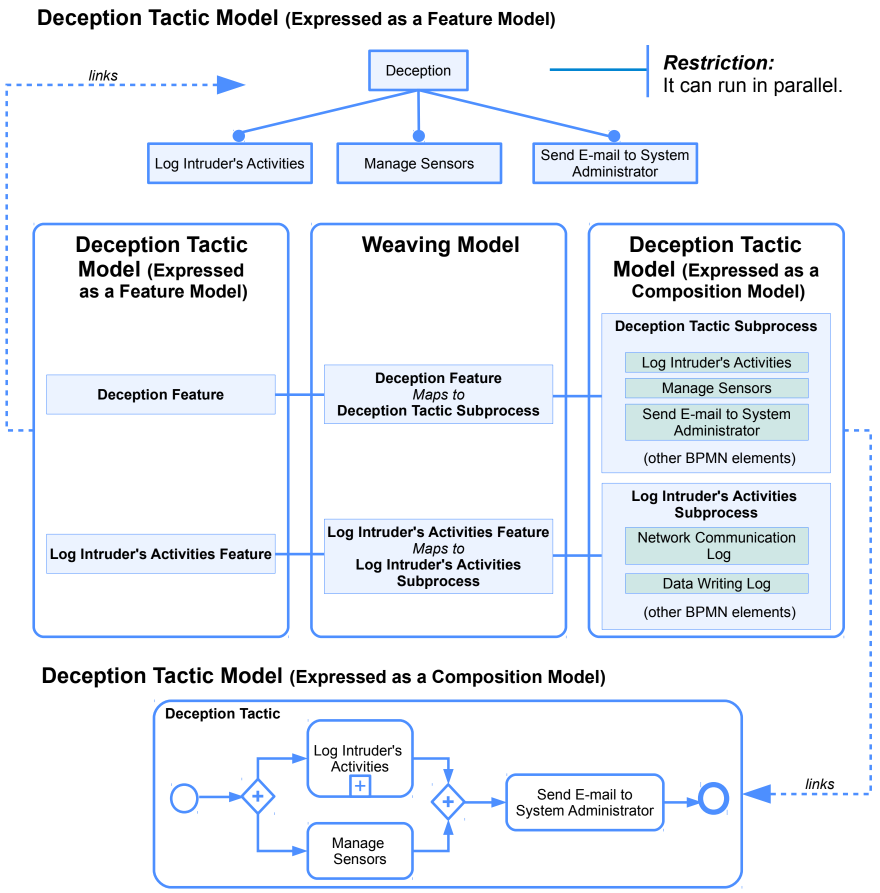

Deception Tactic Model (Expressed as a Feature Model)
links
Deception
Restriction:
It can run in parallel.
Log Intruder's Activities
Manage Sensors
Send E-mail to System Administrator
Deception Tactic Model (Expressed as a Feature Model)
Deception Feature
Log Intruder's Activities Feature
Weaving Model
Deception Feature Maps to Deception Tactic Subprocess
Log Intruder's Activities Feature Maps to Log Intruder's Activities Subprocess
Deception Tactic Model (Expressed as a Composition Model)
Deception Tactic Subprocess
Log Intruder's Activities
Manage Sensors
Send E-mail to System Administrator
(other BPMN elements)
Log Intruder's Activities Subprocess
Network Communication Log
Data Writing Log
(other BPMN elements)
Deception Tactic Model (Expressed as a Composition Model)
Deception Tactic
Log Intruder's Activities
Manage Sensors
Send E-mail to System Administrator
links

Figure 6.8: The deception tactic models expressed as a feature model and a composition model, and a fragment of the weaving model.

The `Deception` feature maps to a BPMN subprocess that abstracts the workflow of the `Deception` tactic. This subprocess includes a set of activities that abstract service operations (e.g. `Manage Sensors`) and other workflow-related elements, such as BPMN events and sequence flows among them.

There are not limits for the length or depth of model elements used in tactic models. For example, the `Log Intruder’s Activities` subprocess contains the BPMN activities that describe the invocation of service operations for writing in the network and data logs. The weaving model contains a mapping between the `Log Intruder’s Activities` feature and this subprocess.

In Appendix B, Listing B.20 shows the tactic model for deception expressed as a feature model in XML Metadata Interchange (XMI) format. In turn, Listing B.23 shows the tactic model for deception expressed as a composition model in XMI format. Also, Listing B.24 shows the WS-BPEL code fragment that invokes the deception tactic.

The deception tactic model (expressed as a composition model) maps to a WS-BPEL code fragment with an `<invoke name="deceptionTactic" ...>` instruction. This instruction invokes the service operation implementing this tactic. At runtime, this invocation instruction can be put into the WS-BPEL composition schema in order to evolve the service composition’s orchestration.

Listing 6.1 shows the key WS-BPEL code that implements the invocation of the composite service that implements the deception tactic. The `Deception Tactic` service invokes the `Log Intruders Activities`, the `Manage Sensors`, and the `Send Email to System Administrator` service operations.

Listing 6.1: WS-BPEL code that invokes the deception tactic.

```
<bpel:invoke name="DeceptionTactic" partnerLink="DeceptionTacticPL"
    operation="deceit" portType="ns17:deceptiontactic" inputVariable="
    DeceptionTacticRequest" outputVariable="DeceptionTacticResponse">
</bpel:invoke>
```

6.7 REASONING ABOUT UNKNOWN CONTEXT EVENTS WITH RULE PREMISES

In order to find the requirements that can be affected by unknown context events, our approach evaluates arising context *facts* (i.e., context events) against general *rule premises* in a knowledge base. The knowledge base is implemented as a rule file, which includes Resource Description Framework Schema (RDFS) rules.

For simple service compositions, rules can be obtained from human experts by: 1) collecting empirical data from the current service composition; 2) analyzing collected data to discover the symptoms of problematic situations; and 3) defining general situations in the context that can affect requirements. In complex service compositions, these steps can be extended with methods for generating rules from data (e.g. with heuristics or neural networks). The methods to create these rules are out of the scope of this thesis.

Examples:

It is possible to infer that a service operation can affect the `High Security` requirement when the execution time of a service operation is higher than a particular threshold. For example, Listing 6.2 shows that an observed execution time higher than 5,000 ms can be caused by a security attack (such as a Denial of Service (DoS) attack). In this case, this observed event affects the `High Security` requirement. In order to avoid false positives with an inference over just one observed execution time, which may not attempt against a particular requirement but may be caused by a passing event (such as a particular short delay in the network), a sample of observations can be used to make a decision. For instance, if the same service operation fulfills the `affectedHighSecurityRequirement` rule in a set of η observations, we can be more confident to infer that this service operation is under attack.

Listing 6.2: Rules to find out if a service operation affects the High Security requirement.

```
@prefix j.0: http://my.ontology#
[underAttack: (?s rdf:type j.0:WebService)
    (?s j.0:executionTime ?c)
    greaterThan(?c,5000)
    ->
    (?s rdf:type j.0:UnderAttack)
]

[affectedHighSecurityRequirement: (?s rdf:type j.0:UnderAttack)
    ->
    (?s rdf:type j.0:AffectedHighSecurityRequirement)
]
```

In case of having more context sensors, it is possible to infer that a service operation may attempt against the `High Performance` requirement in a T1 network when its execution time is extremely high, the latency is higher than five ms, and the bandwidth is lower than 1.544 Megabits Per Second (Mbps) (see Listing 6.3).

Listing 6.3: Rule premise to infer if a service operation is under attack with extra context sensors.

```
@prefix j.0: http://my.ontology#
[lowPerformance: (?s rdf:type j.0:WebService)
        (?s j.0:hasExecutionTime ?c) greaterThan (?c,30000)
        (?s j.0:hasLatency ?c) greaterThan (?c,5)
        (?s j.0:hasBandwidth ?c) lessThan (?c,1.544) ->
        (?s rdf:type j.0:LowPerformance)
]

[affectedHighPerformanceRequirement: (?s rdf:type j.0:LowPerformance)
    ->
    (?s rdf:type j.0:AffectedHighPerformanceRequirement)
]
```

6.8 CONCLUSIONS

In this chapter, we have presented a solution to design dynamic evolutions of service compositions. First, we formulated the differences between dynamic adaptation and dynamic evolution of software in Section 6.1. This general view helped us to describe the dimensions of dynamic adaptation and dynamic evolution of service compositions in Section 6.2. This knowledge is fundamental in order to contextualize our approach. Then, we introduced our model-driven approach to face uncertainty at the open world in Section 6.3. The process model to design dynamic evolutions was described in Section 6.4. The requirements model, which abstracts the expected requirements, was described in Section 6.5. The tactic models, which abstract surviving actions to preserve requirements at runtime, were described in Section 6.6. The rule premises to find out the requirements that can be affected by unknown context events were described in Section 6.7.

In brief, our approach to manage uncertainty in the open world can be summarized in the following premises:

Our approach to manage uncertainty in the open world can be summarized in three premises.

1. In the complex open world, it is possible to reduce uncertainty, not to mitigate it completely.

2. Uncertainty in the open world has to be managed in an organized way, otherwise the system can fall into chaos.

3. A set of "knowns" can be used to manage "unknown" context events in the open world.

According to the first premise, our solution tries to reduce the impact of uncertainty in the open world. To this end, we use abstract tactics that preserve expected requirements at runtime when facing unknown context events.

According to the second premise, we believe it is always necessary to count on a seed of knowledge to face uncertainty (e.g. inference rules or tactics in our case). Otherwise, the service composition will not be able to manage evolutions in an organized way; i.e., it could self-adjust in η possible configurations that may not be relevant for the underlying BP. Regarding this, Cheng et al. say: "a system cannot start out as a transport robot and self-adapt into a robot chef" (Cheng et al., 2009a).

According to the third premise, the aforementioned seed of knowledge should be known a priori. Nevertheless, it can be applied to unknown context events at runtime. The key concept is to count on knowledge that is generic enough to try to cover a large amount of unknown context events at runtime.

The contributions presented in this chapter are based on scientific publications (Alférez and Pelechano, 2012a, 2013b).

Part III

DYNAMIC ADJUSTMENT OF SERVICE COMPOSITIONS

The *pupillary light reflex* is an autonomic adaptation mechanism.

Photo credit: Icefreez, Flickr, 2009.

The *pupillary light reflex* is an adaptation mechanism, which automatically adjusts the amount of light that reaches the retina. Greater intensity light causes the pupil to become smaller, whereas lower intensity light causes the pupil to become larger (Purves et al., 2008).
As organisms and organs count on autonomic adaptation mechanisms, we argue that service compositions require a supporting computing infrastructure to face context events during execution. In this part, we propose this infrastructure to detect changes in the context and enable the *dynamic adaptation* and the *dynamic evolution* of the service composition by leveraging the models that are created at design time.
Part III is structured as follows. Chapter 7 presents the computing infrastructure to achieve dynamic adaptation of service compositions through models at runtime. Chapter 8 presents our approach to face unknown context events in the open world with the dynamic evolution of service compositions.

7 ACHIEVING DYNAMIC ADAPTATION THROUGH MODELS AT RUNTIME

"Research is formalized curiosity.
It is poking and prying with a purpose."

— Zora Neale Hurston

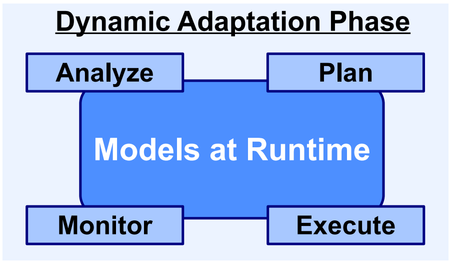


Figure 7.1: Scope of Chapter 7.

An ad-hoc approach to build self-adaptive service compositions is to use existing variability mechanisms (e.g. if-statements or method dispatch) directly in the architecture. However, the arising complexity of dynamic adaptable software can limit the number of dynamic reconfiguration points to a few well-defined ones due to the lack of appropriate approaches (Dinkelaker et al., 2010). Therefore, in the `Dynamic Adaptation Phase` of our framework, we reuse the knowledge captured in models, to guide the dynamic adaptation of service compositions. In response to changes in the context, the system itself can query these models to determine the necessary modifications in the service composition.

This chapter presents our approach to achieve dynamic adaptation of service compositions through models at runtime (see Figure 7.2). This infrastructure is based on the components of the MAPE-K loop

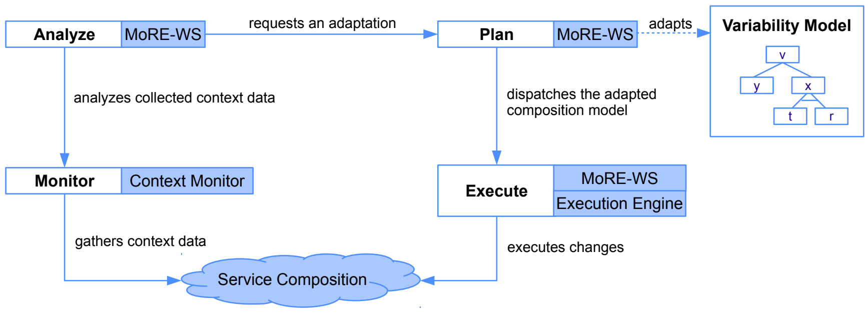


Figure 7.2: Computing infrastructure to deal with the dynamic adaptation of service compositions.

(IBM, 2006), namely `Monitor`, `Analyze`, `Plan`, `Execute`, and `Knowledge`. In our case, the knowledge that is managed in this loop is based on models at runtime.

First, sensors collect information from the context of the service composition. In the `Monitor` component, this information is processed by the Context Monitor. In the `Analyze` component, our Model-based Reconfiguration Engine for Web Services (MoRE-WS) tool, which implements the Model-Based Reconfigurator (as introduced in Chapter 4), updates the context model with the observed context information. Then, MoRE-WS decides if any context condition has been accomplished. If any context condition has been accomplished, then an adaptation is requested. In the `Plan` component, MoRE-WS executes adaptation policies that activate or deactivate features in the variability model. When features are activated or deactivated at runtime in the variability model due to changes in the context, a DSPL architecture supports the dynamic service recomposition. The adapted configuration of the variability model is used to automatically create a *reconfiguration plan* with the adaptation actions to be carried out on the composition model. The adapted composition model is dispatched to the `Execute` component. In this component, MoRE-WS provides the mechanisms to map the elements in the adapted composition model to WS-BPEL code. The generated WS-BPEL code and other required artifacts are deployed at runtime in the Execution Engine.

The remainder of this chapter is structured as follows in the context of MAPE-K components. Section 7.1 describes how the Model-based Reconfiguration Engine (MoRE), a previous model-oriented reconfiguration engine, was extended in order to create MoRE-WS. Section 7.2 describes how the context is monitored. Section 7.3 describes the analysis of the collected context information in order to realize if any adaptation is required. Section 7.4 describes how adaptations are planned. Section 7.5 describes the execution of adaptations on the running service composition. Finally, Section 7.6 presents the conclusions.

7.1 MORE-WS: AN EXTENSION OF MORE FOR SERVICE COMPOSITIONS

In our approach, features in a variability model are activated or deactivated at runtime in order to self-adapt the service composition from one configuration to another. MoRE-WS is the key element to translate context changes into changes in the variability model. Then, MoRE-WS translates these changes into reconfiguration actions that modify the WS-BPEL composition schema accordingly.

MoRE-WS is an extension of MoRE, a previous work in our research group (Cetina et al., 2009; Cetina, 2010). MoRE translates context changes into changes in the activation and deactivation of features in a feature model at runtime. It has been successfully applied to the smart-home domain, demonstrating its feasibility for self-healing and self-configuring the software that manages smart homes.

MoRE makes use of the Open Services Gateway Initiative (OSGi) framework (Marples and Kriens, 2001) for implementing the reconfiguration actions. To this end, MoRE components and the software components of the smart home are built upon OSGi bundles. According to OSGi terminology, system components are known as *bundles*. Bundles can register services within the framework service registry. Then, other bundles can discover registered services and use their functionality. OSGi was particularly attractive to self-adapt smart homes because it offers a dynamic model in which bundles can be installed, started, stopped, updated, and uninstalled without bringing down the whole system.

The reconfiguration actions of MoRE are classified into the following categories (Cetina et al., 2009; Cetina, 2010):

1. **Component actions:** According to (Kramer and Magee, 1990), a component must transit from an active (operational) state to a quiescent (idle) state to perform system adaptation. MoRE implements this pattern by means of the OSGi capabilities to install, start, restart, and uninstall components without restarting the entire system. Components that are irrelevant for the current configuration are in a catalog of quiescent components that do

not consume processor or memory resources but are ready to be started.

2. **`Channel actions:`** Once a component transits to an active state, it must establish communication with other services. MoRE implements these communication channels using the OSGi Wire class. An OSGi wire is an enhanced implementation of the publish-subscribe pattern oriented to dynamic systems. Specifically, an OSGi wire implements the whiteboard pattern.

3. **`Model actions:`** After the system architecture has been modified, MoRE updates the feature model according to the new system functionality. It performs this update by means of a partial reflection of the architecture using model introspection.

MoRE-WS only reuses MoRE bundles that support model actions.

The first two categories of actions are in charge of reconfiguring the smart-home architecture by starting or stopping components (i.e., OSGi bundles), and creating and destroying channels among components (i.e., OSGi wires). Nevertheless, these actions are not enough in the problem domain of this thesis. First, the dynamic adaptation of service compositions represent a more complex problem: service operations have to be executed in a particular order to support a BP. Therefore, activating or deactivating components at runtime is not enough in this domain. Second, MoRE makes use of low-level instructions for implementing and executing reconfiguration actions in OSGi (a.k.a. architecture increments and architecture decrements). Nevertheless, we argue that in a model-driven approach, reconfiguration actions should be stated and analyzed in a higher level of abstraction. Due to these situations, MoRE-WS only reuses MoRE bundles that support model actions.

7.2 MONITORING THE CONTEXT

The main building block in the monitoring component is the Context Monitor.

The monitoring component in the MAPE-K loop involves capturing basic metrics of specific quality attributes from the context that are of significance to the self-* properties of Web services. The main building block in the monitoring component is the Context Monitor, which observes the behavior of the service operations running in the context.

For the sake of flexibility, the Context Monitor is not attached to any specific implementation. In fact, the Context Monitor works as a plugin that can be connected or disconnected from the system, or replaced by other implementations at design time. Therefore, the

CONTEXT MONITOR can be implemented by different mechanisms according to particular needs. For example, the CONTEXT MONITOR could observe congestion in the network or other shared resources (such as from a DoS attack), or exceptions that can arise when a service operation has failed to meet its timing constraints (if the wait time exceeds a certain value) (Bass et al., 2012).

We have implemented a CONTEXT MONITOR that observes the context by triggering automatic requests to service operations and receiving responses from them. Our motivations to implement this CONTEXT MONITOR are as follows: 1) to count on full flexibility in the information to be observed and in the frequency of observations; and 2) to extend this CONTEXT MONITOR in the future via our own source code. Nevertheless, thanks to our plugin-based solution, other tools to observe the context could be used as well, such as SALMON (Ameller and Franch, 2008).

Our CONTEXT MONITOR *follows the ping/echo approach.*

Our CONTEXT MONITOR follows the ping/echo approach, which refers to an asynchronous request/response message pair exchanged between nodes (Bass et al., 2012); in our case, between the CONTEXT MONITOR and the Web services. The ping/echo approach can be used to determine reachability (or availability) and the round-trip delay of service operations.

The CONTEXT MONITOR counts on *sensors* to monitor the context and to get the measures for basic metrics of specific quality attributes. Specifically, our prototype has a sensor for `Availability` that measures the availability of a Web service all the time. Also, it has a sensor for `Execution Time` that measures the current execution time in milliseconds that a Web service takes to execute a job. Several sensors can be chosen at the same time. Also, they can be extended according to particular needs (e.g. in order to monitor other attributes, such as security). The CONTEXT MONITOR frequently updates a file with the collected context information. The description of how this file is updated and other implementation details of the CONTEXT MONITOR are described in Appendix A.1.

Discovering Web service operations falls outside the contributions of this work. Nevertheless, since our CONTEXT MONITOR is implemented as a plugin, we believe it could be easily extended in the future to support this capability by using existing techniques for Web services discovery[1]. A research trend in this area uses the Semantic Web to sup-

1 Web services discovery is a wide field of study with relevant research works in tracks of important conferences in the field of Web services, such as the International Conference on Web Services (ICWS).

port capability-based discovery and interoperation of Web services at runtime. For instance, the DARPA Agent Markup Language for Services (DAML-S) tries to close the gap between the Semantic Web and Web services (Paolucci and Sycara, 2003). Services use the Semantic Web to support capability-based discovery and interoperation at runtime. More recent works also use the Semantic Web to discover service operations (Srinivasan et al., 2005; Klusch and Kapahnke, 2010; Pahl et al., 2011). Web services are semantically described in terms of capabilities offered. Then, inferences are performed to match the capabilities requested with the capabilities offered.

7.3 ANALYZING THE CONTEXT

The file that is updated with the measures taken from the context needs to be queried to determine if any change has to be made in the service composition. This task is in charge of MoRE-WS, which periodically queries this file to find new contextual information. In order to count on a fresh representation of the context, MoRE-WS periodically updates the context model according to the information that has been collected by the Context Monitor. After inserting the events in the context model, MoRE-WS evaluates the values in this model to find out if any context condition has been accomplished.

MoRE-WS keeps a fresh abstract representation of the context.

By means of the SPARQL Protocol and RDF Query Language (SPARQL)[2], we have implemented the operations to insert context information into the context model and reason about context conditions. SPARQL is the W3C recommendation query language for RDF triples. This language is based on graph-matching techniques. Given a data source, a query consists of a pattern which is matched against the data source, and the values obtained from this matching are processed to give the answer. The data source to be queried can be an OWL model as is the one of the ontology in the context model.

A SPARQL query consists of the following three parts: 1) the *pattern matching part* includes several features of pattern matching of graphs, such as optional parts, union of patterns, nesting, filtering (or restricting) values of possible matchings, and the possibility of choosing the data source to be matched by a pattern; 2) the *solution modifiers* allow to modify these values applying classical operators like projection, distinct, order, limit, and offset; and 3) the *output*, which can be of different types: yes/no queries (ASK), selections of values of the variables

2 http://www.w3.org/TR/rdf-sparql-query

which match the patterns (SELECT), creation of new triples (INSERT), and descriptions of resources (DESCRIBE).

The operations that we have implemented in MoRE-WS to insert events into the context model and to reason about context conditions are described as follows (Appendix A.2.1 offers implementation details of these operations):

MoRE-WS uses two operations to manage context events from an abstract point of view.

1. **`Update the context model:`** This operation inserts new context events into the context model. To this end, MoRE-WS uses the SPARQL INSERT form, which inserts new triples in the RDF graph of the context model. Each new triple is in the form of (subject, predicate, object). As we mentioned in Section 7.2, our approach does not consider service discovery at runtime. Therefore, our solution keeps the original structure of the ontology during execution and focuses on inserting values into datatype properties.

2. **`Evaluate context conditions:`** This operation evaluates if a context condition has been accomplished. In order to analyze the collected context conditions, MoRE-WS uses the SPARQL ASK form to test whether or not a query pattern (e.g. a context condition) has a solution. This operation does not return information about the possible query solutions, just whether or not a solution exists. That is, ASK returns a Boolean value indicating whether a query pattern matches or not.

We propose the following strategy to avoid the saturation of MoRE-WS when several problematic context events arise in a tight time frame: 1) in each observation, MoRE-WS retrieves a list with an ordered sequence of new context events ($contEvent_1, contEvent_2, \ldots, contEvent_n$). It is possible to retrieve an ordered sequence of context events because every context event has a unique ascending identification number; 2) MoRE-WS sequentially evaluates the set of context conditions ($contCond_1, contCond_2, \ldots, contCond_n$), which can be triggered by the set of context events. Therefore, only one adaptation is triggered at a time for each collected problematic context event.

MoRE-WS avoids saturations when facing several problematic context events.

Examples:

The SPARQL operation in Listing 7.1 inserts into the context model information about the unavailability of the `Barnes & Noble Books` service operation.

Listing 7.1: An SPARQL INSERT example.

```
PREFIX web: <http://my.ontology#>
INSERT DATA {web: 'Barnes&NobleBooks' web:isAvailable 'false'}
```

The SPARQL query in Listing 7.2 evaluates the `B&NUnavailable` context condition. This listing asks if the `Barnes and Noble Books` service operation is currently unavailable.

Listing 7.2: An SPARQL ASK example.

```
PREFIX web: <http://my.ontology#>
ASK { web:Barnes&NobleBooks web:isAvailable 'false' }
```

7.4 PLANNING THE ADAPTATION

When an adaptation has been requested (i.e., after a context condition has been fulfilled), MoRE-WS carries out the following steps to plan the adaptation of the service composition. The implementation details of each step are described in Appendix A.2.2.

Step 1: Execute a Resolution

In this step, MoRE-WS triggers a resolution associated to a context condition (or a composite context condition), which has occurred. To this end, MoRE-WS carries out the following actions (see Figure 7.3):

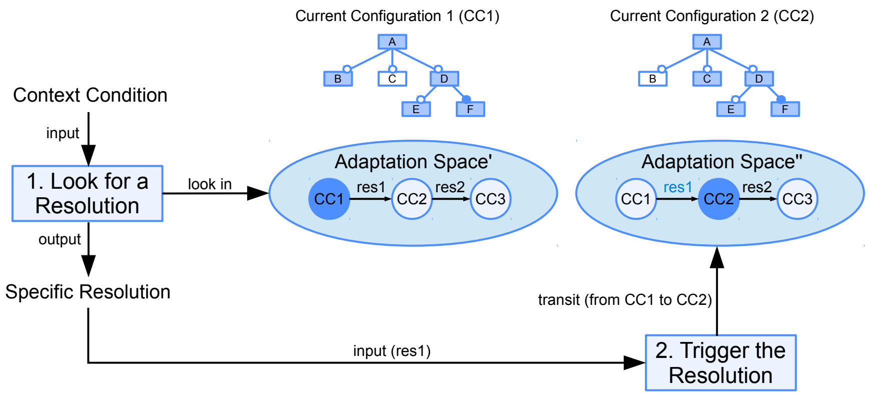


Figure 7.3: Execute a resolution.

1. **`Look for the resolution that is triggered by the context condition:`** In this action, MoRE-WS uses the information of the context condition that has been fulfilled in order to find the resolution that is triggered by this condition. To this end, MoRE-WS looks in the adaptation space for a transition, which represents a resolution, for the context condition. This operation is possible because every transition in the adaptation space encapsulates the information of the resolution and its related context condition. The main benefit of this approach is that MoRE-WS has full control of the different possible variability model configurations and the transitions among them. The output of this action is a resolution to be triggered on the feature model, which abstracts the variability model.

The adaptation space allows to have full control of configurations and transitions in the variability model.

2. **`Trigger the resolution:`** In this action, MoRE-WS triggers the resolution to modify the configuration of the feature model by activating/deactivating its features. The current configuration in the adaptation space transits to a new configuration thanks to the application of a resolution (i.e., transition in terms of the adaptation space). According to this change, MoRE-WS updates the current configuration in the adaptation space in order to use this information for subsequent adaptations.

Example:

MoRE-WS finds out that the `B&NUnavailable` has been fulfilled. With this information, it finds $R_{B\&NUnavailable}$, which deactivates the features related to the `Barnes & Noble Books` service operation (since it is unavailable), and activates the features related to the `Amazon Books` service operation. $R_{B\&NUnavailable}$ is described as follows:

$R_{B\&NUnavailable}$ = {(Barnes & Noble Books, Inactive), (Search Book, Inactive), (Show Book Info, Inactive), (Show Related Books, Inactive), (Amazon Books, Active), (Book Searching, Active), (Book Description, Active), (Related Titles, Active), (Barnes & Noble Shopping Cart, Inactive), (Amazon Shopping Cart, Active)}.

Step 2: Generate a Reconfiguration Plan

In this step, MoRE-WS creates a *reconfiguration plan*, which contains a set of reconfiguration actions to adapt the composition model according to the new configuration of the variability model (which has been

A reconfiguration plan contains a set of reconfiguration actions to adapt the composition model.

modified by a resolution). Reconfiguration actions are stated as *composition model increments* ($CM\triangle$) and *composition model decrements* ($CM\nabla$). These operations take a new configuration of the variability model as input, and they calculate the modifications to the composition model by adding ($CM\triangle$) or removing ($CM\nabla$) variant models (see Figure 7.4). The modified composition model will eventually cause the adaptation of the WS-BPEL code that orchestrates the service operations.

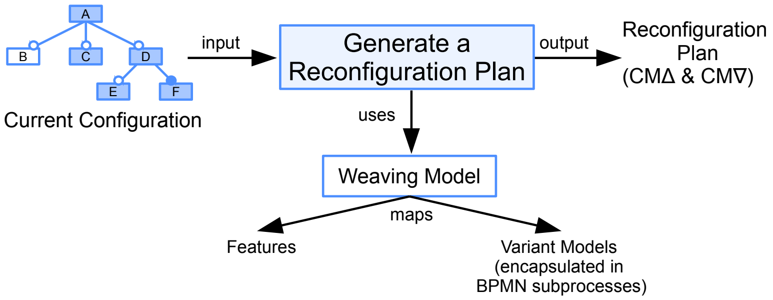

A
B
C
D
E
F
Current Configuration
input
Generate a Reconfiguration Plan
output
Reconfiguration Plan (CMΔ & CM∇)
uses
Weaving Model
maps
Features
Variant Models (encapsulated in BPMN subprocesses)

Figure 7.4: Generate a reconfiguration plan.

In order to generate reconfiguration actions, MoRE-WS queries the weaving model to realize the mappings between the features that are active in the new configuration of the feature model and the related BPMN subprocesses, which abstract variant models. In this way, a given service operation, which is represented in the composition model, will be invoked in the adapted service composition if and only if its related feature in the feature model configuration is active. That is, the composition model is adapted through the activation or deactivation of features. The currently active features, which have not been deactivated in the new configuration of the variability model, still active.

Example:

When MoRE-WS applies $R_{B\&NUnavailable}$ to the initial configuration of the case study (depicted in Figure 4.2), the resulting reconfiguration plan is the following:

$CM\nabla$ = {*Barnes & Noble Books variant model, Barnes & Noble Shopping Cart variant model*} and $CM\triangle$ = {*Amazon Books variant, Amazon Shopping Cart variant model*}.

These actions express how to reorganize elements in the composition model to move from one configuration when the `Barnes & Noble Books` composite service operation fails to another configuration when this operation is replaced by the `Amazon Books`, `Related Titles`, and `Amazon Shopping Cart` Web service operations (according to Figure 5.6).

Step 3: Adapt the Composition Model

In this step, MoRE-WS applies the reconfiguration plan (with $CM\triangle$ and $CM\nabla$ actions) on the composition model. Specifically, it carries out the following actions:

1. MoRE-WS loads the current version of the composition model (i.e., the one that reflects the current situation of the service composition). Let us name the current version of the composition model *"running composition model"*. In order to keep track of the variation points that can be rebound at runtime, the running composition model always keeps the information of the variation points.

2. MoRE-WS deletes all the modeling elements in the variation points of the running composition model that are affected by $CM\nabla$ actions.

3. MoRE-WS loads the XMI file that contains the base composition model and the variant models. The objective of this action is to count on the abstract elements to be incremented into the running composition model according to $CM\triangle$ actions.

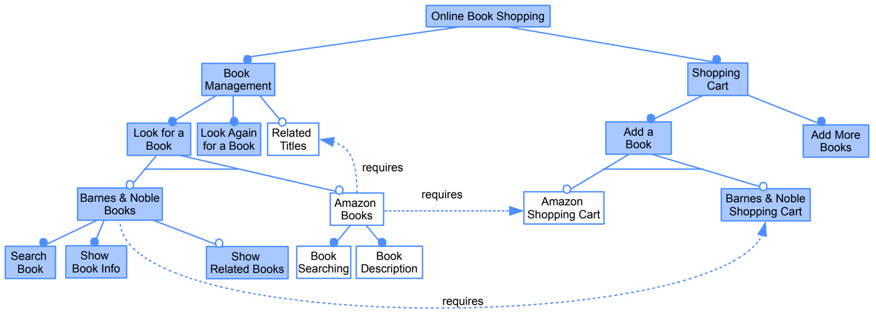
4. MoRE-WS inserts variant models into variation points in the running composition model according to $CM\triangle$ actions.

5. MoRE-WS saves the new version of the running composition model.

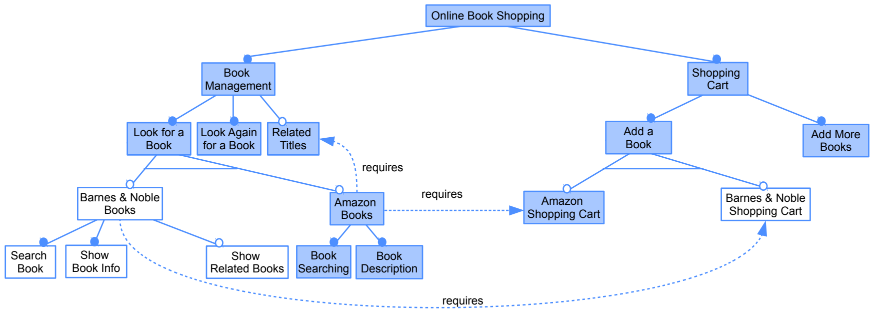
A Summarizing Example

Figure 7.5 shows the model adaptation process when a context condition has been fulfilled.

a) Initial variability model configuration. Features in blue are active (*configuration 0*):



b) New variability model configuration after $R_{B\&NUnavailable}$ has been executed. Features in blue are active (*configuration 1*):



c) Modified running composition model after a reconfiguration plan ($CM\triangle$, $CM\nabla$) has been executed:

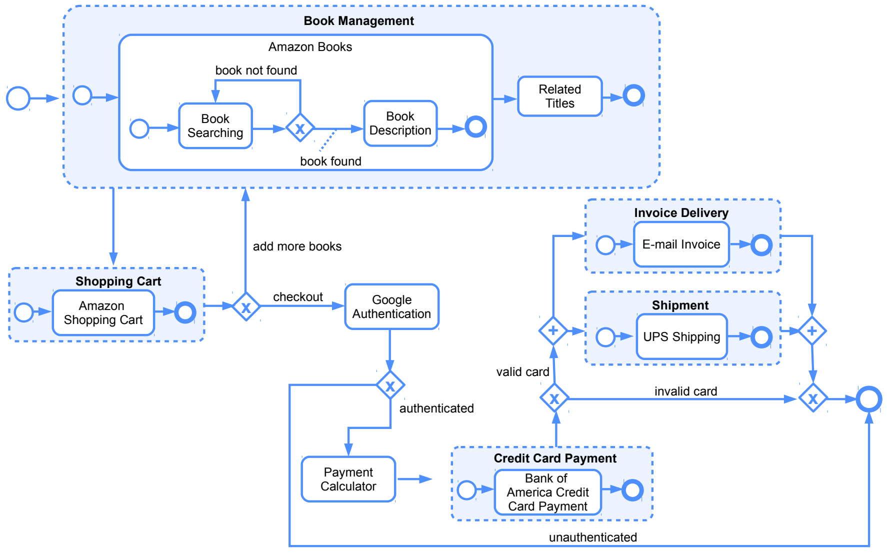


Figure 7.5: Model adaptation process.

Section a shows a fragment of the initial variability model configuration in the case study. At a particular point in time, the `B&NUnavailable` context condition is fulfilled. As a result, MoRE-WS looks for a resolution to solve this situation and finds $R_{B\&NUnavailable}$ that triggers the activation and deactivation of features in the variability model (*section b*). The application of this resolution causes the system to transit from *configuration 0* to *configuration 1*. Finally, MoRE-WS creates a reconfiguration plan and executes it to modify the composition model (*section c*).

7.5 EXECUTING THE ADAPTATION

In order to materialize the changes at the composition model level into the running service composition, the `Execute` component is divided into three steps: 1) MoRE-WS creates a new version of the deployment directory in which the adapted WS-BPEL composition schema will be running; 2) MoRE-WS inserts WS-BPEL code fragments into the WS-BPEL template at runtime. The completed WS-BPEL template (i.e., this template with all its bound variation points) is used to orchestrate the service composition; and 3) MoRE-WS deploys at runtime the generated WS-BPEL code and other required artifacts in the Execution Engine. These steps are described as follows. Implementation details are described in Appendix A.2.3.

Step 1: Create a New Version of the Deployment Directory

First of all, MoRE-WS creates a deployment directory in the Execution Engine. This is a mandatory step because the composite service is executed by invoking the composition schema in this directory.

We have implemented a versioning strategy for the deployment directory to prevent the Execution Engine from deleting all the running instances when a new composition schema is deployed. To this end, a new deployment directory with an increasing version number is deployed with every dynamic adaptation. New instances run according to the composition schema in the directory with the latest version.

Step 2: Insert WS-BPEL Fragments into the WS-BPEL Template

The adaptation of the composition schema is guided by the information contained in the adapted composition model. In this step, we propose an approach to reflect the changes in the composition model into

WS-BPEL code by adding WS-BPEL code fragments into the WS-BPEL template. Specifically, MoRE-WS carries out the following three actions to adapt the composition schema (see Figure 7.6):

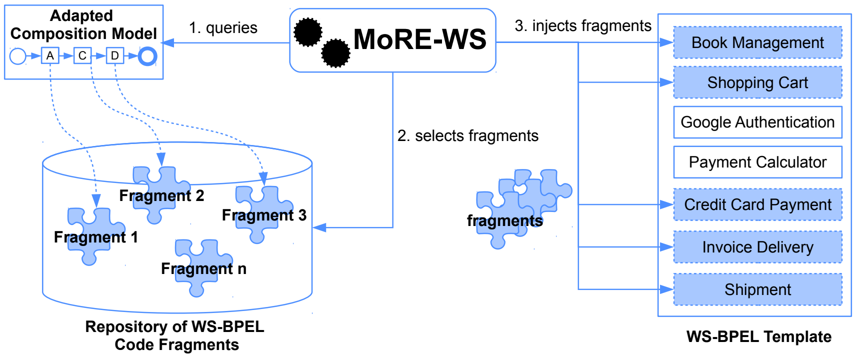


Figure 7.6: Inserting WS-BPEL code fragments into the WS-BPEL template.

1. MoRE-WS gets the current set of variant models in the adapted running composition model (e.g. see *section c* in Figure 7.5). Each variant model maps to a WS-BPEL code fragment, which is stored in a repository (i.e., a directory) as described in Section 5.2.10.

2. MoRE-WS selects the set of WS-BPEL code fragments that map to the variant models present in the current version of the running composition model.

3. MoRE-WS injects the selected WS-BPEL code fragments into the WS-BPEL template. In order to find the variation points in the WS-BPEL template, MoRE-WS looks for `<documentation>` elements. These elements are used to specify the location of variation points in this template (see Section 5.2.10 for details). The WS-BPEL template is saved in the new version of the deployment directory as a new WS-BPEL file. As a result, the original WS-BPEL template is always kept clean (without changes) in order to be used for further adaptations.

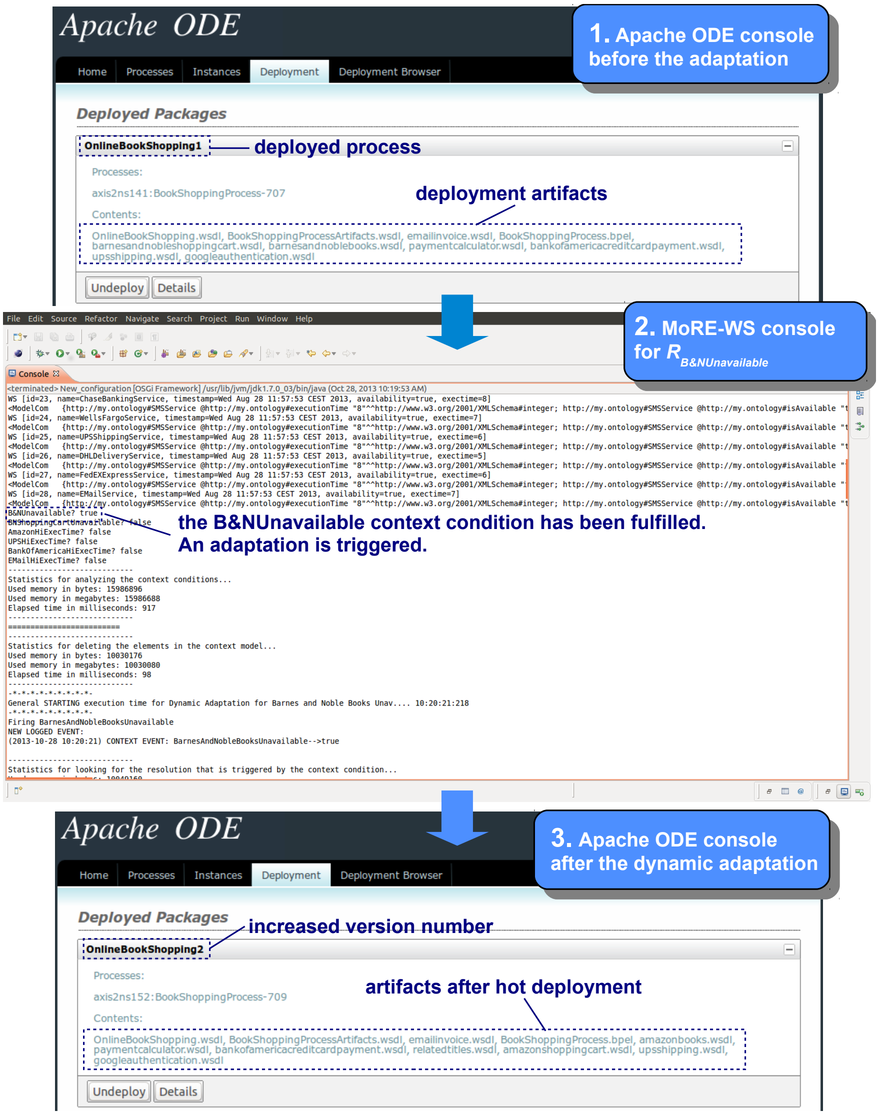
Step 3: Hot Deployment

In this step, MoRE-WS hot deploys the adapted composition schema on the Execution Engine. Apache ODE[3] was chosen as the Execu-

3 http://ode.apache.org

TION ENGINE because it is compliant with WS-BPEL and offers mature hot-deployment support. Instead of extending the functionality of the EXECUTION ENGINE, our approach is transparent to the engine (i.e., it is unchanged). Therefore, our approach could be used with other WS-BPEL engines that support hot deployment. Figure 7.7 shows MoRE-WS and APACHE ODE consoles during a dynamic adaptation according to $R_{B\&NUnavailable}$.

Our approach is transparent to the EXECUTION ENGINE.


Apache ODE
Home
Processes
Instances
Deployment
Deployment Browser
1. Apache ODE console before the adaptation
Deployed Packages
OnlineBookShopping1
deployed process
Processes:
axis2ns141:BookShoppingProcess-707
deployment artifacts
Contents:
Undeploy
Details
File Edit Source Refactor Navigate Search Project Run Window Help
Console
2. MoRE-WS console for $R_{B\&NUnavailable}$
the B&NUnavailable context condition has been fulfilled. An adaptation is triggered.
Apache ODE
Home
Processes
Instances
Deployment
Deployment Browser
3. Apache ODE console after the dynamic adaptation
Deployed Packages
increased version number
OnlineBookShopping2
Processes:
axis2ns152:BookShoppingProcess-709
artifacts after hot deployment
Contents:
Undeploy
Details

OnlineBookShopping.wsdl, BookShoppingProcessArtifacts.wsdl, emailinvoice.wsdl, BookShoppingProcess.bpel, barnesandnobleshoppingcart.wsdl, barnesandnoblebooks.wsdl, paymentcalculator.wsdl, bankofamericacreditcardpayment.wsdl, upsshipping.wsdl, googleauthentication.wsdl

```
<terminated> New_configuration [OSGi Framework] /usr/lib/jvm/jdk1.7.0_03/bin/java (Oct 28, 2013 10:19:53 AM)
WS [id=23, name=ChaseBankingService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=8]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#SMSService @http://my.ontology#isAvailable "t
WS [id=24, name=WellsFargoService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=7]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#SMSService @http://my.ontology#isAvailable "t
WS [id=25, name=UPSShippingService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=6]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#SMSService @http://my.ontology#isAvailable "t
WS [id=26, name=DHLDeliveryService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=5]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#SMSService @http://my.ontology#isAvailable "
WS [id=27, name=FedEXExpressService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=6]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#SMSService @http://my.ontology#isAvailable "
WS [id=28, name=EMailService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=7]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#SMSService @http://my.ontology#isAvailable "t
B&NUnavailable? true
BNShoppingCartUnavailable? false
AmazonHiExecTime? false
UPSHiExecTime? false
BankOfAmericaHiExecTime? false
EMailHiExecTime? false
----------------------------
Statistics for analyzing the context conditions...
Used memory in bytes: 15986896
Used memory in megabytes: 15986688
Elapsed time in milliseconds: 917
----------------------------
=========================
----------------------------
Statistics for deleting the elements in the context model...
Used memory in bytes: 10030176
Used memory in megabytes: 10030080
Elapsed time in milliseconds: 98
----------------------------
-*-*-*-*-*-*-*-*-*-
General STARTING execution time for Dynamic Adaptation for Barnes and Noble Books Unav.... 10:20:21:218
-*-*-*-*-*-*-*-*-*-
Firing BarnesAndNobleBooksUnavailable
NEW LOGGED EVENT:
(2013-10-28 10:20:21) CONTEXT EVENT: BarnesAndNobleBooksUnavailable-->true

----------------------------
Statistics for looking for the resolution that is triggered by the context condition...
```

OnlineBookShopping.wsdl, BookShoppingProcessArtifacts.wsdl, emailinvoice.wsdl, BookShoppingProcess.bpel, amazonbooks.wsdl, paymentcalculator.wsdl, bankofamericacreditcardpayment.wsdl, relatedtitles.wsdl, amazonshoppingcart.wsdl, upsshipping.wsdl, googleauthentication.wsdl

Figure 7.7: MoRE-WS and Apache ODE consoles during a dynamic adaptation.

In addition to copying the adapted composition schema into the deployment directory (as described in Step 2), MoRE-WS also copies the WSDL files, which describe the functionality offered by the Web services, and the deployment descriptor (an XML file). Since the Execution Engine carries out the hot deployment as soon as the deployment descriptor is added into the deployment directory, this is the last operation that is carried out by MoRE-WS.

After the adapted composition schema is deployed, all new instances run according to the new version of the composition schema. The instances that are currently running when an adaptation is triggered, continue their execution according to the old version of the composition schema. The dynamic adaptation of running instances is covered in related work (Weber et al., 2008). In (Weber et al., 2008), the migration of instances from an old composition schema to a new one is carried out when it is safe to do so. That is, only those instances are migrated which are compliant with the old version of the composition schema. Specifically, an instance I is *compliant* with a composition schema S, if the current execution history of I can be created based on S (Rinderle et al., 2004). All other instances remain running according to the old version of the composition schema.

7.6 CONCLUSIONS

In this chapter, we have presented an approach to achieve the dynamic adaptation of service compositions through models at runtime. Since MoRE-WS extends MoRE for supporting the dynamic adaptation of service compositions, we described MoRE in Section 7.1. The following sections were structured according to the components of the MAPE-K loop because MoRE-WS implements them. First, the Context Monitor, which collects information from the context, was described in Section 7.2. Then, the operations that are carried out by MoRE-WS to analyze the collected context information and realize the context conditions that are fulfilled were described in Section 7.3. Afterwards, the actions that are carried out by MoRE-WS to plan the adaptation were described in Section 7.4. Specifically, MoRE-WS executes a resolution, generates a reconfiguration plan, and adapts the composition model. Finally, the actions that are carried out by MoRE-WS to execute the adaptation were described in Section 7.5. To this end, MoRE-WS inserts WS-BPEL code fragments into the WS-BPEL template and hot deploys the adapted composition schema. Appendix B shows a scenario in a case study in which two dynamic adaptations are carried out on a service composition.

Appendix B shows a scenario in a case study in which two dynamic adaptations are carried out.

The contributions presented in this chapter are based on scientific publications (Alférez and Pelechano, 2011a; Ayora et al., 2011, 2012; Alférez et al., 2013; Pelechano et al., 2013).

8 ACHIEVING DYNAMIC EVOLUTION THROUGH MODELS AT RUNTIME

"To improve is to change; to be perfect is to change often."

— Winston Churchill

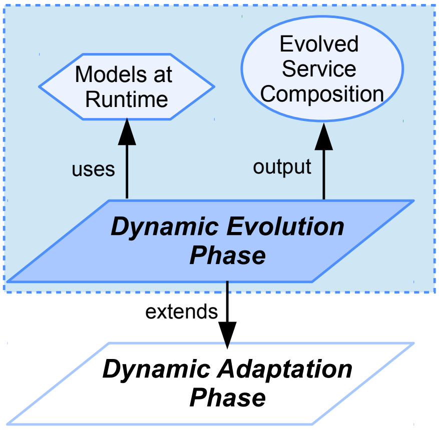


Figure 8.1: Scope of Chapter 8.

Service compositions run in complex contexts where arising events may affect the quality of the system. However, crucial service compositions cannot be stopped in order to apply changes to deal with problematic context events. Therefore, the trend is moving towards context-aware service compositions, which use context information as a basis for autonomic changes. Under the closed-world assumption, the context and possible adaptations are fully known at design time. Nevertheless, it is difficult to foresee all the possible situations arising in uncertain contexts.

In the unpredictable open world, service compositions should react to continuous and unanticipated events in complex and uncertain contexts. As described in Chapter 7, it is possible to use the knowledge in

In the open world, models should be able to evolve for better functioning and system "survival".

the models that are created at design time to guide the dynamic adaptation of service compositions in the closed world. In order to face uncertainty in the open world, the models should be able to evolve during execution for better functioning and system "survival". Otherwise, the benefits of using models at runtime to guide adjustments of service compositions will be diminished with static abstractions that fall short to manage unforeseen context events.

In this chapter, we describe a computing infrastructure that leverages models at runtime to guide the dynamic evolution of context-aware service compositions. This infrastructure supports the `Dynamic Evolution Phase` of our framework. Dynamic evolutions are carried out to deal with unknown context events (i.e., unforeseen at design time) in the open world, where the service composition is deployed. In order to manage uncertainty, the proposed infrastructure self-evolves the variability model and the composition model to preserve expected requirements. The evolved models guide changes in the underlying WS-BPEL composition schema, which orchestrates the service composition. As a result, dynamic evolutions move the service composition to new versions, which cannot be supported by predefined dynamic adaptations.

Two tools support dynamic evolutions, namely Evolution Planner and MoRE-WS. On one hand, the Evolution Planner plans evolutions by looking for unknown context events and the requirements that can be affected by these events. On the other hand, MoRE-WS evolves the variability model and the composition model, which abstract the service composition. Then, it uses the evolved models to evolve the WS-BPEL composition schema.

The remainder of this chapter is structured as follows. Section 8.1 presents the overview of the computing infrastructure for the dynamic evolution of service compositions. Section 8.2 describes how the Evolution Planner plans evolutions. Section 8.3 describes how MoRE-WS supports model-driven dynamic evolutions. Finally, Section 8.4 presents the conclusions.

8.1 COMPUTING INFRASTRUCTURE FOR DYNAMIC EVOLUTIONS

The knowledge in the models created in the `Design Phase` is used in the `Dynamic Evolution Phase` to guide the dynamic evolution of the service composition. To this end, the `Dynamic Evolution Phase` extends the `Dynamic Adaptation Phase` with dynamic evolution capabilities.

The `Dynamic Evolution Phase` extends the `Dynamic Adaptation Phase` with dynamic evolution capabilities.

There are two operations that are necessary to reason about the dynamic evolution of service compositions. First, it is necessary to analyze unknown context events and plan evolutions to protect expected requirements when these events arise. Since highly-abstract models express the underlying service composition, we argue that evolutions can be planned at the modeling level. The second operation is about applying (or materializing) dynamic adjustments into the service composition. To this end, evolutions that have been planned at the modeling level, have to be reflected into the service composition.

According to the aforementioned aspects to reason about dynamic evolutions, we can say that the `Dynamic Evolution Phase` is particularly interested in extending the following phases of the `Dynamic Adaptation Phase` (see Figure 7.2): 1) `Analyze` and `Plan` phases by reasoning about evolutions at the modeling level; and 2) `Execute` phase by reflecting the evolutions at the modeling level into the service composition.

We propose two building blocks to support the dynamic evolution of service compositions.

In this chapter, we propose two building blocks for analyzing arising unknown context events, planning dynamic evolutions at a highly-abstract level, and executing dynamic evolutions on service compositions (see Figure 8.2). These building blocks are as follows:

1. **`Evolution Planner:`** The EVOLUTION PLANNER constantly looks for unknown context events in the open world. To this end, it *analyzes* the context model with the updated context information. If there is an unknown context event, then it looks for a requirement that can be affected by this event. Afterwards, it *plans* the evolution by looking for a surviving tactic to preserve the requirement. Highly-abstract tactics are the main components to face uncertainty in service compositions that run in the open world.

2. **`MoRE-WS:`** MoRE-WS *executes* the necessary evolution in two main steps. In the first step, MoRE-WS merges the discovered tactic into the variability model. Then, it evolves the composition model according to the evolved variability model. These two models are evolved because they abstract two key aspects of autonomic service compositions: variability and the workflow among service operations. The evolved models support the tactic's functionality to preserve the affected requirement. In the second step,

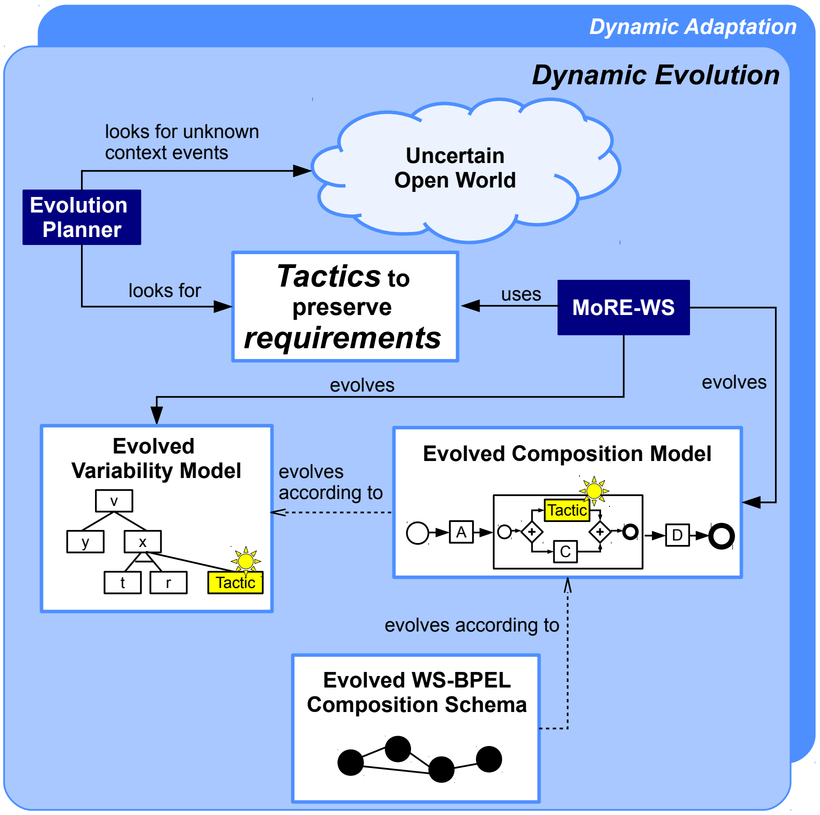


Figure 8.2: Architecture of the dynamic evolution layer to face uncertainty in the open world.

> MoRE-WS evolves the WS-BPEL composition schema according to the evolved composition model. Since the composition model abstracts the workflow in the underlying service composition, the evolution of the composition model can be used to guide the evolution of the service composition.

The operations to update the context model are described in Section 7.2. The operations that are carried out by MoRE-WS to hot deploy WS-BPEL composition schemes are described in Section 7.5. Therefore, this chapter focuses on describing the Evolution Planer and the role of MoRE-WS to support dynamic evolutions.

8.2 PLANNING THE EVOLUTION WITH THE EVOLUTION PLANNER

The main objective of the EVOLUTION PLANNER is to look for the tactics to preserve the requirements that can be affected by unknown context events. The output of this building block is a set of surviving tactics. To this end, the EVOLUTION PLANNER constantly analyzes the context model to find situations that may require the dynamic evolution of the service composition. In order to deal with unknown context events, the EVOLUTION PLANNER carries out the following steps (Appendix A.3.1 describes the implementation details):

The EVOLUTION PLANNER *looks for tactics to preserve requirements.*

Step 1: Search for Unknown Context Events

The first step to trigger a dynamic evolution is to look for unknown context events. To this end, the EVOLUTION PLANNER periodically checks the updated ontology in the context model. An observed context event is considered as *unknown* when there are not predefined context conditions to deal with it. As described in Section 5.2.5, context conditions are Boolean expressions that work as SLAs. If a context condition is fulfilled (i.e., an SLA is violated), then an adaptation is triggered on the service composition to deal with the arising situation.

An observed context event is unknown when there are not predefined context conditions to deal with it.

Step 2: Search for Affected Requirements

The objective of this step is to look for the requirements that can be affected by an unknown context event. This is a key information because the surviving tactics are associated to requirements. In order to find the requirements that can be affected by unknown context events, the EVOLUTION PLANNER uses *forward chaining* (Labhart et al., 1990), a well-known method of reasoning in AI. This method evaluates arising context *facts* (i.e., context events) against general *rule premises* in a knowledge base. A key advantage of forward chaining in the open world is that new context events can trigger new inferences. The knowledge base is implemented as a rule file (see Section 6.7).

Example:

Figure 8.3 shows a basic example when the unknown context event *F1* (a fact) is detected. In this case, the rule *underAttack* has a condition that matches this new fact (step 1). Then, the forward chaining method fires the new fact *F2* (step 2). The process continues until the fact *F3* is fired (step 4). *F3* indicates that *the* ***Barnes & Noble Books service***

operation** can affect the **High Security softgoal. This example shows that evolutions are only triggered when requirements are negatively impacted. The implementation of the rules, which are involved in this example, is described in Listing 6.2.

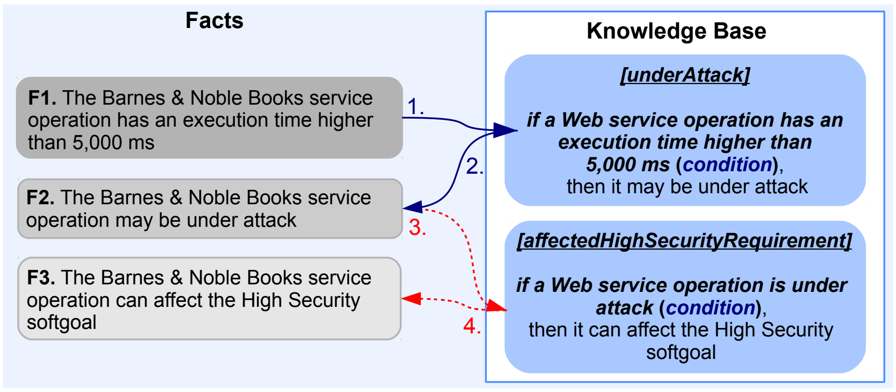

Facts
F1. The Barnes & Noble Books service operation has an execution time higher than 5,000 ms
F2. The Barnes & Noble Books service operation may be under attack
F3. The Barnes & Noble Books service operation can affect the High Security softgoal
1.
2.
3.
4.
Knowledge Base
[underAttack]
if a Web service operation has an execution time higher than 5,000 ms (condition), then it may be under attack
[affectedHighSecurityRequirement]
if a Web service operation is under attack (condition), then it can affect the High Security softgoal

Figure 8.3: Forward chaining inference example.

Step 3: Search for Surviving Tactics

The objective of this step is to discover the tactics to preserve the requirements that can be negatively impacted by an unknown context event. To this end, the Evolution Planner carries out the following steps:

1. If there is not any claim that contributes to the affected softgoals, then it selects the tactics with the most positive contributions on these softgoals.

2. If there is any claim that contributes to the affected softgoals, it checks the claims associated to the affected softgoals. According to claims, it chooses the tactics with the most positive contributions on the affected softgoals.

Examples:

When the Evolution Planner finds that the *Barnes & Noble Books service operation can affect the High Security softgoal* (an unknown context event), it looks for the `High Security` softgoal in the requirements model (see Figure 6.6). The EVOLUTION PLANNER chooses the `Deception` tactic because it has the most positive impact on this softgoal.

In another scenario, an unknown context event affects the `High Security` and the `High Performance` softgoals at the same time. In this case, the `Temporal Separation` and the `Introduce Concurrency` tactics are chosen together. These tactics are selected because they have the most positive contributions on the `High Security` and the `High Performance` softgoals according to a claim (see Figure 6.6).

8.3 USING MORE-WS FOR DYNAMIC EVOLUTIONS

In order to evolve the service composition for preserving requirements with the tactics found by the EVOLUTION PLANNER, it is necessary to answer the following questions:

1. How to vary the service composition in order to protect requirements with tactics?

2. How to keep the main workflow order of the service composition whilst it evolves at runtime (e.g. service operation α comes before – or has to be invoked before – service operation β despite evolutions)?

3. How to evolve the WS-BPEL composition schema by reasoning about evolutions carried out in a highly-abstract level?

In this section, we answer these questions through a set of steps that are carried out by MoRE-WS. Among these steps, the ones related to merging tactical functionality into models at runtime are key to achieve the dynamic evolution of service compositions (i.e., changes at the modeling level will be eventually reflected into the service composition). Implementation details of these model-merging steps are presented in Appendix A.3.2.

Merging tactical functionality into models at runtime is crucial to achieve dynamic evolutions.

Step 1: Merge a Tactic Model into the Variability Model

The variability model abstracts the variant functionalities of the service composition in terms of features. In order to inject the functionality of the discovered tactic into the variability model, this step has two objectives: 1) to identify a tactic model (which is expressed as a feature model) that describes the tactic to be triggered for preserving an affected requirement at runtime; and 2) to merge the required tactic model into the variability model to count on an enriched-evolved variability model.

MoRE-WS takes into account the following aspects for merging tactic models into feature-based variability models. First, the tactic model is considered as a variant that can be added to the variability model. Second, the tactic model is inserted under the variation point of the feature that has affected the requirement.

Example:

In the case of the discovered *"Barnes & Noble Books service operation can affect the High Security softgoal"* context event, the deception tactic model is inserted as a variant under the `Look for a Book` variation point (see figure 8.4).

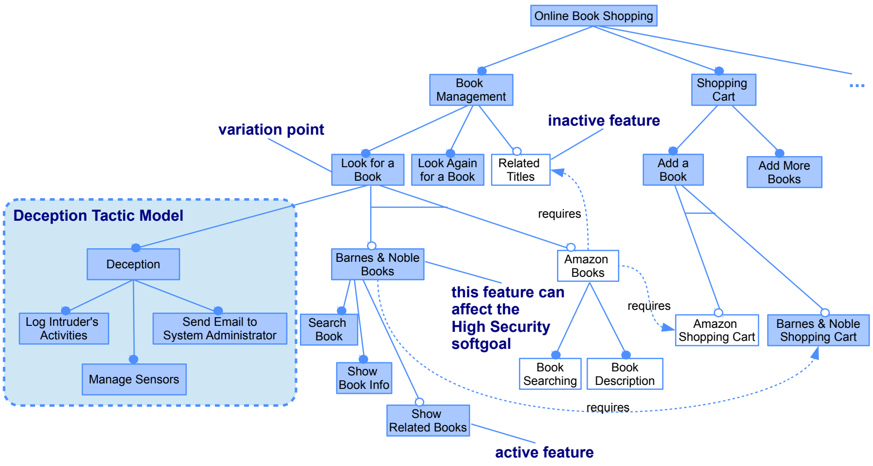


Figure 8.4: A fragment of the evolved variability model in our case study.

Step 2: Generate an Evolution Policy

Merging a tactic model into the variability model may cause the activation or deactivation of features in the current configuration of the variability model. Therefore, the objective of this step is to generate an *evolution policy*, containing evolution actions, to decide which system features need to be activated or deactivated in the evolved variability model. An evolution policy (*EP*) for a particular tactic (*T*) can be expressed as a list of pairs (*F*, *S*) where each pair is made up of a feature (*F*) in the evolved variability model (*EVM*) and the state (*S*) of the feature (*active* or *inactive*):

$$EP_T = \{(F, S) \mid F \epsilon [EVM] \wedge S \epsilon \{Active, Inactive\}\}$$

The generated evolution policies activate or deactivate features according to the following rules:

1. Since the features in the inserted tactic model are necessary to keep a particular requirement working, all the features in the tactic model are activated.

2. If the variant that represents the inserted tactic can run in parallel with other variants (according to a restriction), then the features in the evolved variability model keep their current states.

3. If the tactic model requires the activation or deactivation of features in the initial variability model, then the EVOLUTION PLANNER triggers the necessary changes.

Examples:

The following evolution policy activates the deception functionality in the evolved variability model according to the discovered *"Barnes & Noble Books service operation can affect the High Security softgoal"* context event: $EP_{Deception}$ = *{(Deception, Active), (Log Intruder's Activities, Active), (Manage Sensors, Active), (Send E-mail to System Administrator, Active)}*. The active features in the evolved configuration are highlighted in Figure 8.4. Active features keep their state because this tactic can run in parallel with the current active functionality.

In another example, the activation of the temporal separation tactic causes the problematic feature to become inactive (e.g. `Look for a Book` and its subfeatures). In this scenario, a restriction in the tactic model for temporal separation states that this tactic cannot run in parallel with the problematic functionality. Therefore, the tactic's restriction helps to indicate that the problematic functionality needs to be temporarily separated or inactive when this tactic is activated.

Step 3: Create a Reconfiguration Plan

MoRE-WS creates a reconfiguration plan to evolve the composition model.

MoRE-WS creates a *reconfiguration plan*, which contains a set of reconfiguration actions to evolve the composition model representing the underlying service composition. The evolved composition model keeps the consistency between the evolved variability model and the underlying service composition. Reconfiguration actions in the reconfiguration plan are stated as ($CM\triangle$) and ($CM\nabla$). These operations take an evolution policy as input, and they calculate the modifications to the composition model by adding ($CM\triangle$) or removing ($CM\nabla$) model elements.

The evolution policy is described in terms of the activation or deactivation of features, and the reconfiguration plan is described in terms of elements in the composition model model. Therefore, it is necessary to query the mappings in the weaving model to realize which elements in the composition model should be added or removed according to the features in the evolution policy.

Example:

The generated reconfiguration plan to reorganize elements in the composition model in the case of the discovered *"Barnes & Noble Books service operation can affect the High Security softgoal"* context event is as follows: $CM\triangle$ = *{Deception, LogIntruderActivities, ManageSensors, SendEmailSystAdmin, NetworkComLog, DataWritingLog}*.

Step 4: Merge a Tactic Model into the Composition Model

In order to inject the functionality of the discovered tactic into the service composition, it is necessary to: 1) identify a tactic model, in terms of a composition model, which describes the tactic to be triggered (i.e., $CM\triangle$) for preserving an affected requirement; and 2) to merge the required tactic model into the composition model to count on an enriched composition model that guides changes in the service composition. The merging operation was inspired by the *insert process fragment* pattern described in (Weber et al., 2008).

The set of steps that are carried out in the merging operation are as follows: 1) the activity (e.g. subprocess or task) describing the service operation that can negatively affect a requirement is put into a new subprocess; 2) the discovered tactic is put into the created subprocess; 3) a parallel relationship is created between the problematic activity and the tactic. As a result, the tactic's functionality will be executed when the problematic service operation is invoked; 4) the sequence flows that come in and go out from the problematic activity are redirected to the created subprocess. If subsequently, an activity that has been preserved with a tactic needs to be removed (i.e., in case of a triggered predefined dynamic adaptation), then the tactic is also removed.

Example:

Figure 8.5 shows the evolved composition model after discovering that the `Barnes & Noble Books` service operation can affect the `High Security` softgoal. The numbers in the figure indicate the aforementioned merging operations.

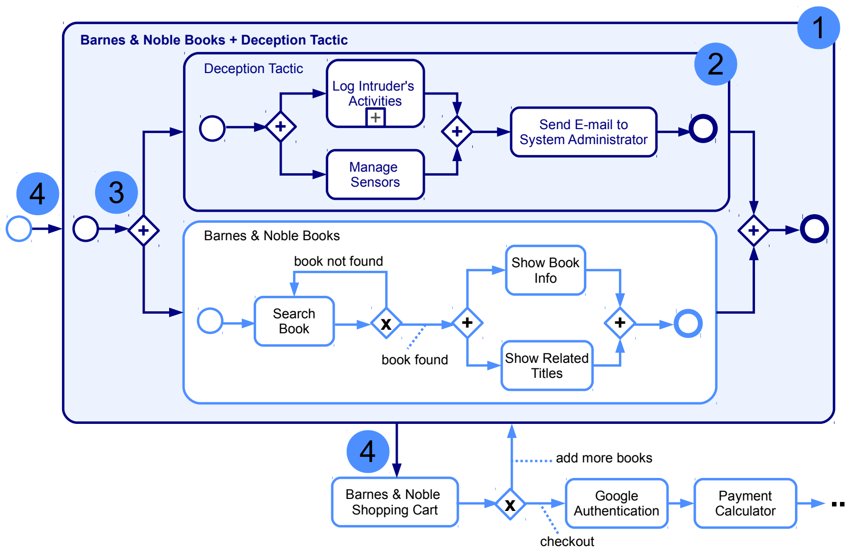


Figure 8.5: Evolved composition model in our case study.

Step 5: Evolve the WS-BPEL Composition Schema

The evolution of the WS-BPEL composition schema is guided by the information contained in the evolved composition model. We propose the following steps to reflect the changes in the evolved composition model into the WS-BPEL composition schema (see Figure 8.6): 1) MoRE-WS looks for the tactic that has been added into the composition model;

2) with this information, MoRE-WS looks for the WS-BPEL code fragment that invokes the tactical functionality. Each tactic model maps to a WS-BPEL code fragment, which is stored in a repository (i.e., a directory). Each code fragment has an associated WSDL, which is used to invoke the tactic's Web service; and 3) MoRE-WS injects the WS-BPEL code fragment that invokes the tactic into the composition schema. A parallel flow is dynamically created between the code that invokes the affected service operation and the code that invokes the tactic's Web service.

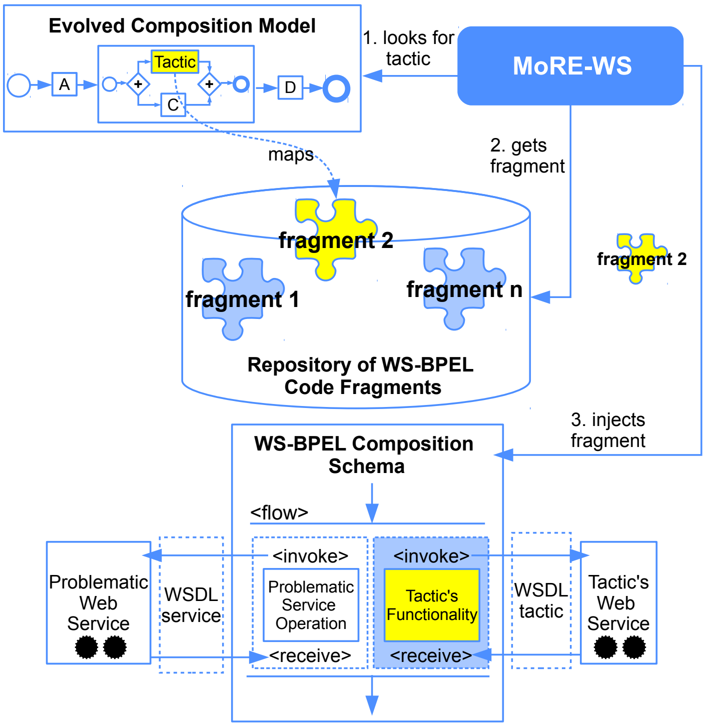


Figure 8.6: Evolution of the WS-BPEL composition schema.

In each evolution, MoRE-WS puts the evolved composition schema and other required artifacts (e.g. WSDL files) into a deployment directory. This directory is hot deployed by the Execution Engine. Each new directory has a higher version to prevent the Execution Engine from deleting all the running instances with new deployments. New instances run according to the evolved composition schema. Therefore, our approach also covers the dimension of Dynamic Evolution of the Composition Schema (see Section 6.2). Existing approaches, such as Weber et al. (Weber et al., 2008), offer solutions that can be used to migrate running instances to cope with the evolved composition schema. Finally, instead of extending the functionality of the Execution Engine, our approach for dynamic evolution offers a transparent solution.

8.4 conclusions

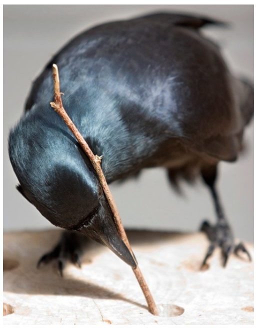
In this chapter, we have presented an approach to achieve the dynamic evolution of service compositions through models at runtime. First, we introduced a computing infrastructure to support dynamic evolutions in Section 8.1. This infrastructure is specially focused on extending the `Analyze`, `Plan` and `Execute` components of our MAPE-K-based solution for the dynamic adaptation of service compositions. Two main building blocks support the dynamic evolution of service composition: 1) the Evolution Planner looks for the tactics to preserve the requirements that can be affected by unknown context events. The Evolution Planner was described in Section 8.2; and 2) MoRE-WS carries out the evolution of the WS-BPEL composition schema by leveraging models at runtime. The model-driven mechanisms that are used by MoRE-WS for dynamically evolving the WS-BPEL composition schema are described in Section 8.3. Our solution for dynamic evolutions can be plugged/unplugged from the Execution Engine where the service composition is deployed.

Appendix B shows a scenario in a case study in which the dynamic evolution of a service composition is carried out. The contributions presented in this chapter are based on scientific publications (Alférez and Pelechano, 2012a, 2013b).

Appendix B shows a dynamic evolution scenario.

Part IV

APPLICABILITY AND PROVABILITY



The New Caledonian Crow uses tools to survive.

Photo credit: Bachild Marcello, Flickr, 2012.

The New Caledonian Crow is a tool-using species of crow, which are able to make hooks (Hunt, 1996). Without tools, it will be difficult for them to find the most profitable prey to survive (i.e., to adapt in the environment). These crows use different plant material for creating hooks. Each type of tools is used in different ways, for instance to grasp food in a narrow hole or to reach food in a difficult place.
As the New Caledonian Crow uses tools to adapt to its environment, our approach requires tools to allow service compositions to adapt to their context. These tools are used to model the autonomic behavior and to leverage models at runtime to guide dynamic adjustments.
Part IV covers the applicability and provability of our approach. This part is structured as follows: Chapter 9 describes the tools that support autonomic service compositions. The set of tools that can be used at design time is described in the context of two method contents. Chapter 10 describes the validation of our approach.

9 TOOL SUPPORT AND METHOD CONTENTS

"Computers are magnificent tools for the realization of our dreams, but no machine can replace the human spark of spirit, compassion, love, and understanding."

— Louis Gerstner

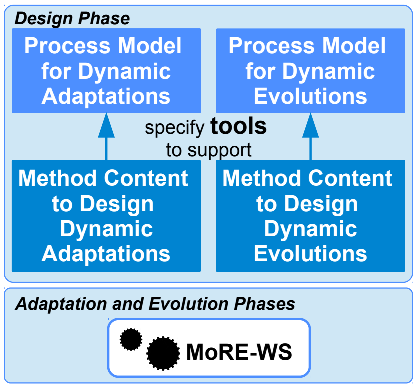

Design Phase
Process Model for Dynamic Adaptations
Process Model for Dynamic Evolutions
specify tools to support
Method Content to Design Dynamic Adaptations
Method Content to Design Dynamic Evolutions
Adaptation and Evolution Phases
MoRE-WS

Figure 9.1: Scope of Chapter 9.

Software engineering tools have existed since the early days of computer programming. They are becoming increasingly important enablers as the demand for software increases, time-to-market decreases, and diversity and complexity grow beyond any imagined a few decades ago (Ossher et al., 2000). In order to be effective, any software engineering approach needs tool support. Therefore, this chapter presents the tools that support the `Design`, `Dynamic Adaptation`, and `Dynamic Evolution` phases of our framework.

In order to facilitate the comprehension and applicability of the tools, it is necessary to describe them in the context of software process models. As a result, it will be possible to answer the following question: *Which activity of a software process can be supported by a particular tool?* In addition, it is necessary to describe who uses the tools, what are the required inputs, and what are the resulting outputs. To this end, we describe the tools in the context of two Software and Systems Process Engineering Meta-Model (SPEM) 2.0 method contents.

The proposed method contents support the activities in the process models for designing the dynamic adaptation and the dynamic evolution of service compositions at the `Design Phase` of our framework. These process models are described in Chapter 5 and Chapter 6, respectively. The method contents state what is done, when, where, and by whom during the software process. Moreover, in order to facilitate the application of the proposed process models in industry, the method contents are plugable.

The most relevant tool at the `Design Phase` is MOSKitt4SPL[1]. This is a free open-source software for modeling SPLs and DSPLs, which was developed by the PROS. Specifically, it implements the Configuration Generator, which can generate the adaptation space by applying a set of resolutions (i.e., adaptation policies) on a variability model. As described in Chapter 7, the adaptation space can be used at runtime to reason about where to transit to when a context condition has been fulfilled. This is a key aspect because changes in variability model configurations (by means of the activation or deactivation of features) are eventually reflected on the underlying service composition at runtime.

In this chapter, we also provide technical details of MoRE-WS, which is the key tool to support the `Dynamic Adaptation` and the `Dynamic Evolution` phases of our framework. MoRE-WS queries models at runtime to determine how the service composition should change, and provides the mechanisms for modifying the WS-BPEL composition schema accordingly.

The remainder of this chapter is structured as follows. Section 9.1 introduces the plugins that encapsulate the method contents. These method contents describe the tools that support the process models in our framework. Section 9.2 describes the tools in the method content that supports the process model to design the dynamic adaptation of service compositions. Section 9.3 describes the tools in the method content that supports the process model to design the dynamic evolution of service compositions. Section 9.4 describes the technical details of MoRE-WS. Finally, Section 9.5 presents the conclusions.

1 http://www.pros.upv.es/m4spl

9.1 PLUGABLE METHOD CONTENTS

We believe that the successful applicability of any software engineering approach is only possible if it is supported by well-defined process models. Several modeling languages have been proposed to describe software processes (Curtis et al., 1992; Münch et al., 2012). A well-known modeling language is SPEM (OMG, 2008). Specifically, SPEM 2.0 is the OMG standard for defining software and systems development processes and their components (OMG, 2008). The conceptual SPEM framework has two key elements:

The successful applicability of any software engineering approach is only possible if it is supported by well-defined process models.

1. `Method content:` It contains libraries of reusable elements, such as roles, work products, and tools. It defines the "who, what, and how" of work increments that have to be done (Münch et al., 2012). Table 9.1 presents a subset of elements used for method content.

Element	**Description**	**Icon**
Task definition	Defines work being performed by one or many role(s). A task has input and output work products.	▷
Step	Tasks can be divided into steps that describe subunits of work needed to perform the task.	👣
Work product definition	Defines any artifact produced, consumed, or modified by a task.	📄
Role definition	Defines a role and thus related skills, competencies, and responsibilities of one person or many persons. A role is responsible for one or many work product(s) and performs one or many task(s).	👤
Tool Definition	Specifies a tool's participation in a task.	🔨

Table 9.1: Subset of elements used for method content.

2. `Process:` It defines the "when" of work increments that have to be done (Münch et al., 2012). The creation of processes is supported on reusable method contents. Due to UML 2.0 compliance,

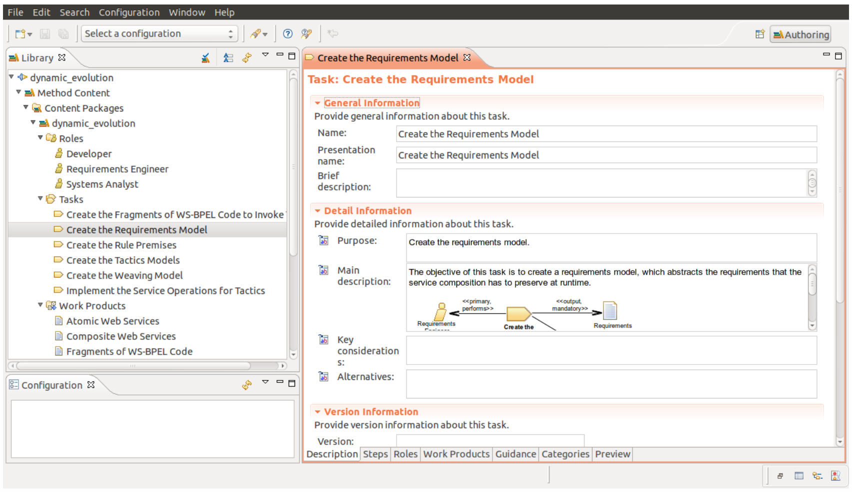
The creation of processes is supported on reusable method contents.

standard UML diagrams such as activity diagrams or state chart diagrams can be used for visualizing processes models.

In Chapters 5 and 6, we propose two process models to engineer autonomic service compositions at the `Design Phase` of our framework. These process models are described as UML activity diagrams with activities that support the design of models and other artifacts for the dynamic adaptation and the dynamic evolution of service compositions in the closed and open worlds, respectively. In order to guide the creation of these process models, we implemented two plugable SPEM-based method contents. One of these ECLIPSE[2] plugins contains the method content that supports the modeling of dynamic adaptations and the other plugin supports the modeling of dynamic evolutions.

Other process models can reference our method-content plugins and reuse their content.

The main benefit of our method-content plugins is that other process models can reference them and reuse their content. These plugins can be downloaded on this thesis website (Alférez and Pelechano, 2013a). Figure 9.2 shows one of these plugins in action. This plugin contains the method content to create the process model for designing the dynamic evolution of service compositions.


File Edit Search Configuration Window Help
Select a configuration
Authoring
Library
Configuration
Create the Requirements Model
Description Steps Roles Work Products Guidance Categories Preview
<<primary, performs>>
<<output, mandatory>>
Requirements
Create the
Requirements

- dynamic_evolution
  - Method Content
    - Content Packages
      - dynamic_evolution
        - Roles
          - Developer
          - Requirements Engineer
          - Systems Analyst
        - Tasks
          - Create the Fragments of WS-BPEL Code to Invoke
          - Create the Requirements Model
          - Create the Rule Premises
          - Create the Tactics Models
          - Create the Weaving Model
          - Implement the Service Operations for Tactics
        - Work Products
          - Atomic Web Services
          - Composite Web Services
          - Fragments of WS-BPEL Code

Task: Create the Requirements Model

General Information
Provide general information about this task.
Name: Create the Requirements Model
Presentation name: Create the Requirements Model
Brief description:

Detail Information
Provide detailed information about this task.
Purpose: Create the requirements model.
Main description: The objective of this task is to create a requirements model, which abstracts the requirements that the service composition has to preserve at runtime.
Key considerations:
Alternatives:

Version Information
Provide version information about this task.
Version:

Figure 9.2: A method-content plugin to create the process model for designing the dynamic evolution of service compositions.

2 http://www.eclipse.org

The Eclipse Process Framework (EPF) Composer[3] was used to create the aforementioned plugins. EPF aims at producing a customizable software process engineering framework, with exemplary process content and tools, supporting a broad variety of project types and development styles. To this end, the EPF Project created the EPF Composer, an open-source tool platform designed for process engineers and project managers to author, tailor, and publish methods and processes for development organizations and projects.

In addition, the EPF Composer was used to create two websites that describe the method contents to design dynamic adaptations and dynamic evolutions of service compositions. Hyperlinks to these websites are available on (Alférez and Pelechano, 2013a).

9.2 DESIGNING DYNAMIC ADAPTATIONS

In this section, we describe the tools in the method content to support the creation of the process model to design dynamic adaptations of service compositions in the closed world. Section 5.2 describes this process model.

This section is organized by tasks that support the creation of the work products to guide dynamic adaptations. Each task has a one-to-one mapping to each activity in the process model depicted in Figure 5.4. Therefore, the information in each task supports the creation of each activity in the process model. The description of each task contains the information about the tools that are used to create work products.

Each task has a one-to-one mapping to each activity in the process model.

Among the proposed tools, MOSKitt4SPL is the most relevant tool to design the dynamic adaptation of service compositions. MOSKitt4SPL is free and open-source, is built on ECLIPSE, and gives support for modeling DSPLs at the `Design Phase` of our framework. MOSKitt4SPL is based on MOSKITT[4]. MOSKitt is a free modeling platform that is being developed by the Valencian Regional Ministry of Infrastructure and Transport. MOSKitt4SPL was developed using the Eclipse Modeling Framework (EMF)[5], the Graphical Modeling Framework (GMF)[6], and the ATL Transformation Language (ATL)[7]. MOSKitt4SPL is composed of the following plugins (see Figure 9.3):

3 http://www.eclipse.org/epf
4 http://www.moskitt.org
5 http://www.eclipse.org/modeling/emf
6 http://wiki.eclipse.org/GMF
7 http://www.eclipse.org/atl

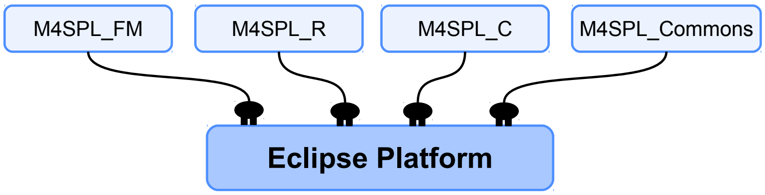


Figure 9.3: MOSKitt4SPL plugins.

- **`M4SPL_FM`:** This plugin manages the feature model and its customization. It contains two packages. First, the `FeatureModelCustomization` package contains the classes to manage the customization of the feature model in terms of features' and relationships' notations. Second, the `FeatureModelPackage` contains the classes to manage the feature model (e.g. the `Feature` and `Mandatory` classes).

- **`M4SPL_R`:** This plugin manages resolutions and context conditions. The `State` class represents the possible states in the adaptation space. The `Transition` class represents the possible transitions (resolutions) in the adaptation space. The `StateMachineModel` class represents the state machine model, which abstracts the adaptation space.

- **`M4SPL_C`:** This plugin manages the configurations of the variability model. This plugin contains two key classes. First, the `FeatureState` class offers getter and a setter methods to get and set information about feature states. Second, the `FeatureStateType` class manages the possible feature states.

- **`M4SPL_Commons`:** This plugin offers common functionalities such as perspective management in Eclipse and a wizard to create models.

Most models are specified in the XMI format to execute queries against them at runtime.

All the models presented in the following sections, with the exception of the context model, are specified in the XMI format in order to execute queries against them at runtime. The main advantage of XMI models in our approach is to avoidthe definition of technological bridges, because the same technologies used at design time for manipulating XMI models can be applied at runtime.

9.2.1 *Create the Initial Composition Model*

The objective of this task is to create the initial composition model that abstracts the underlying service composition (see Figure 9.4). The initial composition model can be created with the SOA Tools Platform (STP) BPMN Modeler[8]. The main goal of this tool is to provide a graphical notation that allows to define BPs. This tool has been developed with GMF.

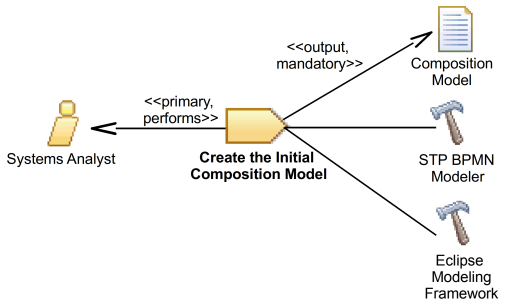


Figure 9.4: Create the Initial Composition Model task.

Composition models created with the STP BPMN Modeler rely on the BPMN metamodel defined by STP[9]. The STP metamodel covers almost all BPMN 1.0 shapes, connections, and markers except the layouts and appearance of the lanes inside a pool and the group-artifact. Specifically, our composition model is based on the following fundamental set of BPMN elements: 1) start and end events; 2) task and subprocess activities; 3) exclusive, parallel, and inclusive gateways; 4) sequence flows; and 5) pools.

Our composition model is based on a set of fundamental BPMN elements.

The STP Project provides a functional editor for BPMN diagrams (see Figure 9.5). Nevertheless, it is also possible to create composition models directly on the infrastructure provided by EMF. These diagrams conform to the BPMN metamodel provided by STP. For instance, Figure 9.6 shows the composition model for our case study in the EMF Generic Editor. Since the metamodel dictates that BPMN elements need to be contained in a *pool*, a pool contains all the elements of this model (see the `OrangeCountryBookStore` pool). The way to choose to create composition models depends on the knowledge that systems analysts have about these tools.

8 http://git.eclipse.org/c/bpmnmodeler
9 http://wiki.eclipse.org/STP

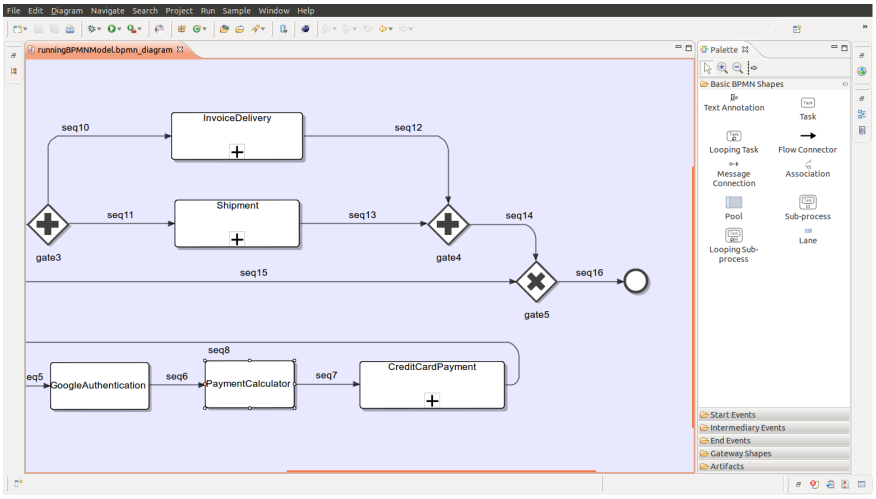
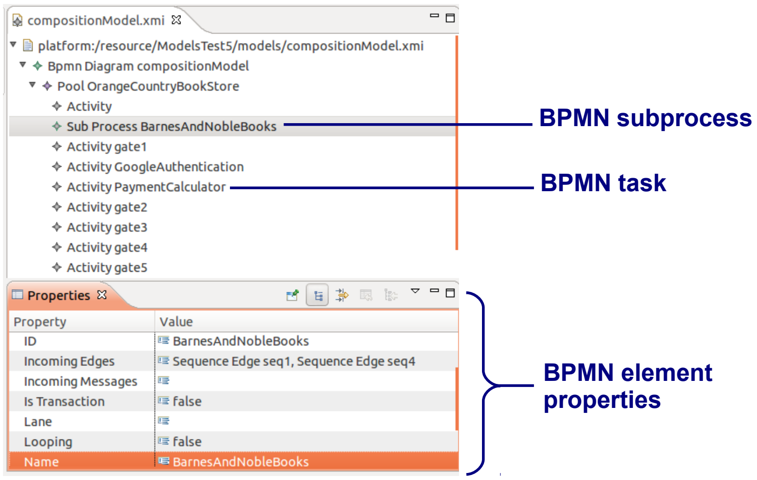


Figure 9.5: STP BPMN Modeler.



compositionModel.xmi

- platform:/resource/ModelsTest5/models/compositionModel.xmi
  - Bpmn Diagram compositionModel
    - Pool OrangeCountryBookStore
      - Activity
      - Sub Process BarnesAndNobleBooks
      - Activity gate1
      - Activity GoogleAuthentication
      - Activity PaymentCalculator
      - Activity gate2
      - Activity gate3
      - Activity gate4
      - Activity gate5

Properties

Property	Value
ID	BarnesAndNobleBooks
Incoming Edges	Sequence Edge seq1, Sequence Edge seq4
Incoming Messages	
Is Transaction	false
Lane	
Looping	false
Name	BarnesAndNobleBooks

Figure 9.6: Initial composition model for our case study in XMI format.

9.2.2 *Create the Variability Model*

The objective of this task is to create the variability model that abstracts the variability of the service composition (see Figure 9.7). The variability model can be described as a feature model with MOSKitt4SPL.

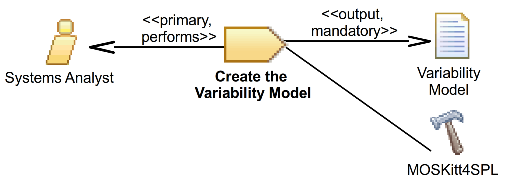


Figure 9.7: Create the Variability Model task.

Feature models are based on the feature model metamodel defined in (Schobbens et al., 2007). The graphical notations used to represent the elements in a feature model can be changed easily in the `Feature Model Editor` of MOSKitt4SPL. This editor is very convenient when dealing with large feature models. Specifically, the `Feature Model Editor` supports the customization of the following feature notations:

- **`Feature with Attributes:`** Features are graphically represented by means of rectangles. These rectangles are composed of two compartments. The top compartment holds the feature name and the bottom compartment holds the features attributes.

- **`Rounded Feature:`** Features are graphically represented by means of ellipses. The feature name is at the ellipse center, whereas feature attributes are not shown.

- **`Fixed Feature:`** Features are represented as rounded features. The diameter depends of the length of the feature name.

- **`Simplified Feature:`** Features are graphically represented by means of ellipses. Neither the feature name is visible, nor the feature attributes. The ellipse diameter is constant.

MOSKitt4SPL also supports the customization of the following relationship notations:

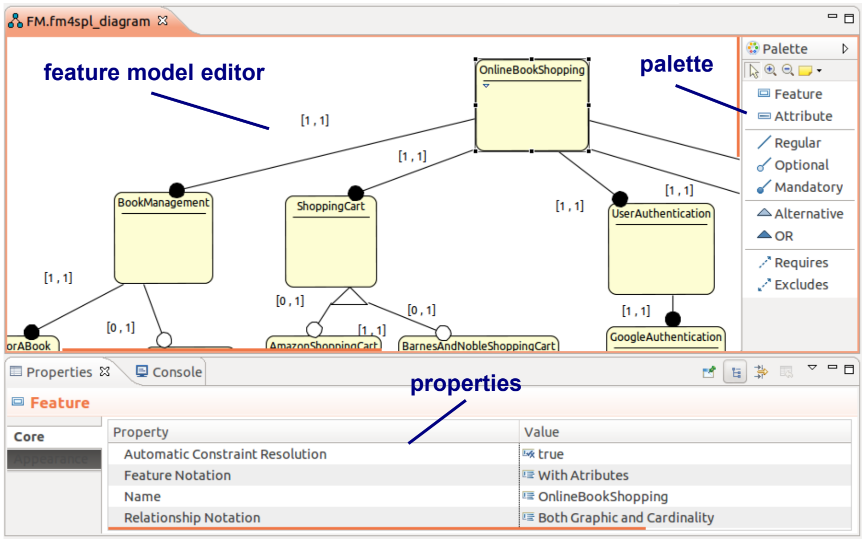
- **`Simplified Relationship:`** Relationships are represented with lines.

- **`Graphic Relationship:`** Relationships are represented with decorated lines.

- **`Cardinality Relationship:`** Relationships are represented with lines and a floating label.

- **`Cardinality-Graphic Relationship:`** Relationships are represented with decorated lines and a floating label. The line decoration indicates the type of relationship. Optional relationships are decorated with a white ellipse and mandatory relationships are decorated with a black ellipse. The label follows the pattern `[min, max]` to indicate the minimum and maximum cardinality of the relationship. Both label and decoration are synchronized between them.

The elements in the feature model can be reorganized in a graphical tree. The element at the top is the root of the tree and the elements at the bottom are the leaves, which are organized by levels. For example, Figure 9.8 shows the MOSKitt4SPL `Feature Model Editor` in action when the variability model in our case study was created. It is possible to choose in the `Properties` view among different types of feature notations and relationship notations. MOSKitt4SPL automatically updates the XMI representation of the feature model.



Property	Value
Automatic Constraint Resolution	true
Feature Notation	With Atributes
Name	OnlineBookShopping
Relationship Notation	Both Graphic and Cardinality

Figure 9.8: MOSKitt4SPL Feature Model Editor.

MOSKitt4SPL also has the `Feature Model Configurator`, which allows to define the initial configuration of the variability model. This configuration defines the set of *active* and *inactive* features when the service composition starts to run. For example, Figure 9.9 shows the features that have been set to active and inactive in our case study. A floating window that appears when clicking on each feature offers the option to activate/deactivate it.

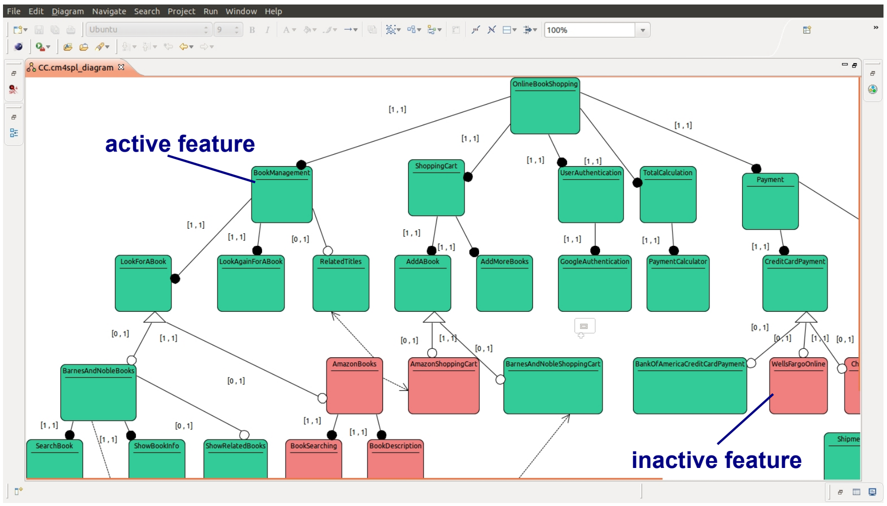


Figure 9.9: Initial variability model configuration in our case study.

9.2.3 *Set Variability at the Composition Model*

The objective of this task is to extend the initial composition model with variability constructs. To this end, we propose the creation of the following models: 1) the *base composition model* extends the initial composition model with semantics for variability. This model preserves the main workflow of the service composition during adaptations. To this end, it defines a set of commonalities and variation points;

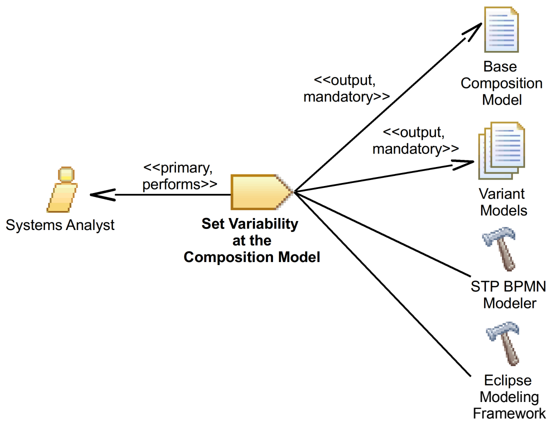


Figure 9.10: Set Variability at the Composition Model task.

and 2) a set of *variant models* can be used to inject variant behavior on the base composition model at runtime. To this end, they can be bound into the variation points of the base composition model during execution (see Figure 9.10).

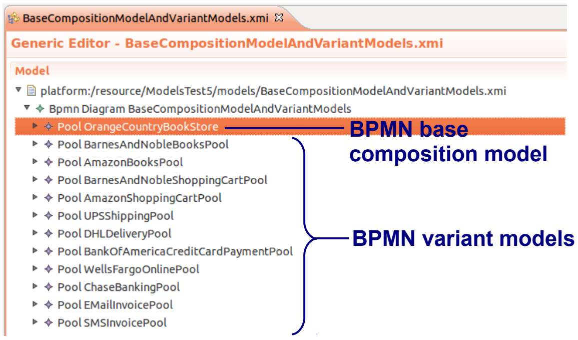
The base composition model and the variant models, which are expressed as composition models, can be created with the STP BPMN Modeler or with the infrastructure provided by EMF. Section 9.2.1 presents the details about these tools.

Specifically, we propose to create one BPMN model with two kinds of pools that contain these models:

1. A *pool* that contains the base composition model. This pool contains events, activities, gateways, sequence flows among activities, and subprocesses that represent variation points.

2. Additional *pools* for each variant model. A BPMN subprocess inside a pool encapsulates the elements in a variant model. Each pool contains at most one high-level subprocess. Nevertheless, this subprocess can contain low-level subprocesses.

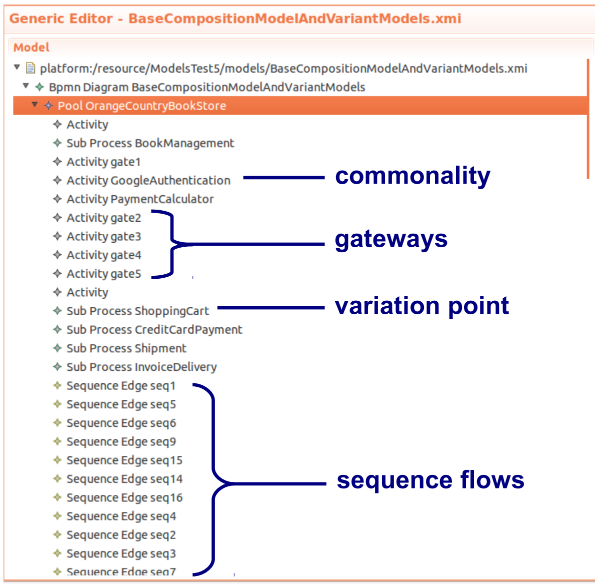
For instance, the `OrangeCountryBookStore` pool in Figure 9.11 contains the base composition model and the other pools contain the variant models (e.g. `AmazonBooksPool`).


BaseCompositionModelAndVariantModels.xmi
Generic Editor - BaseCompositionModelAndVariantModels.xmi
Model
BPMN base composition model
BPMN variant models

```
▾ platform:/resource/ModelsTest5/models/BaseCompositionModelAndVariantModels.xmi
  ▾ Bpmn Diagram BaseCompositionModelAndVariantModels
    ▸ Pool OrangeCountryBookStore
    ▸ Pool BarnesAndNobleBooksPool
    ▸ Pool AmazonBooksPool
    ▸ Pool BarnesAndNobleShoppingCartPool
    ▸ Pool AmazonShoppingCartPool
    ▸ Pool UPSShippingPool
    ▸ Pool DHLDeliveryPool
    ▸ Pool BankOfAmericaCreditCardPaymentPool
    ▸ Pool WellsFargoOnlinePool
    ▸ Pool ChaseBankingPool
    ▸ Pool EMailInvoicePool
    ▸ Pool SMSInvoicePool
```

Figure 9.11: Collapsed base composition model and variant models.

Figure 9.12 shows the expanded pool that contains the elements in the base composition model in our case study.


Generic Editor - BaseCompositionModelAndVariantModels.xmi

Model

- platform:/resource/ModelsTest5/models/BaseCompositionModelAndVariantModels.xmi
  - Bpmn Diagram BaseCompositionModelAndVariantModels
    - Pool OrangeCountryBookStore
      - Activity
      - Sub Process BookManagement
      - Activity gate1
      - Activity GoogleAuthentication
      - Activity PaymentCalculator
      - Activity gate2
      - Activity gate3
      - Activity gate4
      - Activity gate5
      - Activity
      - Sub Process ShoppingCart
      - Sub Process CreditCardPayment
      - Sub Process Shipment
      - Sub Process InvoiceDelivery
      - Sequence Edge seq1
      - Sequence Edge seq5
      - Sequence Edge seq6
      - Sequence Edge seq9
      - Sequence Edge seq15
      - Sequence Edge seq14
      - Sequence Edge seq16
      - Sequence Edge seq4
      - Sequence Edge seq2
      - Sequence Edge seq3
      - Sequence Edge seq7

Figure 9.12: Fragment of the expanded BPMN pool that contains the elements of the base composition model.

Figure 9.13 shows the expanded BPMN pool that contains the Chase-BankingVar BPMN subprocess. This subprocess encapsulates the variant model elements for `Chase Banking`.

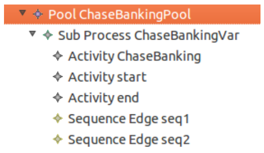

- Pool ChaseBankingPool
  - Sub Process ChaseBankingVar
    - Activity ChaseBanking
    - Activity start
    - Activity end
    - Sequence Edge seq1
    - Sequence Edge seq2

Figure 9.13: Expanded BPMN pool for the `Chase Banking` variant model.

9.2.4 *Create the Context Model*

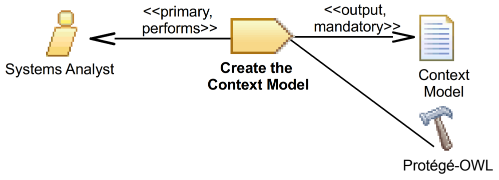


Figure 9.14: Create the Context Model task.

The objective of this task is to create a context model that abstracts the collected context information. Specifically, we propose the creation of an ontology-based context model that leverages Semantic Web technology (see Figure 9.14). The context model can be created with PROTÉGÉ-OWL[10], which is an extension of PROTÉGÉ[11] for OWL. PROTÉGÉ-OWL is tightly integrated with JENA[12], a JAVA framework for building Semantic Web applications. Figure 9.15 shows the graphical interface of PROTÉGÉ-OWL in our case study.

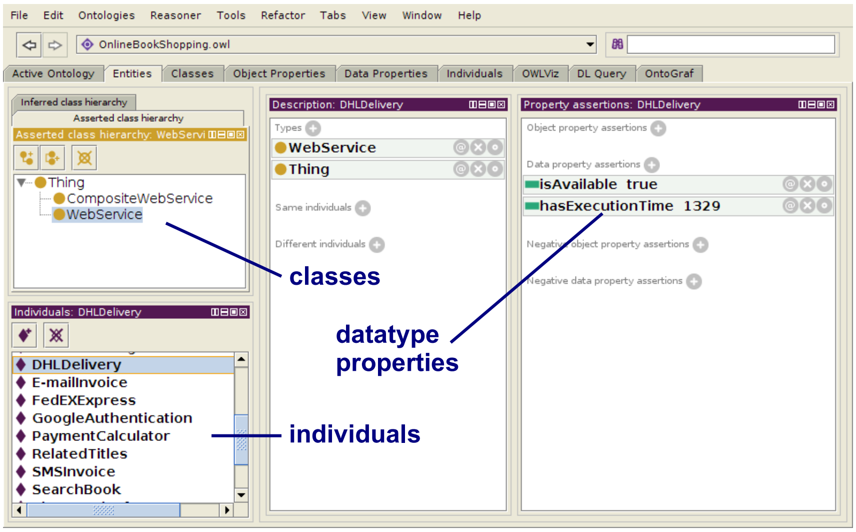


Figure 9.15: Graphical interface of PROTÉGÉ-OWL.

10 http://protege.stanford.edu/overview/protege-owl.html
11 http://protege.stanford.edu
12 http://jena.apache.org

9.2.5 *Define Context Conditions*

The objective of this task is to define context conditions that indicate the compliance to certain situations in the context. Context conditions are extracted from the context model work product as Boolean expressions (see Figure 9.16). The `Resolution Editor` of MOSKitt4SPL allows to define context conditions. For example, Figure 9.18 shows the definition of the the `B&NUnavailable` context condition, which is fulfilled when the `Barnes & Noble Books` composite service is currently unavailable.

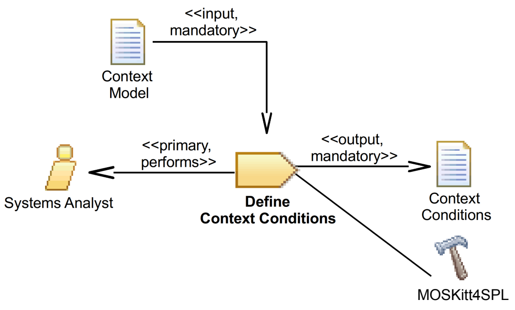


Figure 9.16: Define Context Conditions task.

9.2.6 *Define Resolutions*

The objective of this task is to define the resolutions that represent the set of changes (i.e., activation and deactivation of features) in a feature model triggered by context conditions (see Figure 9.17). The resolutions can be defined in the `Resolutions Editor` of MOSKitt4SPL. For example, Figure 9.18 shows the definition of a resolution to be triggered when the `B&NUnavailable` context condition is fulfilled. *Actions* in this figure express state change of features in the feature model.

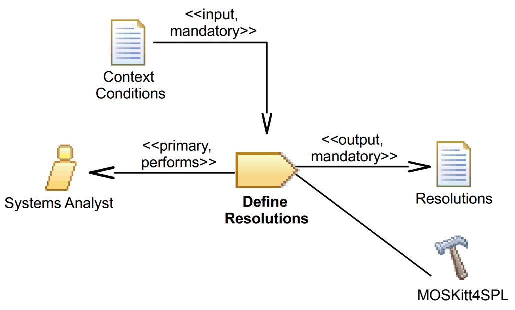


Figure 9.17: Define Resolutions task.

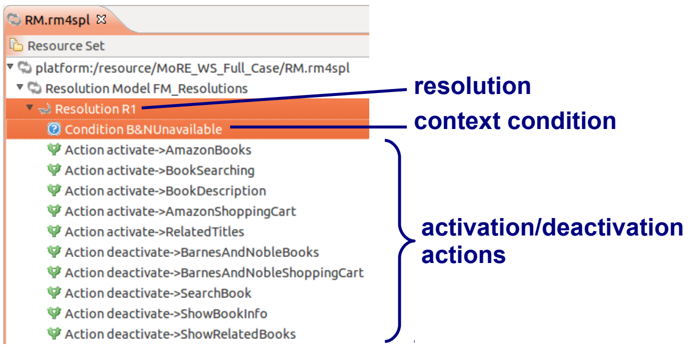

RM.rm4spl
Resource Set
platform:/resource/MoRE_WS_Full_Case/RM.rm4spl
Resolution Model FM_Resolutions
Resolution R1
Condition B&NUnavailable
Action activate->AmazonBooks
Action activate->BookSearching
Action activate->BookDescription
Action activate->AmazonShoppingCart
Action activate->RelatedTitles
Action deactivate->BarnesAndNobleBooks
Action deactivate->BarnesAndNobleShoppingCart
Action deactivate->SearchBook
Action deactivate->ShowBookInfo
Action deactivate->ShowRelatedBooks

Figure 9.18: Resolutions for context conditions in MOSKitt4SPL.

9.2.7 *Generate the Adaptation Space*

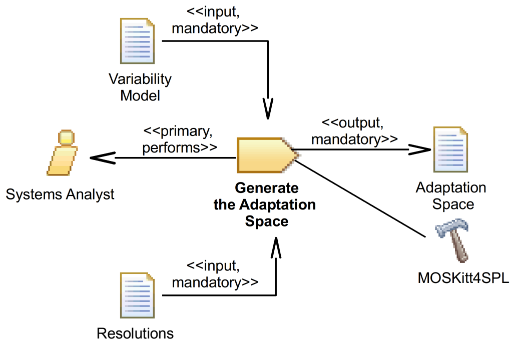


Figure 9.19: Generate the Adaptation Space task.

The objective of this task is to support the automatic generation of the adaptation space (see Figure 9.19). The execution of a DSPL can be abstracted as a highly-connected state machine where the states are the possible service composition configurations and the transitions the migration paths.

MOSKitt4SPL fully generates the implicit adaptation space in form of a state machine. To this end, it uses the variability model and the set of resolutions for particular context conditions.

Just after the adaptation space is generated, MOSKitt4SPL displays the statistics about the time employed to create the state machine and the number of generated states and transitions. Users can select a particular state in the adaptation space to see the current variability model configuration. Moreover, graphical representations of the configurations for particular states can be automatically generated with MOSKitt4SPL.

Figure 9.20 shows the generated adaptation space for our case study and nine resolutions (circles represent variability model configurations and arrows transitions among configurations). MOSKitt4SPL generated an adaptation space with 40 variability model configurations and 360 transitions among configurations in less than four seconds.

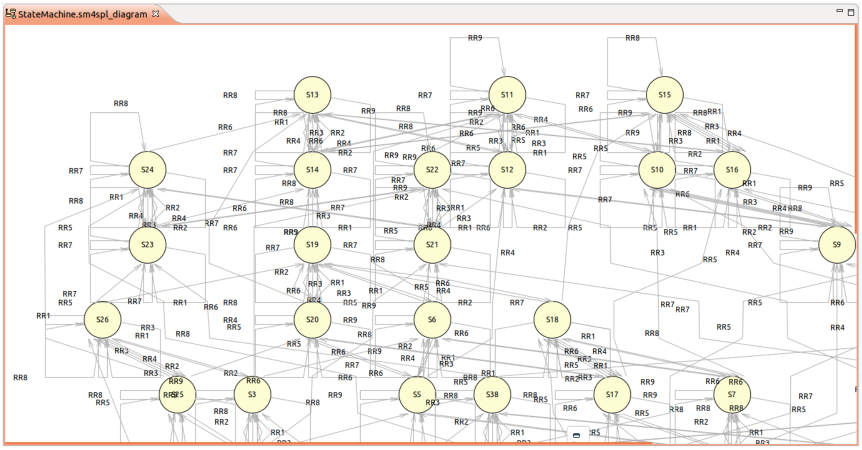


Figure 9.20: Adaptation space for our case study generated by MOSKitt4SPL.

9.2.8 *Link Features to Service Operations*

The objective of this task is to link features in the variability model to: 1) variation points and commonalities in the base composition model, and 2) BPMN subprocesses that encapsulate variant models. The weaving model can be created with the ATLAS Model Weaver (AMW)[13], a tool for establishing relationships or links between models.

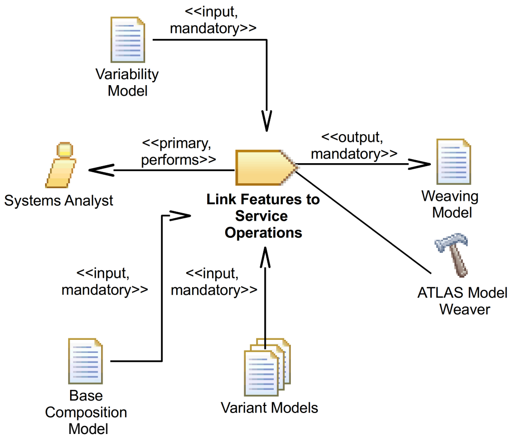


Figure 9.21: Link Features to Service Operations task.

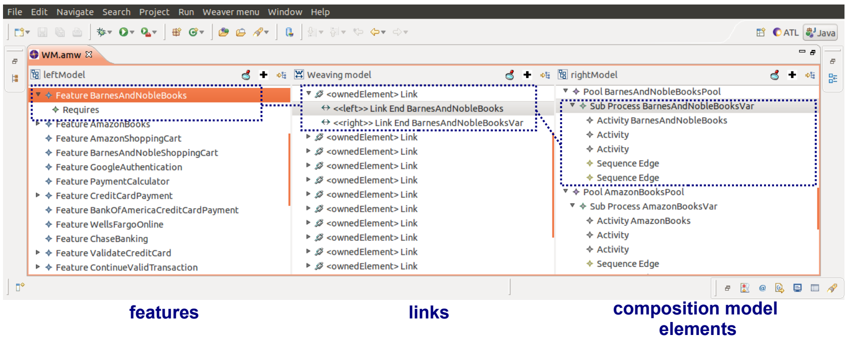
The weaving model is created conforming to a weaving metamodel.

13 http://www.eclipse.org/gmt/amw

This metamodel supports basic link management, i.e., `n:n` relationships between model elements. The identifiers of the referred elements can be saved using different identification methods.

The AMW tool supports only one source model and only one destination model. Therefore, at the implementation level we propose to map the feature model (which implements the variability model) to a unique XMI file that encapsulates the base composition model and the set of variant models.

For instance, Figure 9.22 shows a fragment of the weaving model in AMW that connects the features in the variability model and the variation points and commonalities in the base composition model, and variant models in our case study. It highlights the relationship between the `Barnes & Noble Books` feature and a set of BPMN elements in the `Barnes & Noble Books` variant model (encapsulated in a BPMN subprocess).



leftModel
- Feature BarnesAndNobleBooks
  - Requires
- Feature AmazonBooks
- Feature AmazonShoppingCart
- Feature BarnesAndNobleShoppingCart
- Feature GoogleAuthentication
- Feature PaymentCalculator
- Feature CreditCardPayment
- Feature BankOfAmericaCreditCardPayment
- Feature WellsFargoOnline
- Feature ChaseBanking
- Feature ValidateCreditCard
- Feature ContinueValidTransaction

Weaving model
- <ownedElement> Link
  - <<left>> Link End BarnesAndNobleBooks
  - <<right>> Link End BarnesAndNobleBooksVar
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link
- <ownedElement> Link

rightModel
- Pool BarnesAndNobleBooksPool
  - Sub Process BarnesAndNobleBooksVar
    - Activity BarnesAndNobleBooks
    - Activity
    - Activity
    - Sequence Edge
    - Sequence Edge
- Pool AmazonBooksPool
  - Sub Process AmazonBooksVar
    - Activity AmazonBooks
    - Activity
    - Activity
    - Sequence Edge

Figure 9.22: A fragment of a weaving model in the case study using AMW.

9.2.9 *Verify Reconfigurations*

The objective of this task is to verify the variability model and its possible configurations at design time. Configurations are described in the adaptation space (see Figure 9.23). The variability model and its configurations can be verified with the VERIFIER. As mentioned in Section 5.2.9, the VERIFIER is not attached to any particular implementation. Therefore, the implementation depends on the desirable verification criteria to be applied.

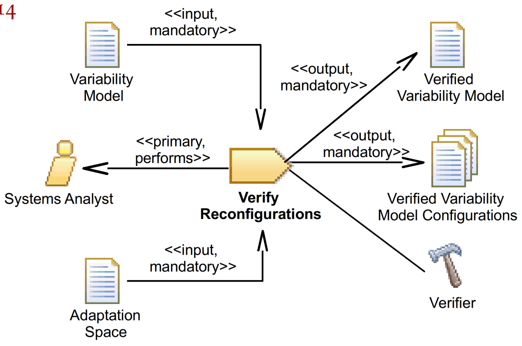
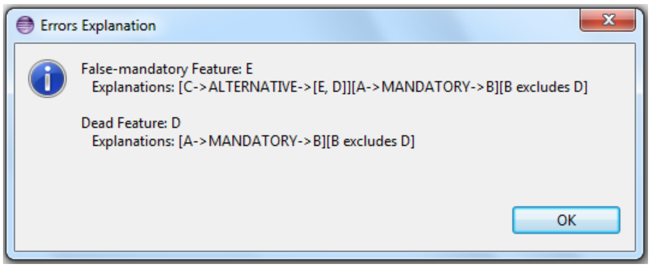


Figure 9.23: Verify Reconfigurations task.

We have chosen FAMA-FW[14] to carry out the generic verification operations (void feature model, right configuration, and anomalies detection). FAMA-FW is a framework for automated analyses of feature models. It integrates some of the most commonly used logic representations and solvers proposed in literature (Binary Decision Diagrams (BDD), Satisfiability (SAT), and CSP solvers).

MOSKitt4SPL supports these operations by invoking the functionality provided by FAMA-FW. The website of MOSKitt4SPL describes these operations in detail[15]. Figure 9.24 shows a screenshot of the explanation of two errors in one configuration of the variability model on MOSKitt4SPL, which uses the functionality of FAMA-FW.



False-mandatory Feature: E
Explanations: [C->ALTERNATIVE->[E, D]][A->MANDATORY->B][B excludes D]

Dead Feature: D
Explanations: [A->MANDATORY->B][B excludes D]

Figure 9.24: Error explanation in one configuration of the variability model on MOSKitt4SPL.

We have implemented the semi-aliveness and stability verification operations with GNU PROLOG[16]. GNU PROLOG is a programming language that includes a powerful constraint solver over finite domains. It uses similar algorithms as other CP solvers and the unification operation to work with PROLOG facts. GNU PROLOG uses very efficient

14 http://www.isa.us.es/fama
15 http://www.pros.upv.es/m4spl
16 http://gprolog.univ-paris1.fr

algorithms based on consistency, propagation, and backtracking techniques. The non-determinism used by GNU Prolog allows to calculate several solutions for the same constraint problem and obtain the best solution (based on several criteria, such as computing time, number of backtracks, and a maximization function) that satisfies the constraints.

9.2.10 *Model-Driven Generation of WS-BPEL Code*

The objective of this task is to generate the WS-BPEL code that is going to be used during dynamic adjustments. Specifically, a WS-BPEL template is created to be instantiated at runtime with a set of WS-BPEL code fragments. The generation of the WS-BPEL template is model-driven (see Figure 9.25).

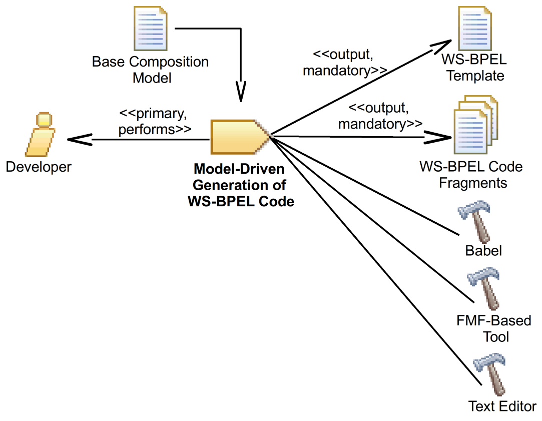


Figure 9.25: Model-Driven Generation of WS-BPEL task.

A partial WS-BPEL template can be generated with Babel[17] from the base composition model. Our FMF-based configuration tool can be used to manually fulfill the missing parts in the WS-BPEL code generated by Babel. Section 5.2.10 introduces these tools. Since the code in WS-BPEL code fragments is short (i.e., it consists on invoking variant service operations), a text editor can be used to create these fragments.

9.3 DESIGNING DYNAMIC EVOLUTIONS

In this section, we describe the elements in the method content for the creation of the process model to design dynamic evolutions of service compositions in the open world. Section 6.4 describes this process model.

This section is organized by tasks to create the work products to guide dynamic evolutions. These tasks map to the activities in the

17 http://www.bpm.scitech.qut.edu.au/research/projects/oldprojects/babel/tools

process model to design dynamic evolutions (this process model is depicted in Figure 6.5). In fact, there is a one-to-one mapping between most of the tasks and activities. The description of each task contains the information about the work products and the tools that are used to create these work products. All the models described in the following sections are specified in the XMI format in order to execute queries against them at runtime.

9.3.1 *Create the Requirements Model*

The objective of this task is to create a requirements model, which abstracts the requirements that the service composition has to preserve at runtime (see Figure 9.26). The requirements engineer role is in charge of creating the requirements model work product.

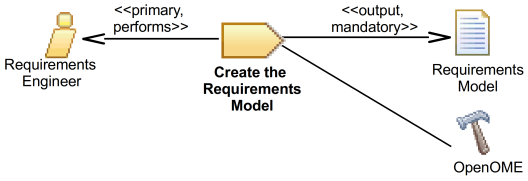


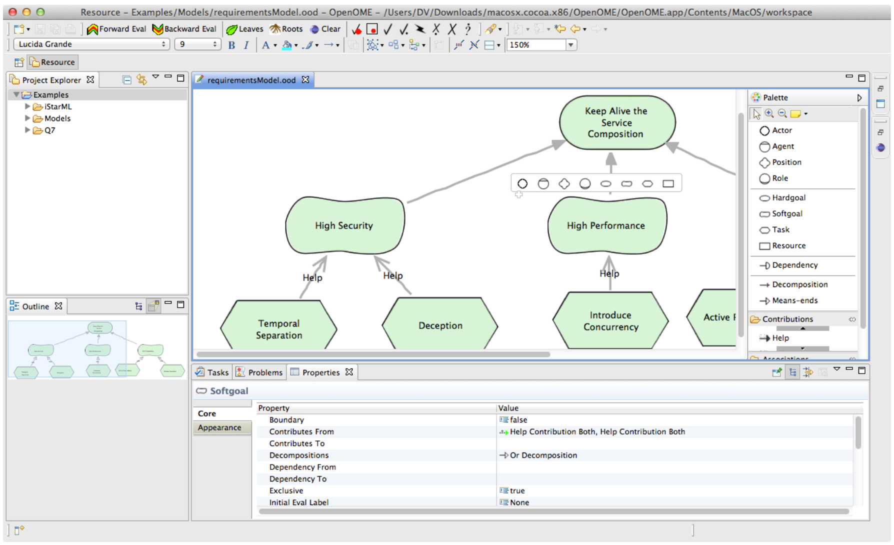
Figure 9.26: Create the Requirements Model task.

We choose OpenOME[18] to create the requirements model because it offers an open metamodel that can be used to create XMI-based requirements models. OpenOME is an open source, general, goal-oriented and/or agent-oriented modeling and analysis tool. It provides users with a graphical interface to develop models, and supports access to a knowledge base that allows for sophisticated computer-aided analysis. OpenOME supports the *i** framework. Figure 9.27 shows the graphical interface of OpenOME.

In our approach, we manage four kinds of OpenOME contributions: 1) the `MakeContribution` is a positive contribution strong enough to satisfice a softgoal (i.e., ++); 2) the `HelpContribution` is a partial positive contribution, not sufficient by itself to satisfy the softgoal (i.e., +); 3) the `BreakContribution` is a negative contribution sufficient enough to deny a softgoal (i.e., - -); and 4) the `HurtContribution` is a partial negative contribution, not sufficient by itself to deny the softgoal (i.e., -).

Although the OpenOME metamodel contains the `Belief` metaclass, its graphical interface misses the representation for claims (or beliefs).

18 https://se.cs.toronto.edu/trac/ome



Property	Value
Boundary	false
Contributes From	Help Contribution Both, Help Contribution Both
Contributes To	
Decompositions	Or Decomposition
Dependency From	
Dependency To	
Exclusive	true
Initial Eval Label	None

Figure 9.27: OpenOME graphical interface.

Therefore, it is necessary to use the EMF generic editor in order to add claims into the requirements model.

9.3.2 *Create the Tactic Models*

The objective of this task is to create a set of tactic models. These models express the tactical functionality to be triggered on the service composition to preserve requirements when facing unknown context events. For each tactic model expressed as a feature model, there is a tactic model expressed as a composition model (see Figure 9.28). The systems analyst role is in charge of creating the tac-

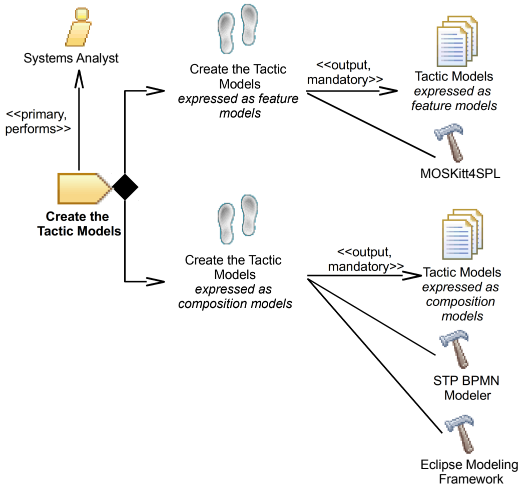

Systems Analyst
<<primary, performs>>
Create the Tactic Models
Create the Tactic Models *expressed as feature models*
<<output, mandatory>>
Tactic Models *expressed as feature models*
MOSKitt4SPL
Create the Tactic Models *expressed as composition models*
<<output, mandatory>>
Tactic Models *expressed as composition models*
STP BPMN Modeler
Eclipse Modeling Framework

Figure 9.28: Create the Tactics Models task.

tic models work products, both expressed as feature models and composition models. This tactic is composed of the following two steps, which can be carried out in parallel:

- *Step 1: Create the Tactic Models Expressed as Feature Models:* This step supports the creation of tactic models expressed as feature models. These models describe the tactical functionality in terms of features to be added into the evolved variability model when facing unknown context events.

  Feature models can be created with MOSKitt4SPL as described in Section 9.2.2. This step maps to the *Create the Tactic Models (Expressed as Feature Models)* activity in the process model to design dynamic evolutions in Figure 6.5.

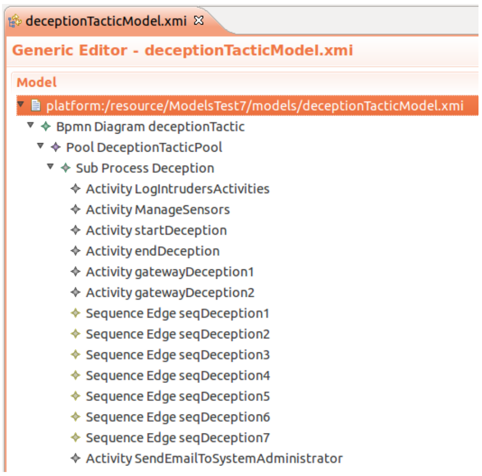
- *Step 2: Create the Tactic Models Expressed as Composition Models:* This step supports the creation of tactic models expressed as composition models. These models describe the workflow among service operations in tactics.

  The set of tactic models can be created with the STP BPMN Modeler or with the infrastructure provided by EMF. Section 9.2.1 presents the details about these tools. This step maps to the *Create the Tactic Models (Expressed as Composition Models)* activity in the process model to design dynamic evolutions in Figure 6.5.

For instance, Figure 9.29 describes the EMF deception tactic model (expressed as a composition model in XMI format) in our case study. Since the STP metamodel dictates that model elements need to be contained in a pool, `DeceptionTacticPool` was created to contain a subprocess with all the elements of this model.

9.3.3 *Create the Weaving Model*

The objective of this task is to create a weaving model to map the features in each tactic model (expressed as a feature model), and the BPMN elements in tactic models (expressed as composition models) (see Figure 9.30). A weaving model reflects the activation of features in tactic models, which are expressed as feature models, on composition models that abstract the workflow among service operations in these tactics.



deceptionTacticModel.xmi

Generic Editor - deceptionTacticModel.xmi

Model

- platform:/resource/ModelsTest7/models/deceptionTacticModel.xmi
  - Bpmn Diagram deceptionTactic
    - Pool DeceptionTacticPool
      - Sub Process Deception
        - Activity LogIntrudersActivities
        - Activity ManageSensors
        - Activity startDeception
        - Activity endDeception
        - Activity gatewayDeception1
        - Activity gatewayDeception2
        - Sequence Edge seqDeception1
        - Sequence Edge seqDeception2
        - Sequence Edge seqDeception3
        - Sequence Edge seqDeception4
        - Sequence Edge seqDeception5
        - Sequence Edge seqDeception6
        - Sequence Edge seqDeception7
        - Activity SendEmailToSystemAdministrator

Figure 9.29: Deception tactic model in XMI format.

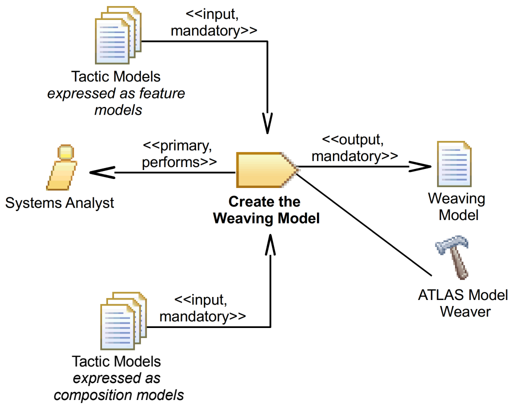


Figure 9.30: Create the Weaving Model task.

Each link in this weaving model has the following endpoints: the first endpoint refers to features in all the tactic models that are expressed as feature models; the second endpoint refers to elements in all the tactic models that are expressed as composition models.

The weaving models can be created with AMW as described in Section 9.2.8. The AMW tool supports only one source model and only one destination model. Therefore, at the implementation level we propose to map a unique XMI file that encapsulates the set of tactic models, expressed as feature models, to a unique XMI file that encapsulates the tactic models, expressed as composition models.

9.3.4 *Implement the Service Operations for Tactics*

The objective of this task is to implement the service operations for each tactic (see Figure 9.31). The developer role is in charge of creating the atomic and composite Web services that implement the tactics.

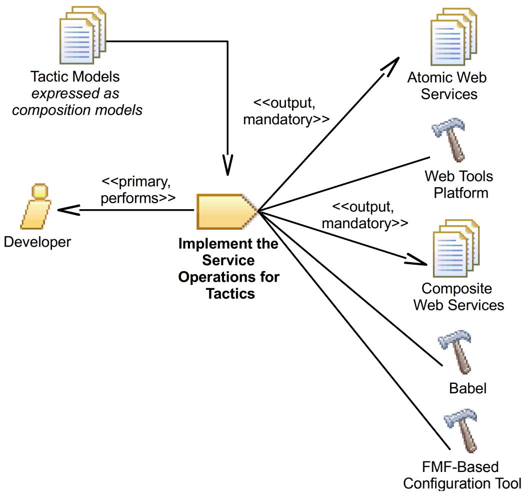


Figure 9.31: Implement the Service Operations for Tactics task.

Some tactics can be implemented as atomic Web services (their implementation is self contained and does not invoke any other services). Different tools (both free and commercial) can be used to facilitate the creation of atomic Web services. We propose the use of the WEB SERVICES PROJECT[19], a subproject in the ECLIPSE WEB TOOLS PLATFORMS PROJECT[20] to implement atomic Web services.

The WEB SERVICES PROJECT has two components. First, the JST WEB SERVICES COMPONENT contains tools for developing and interacting with JAVA Web services. Second, the WST WEB SERVICES COMPONENT contains tools for Web services development which is not JAVA specific. A demonstration of the WEB SERVICES PROJECT is available online[21]. In this demonstration, a top-down approach is followed to create an atomic Web service (i.e., the Web service is created from a WSDL file).

Other tactics can be implemented as composite Web services. These tactics are more complex than the ones implemented as atomic Web services because they can invoke several service operations. In the case of tactics that are based on composite Web services, BABEL and our FMF-based configuration tool can be used to generate WS-BPEL code. First, BABEL can partially generate the WS-BPEL code from tactic models expressed as BPMN composition models. Afterwards, our FMF-based

19 http://www.eclipse.org/webtools/ws
20 http://www.eclipse.org/webtools
21 http://www.eclipse.org/webtools/tutorial

configuration tool can be used to complete the WS-BPEL code. Section 5.2.10 describes these tools.

9.3.5 *Create the Fragments of WS-BPEL Code to Invoke Tactics*

The objective of this task is to create the fragments of WS-BPEL code that will invoke the service operations implementing the tactics (see Figure 9.32). The developer role is in charge of creating the fragments of WS-BPEL code to invoke tactics. Each tactic model maps to a WS-BPEL code fragment. Each WS-BPEL code fragment has an `<invoke>` instruction, which invokes the service operation implementing the tactic. At runtime, invocation instructions are put into the WS-BPEL composition schema in order to evolve the service composition.

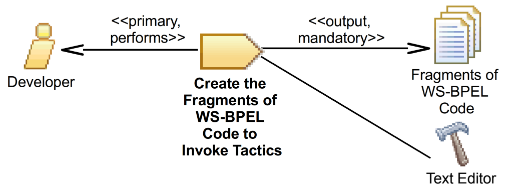

Developer
<<primary, performs>>
Create the Fragments of WS-BPEL Code to Invoke Tactics
<<output, mandatory>>
Fragments of WS-BPEL Code
Text Editor

Figure 9.32: Create the Fragments of WS-BPEL Code to Invoke Tactics task.

A text editor can be used to implement the fragments of WS-BPEL code that invoke service operations implementing tactics. We propose to store each invocation instruction in a separated text file. Nevertheless, other approaches can also be followed to implement fragments of WS-BPEL code. For example, it is possible to store all invocation instructions in a database or to count on a single text file with all the invocation instructions. The key aspect in this task is to count on a fast mechanism to retrieve these instructions at runtime.

9.3.6 *Create the Rule Premises*

The objective of this task is to create a rules file with rule premises. These rules are used at runtime to discover the requirements that can be affected by unknown context events (see Figure 9.33).

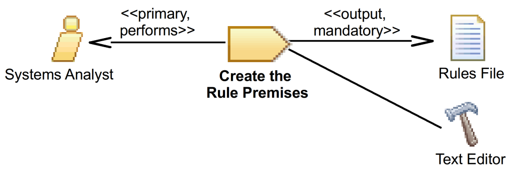


Figure 9.33: Create the Rule Premises task.

The systems analyst role is in charge of creating the rule premises and storing them in a rules file. The rules file works as a knowledge base to keep RDFS rule premises. At runtime, our approach evaluates arising context facts (i.e., context events) against these general rule premises in order to find the requirements that can be affected by unknown context events. A text editor can be used to implement the rules file.

9.4 MORE-WS

In order to enable autonomic behavior, the service composition must adjust from one configuration to another by itself. Since in our approach the reconfiguration of service compositions is performed in terms of features, MoRE-WS translates context changes into changes in the activation/deactivation of features. Then, these changes are translated into the reconfiguration actions that modify the service composition accordingly. MoRE-WS supports the `Dynamic Adaptation` and the `Dynamic Evolution` phases of our framework.

MoRE-WS translates context changes into changes in the activation/deactivation of features.

MoRE-WS is implemented as a set of OSGi[22] bundles. The OSGi technology is a set of specifications that define a dynamic component system for Java. These specifications enable a development model where applications are dynamically composed of many different reusable components. The OSGi specifications enable components to hide their implementations from other components while communicating through services. OSGi was chosen as the underlying technology because code is easier to write and test, reuse is increased, deployment is more manageable, bugs are detected early, and the runtime provides a deep insight into what is running.

As mentioned in Section 7.1, MoRE-WS is built on top of MoRE and only reuses MoRE's bundles that support model actions. The bundles that are reused by MoRE-WS are depicted at the bottom of Figure 9.34[23]. Model-actions-related bundles are invoked in the following order. These bundles have evolved since the first version of MoRE in order to apply it to other problem domains (i.e., not just to OSGi-based architectures). Therefore, these bundles do not specifically reflect the initial set of bundles that are introduced in (Cetina, 2010):

1. **`contextMonitor_Simple`:** This bundle is invoked by MoRE-WS to feed MoRE with the states of the context conditions that can trig-

22 http://www.osgi.org
23 Relationships among bundles are not depicted in this figure to improve readability.

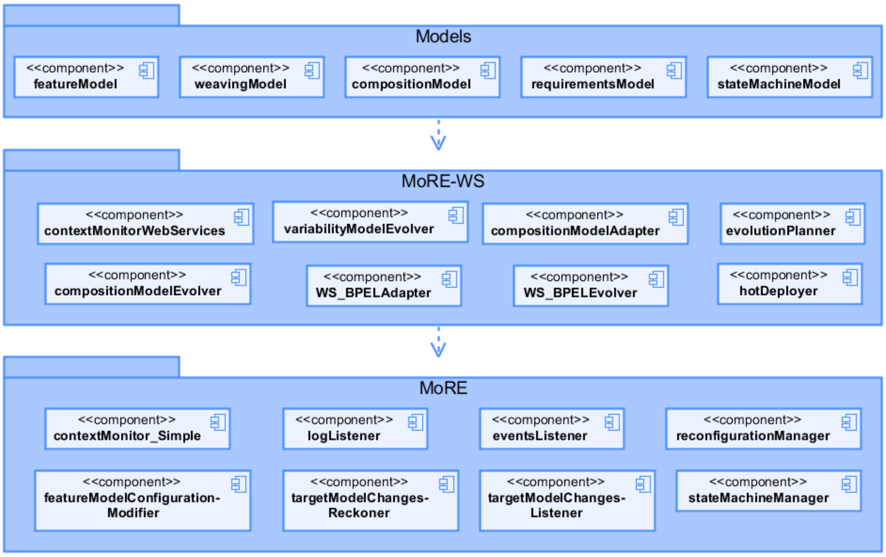

Models
<<component>> featureModel
<<component>> weavingModel
<<component>> compositionModel
<<component>> requirementsModel
<<component>> stateMachineModel
MoRE-WS
<<component>> contextMonitorWebServices
<<component>> variabilityModelEvolver
<<component>> compositionModelAdapter
<<component>> evolutionPlanner
<<component>> compositionModelEvolver
<<component>> WS_BPELAdapter
<<component>> WS_BPELEvolver
<<component>> hotDeployer
MoRE
<<component>> contextMonitor_Simple
<<component>> logListener
<<component>> eventsListener
<<component>> reconfigurationManager
<<component>> featureModelConfiguration-Modifier
<<component>> targetModelChanges-Reckoner
<<component>> targetModelChanges-Listener
<<component>> stateMachineManager

Figure 9.34: Architecture to support autonomic service compositions in terms of OSGi bundles.

ger adaptations. If the state of the context condition is TRUE, then it means that the context condition has been triggered.

2. **`logListener`:** This bundle prints out informative messages in console about the adaptation actions (e.g. to show the information about the fulfilled context condition).

3. **`eventsListener`:** This is a supporting bundle that manages the inter-bundle communication among MoRE's bundles (i.e., it manages the connections between OSGi consumer services and producer services).

4. **`reconfigurationManager`:** This bundle finds out if a context condition has been triggered (according to the information collected by the `contextMonitor_Simple` bundle). If this is the case (i.e., the context condition is TRUE), then MoRE fires a transition from the current configuration of the feature model to a new configuration (according to the state machine model that abstracts the adaptation space).

5. **`featureModelConfigurationModifier`:** This bundle activates or deactivates features in the feature model.

6. **`targetModelChangesReckoner`:** This bundle loads the models to be used at runtime. With the information in these models, MoRE calculates composition model increments and decrements.

7. **`stateMachineManager`:** This bundle starts the state machine. This state machine describes the adaptation space. The initial configuration of the service composition (or state in the state machine) is defined at design time.

8. **`targetModelChangesListener`:** This bundle listens to the changes the `targetModelChangesReckoner` bundle has calculated to be carried out on the target model (the composition model in our case).

MoRE-WS extends the aforementioned generic set of bundles with the following ones in order to support model-driven dynamic adjustments of service compositions. Implementation details are described in Appendix A:

Appendix A describes implementation details.

1. **`contextMonitorWebServices`:** This bundle implements the Context Monitor that observes the context of the service composition. This bundle feeds MoRE's `contextMonitor_Simple` bundle with the current state of context conditions (i.e., TRUE or FALSE).

2. **`compositionModelAdapter`:** This bundle is in charge of adapting the composition model at runtime. This bundle gets the list of composition model increments and decrements that are listened by MoRE's `targetModelChangesListener` bundle.

3. **`evolutionPlanner`:** This bundle constantly looks for unknown context events. If there is an unknown context event, then it looks for a requirement that can be affected by this event. Afterwards, it looks for a surviving tactic to preserve the requirement.

4. **`variabilityModelEvolver`:** This bundle evolves the variability model to face unknown context events in the open world.

5. **`compositionModelEvolver`:** This bundle evolves the composition model to face unknown context events in the open world.

6. **`WS_BPELAdapter`:** This bundle adapts the WS-BPEL composition schema using the information in the adapted composition model.

7. **WS_BPELEvolver:** This bundle evolves the WS-BPEL composition schema using the information in the evolved composition model in order to face unknown context events in the open world.

8. **hotDeployer:** This bundle hot deploys the adjusted WS-BPEL composition schema into the EXECUTION ENGINE (e.g. APACHE ODE[24]).

Also, MoRE-WS uses a set of bundles, which contain classes that implement the models that are leveraged at runtime. The JAVA code for these classes was generated with a set of EMF metamodels. The generated code consists of the following elements: `model` (interfaces and the factory to create the JAVA classes), `model.impl` (concrete implementation of the interfaces defined in `model`), and `model.util` (the `AdapterFactory`). The bundles that implement the models that are used during execution are as follows:

1. **featureModel:** This bundle contains the classes to instantiate feature models. The metamodel for the feature model is taken from MOSKitt4SPL.

2. **weavingModel:** This bundle contains the classes to instantiate weaving models. The metamodel for the weaving model is taken from AMW.

3. **compositionModel:** This bundle contains the classes to instantiate composition models. The metamodel for the composition model is taken from STP.

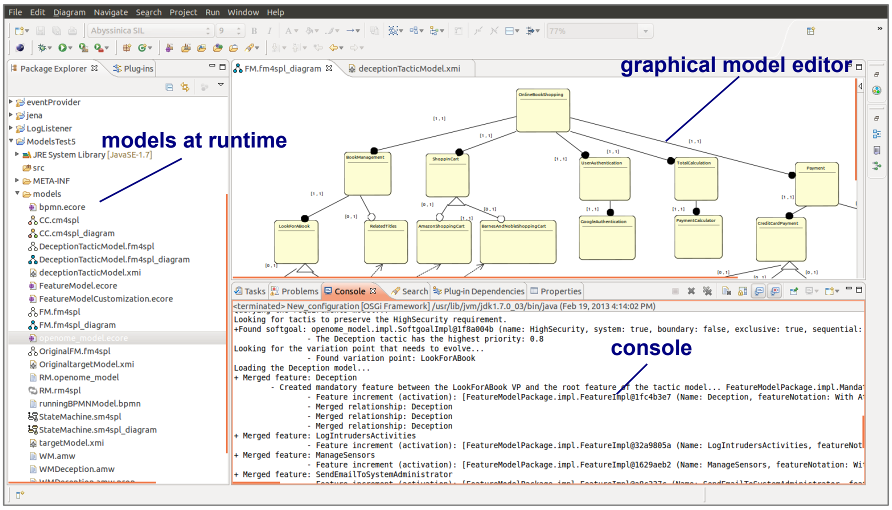
4. **requirementsModel:** This bundle contains the classes to instantiate requirements models. The metamodel for the requirements model is taken from OPENOME.

5. **stateMachineModel:** This bundle contains the classes to instantiate adaptation spaces. The metamodel for the state machine model is taken from MOSKitt4SPL.

Figure 9.35 shows the graphical interface of MoRE-WS. The models that are used at runtime and the bundles can be accessed in the `Package Explorer` at the left. The models can be edited in MoRE-WS in order to facilitate management, as shown at the right. To this end, MoRE-WS imports MOSKitt4SPL EMF-based plugins. The console shows the steps that are carried out during dynamic adjustments.

Appendix B shows a case study in which a service composition is dynamically adapted and evolved with MoRE-WS. Moreover, the website of this thesis (Alférez and Pelechano, 2013a) has two videos with demonstrations of MoRE-WS in action during dynamic adaptations and dynamic evolutions.

Appendix B describes a case study. Also we provide video demonstrations of MoRE-WS.



```
<terminated> New_configuration [OSGi Framework] /usr/lib/jvm/jdk1.7.0_03/bin/java (Feb 19, 2013 4:14:02 PM)
Looking for tactis to preserve the HighSecurity requirement.
+Found softgoal: openome_model.impl.SoftgoalImpl@1f8a004b (name: HighSecurity, system: true, boundary: false, exclusive: true, sequential:
                - The Deception tactic has the highest priority: 0.8
Looking for the variation point that needs to evolve...
                - Found variation point: LookForABook
Loading the Deception model...
+ Merged feature: Deception
        - Created mandatory feature between the LookForABook VP and the root feature of the tactic model... FeatureModelPackage.impl.Manda
                - Feature increment (activation): [FeatureModelPackage.impl.FeatureImpl@1fc4b3e7 (Name: Deception, featureNotation: With A
                - Merged relationship: Deception
                - Merged relationship: Deception
                - Merged relationship: Deception
+ Merged feature: LogIntrudersActivities
                - Feature increment (activation): [FeatureModelPackage.impl.FeatureImpl@32a9805a (Name: LogIntrudersActivities, featureNot
+ Merged feature: ManageSensors
                - Feature increment (activation): [FeatureModelPackage.impl.FeatureImpl@1629aeb2 (Name: ManageSensors, featureNotation: Wi
+ Merged feature: SendEmailToSystemAdministrator
```

Figure 9.35: Graphical interface of MoRE-WS.

9.5 CONCLUSIONS

In this chapter, we presented the tools that support the `Design`, `Dynamic Adaptation`, and `Dynamic Evolution` phases of our framework. First, the plugins that encapsulate the method contents were described in Section 9.1. These method contents indicate the tools that support the process models to design dynamic adaptations and dynamic evolutions of service compositions.

The tools in the method content for designing the dynamic adaptation of service compositions were described in Section 9.2. Then, the tools in the method content for designing the dynamic evolution of service compositions were described in Section 9.3.

Figure 9.36 summarizes the tools that can be used at design time. The STP BPMN Modeler and the EMF Generic Editor can be used to create composition models (e.g. base composition models or variant models). AMW can be used to create two weaving models. The first weaving

24 http://ode.apache.org

model links the features in the variability model, which is created with MOSKitt4SPL, to variation points and commonalities in the base composition model, and variant models. The second weaving model links the features in all the tactic models (expressed as feature models) to the elements in all the tactic models (expressed as composition models). These weaving models are helpful to reflect changes in source models (i.e., feature models) on target models (i.e., composition models).

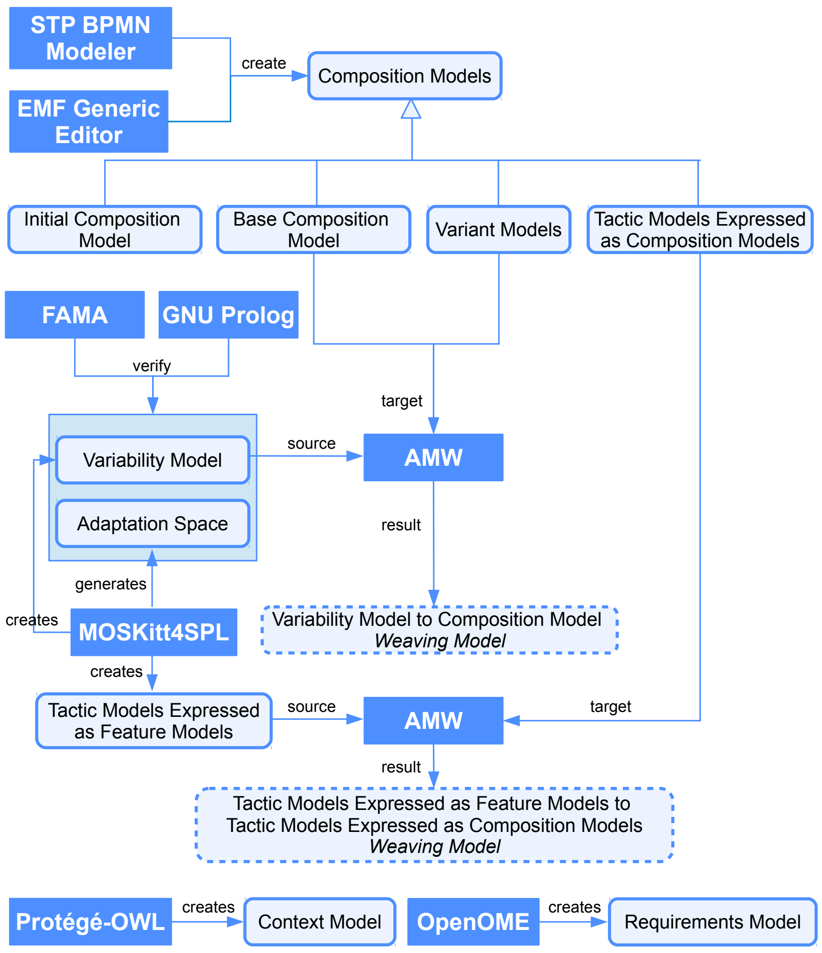


Figure 9.36: Tool-support at design time.

The adaptation space, which abstracts the possible configurations of the variability model (described as a feature model), can be generated with MOSKitt4SPL. Therefore, MOSKitt4SPL implements the Configuration Generator in our framework.

FAMA-FW and GNU Prolog can be used to implement the Verifier. We also proposed tools to create the context model and the requirements model. Specifically, the context model can be created with Protégé-OWL. In turn, the requirements model can be created with OpenOME.

In order to support the adjustment of service compositions at runtime, MoRE-WS was described in Section 9.4. Basically, MoRE-WS leverages models at runtime to determine how the service composition should face arising context events. MoRE-WS guides the dynamic adaptation of services compositions in the closed world, and their dynamic evolution in the open world. MoRE-WS is an extension of the previous MoRE tool.

The contributions presented in this chapter are based on scientific publications (Alférez and Pelechano, 2011a; Alférez et al., 2013; Alférez and Pelechano, 2012a, 2013b).

10 VALIDATION

"Research is creating new knowledge."

— Neil Armstrong

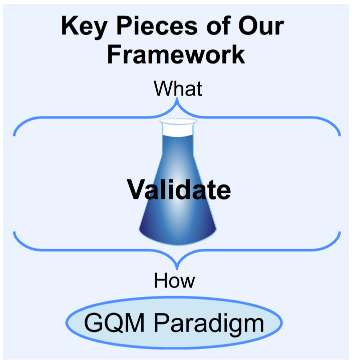


Figure 10.1: Scope of Chapter 10.

In this chapter, we describe the evaluation results of our framework. Among the types of investigations (strategies), we chose to carry out experiments as the empirical investigation strategy. An experiment in software engineering is an empirical inquiry that manipulates one factor or variable of the studied setting (Wohlin et al., 2012).

In order to evaluate our approach, first it is necessary to find out *what* aspects to be evaluated (see Figure 10.1). To this end, we evaluated the key pieces of our framework to confirm that they support the contributions of this thesis. Since our framework spans from design time to runtime, we evaluated key pieces in the `Design`, `Dynamic Adaptation`, and `Dynamic Evolution` phases.

The next step, is to find out *how* to carry out measurements. In order to develop the evaluation metrics, we used the Goal/Question/Metric (GQM) paradigm (Basili et al., 1994). The GQM paradigm was chosen because measurement is defined in a top-down fashion, from goals to metrics. In the experiments, we mostly used the online-book-shopping case study introduced in Section 4.1 and expanded in Appendix B.

The remainder of this chapter is structured as follows. Section 10.1 describes the aspects of our framework that are evaluated. Section 10.2 gives a brief introduction to the GQM paradigm. Section 10.3 describes the computing infrastructure that was used to run the experiments. Section 10.4 presents the results of the experiments that were carried out in the `Design Phase`. Section 10.5 presents the results of the experiments that were carried out in the `Dynamic Adaptation Phase`. Section 10.6 presents the results of the experiments that were carried out in the `Dynamic Evolution Phase`. Finally, Section 10.7 presents the conclusions.

10.1 EVALUATED ASPECTS

In Chapter 4, we introduced the key pieces of our framework. These pieces are used to handle models at design time and to leverage these models at runtime for decision making. **Evaluations were carried out to confirm that these key pieces of our framework give adequate support to the contributions of this thesis**.

At design time, the CONFIGURATION GENERATOR plays a central role for generating the adaptation space with the set of possible variability model configurations. It generates the adaptation space from a variability model and a set of resolutions for particular context conditions. The composition model and the underlying WS-BPEL composition schema are adapted at runtime according to the information in the adaptation space. Therefore, we center our attention on evaluating the following key aspects:

- The efficient generation of the adaptation space, described as a state machine, can facilitate the implementation of our approach and reduce time to market. A slow generation can diminish the effectiveness of the CONFIGURATION GENERATOR. Therefore, we evaluate the generation efficiency of the adaptation space.

- According to (Morin et al., 2009a), fully specifying the adaptation space as a state machine has positive points, such as letting systems analysts perform extensive simulation, validation, and testing of the system's dynamic variability before implementation. Nevertheless, the number of configurations and transitions to be described grows rapidly. Although it is possible to manually specify adaptation spaces for simple adaptive systems, this rapidly becomes a difficult task in the case of large systems. Therefore, it is necessary to measure the complexity reduction of the adaptation space by describing a variability model and a set of resolutions for this model.

Also, the evaluation of the stability and semi-aliveness verification criteria is relevant because it is important to ensure that our solution supports safe service recompositions. Our criteria verify the variability model and its possible configurations prior execution. In this chapter, we evaluate the following aspects:

- It is essential to measure the reduction of anomalies in the variability model and its configurations. Therefore, we evaluate the reduction of anomalies with our verification criteria.

- In order to avoid unnecessary time during verifications, we argue that verifications have to be carried out as fast as possible. Therefore, we evaluate the efficiency of our verification criteria.

At runtime, the Context Monitor collects the context data used by the Model-based Reconfigurator. However, an inefficient context observation mechanism will cause the Model-Based Reconfigurator to work with obsolete data. Therefore, we evaluate the performance of our implementation of the Context Monitor in order to avoid efficiency problems.

The Model-based Reconfigurator is mandatory to detect problematic context events and to carry out the necessary adjustments on the service composition. Therefore, we evaluate the following aspects related to the Model-based Reconfigurator in the closed world:

- The Model-based Reconfigurator leverages models during execution for decision making. This approach arises the following question: Is the management of models at runtime efficient? We argue that the evaluation of the performance of managing models at runtime is key to realize the feasibility of our approach. In this chapter, we evaluate the efficiency of the Model-based Reconfigurator during dynamic adaptations in terms of execution time, memory consumption, and CPU consumption.

- It is possible that several problematic context events arise in very tight time frames. We argue that the Model-based Reconfigurator should continue working fine even in this tough scenario.

Therefore, we evaluate the performance of the Model-based Reconfigurator during dynamic adaptations under stress circumstances.

In order to face uncertainty in the open world, the Model-based Reconfigurator self-evolves models at runtime. To this end, it uses the set of tactics returned by the Evolution Planner. Since dynamic evolutions are a necessity in the complex open world, we evaluate the following aspects:

- Our approach is focused on critical service compositions. Therefore, it is desirable to count on efficient dynamic evolutions of the service composition. In this chapter, we evaluate the efficiency of the Evolution Planner and the Model-based Reconfigurator for dynamic evolutions in terms of execution time, memory consumption, and CPU consumption.
- Dynamic evolutions are performed in order to preserve expected requirements when facing unknown context events. Therefore, we find it essential to count on accurate inferences that are carried out by the Evolution Planner. The objective of these inferences is to find the requirements that can be negatively affected by unknown context events. In this chapter, we evaluate the accuracy of these inferences.

The Execution Engine deploys new versions of the WS-BPEL composition schema. The evaluation of hot-deployment performance is out of the scope of this chapter. We focus on evaluating the pieces in our framework that support the model-driven operations for self-adjusting service compositions.

We choose a top-down approach to define measurements.

The next step, after having the set of aspects to be evaluated, is to carry out measurements. In this chapter, we choose a top-down approach to define measurements, i.e., software measures are derived from measurement goals and questions. In our case, a bottom-up approach does not work because there are several observable characteristics in SAS, but which metrics to be used and how one interprets them is not clear without the appropriate models and goals. Therefore, in Section 10.2 we briefly describe the GQM paradigm, which can be used to define measurements in a top-down fashion.

10.2 A BRIEF INTRODUCTION TO THE GQM PARADIGM

The GQM paradigm (sometimes called the GQM approach) is a mechanism for defining and evaluating a set of operational goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products, and quality perspectives of interest (Basili, 1992). In other words, the GQM paradigm supports a top-down approach to define the goals behind measuring software processes and products, and using these goals to decide precisely what to measure (choosing metrics) (Differding et al., 1996).

The GQM paradigm defines a measurement model with goals, questions, and metrics.

The GQM paradigm defines a measurement model with three levels:

1. **Conceptual Level (Goal):** A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment.

2. **Operational Level (Question):** A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try to characterize the object of measurement with respect to a selected quality issue and to determine its quality from the selected viewpoint.

3. **Quantitative Level (Metric):** A set of data is associated with every question in order to answer it in a quantitative way. The data can be: 1) objective, if they depend only on the object that is being measured and not on the viewpoint from which they are taken; or 2) subjective, if they depend on both the object that is being measured and the viewpoint from which they are taken.

A GQM model is a hierarchical structure (see Figure 10.2) starting with a goal (specifying purpose of measurement, issue to be measured, object to be measured, and viewpoint from which the measure is taken). The goal is refined into several questions that usually break down the issue into its major components. Each question is then refined into metrics.

10.3 COMPUTING INFRASTRUCTURE FOR THE EXPERIMENTS

Figure 10.3 shows the UML deployment diagram that depicts the computing infrastructure used in the experiments and in the online-book-shopping case study (described in Appendix B). The Web services ran

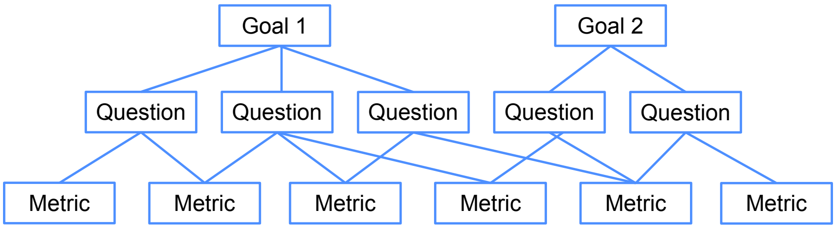


Figure 10.2: GQM model hierarchical structure (Basili et al., 1994).

on Apache Axis2[1] version 1.6.1, which is deployed as a Web application ARchive (WAR) distribution on Apache Tomcat[2] version 7.0.8. Hot deployment is carried out by the Model-based Reconfigurator on Apache ODE[3] version 1.3.5, which is deployed on a second instance of Apache Tomcat as a WAR distribution. The Context Monitor and the Evolution Planner are implemented as OSGi bundles. The aforementioned pieces run on a Personal Computer (PC) with an Intel Core 2 Duo 2.0 GHz processor, 4 GB RAM, 64-bit Ubuntu version 12.10, and Kernel Linux version 3.5.0-37-generic.

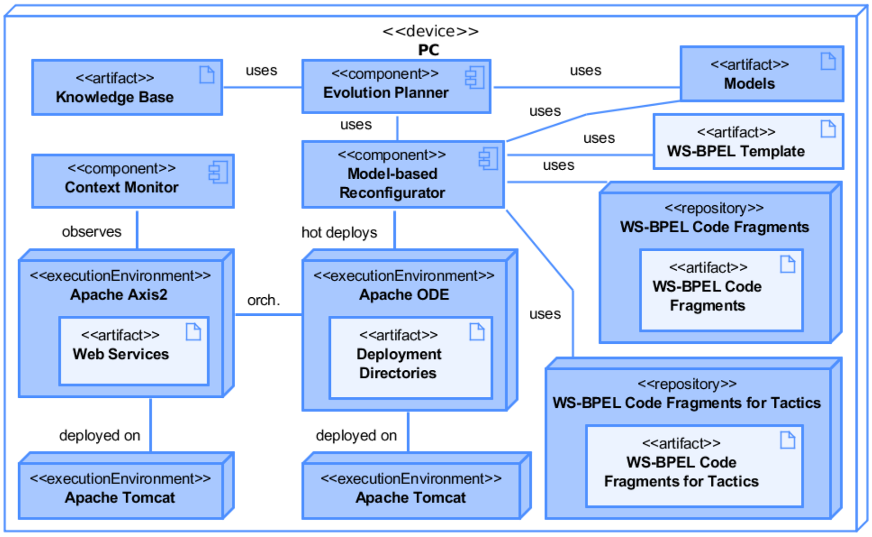


Figure 10.3: UML deployment diagram for the case study.

1 http://axis.apache.org/axis2/java/core/
2 http://tomcat.apache.org/
3 http://ode.apache.org

10.4 VALIDATION IN THE DESIGN PHASE

In this section, we present four GQM models to evaluate key aspects of the Configuration Generator and the Verifier. These pieces are used in the `Design Phase` of our framework. The first two GQM models in Section 10.4.1 and Section 10.4.2 have helped us to evaluate the generation efficiency and complexity reduction of the adaptation space. The other GQM models in Section 10.4.3 and Section 10.4.4 have guided us to evaluate aspects related to the verification of the variability model and its configurations.

10.4.1 *Generation Efficiency of Variability Model Configurations*

Table 10.1 describes the GQM model for the following goal: *"Efficient generation time of variability model configurations from the systems analyst's viewpoint."*

	Purpose	**Efficient**
Goal	**Issue**	**generation time of**
	Object	**variability model configurations**
	Viewpoint	**from the systems analyst's viewpoint**
Question	**Q1**	What is the required time to generate the adaptation space?
Metrics	**M1 - M5**	(**M1**) number of features in the variability model, (**M2**) number of resolutions, (**M3**) number of generated variability model configurations, (**M4**) number of generated transitions among variability model configurations, and (**M5**) generation time of the adaptation space

Table 10.1: GQM model for the "efficient generation time of variability model configurations from the systems analyst's viewpoint" goal.

Efficiency is considered as performing or functioning in the best possible manner with the least waste of time and effort. In order to answer **Q1**, we used the variability model in our running example with 36 features (**M1**) and a set of 9 and 17 resolutions specified for this model (**M2**). With this information, MOSKitt4SPL[4], which implements the Configuration Generator, generated the following adaptation spaces:

- *For 9 resolutions:* 40 variability model configurations (**M3**) and 360 transitions among configurations (**M4**) were generated in 3.6 seconds (**M5**).
- *For 17 resolutions:* 120 variability model configurations (**M3**) and 2,040 transitions among configurations (**M4**) were generated in 4.8 seconds (**M5**).

The manual creation of these large adaptation spaces is clearly unfeasible.

10.4.2 *Complexity Reduction of the Adaptation Space*

Table 10.2 describes the GQM model for the following goal: "*Reduce the complexity of the adaptation space from the systems analyst's viewpoint.*"

Goal	**Purpose**	**Reduce**
	Issue	**the complexity of the**
	Object	**adaptation space**
	Viewpoint	**from the systems analyst's viewpoint**
Question	**Q2**	Do a small variability model and a set of resolutions can be used to describe a large adaptation space?
Metrics	**M6 - M9**	(**M6**) number of features in the variability model, (**M7**) number of resolutions, (**M8**) number of generated variability model configurations, and (**M9**) number of generated transitions among variability model configurations

Table 10.2: GQM model for the "reduce the complexity of the adaptation space from the systems analyst's viewpoint" goal.

4 http://www.pros.upv.es/m4spl

Q2 can be answered by looking at the variability model in our running example with 36 features (**M6**) and nine resolutions (**M7**) specified for this model. This small and easy-to-understand model and a small set of resolutions describe a large adaptation space with 40 variability model configurations (**M8**) and 360 transitions among configurations (**M9**). If we increase the number of resolutions up to 17 (**M7**), the adaptation space grows to 120 variability model configurations (**M8**) and 2,040 transitions among configurations (**M9**).

Therefore, it is possible to say that variability models can reduce (or hide) much of the complexity in the definition of the adaptation space from the point of view of systems analysts. Variability models can provide an intensional rather than extensional description of each possible configuration of the service composition.

10.4.3 *Anomalies Reduction in the Variability Model and its Configurations*

Table 10.3 describes the GQM model for the following goal: "*Reduce anomalies in the variability model and in its configurations from the systems analyst's viewpoint.*"

Goal	**Purpose**	**Reduce**
	Issue	**anomalies in the**
	Object	**variability model and its configurations**
	Viewpoint	**from the systems analyst's viewpoint**
Question	**Q3**	What is the percentage of identified anomalies?
Metrics	**M10 - M11**	(**M10**) number of features in the variability model and (**M11**) percentage of identified anomalies
Question	**Q4**	What is the percentage of false positives?
Metrics	**M12 - M13**	(**M12**) number of features in the variability model and (**M13**) percentage of false positives

Table 10.3: GQM model for the "reduce anomalies in the variability model and in its configurations from the systems analyst's viewpoint" goal.

In order to answer **Q3** and **Q4**, we evaluated 47 models, out of which 45 were taken from the SPLOT repository (Mendonca et al., 2009). The other two models were developed during industry collaboration projects at the CRI (Lora-Michiels et al., 2010). The number of features in the models is distributed as follows (**M10** and **M12**): 30 models contained from 9 to 49 features, 4 models from 50 to 99 features, 4 models from 100 to 999 features, and 9 models from 1,000 to 2,000 features.

The variability models covered various domains such as insurance, entertainment, Web applications, home automation, search engines, and databases (Mazo, 2011). GNU Prolog 1.3.0[5] was used to implement the stability and semi-aliveness verification operations of the Verifier. Experiments showed that our verification approach identified 100% of the anomalies (**M11**) with 0% false positives (**M13**).

10.4.4 *Verification Efficiency*

Table 10.4 describes the GQM model for the following goal: "*Efficient stability and semi-aliveness verification of the variability model from the systems analyst's viewpoint.*"

Goal	**Purpose**	**Efficient**
	Issue	**stability and semi-aliveness of the**
	Object	**variability model**
	Viewpoint	**from the systems analyst's viewpoint**
Question	**Q5**	What are the execution times for each verification criterion?
Metrics	**M14 - M16**	(**M14**) number of features in the variability model, (**M15**) average execution time for the stability criterion, and (**M16**) average execution time for the semi-aliveness criterion

Table 10.4: GQM model for the "efficient verification of the variability model from the systems analyst's viewpoint" goal.

In response to **Q5**, the execution time for each verification criterion is as follows. We used the same variability models, with the same num-

5 http://gprolog.univ-paris1.fr

ber of features per model (**M14**), that were used to evaluate the GQM model in Section 10.4.3:

- *Stability:* The evaluation shows that it takes 0.9 ms in average to verify models between 9 and 100 variables. In the worst case, it takes 16.2 seconds in average to verify models between 100 and 2,000 variables (**M15**).

- *Semi-aliveness:* The performance of this verification criterion depends on the number of partitions and the number of variables per partition. Let us recall that a CSP is theoretically non-polynomial. However, modern solvers use highly-optimized algorithms that are able to solve these CSPs efficiently (if needed, the programmer can help the solver with redundant constraints, alternative modeling, heuristics, etc.). Our preliminary tests show that this operation scales well on our case study: 6.2 seconds for seven partitions and four context variables per partition (**M16**). Since we do not currently have an algorithm to find the partitions, further experiments are required to test semi-aliveness. We found partitions manually in our case study as described in the example of Section 5.2.9.2.

10.5 VALIDATION IN THE DYNAMIC ADAPTATION PHASE

This section presents three GQM models, which are used to validate key aspects of the CONTEXT MONITOR and the MODEL-BASED RECONFIGURATOR. In our prototype, we implemented our own CONTEXT MONITOR. In addition, our MoRE-WS tool implements the MODEL-BASED RECONFIGURATOR. These two tools are used to support the `Dynamic Adaptation Phase` of our framework. The GQM model in Section 10.5.1 has helped us to evaluate the efficiency of the CONTEXT MONITOR, which feeds MoRE-WS with context data. The other models have allowed us to evaluate the following key aspects about dynamic adaptations with MoRE-WS. First, the GQM model in Section 10.5.2 has allowed us to evaluate the efficiency of MoRE-WS during dynamic adaptations. Second, the GQM model in Section 10.5.3 has helped us to evaluate the capacity of MoRE-WS to avoid saturations under stress circumstances.

10.5.1 *Context Observation Efficiency*

Table 10.5 describes the GQM model for the following goal: *"Efficient context observation of service operations from the* CONTEXT MONITOR*'s viewpoint."*

Goal	**Purpose**	**Efficient**
	Issue	**context observation of**
	Object	**service operations**
	Viewpoint	**from the Context Monitor's viewpoint**
Question	**Q6**	Is the CONTEXT MONITOR efficient to observe service operations?
Metrics	**M17 - M21**	(**M17**) execution time for observing service operations, (**M18**) memory consumption when observing service operations, (**M19**) number of observed service operations, (**M20**) observation period, and (**M21**) elapsed time

Table 10.5: GQM model for the "efficient context observation of service operations from the CONTEXT MONITOR's viewpoint" goal.

In order to answer **Q6**, we measured the execution time (**M17**) and the memory consumption (**M18**) of our implementation of the CONTEXT MONITOR. Measures were taken when the CONTEXT MONITOR was observing the context and updating a file with these observations. Specifically, the CONTEXT MONITOR observed a number of service operations (**M19**) measured in a period of time (**M20**) during an elapsed time (**M21**). In our case, it observed the 15 service operations of our case study (common and variant operations) in sequence every five seconds (**M20**) during one hour (**M21**). In every observation, the CONTEXT MONITOR saved the observed data in a file.

Figure 10.4 shows the resulting execution times in this experiment. The execution time to carry out a set of observations has a linear growth as the file with the observations grows. There is always a peak at the very beginning of context observations because resources are assigned to APACHE AXIS2.

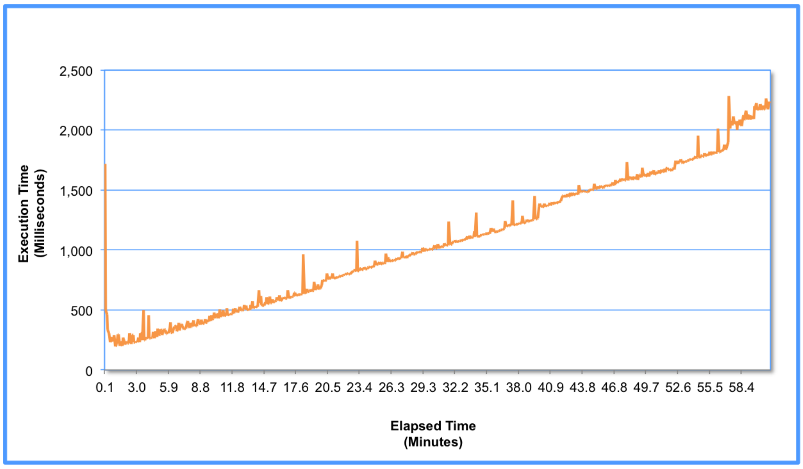


Figure 10.4: Resulting execution time for the experiment to answer **Q6**.

Figure 10.5 shows the resulting memory consumption in this experiment. When the heap reaches a minimum percentage of heap free after garbage collection, the Java Virtual Machine (JVM) increases the amount of free memory. Therefore, this figure shows periodic and sudden memory improvements.

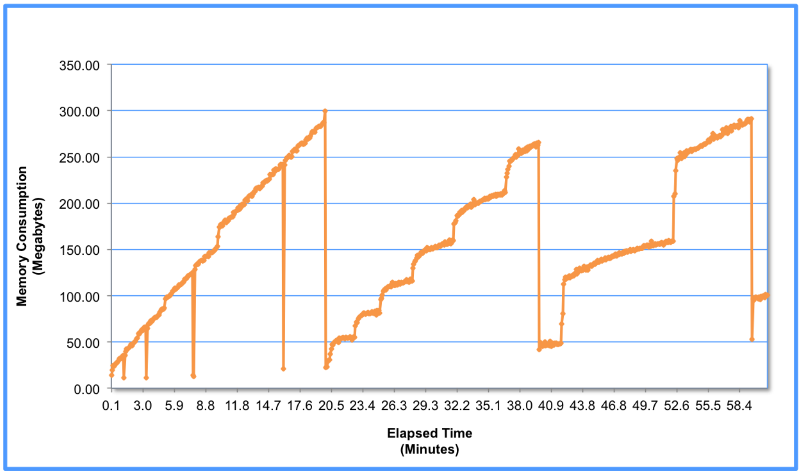


Figure 10.5: Resulting memory consumption for the experiment to answer **Q6**.

In a proof-of-concept scenario, our implementation of the Context Monitor can be used to store context data for short periods of time without excessive execution time or memory problems (e.g. out-of-memory errors). Moreover, the data collected by our Context Monitor fulfills its main goal: to feed MoRE-WS with context data, which can be analyzed at runtime. However, execution time and memory consumption increase as the file with the observations grows. The constant and rapid updates of the file with context observations require a more efficient solution in industrial scenarios, which is out of the scope of this thesis. We propose to use other alternative approaches, such as managing the data in a stream database, which can handle continuous data streams.

10.5.2 *Dynamic Adaptation Efficiency*

Table 10.6 describes the GQM model for the following goal: *"Efficient dynamic adaptation of service compositions from MoRE-WS's viewpoint."*

Goal	**Purpose**	**Efficient**
	Issue	**dynamic adaptation of**
	Object	**service compositions**
	Viewpoint	**from MoRE-WS's viewpoint**
Question	**Q7**	Is MoRE-WS efficient to carry out the dynamic adaptation of service compositions?
Metrics	**M22 - M26**	(**M22**) average execution time of the operations that are carried out by MoRE-WS for dynamic adaptation, (**M23**) average memory consumption of the operations that are carried out by MoRE-WS for dynamic adaptation, (**M24**) CPU consumption in a time frame, (**M25**) memory consumption in a time frame, and (**M26**) time frame

Table 10.6: GQM model for the "efficient dynamic adaptation of service compositions from MoRE-WS's viewpoint" goal.

In order to answer **Q7**, we carried out two experiments. In the first experiment, we measured the average execution time (**M22**) and memory consumption (**M23**) of the operations that are carried out by MoRE-WS to dynamically adapt the service composition to face the `B&NUnavailable` context condition. This context condition is triggered when the `Barnes & Noble Books` composite service is currently unavailable. In order to face this context condition, $R_{B\&NUnavailable}$ deactivates the `Barnes & Noble Books` and the `Barnes & Noble Shopping Cart` functionalities. In turn, this resolution activates the `Amazon Books`, `Amazon Shopping Cart`, and `Related Titles` functionalities. This resolution was chosen because one context event affects several functionalities.

In the second experiment, we measured the CPU consumption (**M24**) and the memory consumption (**M25**) during dynamic adaptations for the following four context conditions. Section B.2 describes these context conditions:

- `B&NUnavailable`
- `BankOfAmericaHiExecTime`$\bigwedge$`WellsFargoExecTimeLowerThanChase-Banking`
- `UPSHiExecTime`$\bigwedge$`DHLExecTimeLowerThanFedEX`
- `EmailInvoiceHiExecTime`

The problematic context events happened sequentially in a time frame of less than a minute (**M26**). In both experiments, we used the following files with models at runtime: a file with the variability model (23.0 kB); a file with the base composition model and variant models (25.1 kB); and a file with the weaving model that links the elements between the aforementioned models (11.4 kB).

Results of the First Experiment

Table 10.7 shows the summary of the average execution time in ms and the average memory consumption in megabytes (MB) for MoRE-WS operations during the dynamic adaptation. We ran the same adaptation three times and calculated the average of the measures in order to give results as accurate as possible.

In the `Analyzing the Context` section of Table 10.7, the "creating an empty context model" and "inserting context events into the context model" operations got the highest execution time. Nevertheless, the "creating an empty context model" operation is carried out just one

Operation	**Execution Time (ms)**	**Memory Consumption (MB)**
Analyzing the Context		
Creating an empty context model	487.6	16.1
Inserting context events into the context model	486.6	24.3
Evaluating context conditions	154	16.5
Deleting the elements in the context model	17.6	8.5
Planning the Adaptation		
Looking for the resolution that is triggered by the context condition	0.6	7.7
Executing a resolution	532.6	7.9
Updating the feature model	1.3	7.6
Querying the weaving model	2.4	9
Generating a reconfiguration plan	449	9
Adapting the composition model	430	12
Executing the Adaptation		
Looking for the adapted variation points in the composition model	340	8.5
Inserting WS-BPEL fragments into the WS-BPEL template	107.3	35.2
Creating the deployment directory	16.3	8.5
Copying the WSDL files into the deployment directory	2.3	8.3

Table 10.7: Summary of the dynamic adaptation results to answer **Q7**.

time when MoRE-WS starts. The "inserting context events into the context model" operation covers two operations: 1) querying the file with the context observations; and 2) updating the context model. Therefore, the efficiency of this operation depends on the number of observations to be put into the context model. In order to make the querying operation as efficient as possible, this operation was implemented with the Streaming API for XML (StAX)[6].

In the `Planning the Adaptation` section of Table 10.7, the operations that guide model-driven dynamic adaptations got the highest execution time. First, the "executing a resolution" operation triggers the activation and deactivation of features in the variability model by invoking the "updating the feature model" operation. Second, the "generating a reconfiguration plan" operation calculates $CM\triangle$ and $CM\nabla$

6 http://stax.codehaus.org

actions. Finally, the "adapting the composition model" operation looks for the variation points that have to be rebound and does the necessary rebindings with variant models. Nevertheless, each one of these operations was carried out in less than 0.6 seconds. Also, memory consumption was very low in these operations.

In the `Executing the Adaptation` section of Table 10.7, the "looking for the adapted variation points in the composition model" got the highest execution time. This operation searches sequentially in the adapted composition model for the variation points that have been rebound with variant models.

It is important to notice the following about the strategy of merging WS-BPEL fragments into the WS-BPEL template: 1) this strategy got a fast execution time; and 2) the memory consumption of this operation was higher than the other operations because of file management. Nevertheless, the required memory can be easily supported by current servers.

Overall, the resulting execution times and memory consumptions in Table 10.7 demonstrate that MoRE-WS is efficient to carry out dynamic adaptations.

Results of the Second Experiment

Figure 10.6 shows the percentage of CPU consumption of MoRE-WS during four dynamic adaptations in a time frame of less than a minute. These results were obtained with Java VisualVM[7].

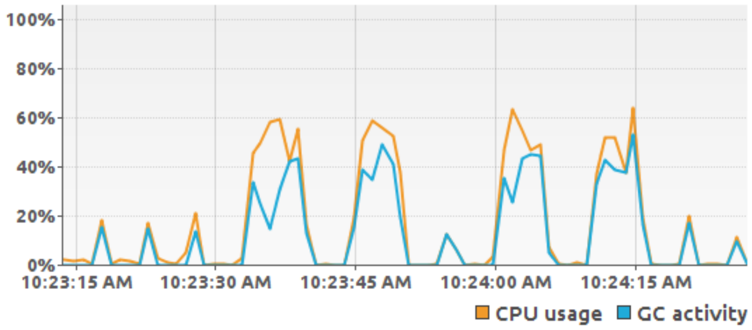


Figure 10.6: CPU consumption of MoRE-WS during four dynamic adaptations (line in orange).

7 http://visualvm.java.net

The line in orange indicates the percentage of CPU consumption. The line in blue is for the garbage collector. On one hand, MoRE-WS does not carry out any dynamic adaptation in the lower peaks. At these times, MoRE-WS updates the context model and analyzes whether any context condition has been fulfilled or not. MoRE-WS only spends around 20% of the CPU when it is on context-observation mode. On the other hand, MoRE-WS carries out the dynamic adaptations for the four fulfilled context events in the four higher peaks. During dynamic adaptations, the CPU consumption grows up to 60% for a few seconds.

In Figure 10.7, the line in blue shows the memory consumption of MoRE-WS for the aforementioned four dynamic adaptations. The line in orange is for the heap size, which is automatically assigned by the JVM. Memory consumption is constant and low even during adaptations.

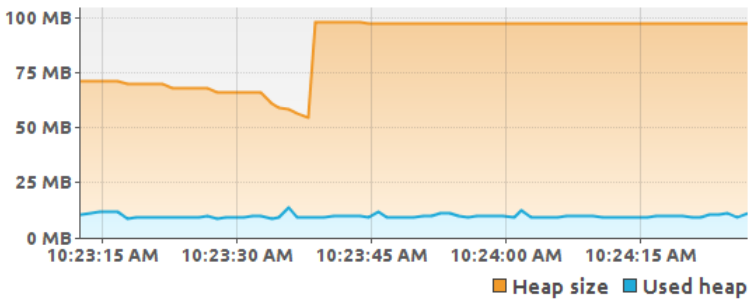


Figure 10.7: Memory consumption of MoRE-WS during four dynamic adaptations (line in blue).

10.5.3 *Operability under Stress*

Table 10.8 describes the GQM model for the following goal: "*Avoid saturation under stress circumstances of* MoRE-WS *from* MoRE-WS*'s viewpoint.*" In order to answer **Q8**, we have manually injected the following four problematic context events into the file with the observations log (**M27**): 1) the `Barnes and Noble Books` service operation is unavailable; 2) the execution time of the `Bank of America Credit Card Payment` service operation is 2,751 ms and the execution time of the `Wells Fargo Online` service operation is lower than the execution time of the `Chase Banking` service operation; 3) the execution time of the `UPS Shipping` service operation is 1,653 ms and the execution time of the `DHL Delivery` service operation is lower than the execution time of the `FedEX Express` service operation; and 4) the `E-mail Invoice` service operation has an execution time of 1,132 ms. These events are separated by very small time frames of less than one second (**M28**).

Goal	**Purpose**	**Avoid**
	Issue	**saturation under stress circumstances of**
	Object	**MoRE-WS**
	Viewpoint	**from MoRE-WS's viewpoint**
Question	**Q8**	Does MoRE-WS have a good performance and memory consumption under stress circumstances?
Metrics	**M27 - M32**	(**M27**) number of problematic context events, (**M28**) time frame between problematic context events, (**M29**) frequency to observe the context, (**M30**) number of context conditions that could be affected by arising context events, (**M31**) execution time under stress circumstances, and (**M32**) memory consumption under stress circumstances

Table 10.8: GQM model for the "avoid saturation under stress circumstances of MoRE-WS from MoRE-WS's viewpoint" goal.

In the fist run (or observation), MoRE-WS retrieved the aforementioned set of problematic events at once because it queries the information collected by the CONTEXT MONITOR every five seconds (**M29**). Then, MoRE-WS evaluated four context conditions that could be affected by these events (**M30**): `B&NUnavailable`, `BankOfAmericaHiExecTime`$\wedge$`WellsFargoExecTimeLowerThanChaseBanking`, `UPSHiExecTime`$\wedge$`DHLExecTimeLowerThanFedEX`, and `EmailInvoiceHiExecTime`. Section B.2 describes these context conditions in detail.

Table 10.9 shows execution time (**M31**) and memory consumption (**M32**) results for dynamic adaptations under this stressful situation. Since context conditions are evaluated in sequence (a dynamic adaptation for *context condition 1* is always triggered before an adaptation for *context condition 2*), we did not experience performance decrease (i.e., the execution time is similar to the execution time without stress). Moreover, the memory consumption was low because only one dynamic adaptation is carried out at a time. Therefore, we can conclude that MoRE-WS is efficient under stress circumstances when several problematic context events arise in tight time frames.

Context Condition	**Execution Time (ms)**	**Memory Consumption (MB)**
B&NUnavailable	6,645	11.8
BankOfAmericaHiExecTime $\bigwedge$ WellsFargoExecTime LowerThanChaseBanking	4,858	10.7
UPSHiExecTime $\bigwedge$ DHLExecTimeLowerThanFedEX	5,138	10.7
EmailInvoiceHiExecTime	4,571	10.7

Table 10.9: Execution time and memory consumption during dynamic adaptations for four context events in a very tight time frame.

10.6 VALIDATION IN THE DYNAMIC EVOLUTION PHASE

This section presents two GQM models, which are used to evaluate key aspects of the Evolution Planner and the Model-based Reconfigurator. In our prototype, we implemented the Evolution Planner. In addition, our MoRE-WS tool implements the Model-based Reconfigurator. These tools are used to support the `Dynamic Evolution Phase` of our framework. The GQM model in Section 10.6.1 has allowed us to evaluate the efficiency of the Evolution Planner and MoRE-WS during dynamic evolutions. The GQM model in Section 10.6.2 has helped us to evaluate the accuracy of the inferences that are carried out by the Evolution Planner. The Evolution Planner uses these inferences to look for the requirements that can be negatively affected by unknown context events.

10.6.1 *Dynamic Evolution Efficiency*

Table 10.10 describes the GQM model for the following goal: *"Efficient dynamic evolution of service compositions from the* Evolution Planner*'s and* MoRE-WS*'s viewpoint."* In order to answer **Q9**, we carried out two experiments.

Goal	**Purpose**	**Efficient**
	Issue	**dynamic evolution of**
	Object	**service compositions**
	Viewpoint	**from the Evolution Planner's and MoRE-WS's viewpoints**
Question	**Q9**	Are the Evolution Planner and MoRE-WS efficient to carry out dynamic evolutions?
Metrics	**M33 - M38**	(**M33**) average execution time of the operations that are carried out by the Evolution Planner to plan an evolution, (**M34**) average memory consumption of the operations that are carried out by the Evolution Planner to plan an evolution, (**M35**) average execution time of the operations that are carried out by MoRE-WS for dynamic evolution, (**M36**) average memory consumption of the operations that are carried out by MoRE-WS for dynamic evolution, (**M37**) overall CPU consumption, and (**M38**) overall memory consumption

Table 10.10: GQM model for the "efficient dynamic evolution of service compositions from the Evolution Planner's and MoRE-WS's viewpoint" goal.

In the first experiment, we measured the following: 1) the average execution time (**M33**) and the average memory consumption (**M34**) of the operations that are carried out by the Evolution Planner to plan an evolution; and 2) the average execution time (**M35**) and the average memory consumption (**M36**) of the operations that are carried out by MoRE-WS to evolve the service composition. In this experiment, we chose the following unknown context event: *the Barnes & Noble Books service operation has an execution time higher than 5,000 ms*. This context event has been used throughout this work to illustrate our approach. In the second experiment, we measured the overall CPU consumption (**M37**) and the overall memory consumption (**M38**) during the dynamic evolution for the aforementioned unknown context event.

In both experiments, we used the following files with models at runtime: a file with the requirements model (3.3 kB); a file with the deception tactic implemented as a feature model (0.96 kB); a file with the deception tactic implemented as a composition model (2.18 kB); a file with the weaving model between the deception tactic implemented as a feature model, and the deception tactic implemented as a composition model (2.7 kB); a file with the composition model (6.8 kB); and a file with the variability model (23.0 kB).

Results of the First Experiment

Table 10.11 shows the summary of the average execution time in ms and the average memory consumption in MB for the operations that are carried out by the Evolution Planner and MoRE-WS during a typical dynamic evolution. We ran the same evolution three times and calculated the average of the measures in order to give results as accurate as possible.

In Table 10.11, the results of memory consumption in all the operations were similar and very low. In the `Evolution Planner` section, the execution time of the "searching for the requirements that may be affected by an unknown context event" operation was faster than the execution time of the "searching for surviving tactics" operation. In the first operation, the Evolution Planner uses the forward chaining method, which is very efficient in our case with small knowledge bases. The implementation of the second operation is based on EMF.

It is important to notice that the computational complexity of forward chaining in a rule system that consists of η rules is $O(\eta^2)$. The proof is that the worst case to search among η rules consists of η iterations. The maximum sum of iterations is $\eta + \eta{-}1 + \eta{-}2 + \ldots + 1 = \eta(\eta{-}1)/2 = O(\eta^2)$. With this exponential complexity, the system will perform quite slowly for a big rule-base with a lot of rules. In case of requiring large knowledge bases, complexity can be reduced with the Rete algorithm (Forgy, 1982). This algorithm reduces the number

Operation	**Execution Time (ms)**	**Memory Consumption (MB)**
Evolution Planner		
Searching for the requirements that may be affected by an unknown context event	22.7	11.2
Searching for surviving tactics	208	11.2
MoRE-WS		
Merging a tactic model into the variability model and generating an evolution policy	87.6	12.4
Creating a reconfiguration plan and merging a tactic model into the composition model	215	14.8
Evolving the WS-BPEL composition schema	138.6	15.8

Table 10.11: Summary of the dynamic evolution results to answer **Q9**.

of comparisons between rule conditions and assertions in the working memory. This kind of improvements is outside the scope of this work.

In the section about MoRE-WS in Table 10.11, the most expensive operations were the creation of a reconfiguration plan and merging a tactic model into the composition model. In these operations, MoRE-WS carries out several tasks. First, it creates a reconfiguration plan with $CM\triangle$ and $CM\nabla$ actions. Then, it merges the tactic model into the composition model by creating a parallel relationship between the abstraction of the problematic service operation (which is previously found) and the tactic model.

In general, the resulting execution time and low memory consumption in Table 10.11 demonstrate that our proposed computer infrastructure is efficient to carry out dynamic evolutions.

Results of the Second Experiment

Figure 10.8 shows the percentage of CPU consumption of MoRE-WS during the dynamic evolution for the following unknown context event: *the Barnes & Noble Books service operation has an execution time higher than 5,000 ms*. The line in orange indicates the percentage of CPU consumption. In the lower peaks, MoRE-WS evaluates whether or not there is any arising unknown context event. During execution, MoRE-WS only spends around 20% of the CPU in this operation. The dynamic evolution occurs around 11:58 AM, which has the highest peak. During this dynamic evolution, the CPU consumption grows up to 40% for a few seconds.

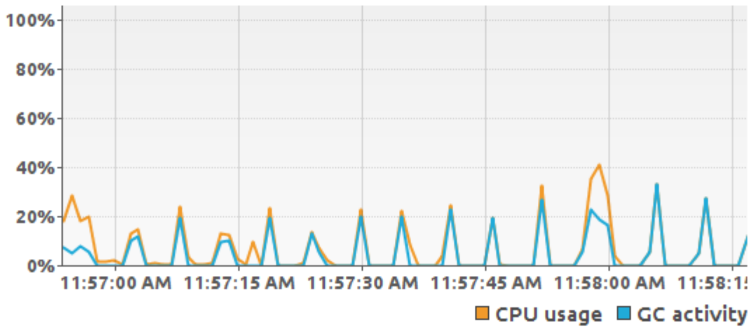


Figure 10.8: CPU consumption of MoRE-WS during a dynamic evolution around 11:58 AM (line in orange).

In Figure 10.9, the line in blue shows the memory consumption of MoRE-WS for the aforementioned evolution around 11:58 AM. Memory consumption increased just a little during this dynamic evolution. The reason of the peak at the beginning of this figure is because MoRE-WS starts to run at that time.

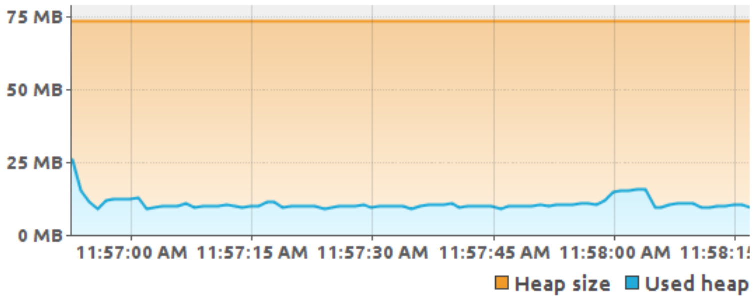


Figure 10.9: Memory consumption of MoRE-WS during a dynamic evolution around 11:58 AM (line in blue)

10.6.2 *Inferences Accuracy*

Table 10.12 describes the GQM model for the following goal: "*Accurate inferences to find the requirements that can be negatively affected by unknown context events in the open world from the* EVOLUTION PLANNER*'s viewpoint*." In order to answer **Q10**, we evaluated the accuracy of the inferences that the EVOLUTION PLANNER carries out to look for the requirements that can be negatively affected by unknown context events. This is a key aspect to be evaluated because dynamic evolutions are performed in order to preserve requirements at runtime.

Goal	**Purpose**	**Accurate**
	Issue	**inferences to find**
	Object	**the requirements that can be negatively affected by unknown context events in the open world**
	Viewpoint	**from the Evolution Planner's viewpoint**
Question	**Q10**	Does the EVOLUTION PLANNER perform accurate inferences to find the requirements that can be negatively affected by unknown context events in the open world?
Metrics	**M39 - M42**	(**M39**) number of unknown context events, (**M40**) number of runs, (**M41**) number of inference rules in the knowledge base, and (**M42**) percentage of discovered affected requirements

Table 10.12: GQM model for the "accurate inferences to find the requirements that can be negatively affected by unknown context events in the open world from the EVOLUTION PLANNER's viewpoint" goal.

To this end, we purposely injected a set of context events that were not predefined at design time to simulate uncertainty in the open world (**M39**). We simulated performance decrease in some service operations (by manually modifying the execution times in the file with the log of context observations), and the unavailability of other operations.

Figure 10.10 summarizes the results of 16 runs (**M40**) with an increasing number of rules in the knowledge base (**M41**). Our approach found the affected requirements in 83.9% cases (**M42**). The number of discovered affected requirements is directly proportional to the number of rules.

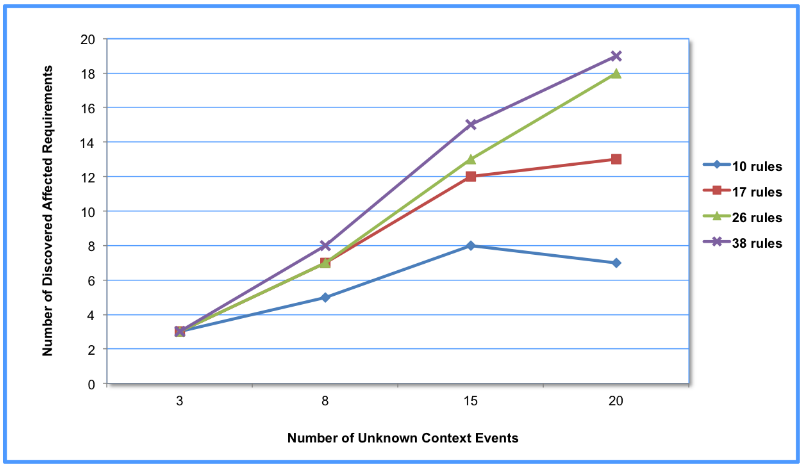


Figure 10.10: Number of discovered affected requirements by the Evolution Planner.

10.7 CONCLUSIONS

In this chapter, we have presented the evaluations that were carried out to confirm that the key pieces of our framework provide adequate support to the contributions of this thesis. These pieces are used in the `Design`, `Dynamic Adaptation`, and `Dynamic Evolution` phases of our framework.

The evaluated aspects were described in Section 10.1. Measurement was guided by the GQM paradigm. A brief introduction to this paradigm was presented in Section 10.2. Then, the computing infrastructure to carry out the experiments was described in Section 10.3.

The experiments that were carried out to validate the key pieces in the `Design Phase` of our framework were described in Section 10.4. The main conclusions are as follows:

- According to the GQM model in Section 10.4.1: The time required to generate the adaptation space with MOSKitt4SPL is very short.

- According to the GQM model in Section 10.4.2: In our approach, features are activated or deactivated by means of model-driven resolutions. Therefore, the adaptation paths between different variability model configurations (which describe service composition configurations) can be expressed in a declarative manner without the need for an exhaustive definition. Therefore, a small variability model and a set of resolutions defined in MOSKitt4SPL can describe a large adaptation space.

- According to the GQM model in Section 10.4.3: Our stability and semi-aliveness verification criteria, which run on GNU Prolog, are helpful to find anomalies and to avoid false positives in variability models and its configurations. Hence, problematic situations can be fixed at design time to avoid negative effects at runtime.

- According to the GQM model in Section 10.4.4: The execution times of the stability and semi-aliveness verification criteria on GNU Prolog is notable.

The experiments that were carried out to validate the pieces in the `Dynamic Adaptation Phase` of our framework were described in Section 10.5. The main conclusions are as follows:

- According to the GQM model in Section 10.5.1: In our proof-of-concept prototype, the Context Monitor is efficient to store context data for short periods of time without excessive execution time or memory problems. However, in industrial scenarios with a larger set of context observations, we propose to use other alternative approaches, such as a stream database.

- According to the GQM model in Section 10.5.2: MoRE-WS is efficient to carry out the dynamic adaptation of service compositions in terms of execution time, memory consumption, and CPU consumption. The analysis of context observations is processor efficient (20% of CPU consumption). Although MoRE-WS requires around 60% of CPU power during dynamic adaptations, this usage is just for a few seconds. Overall, the memory consumption is constant and does not have significant variations during dynamic adaptations.

- According to the GQM model in Section 10.5.3: MoRE-WS has a good performance and memory consumption under stress circumstances when multiple context conditions are fulfilled in very closed time frames.

The experiments that were carried out to validate the pieces in the `Dynamic Evolution Phase` of our framework were described in Section 10.6. The main conclusions are as follows:

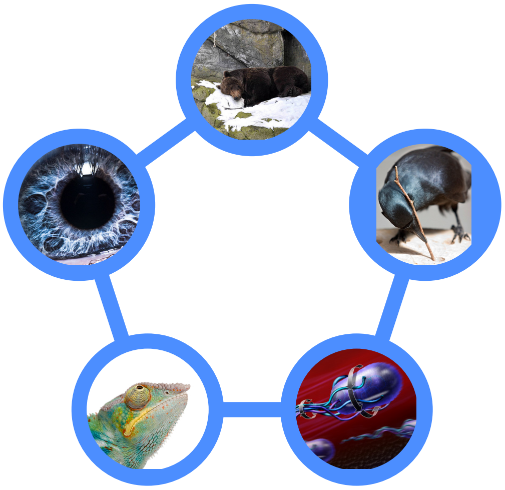
- According to the GQM model in Section 10.6.1: The Evolution Planner is efficient to carry out evolution planning. In turn, MoRE-WS is efficient to carry out dynamic evolutions in terms of execution time, memory consumption, and CPU consumption. The analysis of unknown context events takes around 20% of CPU consumption. A dynamic evolution takes around 40% of CPU consumption for a few seconds. Memory consumption for analyzing unknown context events and during dynamic evolutions tends to be constant.

- According to the GQM model in Section 10.6.2: The accuracy of the Evolution Planner in terms of the number of discovered requirements that can be negatively affected by unknown context events is directly proportional to the number of inference rules. Therefore, in order to increase the effectiveness of our approach in the open world, the number of rules in the knowledge base, and their related abstract requirements and surviving tactics have to be proportional. Although our approach does not solve uncertainty completely, it is an important step towards uncertainty management.

The results presented in this chapter encourage the following: 1) the key pieces of our framework give support to the contributions of this thesis; and 2) models at runtime are a feasible way to guide dynamic adjustments of autonomic service compositions. The experiments and results in this chapter are based on scientific publications (Alférez and Pelechano, 2013b; Alférez et al., 2013).

Part V

CLOSING REMARKS



Nature offers incredible examples of adaptation.

In this thesis, we propose a tool-supported framework to guide autonomic adjustments of context-aware service compositions in the closed and open worlds using models at runtime. As introduced in each part of this document, several examples of adaptation in nature have inspired our contributions.
Part V presents the closing remarks of our approach. This part has one chapter with the conclusions of this thesis and future work.

11

CONCLUSIONS AND FUTURE WORK

"The more original a discovery,
the more obvious it seems afterwards."

— Arthur Koestler

My Quest to Scientific Knowledge
Future Work
Codirected Master Thesis
International Collaboration
Publications
Contributions

Figure 11.1: Scope of Chapter 11.

The present work has introduced a tool-supported framework to guide autonomic adjustments of context-aware service compositions in the closed and open worlds using models at runtime. In this way, the modeling effort made at design time is not only useful for producing the service composition but also offers a rich base for autonomic behavior during execution. This work has resulted in innovative contributions, which have been presented in important forums. Moreover, the presented work has arisen interest in several research communities, which call for further work and international collaborations.

The remainder of this chapter is structured as follows (see Figure 11.1). Section 11.1 summarizes the contributions of this thesis. Section 11.2 describes the publications that have been produced throughout the development of this work. Section 11.3 describes international collaboration. Section 11.4 describes a codirected master thesis. Section 11.5 identifies future research work. Finally, Section 11.6 presents final remarks.

11.1 CONTRIBUTIONS

The main contribution of this work is a **tool-supported framework to guide autonomic adjustments of context-aware service compositions in the closed and open worlds using models at runtime**. This framework offers tools and techniques from design time to runtime for achieving autonomic service compositions by combining AC, MDE, and DSPLE. In particular, we have demonstrated that service compositions can make use of the knowledge in models at runtime to guide their autonomic behavior to face known and unknown context events.

The contributions of this thesis answer three research questions.

In Chapter 1, we proposed three research questions. In this section, we reexamine these research questions in order to describe the following contributions of this thesis:

1. The first contribution is **a set of models to support the dynamic adaptation and the dynamic evolution of service compositions**. This contribution is related to `Research Question 1`: How to design the autonomic behavior of context-aware service compositions by means of easy-to-understand models? In order to answer this question, several models are introduced in the `Design Phase` of our framework, which is described in Chapter 5 and Chapter 6.

   In order to support dynamic adaptations, we propose the following models. A *composition model* describes the service composition. A *variability model* describes the dynamic configurations of the service composition in terms of activation or deactivation of features. Since the initial composition model may lack support for variability, we propose to extend this model with variation points. In addition, we propose the creation of two additional supporting models. First, a *context model* formalizes collected context knowledge. Second, since changes in the variability model guide adaptations in the service composition, which is represented in the composition model, we propose a *weaving model* to connect these two models.

In order to support dynamic evolutions, we propose the creation of *tactic models* with the corrective actions to deal with uncertainty. Requirements are abstracted in a *requirements model*.

2. The second contribution is **a tool-supported software engineering approach for the development of context-aware service compositions from design time to execution**. This contribution is related to `Research Question 2`: How to implement a tool-supported software engineering approach for the development of context-aware service compositions from design time to execution? In order to answer this question, the proposed framework covers the construction of models and other supporting artifacts at design time, and the computing infrastructure to guide dynamic adjustments at runtime. Tool support is summarized as follows:

   - At the `Design Phase` of our framework, there are two tools that provide variability reasoning at design time: 1) the Configuration Generator automatically generates the adaptation space with all the possible configurations of the variability model. The Configuration Generator is implemented with MOSKitt4SPL[1]; and 2) the Verifier verifies the variability model and checks that the generated variability model configurations respect the constraints imposed by the variability model. This Verifier is not attached to any specific tool. Therefore, we have chosen FAMA-FW[2] and GNU Prolog[3] to implement the Verifier depending on the desired verification operations. We propose two SAS-oriented verification operations to ensure safe recompositions.

     From a methodological point of view, we propose two software process models. These process models can be used to create the models and other supporting artifacts for the dynamic adaptation and the dynamic evolution of service compositions. In Appendix B, we describe two tool-supported method contents based on SPEM 2.0 (OMG, 2008) for the creation of these process models.

     In Chapter 10, we demonstrated the following facts related to this contribution at design time: 1) the time required

1 http://www.pros.upv.es/m4spl
2 http://www.isa.us.es/fama
3 http://gprolog.univ-paris1.fr

to generate the adaptation space with MOSKitt4SPL is very short; 2) by defining activation or deactivation of features, the adaptation paths between different variability model configurations can be expressed in a declarative manner; 3) the verification process is helpful to find anomalies and to avoid false positives in variability models and its configurations; and 4) the execution times of the stability and semi-aliveness verification criteria on the variability model is notable.

- At the `Dynamic Adaptation Phase` of our framework, the models that are created in the `Design Phase` are used to guide the self-adaptation of the service composition in the closed world. To this end, we propose a Model-Based Reconfigurator, which is materialized by our MoRE-WS tool.

  MoRE-WS implements the components of the MAPE-K loop (IBM, 2006). First, it queries the context information that is collected by the Context Monitor. When a problematic event is detected, MoRE-WS uses a variability model for decision-making. The activation and deactivation of features in the variability model result in changes in the composition model. The composition model abstracts the workflows among service operations in the service composition. In turn, modifications in the composition model are reflected into the service composition by adding or removing fragments of WS-BPEL code, which are hot deployed in the Execution Engine. The Execution Engine uses the adapted WS-BPEL code to orchestrate the service composition. Our approach is transparent to this engine.

  In Chapter 10, we demonstrated the following facts related to this contribution: 1) in our proof-of-concept prototype, the Context Monitor is efficient to store context data for short periods of time without excessive execution time or memory problems; 2) MoRE-WS is efficient to carry out the dynamic adaptation of service compositions in terms of execution time, memory consumption, and CPU consumption; and 3) MoRE-WS has a good performance and memory consumption under stress circumstances when multiple context conditions are fulfilled in closed time frames.

3. The third contribution is **an approach based on models at runtime to dynamically evolve service compositions to protect expected requirements when facing unknown context events in the open world**. This contribution is related to `Research Question 3`: How to dynamically evolve service compositions to protect expected requirements when facing unknown context events in the open world? In order to answer this question, we propose the `Dynamic Evolution Phase` in our framework. This phase uses the knowledge in models at runtime for guiding the dynamic evolution of the service composition in the open world. In this phase, our EVOLUTION PLANNER is in charge of planning the evolution. In turn, MoRE-WS evolves the models at runtime and generates the evolved WS-BPEL code. MoRE-WS hot deploys the evolved WS-BPEL code on the EXECUTION ENGINE.

   In Chapter 10, we demonstrated the following facts related to this contribution: 1) the EVOLUTION PLANNER is efficient to carry out evolution planning. In turn, MoRE-WS is efficient to carry out dynamic evolutions in terms of execution time, memory consumption, and CPU consumption; and 2) the accuracy of the EVOLUTION PLANNER in terms of the number of discovered requirements that can be negatively affected by unknown context events is directly proportional to the number of inference rules.

This work demonstrates the feasibility of models at runtime to achieve autonomic service compositions.

The aforementioned contributions demonstrate that **it is feasible to use models at runtime to achieve autonomic service compositions**. Specifically, this work demonstrates that **the use of models at runtime has the following benefits**:

- The modeling effort made at design time is not only useful for producing the service composition but also provides a rich semantic base for autonomic behavior during execution.

- Since models are causally connected to the underlying service composition, models provide up-to-date information to drive subsequent adaptation or evolution decisions.

- The same model representations that are used at design time are kept at runtime. This avoids the need for technological bridges, making it possible to apply the same technologies used at design time to manipulate models at runtime.

We hope that the aforementioned contributions encourage researchers and practitioners to use models at runtime in order to achieve autonomic service compositions in computing infrastructures that combine Web services, AC, MDE, and DSPLE.

11.2 PUBLICATIONS

The contributions presented in this thesis are supported by several publications in relevant forums as depicted in Figure 11.2.

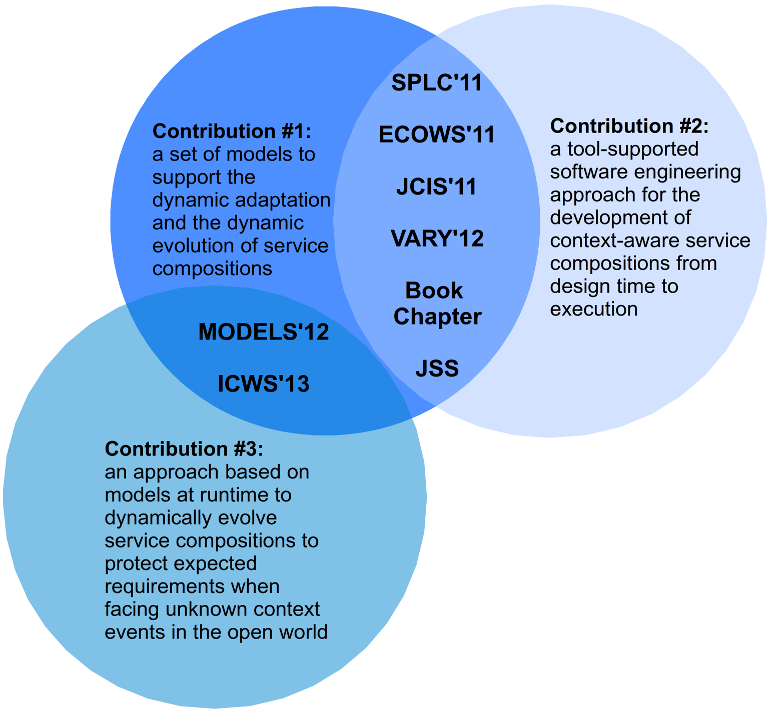

Contribution #1:
a set of models to support the dynamic adaptation and the dynamic evolution of service compositions
SPLC'11
ECOWS'11
JCIS'11
VARY'12
Book Chapter
JSS
Contribution #2:
a tool-supported software engineering approach for the development of context-aware service compositions from design time to execution
MODELS'12
ICWS'13
Contribution #3:
an approach based on models at runtime to dynamically evolve service compositions to protect expected requirements when facing unknown context events in the open world

Figure 11.2: The contributions of this thesis are supported by several publications.

Our strategy was to validate our approach in top conferences in Web services (ICWS and the European Conference on Web Services (ECOWS)), MDE (International Conference on Model Driven Engineering Languages and Systems (MODELS)), and DSPLE (International Software Product Line Conference (SPLC)). In addition to these conferences, we published our work in the Journal of Systems and Software

(JSS), which is a relevant journal with Journal Citation Reports (JCR). Other publications are as follows: a book chapter, a paper presented at the Jornadas de Ciencia e Ingeniería de Servicios (JCIS) (a national conference in Service Computing), and another paper presented at VARiability for You (VARY) (a workshop). AC is a crosscutting concern that is covered in all the publications. It is important to notice that the papers presented at SPLC and ECOWS have been referenced in articles of relevant journals (Bencomo et al., 2012; dos Santos Rocha and Fantinato, 2013).

We present our publications as follows. Each publication is organized under a category (e.g. journal or conference) and in chronological order. The position of the name of the author of this thesies is used as an indicator of the degree of contribution in each publication. In addition, each publication has an associated quality level (e.g. the JCR Impact Factor for journals[4] and the Computing Research and Education Association of Australasia (CORE) ranking[5] for conferences). Under each publication, we also provide a short description of the relevance of the forum where it was published.

Journal

Alférez, G. H., Pelechano, V., Mazo, R., Salinesi, C., Diaz, D., 2013. Dynamic adaptation of service compositions with variability models. Journal of Systems and Software (*accepted article in press*).

- JSS publishes papers covering all aspects of programming methodology, software engineering, and related hardware-software-systems issues. According to the JCR, this journal has a 2012 Impact Factor of 1.135 and a 5-Year Impact Factor of 1.322. It is ranked as follows: 1) 41 out of 105 journals in the "*Computer Science, Software Engineering*" category. This journal is in the second quartile (Q2) of this category; and 2) 30 out of 100 journals in the "*Computer Science, Theory & Methods*" category. This journal is in the second quartile (Q2) of this category.

Conferences

- **Alférez, G. H.**, Pelechano, V., 2011a. Context-aware autonomous web services in software product lines. In: Proceedings of the

4 http://thomsonreuters.com/journal-citation-reports
5 http://core.edu.au

2011 15th International Software Product Line Conference. SPLC '11. IEEE Computer Society, Washington, DC, USA, pp. 100–109.

  - SPLC is the most prestigious and leading forum for researchers, practitioners, and educators in the field of SPLE. This paper was referenced in two important journal articles:
    - *"A View of the Dynamic Software Product Line Landscape"* in COMPUTER (Bencomo et al., 2012).
    - *"The Use of Software Product Lines for Business Process Management: A Systematic Literature Review"* in INFORMATION AND SOFTWARE TECHNOLOGY (dos Santos Rocha and Fantinato, 2013).

    This fact is a strong evidence of the positive impact of our work in the SPLE community.
  - Conference Acceptance Ratio: **29%**.

- **Alférez, G. H.**, Pelechano, V., 2011b. Systematic reuse of web services through software product line engineering. In: 2011 9th IEEE European Conference on Web Services. ECOWS '11. pp. 192–199.
  - ECOWS was the premier conference series for both researchers and practitioners to discuss the latest advances in the state of the art and practices of Web services and to foster cross-European collaboration and exchange. In 2012, ECOWS and ICWS formally merged into ICWS. Ever since, the Services Computing community aims to combine the efforts and focus on one international forum: ICWS.

    This paper was referenced in an article entitled *"The Use of Software Product Lines for Business Process Management: A Systematic Literature Review"* in a relevant journal (dos Santos Rocha and Fantinato, 2013).
  - CORE 2008: **C**. Conference Acceptance Ratio: **24%**.

- Ayora, C., **Alférez, G. H.**, Torres, V., Pelechano, V., 2011. Procesos de negocio auto-adaptables al contexto. In: Álvarez, P., Del Arco, J., Luaces, M. (Eds.), Actas de las VII Jornadas de Ciencia e Ingeniería de Servicios. JCIS '11. Universidade da Coruña, pp. 147–160.

  URL http://www.sistedes.es/.../2011.pdf.

  - JCIS is the main Spanish conference on Services Computing.

- **Alférez, G. H.**, Pelechano, V., 2012b. Dynamic evolution of context-aware systems with models at runtime. In: France, R., Kazmeier, J., Breu, R., Atkinson, C. (Eds.), Model Driven Engineering Languages and Systems. Vol. 7590 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 70–86.
  - MODELS is the premier venue for the exchange of innovative ideas and experiences of model-based approaches in the development of complex systems.
  - CORE 2008: **B**. Conference Acceptance Ratio: **23%**.

- **Alférez, G. H.**, Pelechano, V., 2013b. Facing uncertainty in web service compositions. In: Proceedings of the 20th IEEE International Conference on Web Services. ICWS '13. pp. 219–226.
  - ICWS is a prime international forum for both researchers and industry practitioners to exchange the latest fundamental advances in the state of the art and practice of Web-based services, identify emerging research topics, and define the future of Web-based services.
  - CORE 2008: **A**. Conference Acceptance Ratio: **19%**.

Book Chapter

Pelechano, V., Fons, J., **Alférez, G. H.**, Gómez, M., 2013. Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas. RA-MA Editorial, Ch. Modelos en Tiempo de Ejecución, pp. 529–551.

- This chapter is part of a book written by prestigious authors in the area of MDE. Editorial RA-MA publishes high-quality technical books whose authors are mostly from Spain.

Workshop

Ayora, C., **Alférez, G. H.**, Torres, V., Pelechano, V., 2012. Applying CVL to business process variability management. In: Proceedings of VARiability for You. MODELS 2012. pp. 24–29.
URL http://vary2012.irisa.fr/VARY2012Proceedings.pdf.

- VARY is a workshop associated with MODELS 2012. This workshop influences the ongoing standardization efforts within OMG to establish a common variability language, where the organizers of this workshop are also participating in the consolidated submission team.

11.3 INTERNATIONAL COLLABORATION

The author of this thesis led an international collaboration with researchers at the CRI.

An international collaboration with researchers at the CRI[6], Université Paris-1 Panthéon-Sorbonne, France, enriched the preparation of this thesis. The CRI has a long tradition of teaching, industry collaboration, and research in the domains of information systems engineering, requirements engineering, and service systems. The group is under the direction of Professor Camille Salinesi.

Thanks to this collaboration, we enriched the VERIFIER in our framework with two SAS-oriented verification operations. As a result, it is possible to prevent at design time undesirable dynamic adaptations. To this end, the research work of Professor Raúl Mazo and Professor Daniel Diaz was specially important. Although the CRI had previously used CP to verify feature models in several domains (Mazo et al., 2012), it is the first time that this approach was used to verify the variability model configurations to be used at runtime.

As an important result of this collaboration, we published a joint journal article in which the author of this thesis was the leader of the project (Alférez et al., 2013). This collaboration continues with the extension of the set of verification operations at runtime to avoid inconsistent service recompositions in the open world. For example, settling time will be verified to avoid overheads. Also, changes in the service composition when dealing with unknown context events will be verified during execution to avoid a negative impact in the expected requirements.

11.4 CODIRECTED MASTER THESIS

The following master thesis was codirected in the context of this work in order to explore some ideas. This thesis is referenced in Chapter 5:

De La Fuente, J., 2012. Herramienta para la generación y despliegue de composiciones de servicios web mediante modelos BPMN. Master's thesis, Universitat Politécnica de Valéncia.
URL http://riunet.upv.es/handle/10251/18022.

6 http://www.univ-paris1.fr/centres-de-recherche/cri

11.5 FUTURE WORK

The work proposed in this thesis can be extended in several ways. The following list summarizes the research directions that are planned to continue this work:

- **Apply Our Solution in Other Domains:** In our approach, models at runtime guide the dynamic adjustment of autonomic service compositions. Nevertheless, we believe our approach can be easily extended to other domains, which require dynamic adjustments as well. The main idea is to reflect changes in models at runtime on the system architecture with our computing infrastructure. For instance, our framework could be applied to smart cities, where service-based systems need to deal with the variability of the infrastructure, operations, and people. To this end, we have started a collaboration with researchers at the Center for Information Technology, Fondazione Bruno Kessler[7], Italy. Moreover, our model-driven approach could be used to guide dynamic adjustments in robots. To this end, there is a possible collaboration with researchers at the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center[8].

- **Use Models at Runtime to Migrate Running Instances:** In our approach, dynamic adjustments are carried out in the composition schema. New instances run according to the latest version of the composition schema. We believe that the knowledge in the composition model can be used at runtime to migrate the running instances according to the latest version of the composition schema. This is a big research area with several challenges. For example, integrity of data in migrated transactions has to be ensured, and instances should be migrated efficiently and safely (i.e., without errors).

- **Proactive Dynamic Adaptations:** Our approach will be extended to proactively discover problematic context events and carry out the necessary changes in the architecture. One way to carry out proactive dynamic adaptations is with machine learning. A recent definition of machine learning is the following: "A computer program is said to learn from experience *E* with respect

7 http://www.fbk.eu
8 http://www.nasa.gov/centers/goddard/home/index.html

to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" (Mitchell, 1997). Since the Context Monitor collects data constantly, and data logs can be kept from different autonomic systems, we believe that machine learning can learn from these data to make further decisions.

- **Apply Alternative Methods for Reasoning about Unknown Context Events:** In our current solution, the Evolution Planner uses highly-generic rule premises to find out whether an unknown context event can affect a requirement or not. Other methods, such as data mining could be applied to discover unknown properties on the collected context data. Data mining combines tools from statistics and AI with database management to analyze large digital collections, known as data sets.

- **Concurrency:** In our approach, MoRE-WS carries out a single adjustment at a time (i.e., one adjustment after the other). An interesting area of future work is to study the potential of concurrent dynamic adjustments to save operational time and offer faster results. In this area, it will be important to guarantee the integrity of the resulting configurations of the service composition after applying concurrent adjustments.

- **Verification at Runtime:** In this work, we demonstrated that verification at design time is very useful to prevent undesirable errors during execution. We believe verification should be extended at runtime in the closed and open worlds as follows:

  - **Verification of Dynamic Adaptations:** An interesting area of future work is to verify that the service composition self-adapts only when it is necessary, not due to short exceptional situations (e.g. a passing short transmission delay). Since dynamic adaptations consume resources in the computer infrastructure, we argue that unnecessary adaptations should be avoided as much as possible.

    In addition, we are planning to carry out consistency checking at runtime to guarantee that the current configuration of the variability model is semantically consistent with the composition model. To this end, we are currently studying the possibility to extend the approach presented in (Alférez et al., 2011; Alférez, 2012) from design time to runtime. Basically, this approach offers an automated technique to detect intermodel semantic inconsistencies between the variability model and other models.

Also, we think it is important to create a catalog of verification operations for particular situations. For example, this catalog may include the verification operation to avoid falling into an "infinite loop" of adaptations. In this exceptional scenario, the system can replace service operation α with service operations β and γ at runtime. Nevertheless, service operations β and γ do not reach the expected SLA. Therefore, the system reactivates service operation α, which does not offer an optimal solution. This loop can continue back and forth, maybe for a long time. This situation can decrease the quality of the whole system.

– **Verification of Dynamic Evolutions:** The service composition can grow or shrink at runtime with tactical functionalities, which are added or removed during dynamic evolutions. There are several questions to be answered when recompositions are verified at runtime. For example: Does verification of dynamic evolutions at runtime imply high overheads? Does a merged tactic accomplish its objective at runtime? or Does the quality of the requirement continue decreasing? If a merged tactic is not preserving a requirement as expected, When does another tactic has to be tried? How to verify that the service composition does not grow excessively with a large amount of merged tactics, which can make the system complex or slow? In this way, some tactics could be automatically removed when the service composition has reached an stable state.

11.6 MY QUEST TO SCIENTIFIC KNOWLEDGE

The contributions presented in this thesis are the result of a sequence of key events in my life. Some may say things happen as the result of mere chance. However, it is my belief events are designed with a purpose.

My interest in software design is not new. It all started when I was a kid and my parents used to encourage my brother and I to design sophisticated artifacts with LEGO blocks. As a result, both of us focused our PhD theses on software modeling. Also, we work in areas related to MDE in academia.

My interest in software engineering research has been a quest on four continents. This journey started as a research assistant at the university where I studied computer science engineering in my country. Since that time, research has opened my mind to wonderful new perspectives in science and industry. Then, I moved to Asia where I started working as a lecturer at a university with a vibrant international environment. The influence of several colleagues and master's professors, and the impact of research and technology in Asia increased my passion for research. My first research publications saw light at that time.

After working in different positions in Asia and as a sessional lecturer for a college in Australia, I went to work as a lecturer for undergraduate and graduate software engineering courses in a university in North America. I decided to study a PhD in computer science at the Universitat Politècnica de València after being inspired by an article about models at runtime written by Professor Vicente Pelechano and his colleagues.

H. G. Wells, a prolific writer, once said: "Adapt or perish, now as ever, is nature's inexorable imperative." Adaptation in nature is not a luxury but a necessity to survive. It is also the case in critical service compositions, which run in complex contexts. At the metalevel, I based the ideas presented in this thesis on naturalistic observations. As a result, our solution to achieve autonomic service compositions is as lifelike as possible.

Although this thesis has come to an end, research on model-driven autonomic service compositions is not a closed topic. In fact, it is full of interesting future research directions that I plan to carry on with colleagues and students. Although I have acquired new knowledge and achieved my personal goals thus far, my quest to scientific knowledge will continue.

Part VI

APPENDICES

A

IMPLEMENTATION DETAILS

This section presents the implementation details of our tools that support the dynamic adjustment of service compositions. The remainder of this appendix is structured as follows: Section A.1 presents the implementation details of the Context Monitor. Section A.2 presents the implementation details of MoRE-WS for the dynamic adaptation of service compositions in the closed world. Finally, Section A.3 presents the implementation details of the Evolution Planner and MoRE-WS for the dynamic evolution of service compositions in the open world.

A.1 CONTEXT MONITOR OPERATIONS TO OBSERVE THE CONTEXT

The Context Monitor observes the behavior of the service operations, which run in the context. Our implementation of the Context Monitor materializes the `Monitor` component of the MAPE-K loop as described in Figure A.1. The steps that are carried out on and by the Context Monitor are described in the following subsections.

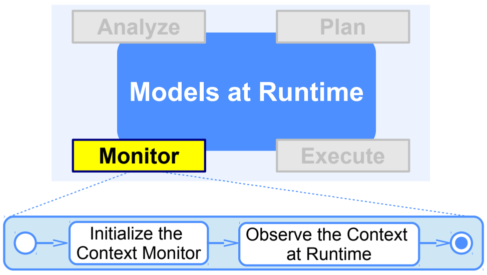

Analyze
Plan
Models at Runtime
Monitor
Execute
Initialize the Context Monitor
Observe the Context at Runtime

Figure A.1: Implementing the `Monitor` component of the MAPE-K loop with the Context Monitor.

Step 1: Initialize the Context Monitor

[1. Initialize the Context Monitor]

↓

2. Observe the Context at Runtime

In this step, the Context Monitor is initialized before execution. The actions that are carried out in this step are as follows:

1. The systems analyst specifies requests for *all* the service operations that will be observed at runtime. The objective of this step is to count on a complete view of the behavior of the service operations that could be used at runtime, both commonalities and variabilities.

2. The systems analysts defines the frequency of context observations.

3. As soon as the Context Monitor starts running, it creates an XML file to store the observed context information. This document has the following mandatory attributes: 1) the `ID` attribute keeps the identification of the observation. This is an increasing sequential integer; 2) the `name` attribute keeps the name of the service operations that are observed; and 3) the `time stamp` attribute keeps the time in which the observation was recorded. This initial set of attributes can be extended with other attributes, such as availability and execution time. XML was chosen to implement the file with the log of context observations in order to structure the information and count on efficient queries at runtime.

Step 2: Observe the Context at Runtime

1. Initialize the Context Monitor

↓

[2. Observe the Context at Runtime]

At runtime, the Context Monitor sends requests to service operations. These requests work as sensors that observe services' behavior. A fragment of the Java code of this operation is described in Listing A.1. Requests are sent continually (see line 2), periodically (see line 4), and sequentially (see the blocks of code starting in lines 6 and 18). The `updateXML` method in line 13 is in charge of updating the XML file according to the information managed by the observation-related methods (see line 10). The `ID` of the observation is increased sequentially after each observation (see line 16).

Listing A.1: A fragment of the Java code to observe the context.

```
 1 // An infinite loop:
 2 while(true){
 3     // Triggering the requests according to the frequency of observation:
 4     Thread.sleep(timer);
 5 
 6     // Observing the first service operation:
 7     System.out.Println("Observing the first service operation");
 8 
 9     // Sending a request to the first service:
10     observingFirstService();
11 
12     // Updating the XML file:
13     updateXML(increasingID, "FirstService", FirstServiceAvailability,
          FirstServiceExecutionTime);
14 
15     // Increasing the observation ID sequentially:
16     increasingID = increasingID + 1;
17 
18     // Observing the second service operation:
19     System.out.println("Observing the second service operation");
20     //...
21 }
```

The steps that are followed by the implementation of every service observation method (such as the ones in line 10), is as follows: 1) an object of the class that implements the Web service client is instantiated; 2) the chronometer to measure the execution time is initialized with the current time; 3) the service operation is invoked through the client; 4) the resulting execution time is the result of the subtraction of the current execution time (after the service operation has been invoked) minus the time when the chronometer was initialized; and 5) a thrown exception indicates that the service operation is unavailable.

A.2 MORE-WS OPERATIONS FOR DYNAMIC ADAPTATIONS

This section describes the implementation details of MoRE-WS to support the dynamic adaptation of service compositions in the closed world. Specifically, this section shows how MoRE-WS materializes the `Analyze`, `Plan`, and `Execute` components of the MAPE-K loop to guide adaptations.

A.2.1 *Analyzing the Context*

The XML file that is updated with the measures taken from the context needs to be queried to determine if any change has to be made in the service composition. This task is in charge of MoRE-WS, which periodically queries this file to find new contextual information. In order to count on a constant representation of the context, MoRE-WS periodically updates the context model according to the information in the XML file. Afterwards, MoRE-WS evaluates the values in this model to find out if any context condition has been fulfilled.

The operations that MoRE-WS implements to insert events into the context model and to reason about context conditions are described as follows. These two operations support the `Analyze` component of the MAPE-K loop (see Figure A.2).

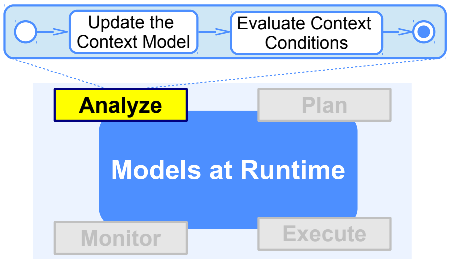


Figure A.2: Implementing the `Analyze` component of the MAPE-K loop with MoRE-WS.

Step 1: Update the Context Model

*[**1. Update the Context Model]***
↓
2. Evaluate Context Conditions

In order to count on a constant representation of the context, MoRE-WS updates the context model with the log of context observations in the updated XML file. To this end, MoRE-WS uses the Java code in Listing A.2. The key idea in this algorithm is to fragment the data taken from the XML file into partitions. As a result, only the partition with the latest set of observations is analyzed at runtime (old data is not analyzed every time in order to save resources). In order to create the data partitions, the algorithm keeps the record of the latest observation in the XML file (see line 28). In iterations greater than the first one (iteration greater than zero), the value of the first observed service operation in a new data partition is calculated from the last observation in the previous iteration – last observation plus one in order to avoid reading the same last `ID` repeatedly (see line 21).

Listing A.2: A fragment of the Java code for reading the XML file with the set of observed context events and inserting these values into the context model.

```
 1 // Creating a new context model when the system starts in order to avoid
       inconsistencies with previous data:
 2 createContextModel();
 3 
 4 // An infinite loop:
 5 while(true){
 6     // Updating the context model periodically:
 7     Thread.sleep(timerForContextModelUpdate);
 8 
 9     // Defining two objects to keep the information of the first and last
           observed service operation in a partition. The WebService class
           defines getter and setter methods for all the attributes
          related to a service operation:
10     WebService firstObservedServiceOperation = null;
11     WebService lastObservedServiceOperation = null;
12 
13     // Putting all the observations from the XML file with context
          observations into a list:
14     List<WebService> readSet = readXML("file.xml");
15 
16     if(iteration = 0){
17         // If it is the first iteration, MoRE-WS gets the first
               observation in the list of WebService objects (i.e., with
               index 0):
18         firstObservedServiceOperation = readSet.get(0);
19     }else{
20         // Otherwise, the first observation is the last observation of
               the previous iteration plus one:
21         firstObservedServiceOperation = readSet.get(
               getLastIDFromPreviousIteration() + 1);
22     }
23 
24     // Getting the last observation. Since the size() method returns the
          number of elements in the list, it is necessary to deduct one in
           order to get the last index from the list:
25     lastObservedServiceOperation = readSet.get(readSet.size() - 1);
26 
27     // Storing the ID of the last observation into a file:
28     storeLastIDOfThisIteration(lastObservedServiceOperation.getid());
29 
30     // Inserting the set of observations in the partition into the context model:
31     for(int i = firstObservedServiceOperation.getid()); i < lastObservedServiceOperation.getid(); i++){
32         WebService ws = readSet.get(i);
33         insertIntoContextModel(ws.getname(), ws.getavailability(), ws.getexectime());
34     }
35 }
```

In the insertIntoContextModel method, which is invoked in line 33 of Listing A.2, MoRE-WS uses SPARQL[1] to update the context model according to new context events. In order to automatize the triggering of SPARQL INSERT operations into the context model, MoRE-WS uses the Java libraries of Jena[2]. Listing A.3 shows the implementation of the insertIntoContextModel method. This code updates the isAvailable and executionTime datatype properties.

Listing A.3: A method in Java to trigger SPARQL INSERT operations through Jena.

```
private static Model model = null;
String contextModel = "contextModel.rdf";

// This method receives the service's name, availability, and execution time, and inserts these values into the context model:
public void insertIntoContextModel(String serviceName, Boolean availability, long executionTime){

    try{
        // Inserting availability data into the context model:
        String pref = "PREFIX web: <http://my.ontology#>";
        String queryString1 = pref + NL + " INSERT DATA {web:"+serviceName+" web:isAvailable '"+availability +"'}";
        UpdateRequest request1 = UpdateFactory.create(queryString1);
        UpdateAction.execute(request1, model);

        // Inserting execution time data into the context model:
        String queryString2 = pref + NL + " INSERT DATA {web:"+serviceName+" web:executionTime "+executionTime +"}";
        UpdateRequest request2 = UpdateFactory.create(queryString2);
        UpdateAction.execute(request2, model);

        // Other datatype properties may be updated here according to the input parameters of this method.
```

1 http://www.w3.org/TR/rdf-sparql-query
2 http://jena.apache.org

```

        // Saving the context model into the system:
        FileOutputStream fout = new FileOutputStream(contextModel);
        model.write(fout);
    }catch(Exception e){
        System.out.println("Error: "+e);
    }
}
```

For instance, a fragment of the generated context model in our case study (for the `Barnes & Noble Books` service operation) is shown in Listing A.4. Datatype properties are kept for all the observations in the partition for this operation (e.g. in partition η, the `isAvailable` datatype property is TRUE in the first observation – in line 3 – and FALSE in a subsequent observation – in line 5). This approach helps to analyze whole sets of data, such as by calculating the average execution time of all the observations in a partition. In order to avoid inconsistencies with old context observations in previous partitions, all the elements in the context model are deleted and then refreshed with the context observations in the latest partition.

Listing A.4: A fragment of the generated context model for the case study.

```
1 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:j
      .0="http://my.ontology#">
2     <j.0:WebService rdf:about="http://my.ontology#
          BarnesAndNobleBooksService">
3         <j.0:isAvailable>true</j.0:isAvailable>
4         <j.0:executionTime rdf:datatype="http://www.w3.org/2001/XMLSchema
              #integer">38</j.0:executionTime>
5         <j.0:isAvailable>false</j.0:isAvailable>
6     </j.0:WebService>
7 </rdf:RDF>
```

Step 2: Evaluate Context Conditions

1. Update the Context Model

↓

[2. Evaluate Context Conditions]

Context conditions are evaluated just after a partition of context observations have been inserted into the context model. MoRE-WS uses the Java libraries of Jena to automatize the analysis of the collected context information. Specifically, MoRE-WS triggers SPARQL ASK queries on the context model.

The fragment of code in Listing A.5 shows the evaluation of two context conditions, `serviceIsUnavailable` in line 22 (i.e., the service operation is unavailable) and `serviceHasHiExecTime` in line 23 (i.e., the service operation has an execution time greater than 1,500 milliseconds). For instance, the operation in line 22 goes through the set of observations that have been made on the `isAvailable` datatype property. The result is TRUE if any context observation states that `isAvailable` is FALSE (see lines 11-13).

Just after the context conditions have been analyzed for the whole set of observed context events in the current partition, MoRE-WS checks the private variables that keep the values of SPARQL ASK queries in lines 2 and 3 of Listing A.5. These variables are accessed through public methods. If any of these variables is TRUE, then a dynamic adaptation is required.

Listing A.5: A method in Java to trigger SPARQL ASK queries through Jena.

```
1  // Private variables that keep the results of SPARQL ASK queries:
2  private static Boolean serviceIsUnavailable = true;
3  private static Boolean serviceHasHiExecTime = true;
4
5  public void analyzeContextConditions(String serviceOperation){
6
7      String pref = "PREFIX web: <http://my.ontology#>";
8      String pref2 = "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>";
9
10     // Is serviceOperation unavailable?
11     String queryString0 = pref + NL + " ASK { web: " + serviceOperation
           + " web:isAvailable 'false' }";
12     Query query0 = QueryFactory.create(queryString0);
13     QueryExecution qexec0 = QueryExecutionFactory.create(query0, model);
14
15     // Is serviceOperation's execution time > 1500?
16     String queryString1 = pref + NL +pref2 + NL + " ASK { web:" +
           serviceOperation + " web:executionTime ?executionTime. FILTER
               (xsd:int(?executionTime) >1500)}";
17     Query query1 = QueryFactory.create(queryString1);
18     QueryExecution qexec1 = QueryExecutionFactory.create(query1, model);
19
20     try{
21         // Executing queries for context conditions:
22         serviceIsUnavailable = qexec0.execAsk();
23         serviceHasHiExecTime = qexec1.execAsk();
24
25         // Printing out the results:
26         System.out.println("is " + serviceOperation + "
               serviceIsUnavailable? " + serviceIsUnavailable);
27         System.out.println("is the execution time of " +
               serviceHasHiExecTime + " > 1,500? " + serviceHasHiExecTime);
28
```

```
    }finally{
    // Closing the connections:
        qexec0.close();
        qexec1.close();
    }
}
```

A.2.2 *Planning the Adaptation*

MoRE-WS carries out the following steps to plan the adaptation of the service composition when an adaptation has been requested (i.e., after a context condition has been fulfilled). These steps support the `Plan` component of the MAPE-K loop (see Figure A.3).

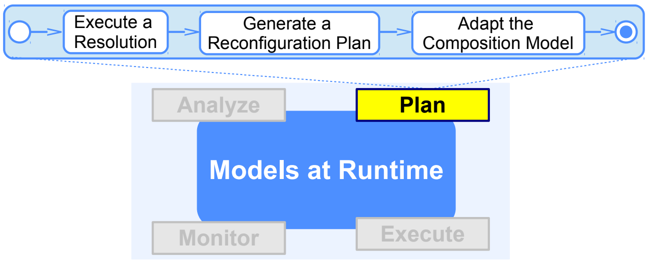


Figure A.3: Implementing the `Plan` component of the MAPE-K loop with MoRE-WS.

Step 1: Execute a Resolution

In order to execute a resolution, MoRE-WS carries out the following actions:

[1. Execute a Resolution]

↓

2. Generate a Reconfiguration Plan

↓

3. Adapt the Composition Model

1. MoRE-WS looks for the resolution that is triggered by the context condition. To this end, MoRE-WS gets the set of all the possible transitions (or resolutions) that are related to the current configuration in the adaptation space (see Listing A.6).

   Listing A.6: A method in JAVA to get the transitions that are related to the current configuration.

```
public ArrayList<Transition> getTransitionsToCheck(){

    ArrayList<Transition> transitionsToCheck = new ArrayList<
        Transition>();

    // Getting all the transitions that are related to the current
        state (configuration):
    EList<Transition> transitions = this.getCurrentState().
        getStateTransitions();

    for(Transition transition : transitions){
        transitionsToCheck.add(transition);
    }
    return transitionsToCheck;
}
```

With the information about transitions, MoRE-WS gets the specific transition that moves to a new configuration of the feature model. Every transition in the adaptation space encapsulates the name of the resolution and the name of the context condition that triggers it. Therefore, MoRE-WS uses the information of the context condition that has been fulfilled in order to find the resolution to be triggered. This operation is depicted in Listing A.7.

Listing A.7: A method in Java to get the specific transition (resolution) in the adaptation space to move to a new configuration.

```
for(Transition transition : transitionsToCheck){
    if(esp.checkCondition(eventOccurred.getEventKey(),transition.
        getResolution().getCondition().getDescription())){
            smManager.fireTransition(transition);
    }
}
```

2. MoRE-WS triggers a resolution (see Listing A.8). In other words, it triggers a transition in the adaptation space to move from the current configuration of the feature model to a new one in order to face a context condition. Line 16 in Listing A.8 is key to apply the transition to a new configuration. The `applyTransition` method, which is invoked in this line, has two parameters: 1) an object of the `ConfigurationModel` class that keeps the information of the current configuration (i.e., active and inactive features in the feature model); and 2) an object of the `Transition` class that indicates the transition to move to.

Listing A.8: A method in Java to trigger a resolution to a new configuration in the adaptation space.

```java
public void fireTransition(Transition tr) throws Exception{

    // Returning a ServiceReference object for a service that implements and was registered under the specified class:
    ServiceReference<IMoREFeatureModelConfigurationAdapter> sr = this.theContext.getServiceReference(IMoREFeatureModelConfigurationAdapter.class);
    if(sr!=null){
        // Returning the service object referenced by the specified ServiceReference object (sr):
        IMoREFeatureModelConfigurationAdapter fmcAdapter = this.theContext.getService(sr);

        // In order to count on the representation of the current configuration of the feature model before the adaptation, the createModelCopy method copies the current configuration of the feature model:
        this.lastConfiguration = createModelCopy(this.currentConfiguration);

        // Checking that the transition moves to a new configuration (state) in the state machine:
        if(!tr.getTransitionTo().equals(tr.getTransitionFrom())){

            // Applying the transition on the state machine:
            this.currentConfiguration = fmcAdapter.applyTransition(this.currentConfiguration, tr);
        }
        // Updating the following information when the state machine changes:
        this.lastState = this.currentState;
        this.currentState = tr.getTransitionTo();
        this.machineChanged = !this.lastState.equals(this.currentState);
        this.lastTransition = tr;
    }else{
        throw new Exception("No feature model configuration adapter was found");
    }
}
```

The `applyTransition` method in Listing A.9 applies the transition (i.e., applies the resolution) in the adaptation space to move from the current configuration in the adaptation space to a new configuration. The `updateFeatureState` method is invoked in line 11 to activate/deactivate features in the feature model according to the actions (activations/deactivations) of features that are returned in line 7.

Listing A.9: A method in Java to apply a transition to move to a new configuration of the feature model.

```
1  public ConfigurationModel applyTransition(ConfigurationModel
        cmToAdapt, Transition transitionToAply){
2
3      ConfigurationModel cmAdapted = cmToAdapt;
4      ArrayList<Feature> presentFeatures = new ArrayList<Feature>();
5
6      // Returning the list of activations and deactivations of
           features with the getActions method:
7      EList <Action> actionsToAply = transitionToApply.getResolution
           ().getActions();
8      for(Action action : actionsToAply){
9
10         // Updating the current configuration of the feature model
               with the updateFeatureState method:
11         cmAdapted = updateFeatureState(cmAdapted, action.getFeature
               (), action.getActionType());
12     }
13     return cmAdapted;
14 }
```

Listing A.10 shows the implementation of the `updateFeatureState` method. This method receives the configuration of the feature model to be adapted, a feature to be updated, and an action to be carried out (i.e., activation or deactivation). The `setState` method, which is invoked in lines 6 and 8, changes the states of the features.

Listing A.10: A method in Java to update the states of features.

```
1  private ConfigurationModel updateFeatureState(ConfigurationModel
        cmToAdapt, Feature featureToUpdate, ActionType actionToTake){
2
3      for(FeatureState fs:cmToAdapt.getFeatureStates()){
4          if(fs.getFeature().equals(featureToUpdate)){
5              if(actionToTake.getValue() == ActionType.ACTIVATE_VALUE
                   ){
6                  fs.setState(FeatureStateType.ACTIVE);
7              }else{
8                  fs.setState(FeatureStateType.DEACTIVE);
9              }break;
```

```
        }
    }
    return cmToAdapt;
}
```

Finally, MoRE-WS keeps a log of the changes that were carried out in the adaptation space and in the configuration of the feature model (see Listing A.11).

Listing A.11: Keeping a log of the changes that are carried out in the configuration of the feature model and in the state machine.

```
// If the state machine has changed:
if(smManager.hasTheMachineChanged()){
    DiffModel differencesModel = DiffService.doDiff(MatchService.
         doMatch((EObject)smManager.getLastConfiguration(),(EObject
         )smManager.getCurrentConfiguration(), Collections.<String,
         Object>emptyMap()), false);
    FeatureModelConfigurationChanges thisSMChanges=getDifferences(
         differencesModel);
    thisSMChanges.setStateMachineModelName(smManager.
         getStateMachineName());
    this.systemChanges.add(thisSMChanges);

    // Keeping the following information:
    broadcastEvent(new LogConfigurationChangeEvent(thisSMChanges,
         smManager.getLastState().getName(), smManager.
         getCurrentState().getName(), smManager.
         getLastConfiguration(), smManager.getCurrentConfiguration
         (), smManager.getLastTransitionApplied().getResolution()))
         ;
}
```

Step 2: Generate a Reconfiguration Plan

In this step, MoRE-WS creates a reconfiguration plan, which contains a set of reconfiguration actions to adapt the composition model according to the new configuration of the feature model. First of all, MoRE-WS checks the activations and deactivations of features that have been carried out in the new configuration of the feature model (see lines 12-14 and 22-24 in Listing A.12). According to the features that have been activated or deactivated in the feature model, MoRE-WS calculates the variant models to be bound/unbound from the base composition model. This task is in charge of the `getTargetModelElements`

1. Execute a Resolution

↓

[2. Generate a Reconfiguration Plan]

↓

3. Adapt the Composition Model

method (which is invoked in lines 17 and 27). The `setIncrements` and `setDecrements` methods in these lines fill two lists with the resulting variants to be incremented ($CM\triangle$) and decremented ($CM\nabla$) from the base composition model, respectively.

Listing A.12: A method in Java to reckon the increments and decrements in the composition model.

```java
1  private TargetModelChanges reckonIncrementsAndDecrements(
       FeatureModelConfigurationChanges configurationChanges){
2
3      TargetModelChanges temp = new TargetModelChanges();
4      temp.setStateMachineModelName(configurationChanges.
           getStateMachineModelName());
5      this.loadModelURIs();
6
7      // Loading the models that are used at runtime:
8      RunTimeModel rtm = new RunTimeModelImpl(sourceModelURI,
           sourceModelPackage, weavingModelURI, targetModelURI,
           targetModelPackage);
9      ArrayList<EObject> aux = new ArrayList<EObject>();
10
11     // Getting the features that have been activated:
12     for(Feature featAux : configurationChanges.getAdditions()){
13         aux.add((EObject)featAux);
14     }
15
16     // Composition model increments:
17     temp.setIncrements(rtm.getTargetModelElements(aux));
18
19     aux = new ArrayList<EObject>();
20
21     // Getting the features that have been deactivated:
22     for(Feature featAux : configurationChanges.getRemovals()){
23         aux.add((EObject)featAux);
24     }
25
26     // Composition model decrements:
27     temp.setDecrements(rtm.getTargetModelElements(aux));
28
29     return temp;
30 }
```

Listing A.13 shows the implementation of the `getTargetModelElements` method. This method collects the BPMN subprocesses that encapsulate the logic of variant models. These subprocesses are linked to the features that have been activated or deactivated in the current configuration of the feature model.

Listing A.13: A method in Java to get the elements in variant models.

```java
public ArrayList<EObject> getTargetModelElements(ArrayList<EObject>
    sourceModelElements){

   ArrayList<EObject> targetModelElements = new ArrayList<EObject>();
   for(EObject element : sourceModelElements){
      targetModelElements.addAll(getTargeModelElementForSourceElement(
          element));
   }
   return targetModelElements;
}
```

Listing A.14 shows the implementation of the `getTargeModelElementForSourceElement` method. This method receives a feature as input. The `superImposeFeature` method uses this information to go over the weaving model in order to get the BPMN subprocesses that encapsulate the logic of the related variant models. The `getEobjectIDREF` method gets the unique identifier of a feature.

Listing A.14: A method in Java to get the variant models.

```java
public ArrayList<EObject> getTargeModelElementForSourceElement(EObject
    sourceElement){

   ArrayList<EObject> targetModelElements = new ArrayList<EObject>();

   // The superImposeFeature method goes over the weaving model:
   List<String> superImposedRefs = this.weavingModel.superImposeFeature(
       sourceModel.getEobjectIDREF(sourceElement));

   // Adding the BPMN subprocesses that encapsulate variant models into
       an ArrayList:
   for(String idREF : superImposedRefs){
      targetModelElements.add(targetModel.getModelElementByIDREF(idREF)
          );
   }
   return targetModelElements;
}
```

Listing A.15 shows the implementation of the `superImposeFeature` method. This method returns the list of BPMN subprocesses that map to a particular feature.

Listing A.15: A method in Java to get the BPMN subprocesses that map to a particular feature.

```
public List<String> superImposeFeature(String FeatureIDREF){

    ArrayList<String>res = new ArrayList<String>();
    WModel model = this.getModel();
    EList<ElementEqual> ownedElements = model.getOwnedElement();
    for(ElementEqual ee:ownedElements){
        String name = ee.getLeft().getName();
            if(name.equals(FeatureIDREF)){
                res.add(ee.getRight().getElement().getRef());
            }
    }
    return res;
}
```

Step 3: Adapt the Composition Model

1. Execute a Resolution

↓

2. Generate a Reconfiguration Plan

↓

[3. Adapt the Composition Model]

The actions that are carried out by MoRE-WS for reflecting the changes in the variability model configuration into the running composition model are the following:

1. MoRE-WS loads the current version of the composition model (i.e., the running composition model). To this end, MoRE-WS uses the instructions in Listing A.16 to invoke the `load` method.

   Listing A.16: Invoking the `load` method.

   ```
   EMFModelLoad loader = new EMFModelLoad();
   BpmnDiagram runningDiagram = loader.load("ModelsAtRuntime", "models
        /runningBPMNModel.bpmn");
   ```

   The `load` method is described in Listing A.17. It requires two inputs: the name of the bundle where the model is running and the location of the model.

   Listing A.17: A method in Java to load the running composition model.

   ```
   public BpmnDiagram load(String bundleName, String location){

       XMIResourceFactoryImpl _xmiFac = new XMIResourceFactoryImpl();
       ResourceSet rSet = new ResourceSetImpl();
       rSet.getResourceFactoryRegistry().getExtensionToFactoryMap().
            put("*", _xmiFac);
       URL smModelURL = null;
   ```

```
    try{
        smModelURL = FileLocator.resolve(Platform.getBundle(
            bundleName).getEntry(location));
    }catch(IOException e){
        e.printStackTrace();
    }
    EPackage.Registry.INSTANCE.put(BpmnPackage.eNS_URI, BpmnPackage
        .eINSTANCE);
    URI domainModelURI = URI.createFileURI(smModelURL.getPath().
        toString());
    Resource res = rSet.getResource(domainModelURI, true);
    BpmnDiagram bpmnDiag = (BpmnDiagram) res.getContents().get(0);
    return bpmnDiag;
}
```

2. MoRE-WS deletes all the modeling elements in the variation points of the running composition model that are affected by *CM∇* actions. Listing A.18 implements this functionality. The `Documentation` attribute in each BPMN subprocess, which encapsulates a variant model, states its related variation point. MoRE-WS uses this information to find out the variation points to be rebound.

Listing A.18: Deleting the modeling elements in affected variation points according to *CM∇* actions.

```
Activity actDec = null;
Vector<Activity> variantsToDecrement = new Vector<Activity>();
SubProcess affectedVP = null;
Vector<SubProcess> affectedVPs = new Vector<SubProcess>();
EObject obj = null;
Vector<EObject> elementsToBeDeleted = new Vector<EObject>();
EList<Pool> runningDiagramPools = runningDiagram.getPools();

// The running composition model is inside a unique pool in a BPMN
    diagram:
List<Vertex> verticesRunningDiagram =  runningDiagramPools.get(0).
    getVertices();

// This loop goes through an ArrayList with the information of BPMN
     subprocesses, which encapsulate variant models, to be
    incremented/decremented in the composition model:
for(TargetModelChanges targetModelChanges : targetChanges){

    // Getting the variant models to be decremented:
    for(EObject  decrement : targetModelChanges.getDecrements()){
        actDec = (Activity)decrement;
        variantsToDecrement.add(actDec);
```

```java
    }

    for(int y = 0; y < variantsToDecrement.size(); y++){
        for(Vertex v : verticesRunningDiagram){
            if((v instanceof Activity) && (variantsToDecrement.get(
                 y).getDocumentation().equals(((Activity) v).
                 getName()))){
                // Looking for the affected variation point:
                affectedVP = (SubProcess)v;
                affectedVPs.add(affectedVP);

                // Getting all the elements in the variation point:
                for(TreeIterator<EObject> it = affectedVP.
                     eAllContents(); it.hasNext();){
                    obj = it.next();
                    elementsToBeDeleted.add(obj);
                }

                // Deleting the elements in the variation point:
                for(int i = 0; i < elementsToBeDeleted.size(); i++)
                     {
                    EcoreUtil.delete((EObject)elementsToBeDeleted.
                         get(i), true);
                }
            }
//...
}
```

3. MoRE-WS needs to count on the abstract elements to be incremented into the running composition model according to $CM\triangle$ actions. To this end, MoRE-WS reuses Listing A.17 to load the XMI file that contains the base composition model and the variant models. This is possible because the running composition model and the XMI file with the base composition model and the variant models are BPMN diagrams.

4. MoRE-WS inserts variant models into variation points in the running composition model according to $CM\triangle$ actions. Listing A.19 describes the algorithm to carry out this operation. The key aspect of this algorithm is to go through the pools that contain the variant models to be incremented according to $CM\triangle$ actions. Specifically, each one of these pools contains a BPMN subprocess, which encapsulates the elements in a variant model. This code also shows how to get the incoming and outgoing sequence flows from a BPMN activity in the variant model, and to add them

into a variation point. This algorithm works for several variation points that need to be adapted.

Listing A.19: Inserting variant models into variation points in the running composition model according to *CM*△ actions.

```java
Activity actInc = null;
EObject obj = null;
Vector<Activity> variantsToIncrement = new Vector<Activity>();

// This loop gets the variant models to be incremented:
for(EObject  increment : targetModelChanges.getIncrements()){
    actInc = (Activity)increment;
    variantsToIncrement.add(actInc);
}

// This loop goes through the affected variation points:
for(int n = 0; n < affectedVPs.size(); n++){

    // This loop checks all the variant models to be incremented:
    for(int w = 0; w < variantsToIncrement.size(); w++){

        // This loop goes through the pools that contain the
             variant models. The counter starts from 1 because the
             first pool (pool number 0) contains the base
             composition model:
        for(int i = 1; i < pools.size(); i++){
            verticesVP = pools.get(i).getVertices();

            // This loop discovers the variant models to be
                 incremented:
            for(Vertex v : verticesVP){
                if((v instanceof Activity) && (variantsToIncrement.
                     get(w).getName().equals(((Activity) v).getName
                     ()))                   && (variantsToIncrement
                     .get(w).getDocumentation().equals(affectedVPs.
                     get(n).getName())))){
                    foundVariant = (Activity)v;

                    // This loop goes through all the elements in
                         the discovered variant model:
                    for(TreeIterator<EObject> it = foundVariant.
                         eAllContents();it.hasNext();){
                        obj = it.next();

                        // If the element in the iteration (obj) is
                             an instance of the Activity class,
                            then an object of this class is
                            instantiated:
```

```
                if(obj instanceof Activity){
                    Activity activity = BpmnFactory.
                        eINSTANCE.createActivity();

                    // If the element in the iteration is a
                         BPMN task, then several
                        attributes are intialized in the
                        activity object. The complete
                        program also checks other BPMN
                        elements inside the variant model,
                         such as start and end events:
                    if(((Activity) obj).getActivityType().
                        equals(ActivityType.TASK_LITERAL))
                        {
                        activity.setActivityType(
                            ActivityType.TASK_LITERAL);
                        activity.setID(((Activity)obj).
                            getID());
                        activity.setName(obj.eClass().
                            getName());
                        activity.setName(((Activity) obj).
                            getDocumentation());
                        activity.setNcname(((Activity) obj)
                            .getNcname());
                        activity.setLooping(((Activity) obj
                            ).isLooping());

                        // Adding the newly created
                            activity into the affected
                            variation point:
                        affectedVPs.get(n).getVertices().
                            add(activity);

                        // Managing incoming and outgoing
                            sequence flows:
                        for(int a = 0; a < ((Activity) obj)
                            .getIncomingEdges().size(); a
                            ++){
                            affectedVPs.get(n).
                                getSequenceEdges().add(((
                                Activity)obj).
                                getIncomingEdges().get(a))
                                ;
                        }

                        for(int b = 0; b < ((Activity) obj)
                            .getOutgoingEdges().size(); b
                            ++){
                                    affectedVPs.get(n).
                                        getSequenceEdges().add(((
                                        Activity)obj).
                                        getOutgoingEdges().get(b);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

5. Finally, MoRE-WS saves the new version of the running composition model. The implementation of this step in Listing A.20 shows how the `saveModel` method receives the object with the adapted composition model and saves it. The adapted composition model will eventually cause the adaptation of the WS-BPEL composition schema.

Listing A.20: A method in Java to save the adapted composition model.

```
public void saveModel(EObject adaptedCompositionModel){

    try{
        URI fileURI = URI.createFileURI("models/runningBPMNModel.
            bpmn");
        Resource poResource = new XMLResourceFactoryImpl().
            createResource(fileURI);
        poResource.getContents().add(project);
        poResource.save(null);
    } catch(Exception ex){
        ex.printStackTrace();
    }
}
```

A.2.3 *Executing the Adaptation*

In order to materialize the changes at the composition model level into the running service composition, this phase is composed of three steps: 1) MoRE-WS creates a new version of the deployment directory in which the adapted WS-BPEL composition schema will be running; 2) MoRE-WS inserts WS-BPEL code fragments into the WS-BPEL template at runtime.

The completed WS-BPEL template is used to orchestrate the service composition; and 3) MoRE-WS deploys at runtime the generated WS-BPEL code and other required artifacts in the Execution Engine. Implementation details about these steps are described as follows. These steps support the `Execute` component of the MAPE-K loop (see Figure A.4).

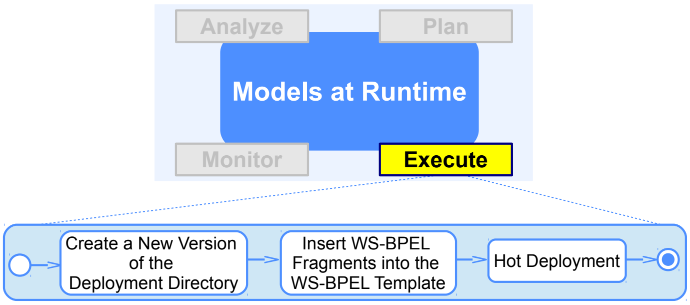

Analyze
Plan
Models at Runtime
Monitor
Execute
Create a New Version of the Deployment Directory
Insert WS-BPEL Fragments into the WS-BPEL Template
Hot Deployment

Figure A.4: Implementing the `Execute` component of the MAPE-K loop with MoRE-WS.

Step 1: Create a New Version of the Deployment Directory

[1. Create a New Version of the Deployment Directory]

↓

2. Insert WS-BPEL Fragments into the WS-BPEL Template

↓

3. Hot Deployment

In this step, MoRE-WS uses the `createDirectory` method in Listing A.21 to create a new deployment directory (see line 8). This directory will contain the adapted WS-BPEL composition schema and other necessary artifacts to run the adapted service composition (i.e., the deployment descriptor – an XML file – and the WSDL files, which describe the functionality offered by the Web services).

Listing A.21: Creating a new version of the deployment directory.

```
1 // counterFile contains the current version of the deployment directory:
2 String counterFile = "counterFile.txt";
3 
4 // strDirectory contains the directory where the adapted composition
     schema will be deployed. For example: "/usr/local/tomcat/webapps/ode
     /WEB-INF/processes/BookShopping":
5 String strDirectory = "";
6 
7 // Creating the new version of the deployment directory:
8 createDirectory(counterFile, strDirectory);
```

Every new deployment directory has an increasing version number. As a result, new invocations to the service composition will be carried out on the adapted version of the service composition. The name of the deployment directory has the following format: `deploymentDirectoryN`, where `N` is the version number (an integer). In order to realize the version of the latest deployment directory, the `counterFile` file keeps an integer with this value.

Listing A.22 shows the implementation of the `createDirectory` method. The version value of the deployment directory is increased in line 12. The new version of the deployment directory is saved in every adaptation as described in lines 24-28.

Listing A.22: A method in JAVA to create a new version of the deployment directory.

```
1  private void createDirectory(String counterFile, String strDirectory){
2
3      String counterStr = "";
4      int counter = 0;
5      boolean success;
6
7      try{
8          BufferedReader br = new BufferedReader(new FileReader(counterFile
               ));
9          counterStr = br.readLine();
10         br.close();
11         counter = Integer.parseInt(counterStr);
12         counter++;
13
14         // Creating the new version of the deployment directory:
15         deployedDirectory = strDirectory + counter;
16         success = (new File(deployedDirectory)).mkdir();
17         if(success){
18
19             // Deleting the old counterFile file:
20             File fileToBeDeleted = new File(counterFile);
21             fileToBeDeleted.delete();
22
23             // Recreating the counterFile file:
24             FileWriter fstream = new FileWriter(counterFile);
25             FileWriter write = new FileWriter(counterFile, false);
26             PrintWriter print_line = new PrintWriter(write);
27             print_line.printf(Integer.toString(counter++));
28             print_line.close();
29         }
30     }catch(Exception e){
31             System.out.println("Error: " + e.getMessage());
```

```
    }
}
```

Step 2: Insert WS-BPEL Fragments into the WS-BPEL Template

1. Create a New Version of the Deployment Directory

↓

[2. Insert WS-BPEL Fragments into the WS-BPEL Template]

↓

3. Hot Deployment

Listing A.23 shows a fragment of code that is used by MoRE-WS to insert a WS-BPEL fragment into the WS-BPEL template. This algorithm reads the file with the WS-BPEL fragment (see line 2). Then, it invokes the `generateWSBPEL` method in line 5, which adapts the WS-BPEL template by inserting the WS-BPEL fragment into a particular variation point. A variation point has the following format: `<documentation>VP</documentation>`.

Listing A.23: Invoking the `generateWSBPEL` method.

```
1 // Reading the WS-BPEL fragment:
2 String fragmentInfo = readBPELFragment("WSBPELCode.fragment");
3
4 // Adding the WS-BPEL fragment into the WS-BPEL template:
5 generateWSBPEL(WSBPELTemplate, fragmentInfo, variationPoint,
      deploymentDirectoryN+"/"+"theProcess.bpel");
```

Listing A.24 shows the implementation of the `generateWSBPEL` method. The first part of this method reads the WS-BPEL template (see lines 5-11). Then, the text in the WS-BPEL code fragment file is inserted into the WS-BPEL template file under the element that indicates a variation point (i.e., `<documentation>VP</documentation>`). To this end, the `VP` variable is replaced with the `VP` variable and a new line with the instructions in the `BPELCodeFragment` variable (see line 14). Afterwards, MoRE-WS puts the adapted composition schema into the new version of the deployment directory (see lines 15-17).

Listing A.24: A method in Java to generate an adapted WS-BPEL file.

```
 1 private void generateWSBPEL(String inBPEL, String BPELCodeFragment,
       String VP, String outBPEL){
 2
 3     try{
 4         // Reading the WS-BPEL template:
 5         File file = new File(inBPEL);
 6         BufferedReader reader = new BufferedReader(new FileReader(file));
 7         String line = "", oldtext = "";
 8         while((line = reader.readLine()) != null){
 9             oldtext += line + "\r\n";
10         }
```

```java
        reader.close();

        // Adapting the WS-BPEL file:
        String mergedCode = oldtext.replaceAll(VP, VP+"\n"+
            BPELCodeFragment);
        FileWriter writer = new FileWriter(outBPEL);
        writer.write(mergedCode);
        writer.close();
    }catch(IOException ioe){
        ioe.printStackTrace();
    }
}
```

Step 3: Hot Deployment

In this step, MoRE-WS hot deploys the adapted WS-BPEL composition schema on the Execution Engine. To this end, MoRE-WS carries out the following actions: 1) it copies the WSDL files that are invoked by the WS-BPEL composition schema. These WSDL files are copied from a repository of WSDLs into the latest version of the deployment directory (see Listing A.25); and 2) it copies the deployment descriptor. This is the last operation because the Execution Engine carries out the hot deployment as soon as the deployment descriptor is added into the deployment directory.

1. Create a New Version of the Deployment Directory

↓

2. Insert WS-BPEL Fragments into the WS-BPEL Template

↓

[3. Hot Deployment]

Listing A.25: A method in Java to copy the WSDL files into the deployment directory.

```java
private void copyWSDLsIntoDeploymentDirectory(Vector<String> WSDLs){

    String srFile = "";
    String dtFile = "";
    String temp = "";
    try{
        for(int i = 0; i < WSDLs.size(); i++){
            srFile = WSDLs.elementAt(i);
            dtFile = deployedDirectory +"/"+ WSDLs.elementAt(i);
            copyFile(srFile, dtFile);
        }
    }catch(Exception e){
        System.err.println("Error: " + e.getMessage());
    }
}
```

A.3 OPERATIONS FOR DYNAMIC EVOLUTIONS

This section describes the implementation details of the Evolution Planner and MoRE-WS to support the dynamic evolution of service compositions in the open world.

A.3.1 *Planning the Evolution with the Evolution Planner*

The Evolution Planner looks for tactics to preserve the requirements that can be affected by unknown context events. The steps that are carried out by the Evolution Planner can be organized according to the components of the MAPE-K loop. Specifically, these steps extend the following components of the MAPE-K loop as described in Section 8.1 (see Figure A.5):

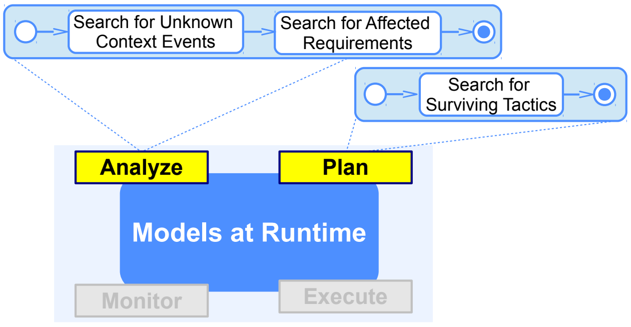

Search for Unknown Context Events
Search for Affected Requirements
Search for Surviving Tactics
Analyze
Plan
Models at Runtime
Monitor
Execute

Figure A.5: Extending the `Analyze` and `Plan` components of the MAPE-K loop in the open world.

- The `Analyze` component of the MAPE-K loop is extended by the Evolution Planner in order to: 1) search for unknown context events in the open world; and 2) look for the expected requirements of the service composition that can be affected by arising unknown context events.

- The `Plan` component of the MAPE-K loop is extended by the Evolution Planner in order to discover tactics to preserve requirements that can be negatively impacted by unknown context events.

The following subsections describe the steps that are carried out by the EVOLUTION PLANNER.

Step 1: Search for Unknown Context Events

In this step, the EVOLUTION PLANNER evaluates the context model to decide whether or not an observed context event is unknown (see Listing A.26). All the context events that do not trigger an adaptation are considered as unknown. Unknown context events are evaluated by the `runEngine` method in line 16. This method applies the Forward Chaining method to find out whether or not these events negatively affect expected requirements. The `runEngine` method receives two parameters: the knowledge base with all the rule premises and the updated context model.

[1. Search for Unknown Context Events]

↓

2. Search for Affected Requirements

↓

3. Search for Surviving Tactics

Listing A.26: Searching for unknown context events.

```
1  Activator.context = bundleContext;
2  IEnvironmentStateProvider contextMonitor = getContext().getService(
       getContext().getServiceReference(IEnvironmentStateProvider.class));
3
4  ContextModelOp cm = new ContextModelOp();
5  cm.loadModel();
6
7  // The following evaluation is carried out for each context condition:
8  if(cm.getContextConditionX() == true){
9      contextMonitor.addNewEvent("contextConditionX", true);
10 }
11
12 // Other evaluations of context conditions can be added here...
13
14 // If an event does not fulfill any context condition, then the Evolution
        Planner analyzes this "unknown" context event with the Forward
       Chaining method:
15 ForwardChaining fc = new ForwardChaining();
16 fc.runEngine("test.rules", cm.getContextModel());
```

Step 2: Search for Affected Requirements

In this step, the EVOLUTION PLANNER looks for the requirements that can be affected by an unknown context event. Listing A.27 shows a method to create and execute the rule-based inference engine implemented in JENA, which carries out the necessary inferences on the knowledge base. The `checkAffectedRequirements` method, which is invoked in line 43, discovers the requirements that may be affected.

1. Search for Unknown Context Events

↓

[2. Search for Affected Requirements]

↓

3. Search for Surviving Tactics

Listing A.27: A method in Java to create an execute the rule-based inference engine.

```java
public void runEngine(String ruleFile, String rdfFile){

    Reasoner reasoner; // The reasoner
    Model model; // The RDF model
    Resource configuration; // A resource to configure the engine
    InfModel infModel; // The inference model

    public ForwardChaining(){
        this.reasoner = null;
        this.model = null;
        this.configuration = null;
        this.infModel = null;
    }

    // Creating a default empty model:
    model = ModelFactory.createDefaultModel();

    // Creating an empty resource:
    configuration = model.createResource();

    // Setting engine mode. In the hybrid mode, all types of rules:
         backward, forward, and hybrid can be executed:
    configuration.addProperty(ReasonerVocabulary.PROPruleMode, "hybrid");

    // Setting the rules file:
    configuration.addProperty(ReasonerVocabulary.PROPruleSet,  ruleFile);

    // Creating an instance of such a reasoner:
    reasoner = GenericRuleReasonerFactory.theInstance().create(
         configuration);

    // Loading the context model:
    model = FileManager.get().loadModel(rdfFile);

    // Creating the inference model:
    infModel = ModelFactory.createInfModel(reasoner, model);

    // Forcing to start the rule execution:
    infModel.prepare();

    // It is necessary to check the inference results in every subject
         (i.e., service operations in our case):
```

```java
    Resource serviceX = infModel.getResource("http://my.ontology#serviceX");

    // Infering if a subject of the ontology (i.e., a service operation) can affect a requirement. The following instruction has to be done for every subject (i.e., service operation) specified in the context model:
    checkAffectedRequirements(infModel, serviceX, null, null);

    // More inferences can be carried out here...
}
```

Listing A.28 shows a fragment of the implementation of the `checkAffectedRequirements` method. The loop in lines 8-20 checks all the rules in the knowledge base. For the sake of simplicity, just one rule is checked in this fragment of code. The condition in line 14 is fulfilled when a rule is fulfilled (i.e., a requirement can be affected by an unknown context event).

Listing A.28: A method in Java to find a requirement that can be affected by an unknown context event.

```java
1  public static void checkAffectedRequirements(Model m, Resource s, Property p, Resource o){
2  
3      String subject = "";
4      String object = "";
5      PrintUtil.registerPrefix("x", "http://my.ontology#");
6      Vector<String> affectedReqs = new Vector<String>();
7  
8      for(StmtIterator i = m.listStatements(s,p,o); i.hasNext();){
9          Statement stmt = i.nextStatement();
10         subject = stmt.getSubject().toString();
11         object = stmt.getObject().toString();
12 
13         // This operation has to be repeated for every requirement to be preserved:
14         if(object.equals("http://my.ontology#AffectedRequirementY")){
15             System.out.println("The "+subject +" Web service operation has affected the Requirement Y!");
16             affectedReqs.add("AffectedRequirementY");
17         }
18 
19         // Other rule checkings can be added here...
20     }
21     //...
22 }
```

Step 3: Search for Surviving Tactics

1. Search for Unknown Context Events

↓

2. Search for Affected Requirements

↓

[3. Search for Surviving Tactics]

In this step, the EVOLUTION PLANNER searches for the tactics to preserve the requirements that can be negatively impacted by an unknown context event. To this end, the EVOLUTION PLANNER invokes the `checkTactic` method as described in Listing A.29. This piece of code was copied from line 21 in Listing A.28. An unknown context event can affect multiple requirements. Therefore, the `affectedReqs` structure, which contains all these requirements, is sent to the `checkTactic` method. The `subject` variable indicates the name of the problematic service operation.

Listing A.29: Invoking a method that queries the requirements model.

```
// Querying the requirements model to realize which tactic to use:
RequirementsModel reqModel = new RequirementsModel();
reqModel.checkTactic(affectedReqs, subject);
```

In order to find the tactics to preserve requirements at runtime, the `checkTactic` method loads the requirements model as described in Listing A.30. A requirements model created in OPENOME[3] includes intentions and contributions among intentions. *Intentions* can be goals, softgoals, and beliefs (or claims). In turn, *contributions* can be of several kinds, such as help and make. In order to find surviving tactics, our algorithm is particularly interested in analyzing contributions. As a result, it loops through the contributions in the requirements model as shown in lines 7-10 of Listing A.30.

Listing A.30: Loading the requirements model.

```
1  // Loading the requirements model:
2  EMFModelLoad loader = new EMFModelLoad();
3  Model requirementsModel = loader.load("ModelsAtRuntime", "models/RM.
       openome_model");
4  
5  // Looping through the contributions in the requirements model:
6  EList<Contribution> contributionsInModel = requirementsModel.
       getContributions();
7  for(int r = 0; r < contributionsInModel.size(); r++){
8      contribution = contributionsInModel.get(r);
9      //...
10 }
```

3 https://se.cs.toronto.edu/trac/ome

A contribution has the following properties:

1. `type` (e.g. `edu.toronto.cs.openome_model:HelpContribution`).
2. `id`, which is a unique identifier.
3. `target`, which indicates the identifier of the intention where the contribution points to.
4. `source`, which indicates the intention where the contribution starts.

The Evaluation Planner collects the contributions in which the `target` is a requirement in the `affectedReqs` structure in Listing A.29. In order to find the targets, the Evaluation Planner uses the `getTarget` method of the `Contribution` class in OpenOME.

Then, the Evolution Planner checks if there are claims associated to the affected softgoals. To this end, our algorithm looks for claims among the sources in the collected contributions (by means of the `getSource` method).

On one hand, if there is not any claim that contributes to the affected softgoals, then the Evolution Planner selects the tactics with the most positive contributions on these softgoals. To this end, it checks the types of the contributions, which go from `edu.toronto.cs.openome_model:MakeContribution` to `edu.toronto.cs.openome_model:BreakContribution`.

On the other hand, the algorithm selects the claims (intentions of type `edu.toronto.cs.openome_model:Belief`), which contribute to the affected requirements. Intentions have the `contributesFrom` property, which is useful to detect the identifiers of the contributions that point to a claim (e.g. contributions that come from tactics). The algorithm loops through these contributions in order to find out the most positive ones.

A.3.2 *Using MoRE-WS for Dynamic Evolutions*

This section presents the implementation details of the operations that are carried out by MoRE-WS to evolve the variability model, the composition model, and the WS-BPEL composition schema. These operations extend the `Execute` component of the MAPE-K loop as described in Figure A.6.

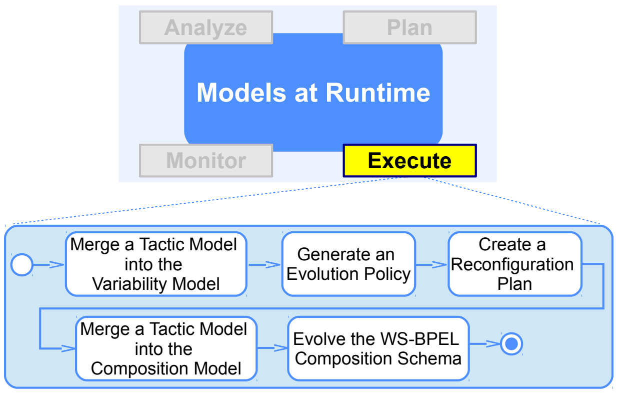

Analyze
Plan
Models at Runtime
Monitor
Execute
Merge a Tactic Model into the Variability Model
Generate an Evolution Policy
Create a Reconfiguration Plan
Merge a Tactic Model into the Composition Model
Evolve the WS-BPEL Composition Schema

Figure A.6: Extending the `Execute` components of the MAPE-K loop in the open world.

Steps 1 and 2: Merge a Tactic Model into the Variability Model and Generate an Evolution Policy

[1. Merge a Tactic Model into the Variability Model & 2. Generate an Evolution Policy]

↓

3. Create a Reconfiguration Plan & 4. Merge a Tactic Model into the Composition Model

↓

5. Evolve the WS-BPEL Composition Schema

In order to merge a tactic expressed as a feature model into the variability model, first it is necessary to find the variation point where the tactical functionality will be merged. Listing A.31 shows a method that implements this operation. The `doMerging` method in line 28 is in charge of doing the merging operation.

Listing A.31: A method in Java to look for the variation point in a feature model where the tactical functionality will be merged.

```
1  private String VP;
2
3  public void lookForVariationPoint(){
4
5          EMFModelLoad loader = new EMFModelLoad();
6          FeatureModel variabilityModel = loader.load("ModelsAtRuntime", "
                models/FM.fm4spl");
7          EObject object = null;
8          EList<EObject> objectsInModel = variabilityModel.eContents();
9
10         // Looping through the whole content in the variability model:
11         for(int i = 0; i < objectsInModel.size(); i++){
12             object =  objectsInModel.get(i);
13
14             // Getting the properties of an object in the variability
                   model:
15             EList<EObject> propertiesInObject = object.eContents();
```

```

            for(int j = 0; j < propertiesInObject.size(); j++){

                // Checking if the property is an optional relationship. Checking mandatory relationships is similar:
                if(propertiesInObject.get(j).getClass().getName().equals("FeatureModelPackage.impl.OptionalImpl")){
                    Optional test = (Optional)propertiesInObject.get(j);

                    // Every relationship indicates the feature from which it starts and the feature where it points to. The following condition checks whether the destination feature (where the relationship points to) is the problematic feature:
                    if(test.getTo().getName().equals(problematicFeature)){

                        // The name of every relationship has the same name of its related variation point. Therefore, it is possible to use this information to realize the affected variation point:
                        VP = test.getName();
                        doMerging();
                    }
                }
            }
        }
}
```

Listing A.32 shows a fragment of the doMerging method. First, this method creates a mandatory relationship between the affected variation point and the root feature of the tactic model (see lines 1-53). Then, it applies the evolution policy, which activates the features in the tactic model (see lines 56-57). The evolution policy is generated with the activation of the features that are obtained by looping the tactic model. Finally, this algorithm merges the relationships among features in the tactic model into the variability model (see lines 59-73).

Listing A.32: A fragment of the doMerging method.

```
1  EMFModelLoad loaderTactic = new EMFModelLoad();
2  FeatureModel tacticModel = loaderTactic.load("ModelsAtRuntime", tempURI); // tempURI is the URI of the tactic model to be merged
3  EList<EObject> objectsInTacticModel = tacticModel.eContents(); // Collecting all the elements inside the tactic model
4  EObject objectTactic;
5  String objectTacticName;
```

```java
XMIResourceFactoryImpl _xmiFac = new XMIResourceFactoryImpl();
ResourceSet rSet = new ResourceSetImpl();
rSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put("*",
     _xmiFac);
URL smModelURL = null;

// Loading the variability model:
try{
    smModelURL = FileLocator.resolve(Platform.getBundle("ModelsAtRuntime"
        ).getEntry("models/FM.fm4spl"));
}catch(IOException e){
    e.printStackTrace();
}

URI domainModelURI = URI.createFileURI(smModelURL.getPath().toString());
Resource res = rSet.getResource(domainModelURI, true);

// Retrieving the default Feature factory singleton:
FeatureModelPackageFactory factory = FeatureModelPackageFactory.eINSTANCE
     ;

// Looping through the tactic model to be merged:
for(int i = 0; i < objectsInTacticModel.size(); i++){
    objectTactic = objectsInTacticModel.get(i);
    objectTacticName = ((Feature)objectsInTacticModel.get(i)).getName();
    Feature feat = factory.createFeature();
    feat.setName(objectTacticName);

    EObject myEvolvedVMOb = res.getContents().get(0);
    FeatureModel myEvolvedVM = (FeatureModel)myEvolvedVMOb;

    // In each iteration, a feature of the tactic model is put into the
        variability model:
    myEvolvedVM.getFeatures().add(feat);

    Feature object = null;
    EList<Feature> featuresInFeatureModel = myEvolvedVM.getFeatures();

    for(int r = 0; r < featuresInFeatureModel.size(); r++){
        object = featuresInFeatureModel.get(r);

        // Creating a mandatory relationship between the variation point
            and the root feature of the tactic model:
        if(object.getName().equals(VP) && feat.getName().equals(
            triggeredTactic)){
            Mandatory mandMain = factory.createMandatory();
            mandMain.setName("tacticRel");
            mandMain.setFrom(object);
            mandMain.setTo(feat);
            object.getCardinalityBased_Relationships().add(mandMain);

            System.out.println("Created mandatory relationship between
                the "+VP+" variation point and the root feature of the
                tactic model... " + mandMain);
        }
    }

    // In each iteration, a feature in the tactic model is activated:
    FeatureModelConfigurationChanges theChanges = new
        FeatureModelConfigurationChanges();
    theChanges.addAddition(feat);

    EList<EObject> propertiesInObjectTactic = objectTactic.eContents();

    // Merging the relationships among features in the tactic model into
        the variability model:
    FeatureModelConfigurationChanges theChanges = new
        FeatureModelConfigurationChanges();
    theChanges.addAddition(feat);
    EList<EObject> propertiesInObjectTactic = objectTactic.eContents();

    // A fragment of code for merging mandatory relationships:
    for(int j = 0; j < propertiesInObjectTactic.size(); j++){
        Mandatory mand = factory.createMandatory();
        mand.setName(((Mandatory)propertiesInObjectTactic.get(j)).getName
            ());
        mand.setFrom(((Mandatory)propertiesInObjectTactic.get(j)).getFrom
            ());
        mand.setTo(((Mandatory)propertiesInObjectTactic.get(j)).getTo());
        object.getCardinalityBased_Relationships().add(mand);
    }
}
```

Steps 3 and 4: Create a Reconfiguration Plan and Merge a Tactic Model into the Composition Model

The reconfiguration plan is created by querying the weaving model, which maps tactic models expressed as feature models to tactic models expressed as composition models. Listing A.15 shows the code to carry out this operation.

Listing A.33 shows a fragment of code to evolve the composition model with the tactic models expressed as BPMN composition models. In this code, MoRE-WS does the following: 1) it creates a parallel relationship between the BPMN activity that abstracts the problematic service operation and the BPMN subprocess that abstracts the tactic (lines 3-25); and 2) it copies all the elements in the BPMN subprocess that encapsulates a tactic into the composition model (lines 28-43).

1. Merge a Tactic Model into the Variability Model & 2. Generate an Evolution Policy

↓

[3. Create a Reconfiguration Plan & 4. Merge a Tactic Model into the Composition Model]

↓

5. Evolve the WS-BPEL Composition Schema

Listing A.33: Evolving the composition model.

```
1  // Looping through a BPMN subprocess that encapsulates the tactic
       functionality:
2  for(int r = 0; r < subprocessThatEncapsulatesTactic.size(); r++){
3      SubProcess su = factory.createSubProcess();
4      su.setName(subprocessThatEncapsulatesTactic.get(r).getName());
5      subServiceTactic.getVertices().add(su);
6
7      // Creating the start and end events, the parallel gateways, and the
           sequence flows between the problematic functionality and the
           merged tactic. These are some code examples for this operation:
8      Activity startEvent = factory.createActivity();
9      startEvent.setName("startEventForTactic");
10     subServiceTactic.getVertices().add(startEvent);
11
12     Activity startGateway = factory.createActivity();
13     startGateway.setName("startGateway");
14     subServiceTactic.getVertices().add(startGateway);
15
16     SequenceEdge seqFromStartEventToStartGateway = factory.
           createSequenceEdge();
17     seqFromStartEventToStartGateway.setName("
           seqFromStartEventToStartGateway");
           seqFromStartEventToStartGateway.setSource(startEvent);
18     seqFromStartEventToStartGateway.setTarget(startGateway);
           subServiceTactic.getSequenceEdges().add(
           seqFromStartEventToStartGateway);
19
20     SequenceEdge seqFromStartGatewayToAffectedSub = factory.
           createSequenceEdge();
21     seqFromStartGatewayToAffectedSub.setName("
           seqFromStartGatewayToAffectedSub");
           seqFromStartGatewayToAffectedSub.setSource(startGateway);
22     seqFromStartGatewayToAffectedSub.setTarget(affectedSub); //
           affectedSub is the subprocess that abstracts the problematic
           service operation
23     objectTargetPool.getSequenceEdges().add(
           seqFromStartGatewayToAffectedSub);
24
25     // Additional elements are created here...
26
27     // Getting the elements in the subprocess that contains the tactic's
           elements:
28     objectSubProcessInTactic = subprocessThatEncapsulatesTactic.get(r);
```

```
    EList<EObject> actiSeqInTactictModel = objectSubProcessInTactic.
        eContents();

    // Merging the tactic's elements into the composition model:
    for(int h = 0; h < actiSeqInTactictModel.size(); h++){
        if(actiSeqInTactictModel.get(h).getClass().getName().equals("bpmn
            .impl.ActivityImpl")){
            Activity act = factory.createActivity();
            act.setName(((Activity)actiSeqInTactictModel.get(h)).getName
                ());
            su.getVertices().add(act);
        }if(actiSeqInTactictModel.get(h).getClass().getName().equals("
            bpmn.impl.SequenceEdgeImpl")){
            SequenceEdge seq = factory.createSequenceEdge();
            seq.setName(((SequenceEdge)actiSeqInTactictModel.get(h)).
                getName());
            su.getSequenceEdges().add(seq);
        }
        //Additional BPMN elements are added here...
    }
}
```

Step 5: Evolve the WS-BPEL Composition Schema

Listing A.34 shows a fragment of code to evolve the WS-BPEL composition schema with tactics. In line 2, MoRE-WS creates a new deployment directory with the method described in Listing A.22. This deployment directory has an increasing version number.

1. Merge a Tactic Model into the Variability Model & 2. Generate an Evolution Policy

↓

3. Create a Reconfiguration Plan & 4. Merge a Tactic Model into the Composition Model

↓

[5. Evolve the WS-BPEL Composition Schema]

Listing A.34: Evolving the WS-BPEL composition schema.

```
1  // Creating a directory that contains all the files that are required for
         deployment. It has an increasing version number:
2  createDirectory(counterFile, strDirectory);
3
4  // Evolving the WS-BPEL composition schema by running in parallel a
        tactic with the problematic service operation:
5  String codeWithPreludeAndPostlude = "<bpel:flow name=\"Flow1\"><bpel:
        sequence name=\"SequenceWithTactic\">" + codeToLookFor + "</bpel:
        sequence>";
6  String invokeTactic = readWSBPEL(WSBPELFragmentTactic);
7  String codeWithTactic = codeWithPreludeAndPostlude+"\n"+invokeTactic+"</
        bpel:flow>\n";
8
9  codeReplacer(previousDeployedDirectoryVersion+"/"+"theProcess.bpel",
        codeToLookFor, codeWithTactic, deployedDirectory+"/"+"theProcess.
        bpel");
```

```
10
11 // Copying the necessary WSDL files into the evolved deployment directory. It includes the tactics’ WSDL files:
12 copyWSDLsIntoDeploymentDirectory(WSDLs);
```

The `codeWithPreludeAndPostlude` variable keeps the concatenation of the WS-BPEL `flow` element and the code of the problematic service operation (i.e., `codeToLookFor`). The `flow` element indicates that the tactic will run in parallel with the problematic service operation.

The `invokeTactic` variable keeps a string with the instructions in the WS-BPEL code fragment that invokes the tactical functionality to be merged. In turn, the `codeWithTactic` variable keeps the concatenation of the `codeWithPreludeAndPostlude` variable and the `invokeTactic` variable.

The `codeReplacer` method is invoked in line 9 to evolve the current version of the WS-BPEL composition schema. This method receives: 1) the WS-BPEL composition schema, which is located in the current version of the deployment directory; 2) the WS-BPEL piece of code that is going to be evolved; 3) the evolved code; and 4) the name of the WS-BPEL composition schema in the new version of the deployment directory. The `copyWSDLsIntoDeploymentDirectory` method is invoked in line 12 to copy the necessary WSDL files into the new version of the deployment directory (including the tactics’ WSDL files). Listing A.25 shows the implementation of this method. Listing A.35 shows the implementation of the `codeReplacer` method.

Listing A.35: A method in Java to merge the tactical functionality into the WS-BPEL composition schema.

```java
private void codeReplacer(String inBPEL, String codeToLookFor, String newCode, String outBPEL){

    try{
        File file = new File(inBPEL);
        BufferedReader reader = new BufferedReader(new FileReader(file));
        String line = "", oldtext = "";
        while((line = reader.readLine()) != null){
            oldtext += line;
        }
        reader.close();
        String mergedCode = oldtext.replaceAll(codeToLookFor, newCode);
        FileWriter writer = new FileWriter(outBPEL);
        writer.write(mergedCode);
        writer.close();
    }catch(IOException ioe){
        ioe.printStackTrace();
    }
}
```

B

CASE STUDY: BOOK SHOPPING PROCESS

In Section 4.1, we introduce a case study about a composite service that supports online book shopping at Orange Country Bookstore. We propose this case study to apply our approach because of the following two reasons: 1) it exemplifies a realistic situation in which dynamic adaptations and dynamic evolutions can be called at runtime in order to keep SLAs. We believe that the lessons learned in this case study can be easily extrapolated to other scenarios; and 2) related work in relevant publications have proposed case studies that introduce service compositions of similar dimensions (Kumar and Yao, 2012; Calinescu et al., 2011; Canfora et al., 2008; Baresi and Guinea, 2011). This fact is a good indicator of the feasibility of the presented case study to demonstrate our approach.

The `Design Phase` of our framework supports the design of the dynamic adaptation and the dynamic evolution of service compositions. Chapter 5 and Chapter 6 present examples, taken from this case study, of the models and other artifacts that are created in this phase. Therefore, the focus of this appendix is not on the `Design Phase`. It focuses on the phases that are carried out at runtime, namely `Dynamic Adaptation` and `Dynamic Evolution`. Specifically, we provide technical details of the case study (including fragments of models at runtime in XMI format), present the outputs in console during dynamic adjustments, and describe the resulting adjusted WS-BPEL composition schema.

The focus of this appendix is on the `Dynamic Adaptation` and the `Dynamic Evolution` phases.

The remainder of this appendix is structured as follows: Section B.1 describes the Web services that are used in this case study. Section B.2 describes a group of context conditions and resolutions that are used in this case study. Section B.3 describes the WS-BPEL template that is used in this case study. Section B.4 describes a scenario in which two dynamic adaptations are triggered. Section B.5 describes a scenario in which a dynamic evolution is triggered.

B.1 WEB SERVICES

The Web services that are used in the book-shopping case study are described as follows:

- **Services for book management:**
  - Barnes and Noble Books: This service allows to look for a book by title or by author. If the book is found, it returns the information about the book and its related titles. To this end, the Barnes and Noble Books composite service orchestrates the following atomic services: 1) the Search Book service looks for a book by title or by author. If the book is not found, it returns a message saying that no book has been found; 2) the Show Book Info service shows the complete names of the authors and the information about the publisher; and 3) the Show Related Titles service shows the books that are related to the book that is searched.
  - Amazon Books: This service allows to look for a book by title or by author. If the book is found, it returns the book's information. To this end, the Amazon Books composite service orchestrates the following atomic services: 1) the Book Searching service looks for a book by title or by author. If the book is not found, it returns a message saying that no book has been found; and 2) the Book Description service shows the complete names of the authors and the information about the publisher.
  - Related Titles: This service shows the books that are related to a particular book.
- **Services for shopping cart:** The Amazon Shopping Cart and the Barnes and Noble Shopping Cart services allow to add books into the shopping cart. They retrieve the gross price of the books in the shopping cart and use this information to calculate the sum of these values.
- **Service for authentication:** The Google Authentication service allows to authenticate users. It receives a user name and a password. If the user information is valid, it returns a welcome message. Otherwise, it returns an error message.

- **`Service to calculate payment:`** The `Payment Calculator` service allows to calculate the total to be paid in US$ by adding the tax to the gross value of the books in the shopping cart.
- **`Services for credit card payment:`** The `Bank of America Credit Card Payment`, `Chase Banking`, and `Wells Fargo Online` services allow to pay with a credit card. To this end, these services get a credit card number and its expiration date (MM/YY). If the input information is wrong (e.g. the credit card has expired), then they return an error message.
- **`Services for invoice delivery:`** The `E-mail Invoice` and `SMS Invoice` services allow to deliver invoices to customers, either by e-mail or by Short Message Service (SMS).
- **`Services for shipment:`** The `UPS Shipping`, `FedEX Express`, and `DHL Delivery` services allow to deliver goods (e.g. books) to customers. These services return a message saying that the goods have been successfully delivered. Otherwise, they return an error message.

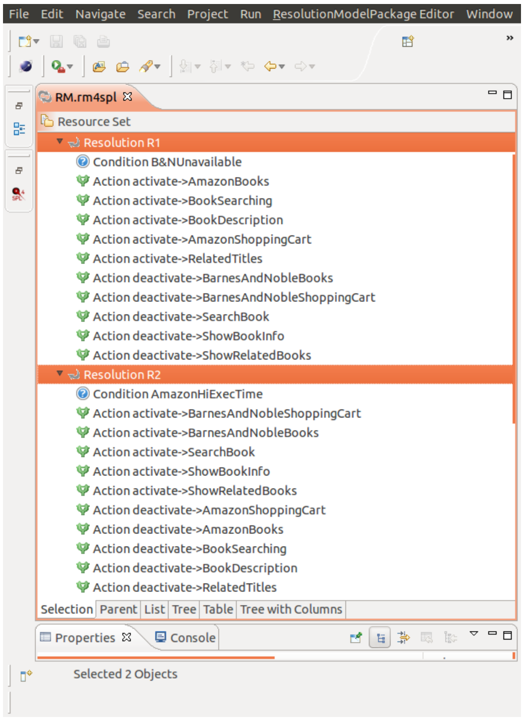
B.2 CONTEXT CONDITIONS AND RESOLUTIONS

Several context conditions can be defined for our case study. For example, some possible context conditions to monitor the availability and the execution time of the `Bank of America Credit Card Payment` service are as follows. These examples demonstrate that composite context conditions can be used to monitor several context events related to availability and execution time:

- `Bank of America Credit Card Payment` service is unavailable and `Wells Fargo` service's execution time is lower than `Chase Banking`'s.
- `Bank of America Credit Card Payment` service is unavailable and `Chase Banking` service's execution time is lower than `Wells Fargo`'s.
- `Bank of America Credit Card Payment` service's execution time is greater than 2,600 ms and `Wells Fargo` service's execution time is lower than `Chase Banking`'s.

- `Bank of America Credit Card Payment` service's execution time is greater than 2,600 ms and `Chase Banking` service's execution time is lower than `Wells Fargo`'s.

In this section, we describe a subset of context conditions and their related resolutions in our case study. Each resolution has a unique identifier (`ID`, where `ID` = 1...9). Resolutions can be defined in the computing infrastructure provided by MOSKitt4SPL[1] (see Figure B.1).



Resource Set
- Resolution R1
  - Condition B&NUnavailable
  - Action activate->AmazonBooks
  - Action activate->BookSearching
  - Action activate->BookDescription
  - Action activate->AmazonShoppingCart
  - Action activate->RelatedTitles
  - Action deactivate->BarnesAndNobleBooks
  - Action deactivate->BarnesAndNobleShoppingCart
  - Action deactivate->SearchBook
  - Action deactivate->ShowBookInfo
  - Action deactivate->ShowRelatedBooks
- Resolution R2
  - Condition AmazonHiExecTime
  - Action activate->BarnesAndNobleShoppingCart
  - Action activate->BarnesAndNobleBooks
  - Action activate->SearchBook
  - Action activate->ShowBookInfo
  - Action activate->ShowRelatedBooks
  - Action deactivate->AmazonShoppingCart
  - Action deactivate->AmazonBooks
  - Action deactivate->BookSearching
  - Action deactivate->BookDescription
  - Action deactivate->RelatedTitles

Selected 2 Objects

Figure B.1: Resolutions defined in MOSKitt4SPL.

- $R1_{B\&NUnavailable}$ is triggered when the `Barnes & Noble Books` service operation is unavailable. In this case, the `Amazon Books` functionality is activated. Since the `Amazon Books` feature requires the `Amazon Shopping Cart` feature, then the `Amazon Shopping Cart` functionality is also activated.

- $R2_{AmazonHiExecTime}$ is triggered when the execution time of the `Amazon Books` service operation is high. In this case, the `Barnes`

1 http://www.pros.upv.es/m4spl

& Noble Books functionality is activated. Since the Barnes & Noble Books feature requires the Barnes & Noble Shopping Cart feature, then the Barnes & Noble Shopping Cart functionality is also activated.

- $R3_{BNShoppingCartUnavailable}$ is triggered when the Barnes & Noble Shopping Cart service operation is unavailable. In this case, the Amazon Shopping Cart functionality is activated. Since the Amazon Shopping Cart functionality works with Amazon Books, then the Amazon Books functionality is also activated.

- $R4_{AmazonShoppingCartUnavailable}$ is triggered when the Amazon Shopping Cart service operation is unavailable. In this case, the Barnes & Noble Shopping Cart functionality is activated. Since the Barnes & Noble Shopping Cart functionality works with Barnes & Noble Books, then the Barnes & Noble Books functionality is also activated.

- $R5_{UPSHiExecTime \wedge DHLExecTimeLowerThanFedEX}$ is triggered when the execution time of the UPS Shipping service operation is higher than 1,500 ms and the execution time of the DHL Delivery service operation is lower than the execution time of the FedEX Express service operation. In this case, the UPS Shipping functionality is deactivated and the DHL Delivery functionality is activated.

- $R6_{DHLUnavailable \wedge FedExExecTimeLowerThanUPS}$ is triggered when the DHL Delivery service operation is unavailable and the execution time of the FedEX Express service operation is lower than the execution time of the UPS Shipping service operation. In this case, the DHL Delivery functionality is deactivated and the FedEX Express functionality is activated.

- $R7_{BankOfAmericaHiExecTime \wedge WellsFargoExecTimeLowerThanChaseBanking}$ is triggered when the execution time of the Bank of America Credit Card Payment service operation is higher than 2,600 ms and the execution time of the Wells Fargo Online service operation is lower than the execution time of the Chase Banking service operation. In this case, the Wells Fargo Online functionality is activated and the Bank of America Credit Card Payment functionality is deactivated.

- $R8_{EmailInvoiceHiExecTime}$ is triggered when the execution time of the E-mail Invoice service operation is greater than 1,000 ms.

In this case, the `SMS Invoice` functionality is activated and the `E-mail Invoice` functionality is deactivated.

- $R9_{SMSInvoiceHiExecTime}$ is triggered when the execution time of the `SMS Invoice` service operation is greater than 1,500 ms. In this case, the `E-mail Invoice` functionality is activated and the `SMS Invoice` functionality is deactivated.

B.3 WS-BPEL TEMPLATE

This section presents the WS-BPEL template that is used in the case study. In order to improve the readability of this template, it is presented in different listings. First, Listing B.1 shows the group of name spaces and the imports in this template (for common and variable service operations).

Listing B.1: Name spaces and imports in the WS-BPEL template.

```
<bpel:process name="BookShoppingProcess"
targetNamespace="http://www.harveyalferez.com/BookShoppingProcess"
suppressJoinFailure="yes"
xmlns:tns="http://www.harveyalferez.com/BookShoppingProcess"
xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
xmlns:ns1="http://www.harveyalferez.com/OnlineBookShopping/"
xmlns:ns2="http://www.example.org/relatedtitles/"
xmlns:ns3="http://www.orangecountry.org/onlineshopping/
    barnesandnoblebooks/"
xmlns:ns4="http://www.example.org/barnesandnobleshoppingcart/"
xmlns:ns5="http://www.orangecountry.org/onlineshopping/
    googleauthentication"
xmlns:ns6="http://www.orangecountry.org/onlineshopping/paymentcalculator/
    "
xmlns:ns7="http://www.orangecountry.org/onlineshopping/
    bankofamericacreditcardpayment/"
xmlns:ns8="http://www.example.org/SendConfirmationEmail/"
xmlns:ns9="http://www.example.org/UPSShipment/"
xmlns:ns10="http://www.example.org/FedExExpress/"
xmlns:ns11="http://www.example.org/DHLDelivery/"
xmlns:ns12="http://www.example.org/SendConfirmationSMS/"
xmlns:ns13="http://www.orangecountry.org/onlineshopping/chasebanking/"
xmlns:ns14="http://www.orangecountry.org/onlineshopping/wellsfargoonline/
    "
xmlns:ns15="http://www.example.org/amazonbooks/"
xmlns:ns16="http://www.example.org/amazonshoppingcart/">

<!-- Import the client WSDLs -->
```

```
<bpel:import namespace="http://www.example.org/amazonshoppingcart/"
    location="amazonshoppingcart.wsdl" importType="http://schemas.
    xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.example.org/amazonbooks/" location="
    amazonbooks.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></
    bpel:import>
<bpel:import namespace="http://www.example.org/relatedtitles/" location="
    relatedtitles.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></
    bpel:import>
<bpel:import namespace="http://www.orangecountry.org/onlineshopping/
    wellsfargoonline/" location="wellsfargoonline.wsdl" importType="
    http://schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.orangecountry.org/onlineshopping/
    chasebanking/" location="chasebanking.wsdl" importType="http://
    schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.example.org/SendConfirmationSMS/"
    location="smsinvoice.wsdl" importType="http://schemas.xmlsoap.org/
    wsdl/"></bpel:import>
<bpel:import namespace="http://www.example.org/DHLDelivery/" location="
    dhldelivery.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></
    bpel:import>
<bpel:import namespace="http://www.example.org/FedExExpress/" location="
    fedEXExpress.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></
    bpel:import>
<bpel:import namespace="http://www.example.org/UPSShipment/" location="
    upsshipping.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"></
    bpel:import>
<bpel:import namespace="http://www.example.org/SendConfirmationEmail/"
    location="emailinvoice.wsdl" importType="http://schemas.xmlsoap.org/
    wsdl/"></bpel:import>
<bpel:import namespace="http://www.orangecountry.org/onlineshopping/
    bankofamericacreditcardpayment/" location="
    bankofamericacreditcardpayment.wsdl" importType="http://schemas.
    xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.orangecountry.org/onlineshopping/
    paymentcalculator/" location="paymentcalculator.wsdl" importType="
    http://schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.orangecountry.org/onlineshopping/
    googleauthentication" location="googleauthentication.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.example.org/barnesandnobleshoppingcart
    /" location="barnesandnobleshoppingcart.wsdl" importType="http://
    schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.orangecountry.org/onlineshopping/
    barnesandnoblebooks/" location="barnesandnoblebooks.wsdl" importType
    ="http://schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:import namespace="http://www.harveyalferez.com/OnlineBookShopping/"
     location="OnlineBookShopping.wsdl" importType="http://schemas.
    xmlsoap.org/wsdl/"></bpel:import>
```

```
<bpel:import location="BookShoppingProcessArtifacts.wsdl" namespace="http://www.harveyalferez.com/BookShoppingProcess" importType="http://schemas.xmlsoap.org/wsdl/"/>
```

Listing B.2 shows the list of potential Web services participating in the service composition.

Listing B.2: List of potential Web services participating in the WS-BPEL process.

```
<bpel:partnerLinks>
    <bpel:partnerLink name="OnlineBookShoppingPL" partnerLinkType="tns:OnlineBookShoppingTypePL" myRole="provider"></bpel:partnerLink>
    <bpel:partnerLink name="BarnesAndNobleBooksPL" partnerLinkType="tns:BarnesAndNobleBooksTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="GoogleAuthenticationPL" partnerLinkType="tns:GoogleAuthenticationTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="BarnesAndNobleShoppingCartPL" partnerLinkType="tns:BarnesAndNobleShoppingCartTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="PaymentCalculatorPL" partnerLinkType="tns:PaymentCalculatorTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="BankOfAmericaPL" partnerLinkType="tns:BankOfAmericaTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="EMailInvoicePL" partnerLinkType="tns:EMailInvoiceTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="UPSShipmentPL" partnerLinkType="tns:UPSShipmentTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="AmazonBooksPL" partnerLinkType="tns:AmazonBooksTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="RelatedTitlesPL" partnerLinkType="tns:RelatedTitlesTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="AmazonShoppingCartPL" partnerLinkType="tns:AmazonShoppingCartTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="ChaseBankingPL" partnerLinkType="tns:ChaseBankingTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="DHLDeliveryPL" partnerLinkType="
        tns:DHLDeliveryTypePL" partnerRole="consumer"></bpel:partnerLink
        >
    <bpel:partnerLink name="FedEXExpressPL" partnerLinkType="
        tns:FedEXExpressTypePL" partnerRole="consumer"></
        bpel:partnerLink>
    <bpel:partnerLink name="SMSInvoicePL" partnerLinkType="
        tns:SMSInvoiceTypePL" partnerRole="consumer"></bpel:partnerLink>
    <bpel:partnerLink name="WellsFargoOnlinePL" partnerLinkType="
        tns:WellsFargoOnlineTypePL" partnerRole="consumer"></
        bpel:partnerLink>
</bpel:partnerLinks>
```

Listing B.3 shows the list of potential variables to be used within the WS-BPEL composition schema.

Listing B.3: List of potential variables within the WS-BPEL composition schema.

```
<bpel:variables>
    <bpel:variable name="OnlineBookShoppingRequest" messageType="
        ns1:buyBookRequest"></bpel:variable>
    <bpel:variable name="OnlineBookShoppingResponse" messageType="
        ns1:buyBookResponse"></bpel:variable>
    <bpel:variable name="RelatedTitlesRequest" messageType="
        ns2:showTitlesRequest"></bpel:variable>
    <bpel:variable name="RelatedTitlesResponse" messageType="
        ns2:showTitlesResponse"></bpel:variable>
    <bpel:variable name="BarnesAndNobleBooksRequest" messageType="
        ns3:searchBookRequest"></bpel:variable>
    <bpel:variable name="BarnesAndNobleBooksResponse" messageType="
        ns3:searchBookResponse"></bpel:variable>
    <bpel:variable name="BarnesAndNobleShoppingCartRequest" messageType="
        ns4:addbookRequest"></bpel:variable>
    <bpel:variable name="BarnesAndNobleShoppingCartResponse" messageType=
        "ns4:addbookResponse"></bpel:variable>
    <bpel:variable name="GoogleAuthenticationRequest" messageType="
        ns5:authenticateRequest"></bpel:variable>
    <bpel:variable name="GoogleAuthenticationResponse" messageType="
        ns5:authenticateResponse"></bpel:variable>
    <bpel:variable name="PaymentCalculatorRequest" messageType="
        ns6:calculateTotalRequest"></bpel:variable>
    <bpel:variable name="PaymentCalculatorResponse" messageType="
        ns6:calculateTotalResponse"></bpel:variable>
    <bpel:variable name="BankOfAmericaRequest" messageType="
        ns7:paymentRequest"></bpel:variable>
    <bpel:variable name="BankOfAmericaResponse" messageType="
        ns7:paymentResponse"></bpel:variable>
    <bpel:variable name="EMailInvoiceRequest" messageType="
        ns8:sendEmailRequest"></bpel:variable>
    <bpel:variable name="EMailInvoiceResponse" messageType="
        ns8:sendEmailResponse"></bpel:variable>
    <bpel:variable name="UPSShipmentRequest" messageType="
        ns9:UPSShipmentRequest"></bpel:variable>
    <bpel:variable name="UPSShipmentResponse" messageType="
        ns9:UPSShipmentResponse"></bpel:variable>
    <bpel:variable name="FedEXExpressRequest" messageType="
        ns10:FedExExpressRequest"></bpel:variable>
    <bpel:variable name="FedEXExpressResponse" messageType="
        ns10:FedExExpressResponse"></bpel:variable>
    <bpel:variable name="DHLDeliveryRequest" messageType="
        ns11:DHLDeliveryRequest"></bpel:variable>
    <bpel:variable name="DHLDeliveryResponse" messageType="
        ns11:DHLDeliveryResponse"></bpel:variable>
    <bpel:variable name="SMSInvoiceRequest" messageType="
        ns12:sendSMSRequest"></bpel:variable>
    <bpel:variable name="SMSInvoiceResponse" messageType="
        ns12:sendSMSResponse"></bpel:variable>
    <bpel:variable name="ChaseBankingRequest" messageType="
        ns13:paymentRequest"></bpel:variable>
    <bpel:variable name="ChaseBankingResponse" messageType="
        ns13:paymentResponse"></bpel:variable>
    <bpel:variable name="WellsFargoOnlineRequest" messageType="
        ns14:paymentRequest"></bpel:variable>
    <bpel:variable name="WellsFargoOnlineResponse" messageType="
        ns14:paymentResponse"></bpel:variable>
    <bpel:variable name="AmazonBooksRequest" messageType="
        ns15:searchBookRequest"></bpel:variable>
    <bpel:variable name="AmazonBooksResponse" messageType="
        ns15:searchBookResponse"></bpel:variable>
    <bpel:variable name="AmazonShoppingCartRequest" messageType="
        ns16:addbookRequest"></bpel:variable>
    <bpel:variable name="AmazonShoppingCartResponse" messageType="
        ns16:addbookResponse"></bpel:variable>
</bpel:variables>
```

Listing B.4 shows the set of activities that coordinate the flow of messages across the services integrated within the BP in the case study. This WS-BPEL template has five variation points: 1) the `Book Management` variation point (`VP_BookManagement`) in line 3; 2) the `Shopping Cart` variation point (`VP_ShoppingCart`) in line 4; 3) the `Credit Cart Payment` variation point (`VP_CreditCardPayment`) in line 37; 4) the `Shipment` variation point (`VP_Shipment`) in line 40; and 5) the `Invoice Delivery` variation point (`VP_InvoiceDelivery`) in line 41.

Listing B.4: Orchestration logic.

```
<bpel:sequence name="main">
    <bpel:receive name="receiveInput" createInstance="yes" partnerLink="
        OnlineBookShoppingPL" operation="buyBook" portType="
        ns1:OnlineBookShopping" variable="OnlineBookShoppingRequest" />
    <documentation>VP_BookManagement</documentation>
    <documentation>VP_ShoppingCart</documentation>
    <bpel:assign validate="no" name="AssignC">
        <bpel:copy>
            <bpel:from>
                <bpel:literal>
                    <tns:authenticate xmlns:tns="http://www.orangecountry
                        .org/onlineshopping/googleauthentication"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                        instance">
                        <userData>
                            <userName>userName</userName>
                            <password>password</password>
                        </userData>
                    </tns:authenticate>
                </bpel:literal>
            </bpel:from>
            <bpel:to variable="GoogleAuthenticationRequest" part="
                parameters"></bpel:to></bpel:copy>
            <bpel:copy>
                <bpel:from part="parameters" variable="
                    OnlineBookShoppingRequest">
                    <bpel:query queryLanguage="
                        urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[userData]]>
                    </bpel:query>
                </bpel:from>
                <bpel:to part="parameters" variable="
                    GoogleAuthenticationRequest">
                    <bpel:query queryLanguage="
                        urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[userData]]>
                    </bpel:query>
                </bpel:to>
            </bpel:copy>
    </bpel:assign>
    <bpel:invoke name="GoogleAuthentication" partnerLink="
        GoogleAuthenticationPL" operation="authenticate" portType="
        ns5:googleauthentication" inputVariable="
        GoogleAuthenticationRequest" outputVariable="
        GoogleAuthenticationResponse"></bpel:invoke>
    <bpel:if name="authorizedUser">
```

```
        <bpel:condition><![CDATA[$GoogleAuthenticationResponse.parameters
            /result = "OK"]]></bpel:condition>
        <bpel:sequence>
        <bpel:invoke name="PaymentCalculator" partnerLink="
            PaymentCalculatorPL" operation="calculateTotal" portType="
            ns6:paymentcalculator" inputVariable="
            PaymentCalculatorRequest" outputVariable="
            PaymentCalculatorResponse">
        </bpel:invoke>
        <documentation>VP_CreditCardPayment</documentation>
        <bpel:sequence name="Sequence">
            <bpel:flow name="Flow">
                <documentation>VP_Shipment</documentation>
                <documentation>VP_InvoiceDelivery</documentation>
            </bpel:flow>
        </bpel:sequence>
    <bpel:else>
        <bpel:assign validate="no" name="AssignJ">
            <bpel:copy>
                <bpel:from>
                    <bpel:literal>
                        <tns:buyBookResponse xmlns:tns="http://www.
                            harveyalferez.com/OnlineBookShopping/"
                            xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                            instance">
                            <result>result</result>
                        </tns:buyBookResponse>
                    </bpel:literal>
                </bpel:from>
                <bpel:to variable="OnlineBookShoppingResponse" part="
                    parameters"></bpel:to>
            </bpel:copy>
            <bpel:copy>
                <bpel:from part="parameters" variable="
                    PaymentCalculatorResponse">
                    <bpel:query queryLanguage="
                        urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[result]]>
                    </bpel:query>
                </bpel:from>
                <bpel:to part="parameters" variable="
                    OnlineBookShoppingResponse">
                    <bpel:query queryLanguage="
                        urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
                        <![CDATA[result]]>
                    </bpel:query>
                </bpel:to>
            </bpel:copy>
        </bpel:assign>
    </bpel:else>
    </bpel:if>
    </bpel:sequence>
<bpel:else>
    <bpel:assign validate="no" name="AssignK">
        <bpel:copy>
            <bpel:from>
                <bpel:literal>
                    <tns:buyBookResponse xmlns:tns="http://www.
                        harveyalferez.com/OnlineBookShopping/" xmlns:xsi
                        ="http://www.w3.org/2001/XMLSchema-instance"><
                        result>result</result>
                    </tns:buyBookResponse>
                </bpel:literal>
            </bpel:from>
            <bpel:to variable="OnlineBookShoppingResponse" part="
                parameters"></bpel:to>
        </bpel:copy>
        <bpel:copy>
            <bpel:from part="parameters" variable="
                GoogleAuthenticationResponse">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0
                    :sublang:xpath1.0">
                    <![CDATA[result]]>
                </bpel:query>
            </bpel:from>
            <bpel:to part="parameters" variable="
                OnlineBookShoppingResponse">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0
                    :sublang:xpath1.0">
                    <![CDATA[result]]>
                </bpel:query>
            </bpel:to>
        </bpel:copy>
    </bpel:assign>
</bpel:else>
</bpel:if>
<bpel:reply name="Reply" partnerLink="OnlineBookShoppingPL" operation="
    buyBook" portType="ns1:OnlineBookShopping" variable="
    OnlineBookShoppingResponse">
</bpel:reply>
</bpel:sequence>
</bpel:process>
```

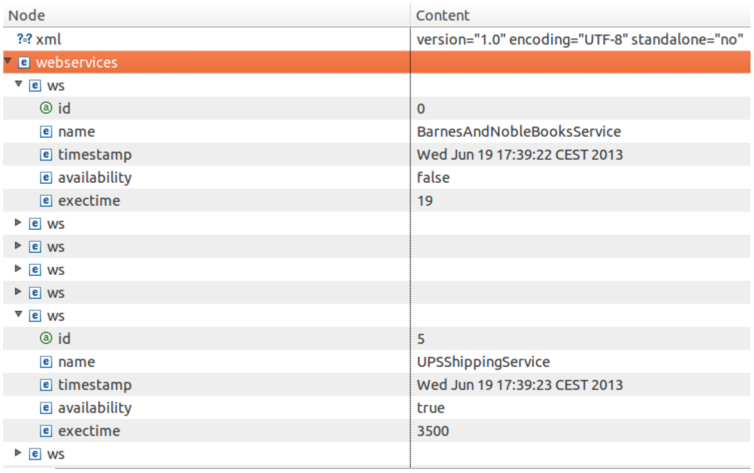
B.4 DYNAMIC ADAPTATION SCENARIO

In this section, we describe a dynamic adaptation scenario to face two context conditions: `B&NUnavailable` and `UPSHiExecTime`. These context conditions are related to resolutions *R1* and *R5* in Section B.2, respectively. They are fulfilled in a tight time frame of one second. The objective of this scenario is to demonstrate the feasibility of our approach even with context conditions that are triggered in tight time frames.

B.4.1 *Requirements and Initial WS-BPEL Composition Schema*

This subsection presents the requirements and the initial WS-BPEL composition schema of this scenario. First, the requirements of this scenario are as follows:

- The XML file, which is updated by the Context Monitor with context information, was modified manually in order to simulate the closed occurrence of the `B&NUnavailable` and the `UPSHiExecTime` context conditions. Figure B.2 shows a fragment of this XML file. In this case, the `Barnes and Noble Books` service operation is currently unavailable and the execution time of the `UPS Shipping` service operation is higher than 1,500 ms. These two events are separated by just one second.



Node	Content
?=? xml	version="1.0" encoding="UTF-8" standalone="no"
▼ webservices	
▼ ws	
id	0
name	BarnesAndNobleBooksService
timestamp	Wed Jun 19 17:39:22 CEST 2013
availability	false
exectime	19
▶ ws	
▶ ws	
▶ ws	
▶ ws	
▼ ws	
id	5
name	UPSShippingService
timestamp	Wed Jun 19 17:39:23 CEST 2013
availability	true
exectime	3500
▶ ws	

Figure B.2: Context data in the dynamic adaptation scenario.

- The initial configuration in the variability model of this scenario has the following *inactive* features: `Amazon Books`, `Book Searching`, `Book Description`, `Related Titles`, `Amazon Shopping Cart`, `Wells Fargo Online`, `Chase Banking`, `SMS Invoice`, `DHL Delivery`, and `FedEX Express`.

- In order to execute this scenario, the following tools have to be started one after the other:

  1. Apache Axis2[2], which runs the Web services.
  2. The Context Monitor.
  3. Apache ODE[3], which executes the service compositions in the deployment directories.
  4. MoRE-WS, which guides the dynamic adaptation of the service composition using models at runtime.

Figure B.3 shows the initial WS-BPEL composition schema for this scenario (visualized in the Eclipse BPEL Designer Project[4]). The `Barnes and Noble Books` composite service is in charge of invoking the `Search Book`, `Show Book Info`, and `Show Related Titles` atomic services. As a result, the WS-BPEL composition schema omits these atomic services. The client sends the set of books that have been put into the shopping cart in the following format: *book*$_1$*/book*$_2$*/.../book*$_n$. Therefore, the WS-BPEL composition schema omits the loop to add books into the shopping cart.

The console of Apache Axis2 shows the output in Listing B.5 when the WS-BPEL composition schema in Figure B.3 is invoked. In this case, the customer looks for two books entitled *"Web Services"* and *"Java Programming"*. For the sake of readability, the following icon helps to visualize the listings that represent outputs in console:

2 http://axis.apache.org/axis2/java/core
3 http://ode.apache.org
4 http://www.eclipse.org/bpel

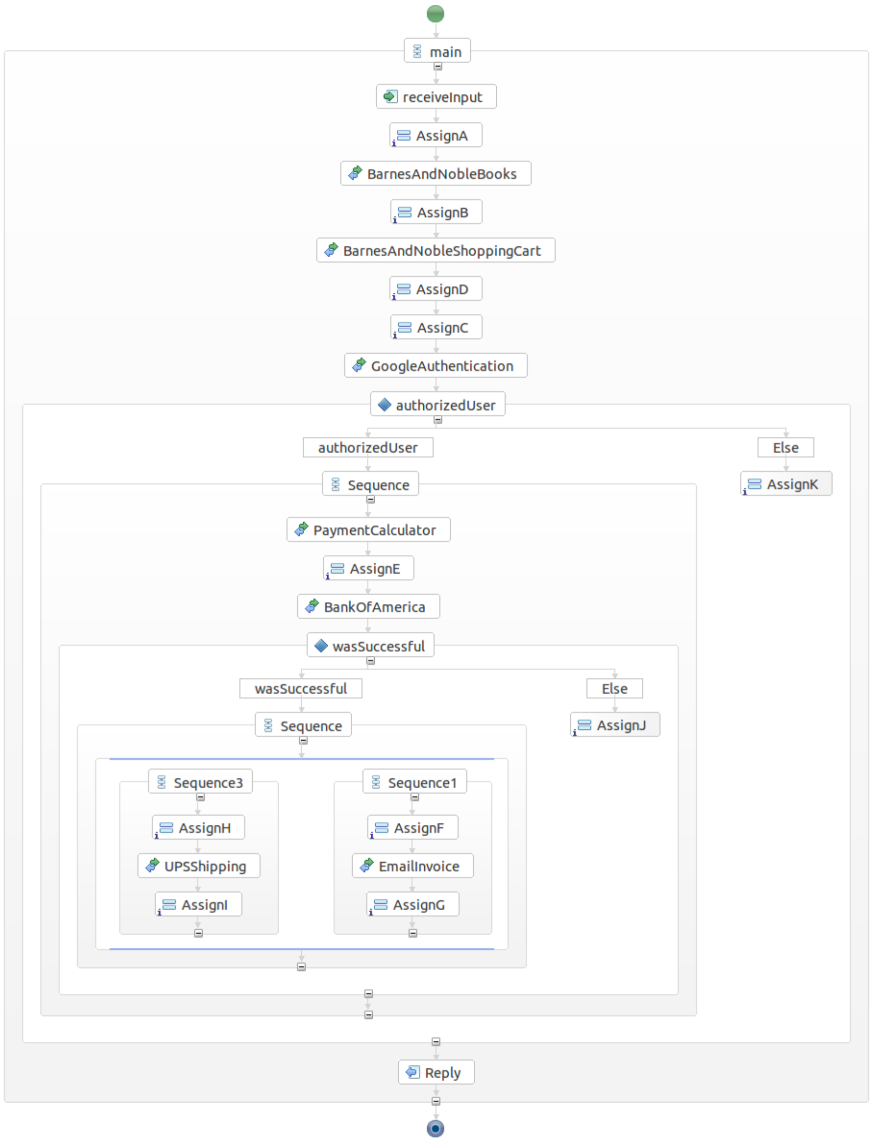


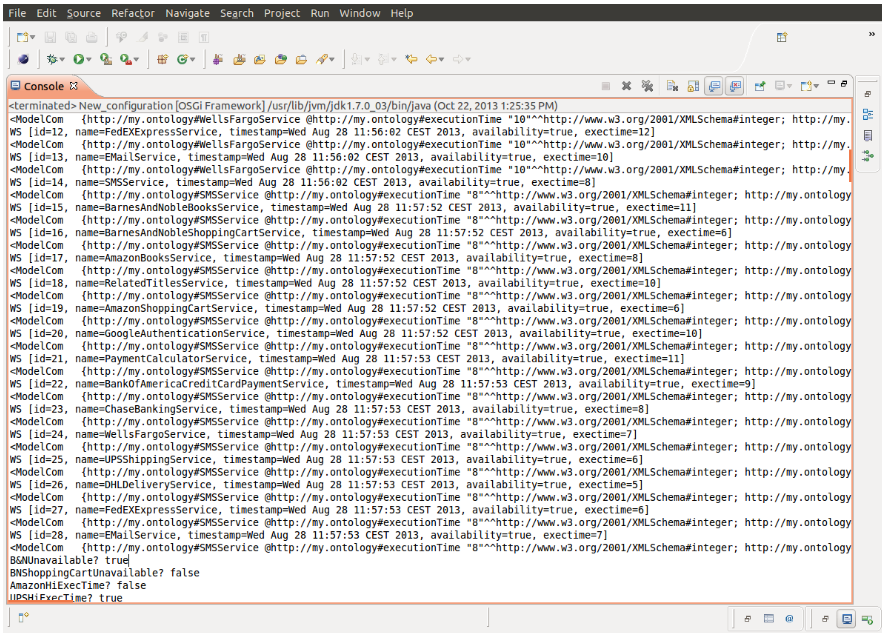
Figure B.3: Initial WS-BPEL composition schema.

Listing B.5: Output in the console of Apache Axis2 when invoking the initial WS-BPEL composition schema.

```
<<Barnes and Noble Books Service>>: Web services:
Authors: Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju.
     Publisher: Springer; 2004 edition (September 15, 2009).
Related titles:
- 1. Web Services Essentials.
- 2. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL
     and RESTful Web Services.
- 3. Web Services, Service-Oriented Architectures, and Cloud Computing:
     The Savvy Manager's Guide.

<<Barnes and Noble Books Service>>: Java programming:
Author: Poornachandra Sarang. Publisher: McGraw-Hill Osborne Media; 1
     edition (January 20, 2012).
Related titles:
- 1. Effective Java.
- 2. Java, the Complete Reference.
- 3. What's New in Java 7.
- 4. Java for Dummies.
- 5. Learning Java.
- 6. Building Java Programs: A Back to Basics Approach.
=========================================================
<<Barnes and Noble Shopping Cart Service>>: Querying price for books...
     $150.0
=========================================================
<<Identify User Service>>: Welcome Harvey!
=========================================================
<<Payment Calculator Service>>: Total (with taxes)... $180.0
=========================================================
<<Bank of America Service>>: Successful payment!
=========================================================
<<Send Invoice E-mail Service>>: Message to: harveyalferez@um.edu.mx.
     Message: Congratulations! It was a successful transaction!
=========================================================
<<UPS Shipment Service>>: Your product has been delivered to: C Santa
     Bárbara 19-3, Alfara del Patriarca, Valencia.
=========================================================
```

MoRE-WS shows in console the adaptation actions that are carried out when context conditions have been fulfilled (see Figure B.4). In the following subsections, we describe the key aspects of the information that is shown in console and what happens in background when the `B&NUnavailable` and the `UPSHiExecTime` context conditions are fulfilled.


File Edit Source Refactor Navigate Search Project Run Window Help
Console

```
<terminated> New_configuration [OSGi Framework] /usr/lib/jvm/jdk1.7.0_03/bin/java (Oct 22, 2013 1:25:35 PM)
<ModelCom   {http://my.ontology#WellsFargoService @http://my.ontology#executionTime "10"^^http://www.w3.org/2001/XMLSchema#integer; http://my.
WS [id=12, name=FedEXExpressService, timestamp=Wed Aug 28 11:56:02 CEST 2013, availability=true, exectime=12]
<ModelCom   {http://my.ontology#WellsFargoService @http://my.ontology#executionTime "10"^^http://www.w3.org/2001/XMLSchema#integer; http://my.
WS [id=13, name=EMailService, timestamp=Wed Aug 28 11:56:02 CEST 2013, availability=true, exectime=10]
<ModelCom   {http://my.ontology#WellsFargoService @http://my.ontology#executionTime "10"^^http://www.w3.org/2001/XMLSchema#integer; http://my.
WS [id=14, name=SMSService, timestamp=Wed Aug 28 11:56:02 CEST 2013, availability=true, exectime=8]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=15, name=BarnesAndNobleBooksService, timestamp=Wed Aug 28 11:57:52 CEST 2013, availability=true, exectime=11]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=16, name=BarnesAndNobleShoppingCartService, timestamp=Wed Aug 28 11:57:52 CEST 2013, availability=true, exectime=6]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=17, name=AmazonBooksService, timestamp=Wed Aug 28 11:57:52 CEST 2013, availability=true, exectime=8]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=18, name=RelatedTitlesService, timestamp=Wed Aug 28 11:57:52 CEST 2013, availability=true, exectime=10]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=19, name=AmazonShoppingCartService, timestamp=Wed Aug 28 11:57:52 CEST 2013, availability=true, exectime=6]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=20, name=GoogleAuthenticationService, timestamp=Wed Aug 28 11:57:52 CEST 2013, availability=true, exectime=10]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=21, name=PaymentCalculatorService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=11]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=22, name=BankOfAmericaCreditCardPaymentService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=9]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=23, name=ChaseBankingService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=8]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=24, name=WellsFargoService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=7]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=25, name=UPSShippingService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=6]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=26, name=DHLDeliveryService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=5]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=27, name=FedEXExpressService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=6]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
WS [id=28, name=EMailService, timestamp=Wed Aug 28 11:57:53 CEST 2013, availability=true, exectime=7]
<ModelCom   {http://my.ontology#SMSService @http://my.ontology#executionTime "8"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology
B&NUnavailable? true
BNShoppingCartUnavailable? false
AmazonHiExecTime? false
UPSHiExecTime? true
```

Figure B.4: Console of MoRE-WS during a dynamic adaptation scenario.

B.4.2 *Dynamic Adaptation for B&NUnavailable*

As soon as MoRE-WS starts, it shows in console the information in Listing B.6. In line 1, MoRE-WS shows that an empty context model is created. Then, it reads the latest set of observations in the stream database. For every context observation, MoRE-WS shows the identification of the observation, the name of the observed service operation, the timestamp of the observation, and the current availability and execution time of the service operation. In addition, MoRE-WS shows the information that has been inserted into the context model. Line 4 shows that the `Barnes & Noble Books` service operation is currently unavailable (i.e., `availability` = false). Line 8 shows that the execution time of the `UPS Shipping` service operation is 3,500 millisecond, which is much higher than the desired SLA (less than 1,500 ms).

Listing B.6: Output in the console of MoRE-WS when observing the context.

```
An empty context model was created: <ModelCom   {} | >
Reading the Stream Database...

WS [id=0, name=BarnesAndNobleBooksService, timestamp=Wed Jun 19 17:39:22
     CEST 2013, availability=false, exectime=19] <ModelCom   {http://my.
     ontology#BarnesAndNobleBooksService @http://my.ontology#
     executionTime "19"^^http://www.w3.org/2001/XMLSchema#integer; http
     ://my.ontology#BarnesAndNobleBooksService @http://my.ontology#
     isAvailable "false"} | >

WS [id=1, name=BarnesAndNobleShoppingCartService, timestamp=Wed Jun 19
     17:39:23 CEST 2013, availability=true, exectime=219] <ModelCom   {
     http://my.ontology#BarnesAndNobleBooksService @http://my.ontology#
     executionTime "19"^^http://www.w3.org/2001/XMLSchema#integer; http
     ://my.ontology#BarnesAndNobleBooksService @http://my.ontology#
     isAvailable "false"; http://my.ontology#
     BarnesAndNobleShoppingCartService @http://my.ontology#executionTime
     "219"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#
     BarnesAndNobleShoppingCartService @http://my.ontology#isAvailable "
     true"} | >
...
WS [id=5, name=UPSShippingService, timestamp=Wed Jun 19 17:39:23 CEST
     2013, availability=true, exectime=3500] <ModelCom   {http://my.
     ontology#PaymentCalculatorService @http://my.ontology#executionTime
     "38"^^http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#
     PaymentCalculatorService @http://my.ontology#isAvailable "true";
     http://my.ontology#UPSShippingService @http://my.ontology#
     executionTime "3500"^^http://www.w3.org/2001/XMLSchema#integer; http
     ://my.ontology#UPSShippingService @http://my.ontology#isAvailable "
     true"; http://my.ontology#BarnesAndNobleBooksService @http://my.
     ontology#executionTime "19"^^http://www.w3.org/2001/XMLSchema#
     integer; http://my.ontology#BarnesAndNobleBooksService @http://my.
     ontology#isAvailable "false"; http://my.ontology#
     BankOfAmericaCreditCardPaymentService @http://my.ontology#
     executionTime "48"^^http://www.w3.org/2001/XMLSchema#integer; http
     ://my.ontology#BankOfAmericaCreditCardPaymentService @http://my.
     ontology#isAvailable "true"; http://my.ontology#
     GoogleAuthenticationService @http://my.ontology#executionTime "40"^^
     http://www.w3.org/2001/XMLSchema#integer; http://my.ontology#
     GoogleAuthenticationService @http://my.ontology#isAvailable "true";
     http://my.ontology#BarnesAndNobleShoppingCartService @http://my.
     ontology#executionTime "219"^^http://www.w3.org/2001/XMLSchema#
     integer; http://my.ontology#BarnesAndNobleShoppingCartService @http
     ://my.ontology#isAvailable "true"} | >
...
```

Listing B.7 shows the next messages that MoRE-WS displays in console. First, MoRE-WS shows in lines 1 and 4 that the `B&NUnavailable` and the `UPSHiExecTime` context conditions have been fulfilled (i.e., they are TRUE). MoRE-WS deals with context events in sequence. Therefore, it carries out a dynamic adaptation first for the `B&NUnavailable` context condition and then for the `UPSHiExecTime` context condition.

In order to deal with the `B&NUnavailable` context condition, line 11 shows that MoRE-WS moved from the configuration (or state) `S1` to the configuration `S2` in the state machine, which abstracts the adaptation space. Line 15 shows the activation and deactivation of features that are carried out in the variability model by the resolution for the `B&NUnavailable` context condition. Since the `Barnes & Noble Books` is unavailable, its functionality is deactivated (together with its related shopping cart) and the `Amazon Books` functionality is activated. Line 17 shows the previous configuration and line 19 shows the new configuration of the variability model.

Listing B.7: Output in the console of MoRE-WS when executing the resolution for the `B&NUnavailable` context condition.

```
1  B&NUnavailable? true
2  BNShoppingCartUnavailable? false
3  AmazonHiExecTime? false
4  UPSHiExecTime? true
5  ...
6
7  =========================
8
9  CONTEXT EVENT: B&NUnavailable-->true
10 CONFIGURATION CHANGE EVENT:
11     - State Machine Model: StateMachine.sm4spl, Previous State: S1,
          Current State: S2
12     - Resolution:
13         - Condition: B&NUnavailable
14
15         - Changes: Activations: {AmazonShoppingCart, RelatedTitles,
              AmazonBooks}, Deactivations: {BarnesAndNobleBooks,
              BarnesAndNobleShoppingCart}
16
17     - Previous Configuration: {OnlineBookShopping->ACTIVE,
          UserAuthentication->ACTIVE, TotalCalculation->ACTIVE,
          BookManagement->ACTIVE, GoogleAuthentication->ACTIVE,
          PaymentCalculator->ACTIVE, LookForABook->ACTIVE,
          BarnesAndNobleBooks->ACTIVE, BarnesAndNobleShoppingCart->ACTIVE,
           ShoppingCart->ACTIVE, AmazonShoppingCart->INACTIVE,
          RelatedTitles->INACTIVE, AmazonBooks->INACTIVE,
          CreditCardPayment->ACTIVE, BankOfAmericaCreditCardPayment->
          ACTIVE, WellsFargoOnline->INACTIVE, ChaseBanking->INACTIVE,
          Payment->ACTIVE, ValidateCreditCard->ACTIVE,
          ContinueValidTransaction->ACTIVE, FinalizeInvalidTransaction->
          ACTIVE, InvoiceDelivery->ACTIVE, Shipment->ACTIVE, EMailInvoice
          ->ACTIVE, SMSInvoice->INACTIVE, UPSShipping->ACTIVE, DHLDelivery
          ->INACTIVE, FedEXExpress->INACTIVE}

      - New Configuration: {OnlineBookShopping->ACTIVE, UserAuthentication
          ->ACTIVE, TotalCalculation->ACTIVE, BookManagement->ACTIVE,
          GoogleAuthentication->ACTIVE, PaymentCalculator->ACTIVE,
          LookForABook->ACTIVE, BarnesAndNobleBooks->INACTIVE,
          BarnesAndNobleShoppingCart->INACTIVE, ShoppingCart->ACTIVE,
          AmazonShoppingCart->ACTIVE, RelatedTitles->ACTIVE, AmazonBooks->
          ACTIVE, CreditCardPayment->ACTIVE,
          BankOfAmericaCreditCardPayment->ACTIVE, WellsFargoOnline->
          INACTIVE, ChaseBanking->INACTIVE, Payment->ACTIVE,
          ValidateCreditCard->ACTIVE, ContinueValidTransaction->ACTIVE,
          FinalizeInvalidTransaction->ACTIVE, InvoiceDelivery->ACTIVE,
          Shipment->ACTIVE, EMailInvoice->ACTIVE, SMSInvoice->INACTIVE,
          UPSShipping->ACTIVE, DHLDelivery->INACTIVE, FedEXExpress->
          INACTIVE}
  ...
```

Then, MoRE-WS shows the generated reconfiguration plan. Line 1 in Listing B.8 shows the BPMN variant models that are incremented. Line 3 shows the BPMN variant models that are decremented. The BPMN `documentation` element indicates the variation point where the BPMN variant model has to be bound.

Listing B.8: Output in the console of MoRE-WS when generating a reconfiguration plan.

```
1 - Composition Model Increments: [bpmn.impl.SubProcessImpl@14b274ea (iD:
      AmazonShoppingCartVar) (documentation: ShoppingCart, name:
      AmazonShoppingCartVar, ncname: null, orderedMessages: null,
      activityType: <unset>, looping: <unset>) (isTransaction: <unset>),
      bpmn.impl.SubProcessImpl@612e0b82 (iD: AmazonBooksVar) (
      documentation: BookManagement, name: AmazonBooksVar, ncname: null,
      orderedMessages: null, activityType: <unset>, looping: <unset>) (
      isTransaction: <unset>)]
2
3 - Composition Model Decrements: [bpmn.impl.SubProcessImpl@69037c66 (iD:
      BarnesAndNobleBooksVar) (documentation: BookManagement, name:
      BarnesAndNobleBooksVar, ncname: null, orderedMessages: null,
      activityType: <unset>, looping: <unset>) (isTransaction: <unset>),
      bpmn.impl.SubProcessImpl@53583279 (iD: BarnesAndNobleShoppingCartVar
      ) (documentation: ShoppingCart, name: BarnesAndNobleShoppingCartVar,
    ncname: null, orderedMessages: null, activityType: <unset>, looping
    : <unset>) (isTransaction: <unset>)]
...
```

Then, MoRE-WS shows the BPMN elements in variant models that were added into the `Book Management` and the `Shopping Cart` variation points (see Listing B.9). BPMN elements that had been previously bound into these variation points are deleted in order to avoid inconsistent behavior.

Listing B.9: Output in the console of MoRE-WS when adding BPMN elements into two variation points.

```
- Additions into: BookManagement

    - Found variant model: AmazonBooksVar

        Added element: bpmn.impl.ActivityImpl@313d6965 (iD: AmazonBooks)
            (documentation: null, name: null, ncname: null,
            orderedMessages: null, activityType: Task, looping: false).
        Added element: bpmn.impl.ActivityImpl@36d27f24 (iD: _nGRGoOBtEeGx
            9pu7hF_iGg) (documentation: null, name: null, ncname: null,
            orderedMessages: null, activityType: EventStartEmpty,
            looping: false).
        Added element: bpmn.impl.ActivityImpl@3599309a (iD: _oo6lIOBtEeGx
            9pu7hF_iGg) (documentation: null, name: null, ncname: null,
            orderedMessages: null, activityType: EventEndEmpty, looping:
             false).
        Added element: bpmn.impl.ActivityImpl@5c45e9f3 (iD: RelatedTitles
            ) (documentation: null, name: null, ncname: null,
            orderedMessages: null, activityType: Task, looping: false):
             Added element: bpmn.impl.SequenceEdgeImpl@1d64b77a (iD: _
            ncQkwOBtEeGx9pu7hF_iGg) (documentation: null, name: null,
            ncname: null, conditionType: <unset>, isDefault: <unset>).
        Added element: bpmn.impl.SequenceEdgeImpl@13c26b5 (iD: _
            ncQkwOBtEeGx9pu7hF_gHA) (documentation: null, name: null,
            ncname: null, conditionType: <unset>, isDefault: <unset>).
        Added element: bpmn.impl.SequenceEdgeImpl@3f49b9a9 (iD: _o5
            pIwOBtEeGx9pu7hF_iGg) (documentation: null, name: null,
            ncname: null, conditionType: <unset>, isDefault: <unset>).

- Additions into: ShoppingCart

    - Found variant model: AmazonShoppingCartVar

        Added element: bpmn.impl.ActivityImpl@72027fd8 (iD:
            AmazonShoppingCart) (documentation: null, name: null, ncname
```

```
          : null, orderedMessages: null, activityType: Task, looping:
          false):
        Added element: bpmn.impl.ActivityImpl@2177862 (iD: _8MSBoOBuEeGx9
          pu7hF_iGg) (documentation: null, name: null, ncname: null,
          orderedMessages: null, activityType: EventStartEmpty,
          looping: false):
        Added element: bpmn.impl.ActivityImpl@52e06b00 (iD: _D3PiMOBvEeGx
          9pu7hF_iGg) (documentation: null, name: null, ncname: null,
          orderedMessages: null, activityType: EventEndEmpty, looping:
           false):
        Added element: bpmn.impl.SequenceEdgeImpl@6a659aeb (iD: _8f5
          hMOBuEeGx9pu7hF_iGg) (documentation: null, name: null,
          ncname: null, conditionType: <unset>, isDefault: <unset>):
        Added element: bpmn.impl.SequenceEdgeImpl@349df0df (iD: _
          AzUowOBwEeGx9pu7hF_iGg) (documentation: null, name: null,
          ncname: null, conditionType: <unset>, isDefault: <unset>):
...
```

Afterwards, MoRE-WS shows a message saying that the new version of the deployment directory has been successfully deployed (e.g. at `BookShopping151`). The new version of the deployment directory is created by merging WS-BPEL code fragments into the WS-BPEL template. For instance, Listing B.10 shows the WS-BPEL code fragment that maps to the `Amazon Books` variant model. This code fragment is used to invoke the `Amazon Books` and the `Related Titles` services.

Listing B.10: `Amazon Books` WS-BPEL code fragment.

```
<bpel:assign validate="no" name="AssignA">
    <bpel:copy>
        <bpel:from>
            <bpel:literal>
                <tns:searchBook xmlns:tns="http://www.example.org/
                     amazonbooks/" xmlns:xsi="http://www.w3.org/2001/
                     XMLSchema-instance">
                    <searchData>
                        <title>title</title>
                        <author>author</author>
                    </searchData>
                </tns:searchBook>
            </bpel:literal>
        </bpel:from>
        <bpel:to variable="AmazonBooksRequest" part="parameters"></bpel:
             to>
    </bpel:copy>
    <bpel:copy ignoreMissingFromData="no">
        <bpel:from part="parameters" variable="OnlineBookShoppingRequest
             ">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                 sublang:xpath1.0">
                <![CDATA[searchData]]>
            </bpel:query>
        </bpel:from>
        <bpel:to part="parameters" variable="AmazonBooksRequest">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                 sublang:xpath1.0">
                <![CDATA[searchData]]>
            </bpel:query>
        </bpel:to>
    </bpel:copy>
</bpel:assign>
<bpel:invoke name="AmazonBooks" partnerLink="AmazonBooksPL" operation="
     searchBook" portType="ns15:amazonbooks" inputVariable="
     AmazonBooksRequest" outputVariable="AmazonBooksResponse" />

<bpel:assign validate="no" name="Assign">
    <bpel:copy>
        <bpel:from>
            <bpel:literal>
                <tns:showTitles xmlns:tns="http://www.example.org/
                     relatedtitles/" xmlns:xsi="http://www.w3.org/2001/
                     XMLSchema-instance">
                    <bookData>
                        <bookName>bookName</bookName>
                    </bookData>
                </tns:showTitles>
            </bpel:literal>
        </bpel:from>
        <bpel:to variable="RelatedTitlesRequest" part="parameters"></bpel
             :to>
    </bpel:copy>
    <bpel:copy>
        <bpel:from part="parameters" variable="OnlineBookShoppingRequest
             ">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                 sublang:xpath1.0">
                <![CDATA[bookData]]>
            </bpel:query>
        </bpel:from>
        <bpel:to part="parameters" variable="RelatedTitlesRequest">
            <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                 sublang:xpath1.0">
                <![CDATA[bookData]]>
            </bpel:query>
        </bpel:to>
    </bpel:copy>
</bpel:assign>
```

```
<bpel:invoke name="RelatedTitles" partnerLink="RelatedTitlesPL" operation
    ="showTitles" portType="ns2:relatedtitles" inputVariable="
    RelatedTitlesRequest" outputVariable="RelatedTitlesResponse"></bpel:
    invoke>
```

Figure B.5 shows a fragment of the adapted WS-BPEL composition schema with the `Amazon Books`, `Related Titles`, and `Amazon Shopping Cart` services. Figure B.6 shows the new version of the deployment directory. MoRE-WS imports all the required artifacts to run the WS-BPEL composition schema (e.g. the WSDL files).

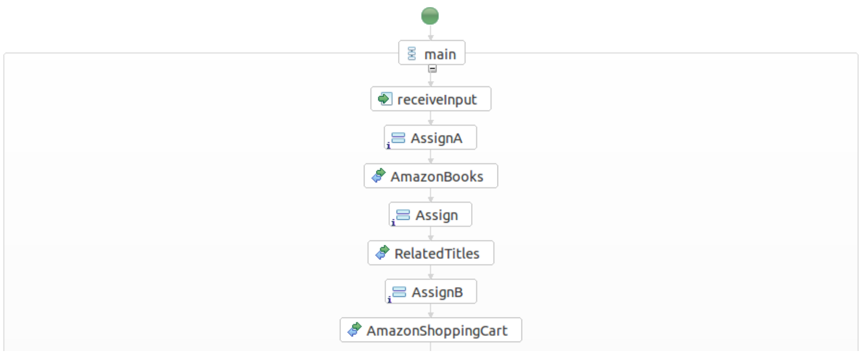


Figure B.5: Adapted WS-BPEL composition schema with the `Amazon Books`, `Related Titles`, and `Amazon Shopping Cart` services.

B.4.3 *Dynamic Adaptation for UPSHiExecTime*

Since a dynamic adaptation has been already triggered to face the `B&NUnavailable` context condition, MoRE-WS triggers the following adaptation for the `UPSHiExecTime` context condition. Listing B.11 shows the result of this new adaptation. In this scenario, the execution time of the `DHL Delivery` service operation is lower than the execution time of the `FedEX Express` service operation. Therefore, the `UPS Shipping` functionality is deactivated and the `DHL Delivery` functionality is activated.

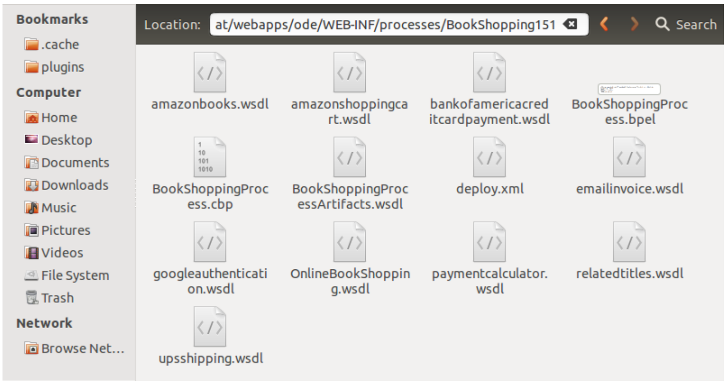


Figure B.6: New version of the deployment directory.

Listing B.11: Output in the console of MoRE-WS for the dynamic adaptation to face the UPSHiExecTime context condition.

```
CONTEXT EVENT: UPSHiExecTime-->true
CONFIGURATION CHANGE EVENT:
    - State Machine Model: StateMachine.sm4spl, previous State: S2,
        current State: S3
    - Resolution:
        - Condition: UPSHiExecTime

        - Changes: Additions: {DHLDelivery}, Removals: {UPSShipping}

    - Previous Configuration: {OnlineBookShopping->ACTIVE,
        UserAuthentication->ACTIVE, TotalCalculation->ACTIVE,
        BookManagement->ACTIVE, GoogleAuthentication->ACTIVE,
        PaymentCalculator->ACTIVE, LookForABook->ACTIVE,
        BarnesAndNobleBooks->INACTIVE, BarnesAndNobleShoppingCart->
        INACTIVE, ShoppingCart->ACTIVE, AmazonShoppingCart->ACTIVE,
        RelatedTitles->ACTIVE, AmazonBooks->ACTIVE, CreditCardPayment->
        ACTIVE, BankOfAmericaCreditCardPayment->ACTIVE, WellsFargoOnline
        ->INACTIVE, ChaseBanking->INACTIVE, Payment->ACTIVE,
        ValidateCreditCard->ACTIVE, ContinueValidTransaction->ACTIVE,
        FinalizeInvalidTransaction->ACTIVE, InvoiceDelivery->ACTIVE,
        Shipment->ACTIVE, EMailInvoice->ACTIVE, SMSInvoice->INACTIVE,
        UPSShipping->ACTIVE, DHLDelivery->INACTIVE, FedEXExpress->
        INACTIVE}

    - New Configuration: {OnlineBookShopping->ACTIVE, UserAuthentication
        ->ACTIVE, TotalCalculation->ACTIVE, BookManagement->ACTIVE,
        GoogleAuthentication->ACTIVE, PaymentCalculator->ACTIVE,
        LookForABook->ACTIVE, BarnesAndNobleBooks->INACTIVE,
        BarnesAndNobleShoppingCart->INACTIVE, ShoppingCart->ACTIVE,
        AmazonShoppingCart->ACTIVE, RelatedTitles->ACTIVE, AmazonBooks->
        ACTIVE, CreditCardPayment->ACTIVE,
        BankOfAmericaCreditCardPayment->ACTIVE, WellsFargoOnline->
        INACTIVE, ChaseBanking->INACTIVE, Payment->ACTIVE,
        ValidateCreditCard->ACTIVE, ContinueValidTransaction->ACTIVE,
        FinalizeInvalidTransaction->ACTIVE, InvoiceDelivery->ACTIVE,
        Shipment->ACTIVE, EMailInvoice->ACTIVE, SMSInvoice->INACTIVE,
        UPSShipping->INACTIVE, DHLDelivery->ACTIVE, FedEXExpress->
        INACTIVE}

----- Generated Reconfiguration Plan: -----

- Composition Model Increments: [bpmn.impl.SubProcessImpl@120f40e6 (iD:
    DHLDeliveryVar) (documentation: Shipment, name: DHLDeliveryVar,
    ncname: null, orderedMessages: null, activityType: <unset>, looping:
     <unset>) (isTransaction: <unset>)]

- Composition Model Decrements: [bpmn.impl.SubProcessImpl@7fb3815e (iD:
    UPSShippingVar) (documentation: Shipment, name: UPSShippingVar,
    ncname: null, orderedMessages: null, activityType: <unset>, looping:
     <unset>) (isTransaction: <unset>)]

-------------

Additions into: Shipment:

- DHLDeliveryVar Added element: bpmn.impl.ActivityImpl@17e01184 (iD:
    DHLDelivery). Added element: bpmn.impl.ActivityImpl@779dfd5a (iD: _
    FNkxEOBxEeGRxeVNv5H7bQ). Added element: bpmn.impl.ActivityImpl@25645
    d10 (iD: _GrSsQOBxEeGRxeVNv5H7bQ). Added element: bpmn.impl.
    SequenceEdgeImpl@471d463d (iD: _FoyGYOBxEeGRxeVNv5H7bQ). Added
    element: bpmn.impl.SequenceEdgeImpl@52ee7344 (iD: _HPQU8OBxEeGRxeVNv
    5H7bQ).

----- Generated WS-BPEL composition schema: -----

Successful deployment at: /usr/local/tomcat/webapps/ode/WEB-INF/processes
    /BookShopping152
```

The composition model is modified in every dynamic adaptation. To this end, MoRE-WS adapts the composition model with the actions in the reconfiguration plan. In order to generate reconfiguration actions, MoRE-WS queries the weaving model with the mappings between the features that are active in the new configuration of the feature model and BPMN subprocesses, which abstract variant models.

Listing B.12 shows a fragment of a mapping in this weaving model. This fragment depicts the relationship between the `DHL Delivery` feature and the BPMN subprocess that encapsulates the elements in the `DHL Delivery` variant model.

Listing B.12: A fragment of the weaving model used in dynamic adaptations.

```
<ownedElement xsi:type="_1:ElementEqual" xmi:id="ElementEqual20">
    <left xsi:type="_1:LeftElement" xmi:id="LeftElement20" name="
        DHLDelivery" element="ElementRef39"/>
    <right xsi:type="_1:RightElement" xmi:id="RightElement20" name="
        DHLDeliveryVar" element="ElementRef40"/>
</ownedElement>
```

In this scenario, the composition model was adapted two times, one to face the `B&NUnavailable` context condition and another time to face the `UPSHiExecTime` context condition. Listing B.13 shows a fragment of the last version of the adapted composition model with the rebound `Book Management`, `Shopping Cart`, and `Shipment` variation points.

Listing B.13: A fragment of the last version of the composition model in XMI format.

```
<vertices xsi:type="bpmn:SubProcess" iD="BookManagement" outgoingEdges="
    seq2" incomingEdges="seq1 seq4" name="BookManagement">
    <vertices xsi:type="bpmn:Activity" iD="AmazonBooks" activityType="
        Task" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="_nGRGoOBtEeGx9pu7hF_iGg"
        activityType="EventStartEmpty" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="_oo6lIOBtEeGx9pu7hF_iGg"
        activityType="EventEndEmpty" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="RelatedTitles" activityType="
        Task" looping="false"/>
    <sequenceEdges iD="_ncQkwOBtEeGx9pu7hF_iGg">
        <source href="targetModel.xmi#_nGRtsOBtEeGx9pu7hF_iGg"/>
            <target href="targetModel.xmi#_iWMC0eBtEeGx9pu7hF_iGg"/>
    </sequenceEdges>
    <sequenceEdges iD="_ncQkwOBtEeGx9pu7hF_gHA">
        <source href="targetModel.xmi#_iWMC0eBtEeGx9pu7hF_iGg"/>
        <target href="targetModel.xmi#RelatedTitles"/>
    </sequenceEdges>
    <sequenceEdges iD="_o5pIwOBtEeGx9pu7hF_iGg">
        <source href="targetModel.xmi#RelatedTitles"/>
        <target href="targetModel.xmi#_oo6lIeBtEeGx9pu7hF_iGg"/>
    </sequenceEdges>
</vertices>

```

```
<vertices xsi:type="bpmn:SubProcess" iD="ShoppingCart" outgoingEdges="seq3" incomingEdges="seq2" name="ShoppingCart">
    <vertices xsi:type="bpmn:Activity" iD="AmazonShoppingCart" activityType="Task" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="_8MSBoOBuEeGx9pu7hF_iGg" activityType="EventStartEmpty" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="_D3PiMOBvEeGx9pu7hF_iGg" activityType="EventEndEmpty" looping="false"/>
    <sequenceEdges iD="_8f5hMOBuEeGx9pu7hF_iGg">
        <source href="targetModel.xmi#_8MSosOBuEeGx9pu7hF_iGg"/>
        <target href="targetModel.xmi#_1RQwQuBuEeGx9pu7hF_iGg"/>
    </sequenceEdges>
    <sequenceEdges iD="_AzUowOBwEeGx9pu7hF_iGg">
        <source href="targetModel.xmi#_1RQwQuBuEeGx9pu7hF_iGg"/>
        <target href="targetModel.xmi#_D3PiMeBvEeGx9pu7hF_iGg"/>
    </sequenceEdges>
</vertices>

<vertices xsi:type="bpmn:SubProcess" iD="Shipment" outgoingEdges="seq13" incomingEdges="seq11" name="Shipment">
    <vertices xsi:type="bpmn:Activity" iD="DHLDelivery" activityType="Task" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="_FNkxEOBxEeGRxeVNv5H7bQ" activityType="EventStartEmpty" looping="false"/>
    <vertices xsi:type="bpmn:Activity" iD="_GrSsQOBxEeGRxeVNv5H7bQ" activityType="EventEndEmpty" looping="false"/>
    <sequenceEdges iD="_FoyGYOBxEeGRxeVNv5H7bQ">
        <source href="targetModel.xmi#_FNlYIOBxEeGRxeVNv5H7bQ"/>
        <target href="targetModel.xmi#_DGRGUeBxEeGRxeVNv5H7bQ"/>
    </sequenceEdges>
    <sequenceEdges iD="_HPQU8OBxEeGRxeVNv5H7bQ">
        <source href="targetModel.xmi#_DGRGUeBxEeGRxeVNv5H7bQ"/>
        <target href="targetModel.xmi#_GrSsQeBxEeGRxeVNv5H7bQ"/>
    </sequenceEdges>
</vertices>
...
```

Figure B.7 shows the last version of the WS-BPEL composition schema. This version uses the `Amazon Books`, `Related Titles`, `Amazon Shopping Cart`, and `DHL Delivery` services. The `DHL Delivery` WS-BPEL fragment was bound in the last adaptation (see Listing B.14).

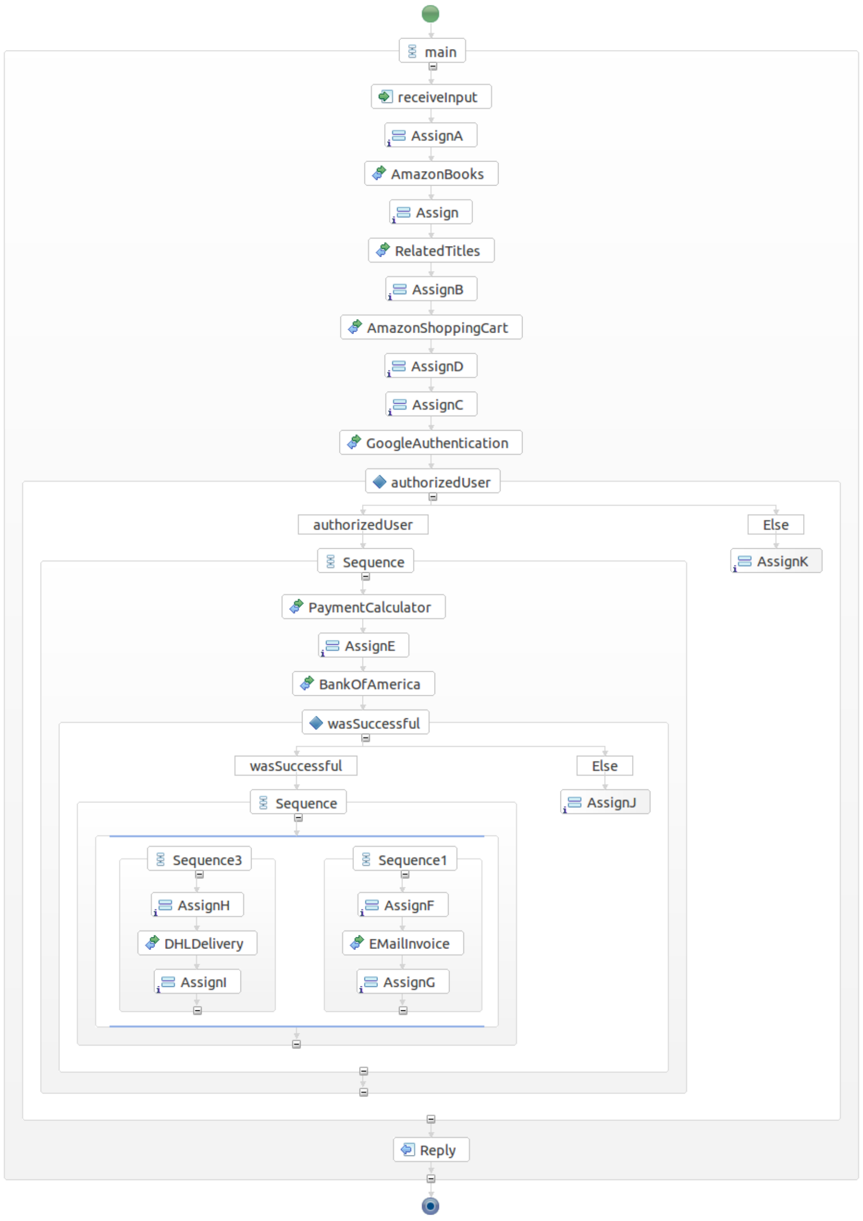


Figure B.7: Last version of the WS-BPEL composition schema with the `Amazon Books`, `Related Titles`, `Amazon Shopping Cart`, and `DHL Delivery` services.

Listing B.14: `DHL Delivery` WS-BPEL code fragment.

```
<bpel:sequence name="Sequence3">
    <bpel:assign validate="no" name="AssignH">
        <bpel:copy>
            <bpel:from>
                <bpel:literal>
                    <tns:DHLDelivery xmlns:tns="http://www.example.org/
                         DHLDelivery/" xmlns:xsi="http://www.w3.org/2001/
                         XMLSchema-instance">
                <sendData>
                    <address>address</address>
                    <phone>phone</phone>
                </sendData>
                    </tns:DHLDelivery>
                </bpel:literal>
            </bpel:from>
            <bpel:to variable="DHLDeliveryRequest" part="parameters"></
                 bpel:to>
        </bpel:copy>
        <bpel:copy>
            <bpel:from part="parameters" variable="
                 OnlineBookShoppingRequest">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                     sublang:xpath1.0">
                    <![CDATA[deliveryData]]>
                </bpel:query>
            </bpel:from>
            <bpel:to part="parameters" variable="DHLDeliveryRequest">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                     sublang:xpath1.0">
                    <![CDATA[sendData]]>
                </bpel:query>
            </bpel:to>
        </bpel:copy>
    </bpel:assign>
    <bpel:invoke name="DHLDelivery" partnerLink="DHLDeliveryPL" operation
        ="DHLDelivery" portType="ns11:DHLDelivery" inputVariable="
        DHLDeliveryRequest" outputVariable="DHLDeliveryResponse"></bpel:
        invoke>
    <bpel:assign validate="no" name="AssignI">
        <bpel:copy>
            <bpel:from>
                <bpel:literal>
                    <tns:buyBookResponse xmlns:tns="http://www.
                         harveyalferez.com/OnlineBookShopping/" xmlns:xsi
                         ="http://www.w3.org/2001/XMLSchema-instance">
                        <result>result</result>
                    </tns:buyBookResponse>
                </bpel:literal>
            </bpel:from>
            <bpel:to variable="OnlineBookShoppingResponse" part="
                 parameters"></bpel:to>
        </bpel:copy>
        <bpel:copy>
            <bpel:from part="parameters" variable="DHLDeliveryResponse">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                     sublang:xpath1.0">
                    <![CDATA[result]]>
                </bpel:query>
            </bpel:from>
            <bpel:to part="parameters" variable="
                 OnlineBookShoppingResponse">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                     sublang:xpath1.0">
                    <![CDATA[result]]>
                </bpel:query>
            </bpel:to>
        </bpel:copy>
    </bpel:assign>
</bpel:sequence>
```

If a client invokes the new version of the WS-BPEL composition schema in the deployment directory named `BookShopping152`, the console of Apache Axis2 shows the output in Listing B.15. In this case, the customer looks for two books entitled *"Web Services"* and *"Java Programming"*. The information of these two books is returned by the `Amazon Books` service. Afterwards, the `Amazon Shopping Cart` service is invoked to add the gross prices of these books. The `DHL Delivery` service is invoked instead of the `UPS Shipping` service. Compare this output with the initial output in Listing B.5.

Listing B.15: Output in the console of Apache Axis2 when invoking the adapted WS-BPEL composition schema.

```
<<Amazon Books Service>>: Web services:
+ Authors: Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju.
+ Publisher: Springer; 2004 edition (September 15, 2009).

<<Amazon Books Service>>: Java programming:
+ Author: Poornachandra Sarang.
+ Publisher: McGraw-Hill Osborne Media; 1 edition (January 20, 2012).
=========================================================
<<Related Titles Service>> Related titles for 'Web services':
    - A. Web Services, Service-Oriented Architectures, and Cloud
         Computing: The Savvy Manager's Guide.
    - B. Service Design Patterns: Fundamental Design Solutions for SOAP/
       WSDL and RESTful Web Services.
    - C. Web Services Essentials.
    - D. Java Web Services: Up and Running.

<<Related Titles Service>> Related titles for 'Java programming':
    - A. Effective Java.
    - B. Java, the Complete Reference.
=========================================================
<<Amazon Shopping Cart Service>>: Gross price for books: $150.0
=========================================================
<<IdentifyUserService>>: Welcome Harvey!
=========================================================
<<Payment Calculator Service>>: Total (with taxes)... $180.0
=========================================================
<<Bank of America Service>>: Successful payment!
=========================================================
<<Send Invoice E-mail Service>>: Message to: harveyalferez@um.edu.mx.
    Message: Congratulations! It was a successful transaction!
=========================================================
<<DHL Delivery Service>>: Your product has been delivered to... C Santa
    Bárbara 19-3 Alfara del Patriarca, Valencia. It is a service brought
     to you by DHL Delivery.
```

B.5 DYNAMIC EVOLUTION SCENARIO

In this section, we describe a dynamic evolution scenario to face an unknown context event in the open world. Figure B.3 shows the initial WS-BPEL composition schema for this scenario. MoRE-WS constantly analyzes the open world to find unknown context events (not considered at design time), which may affect the expected requirements. To this end, just after MoRE-WS checks all the predefined context condition, it looks for problematic unknown events.

Listing B.16 shows the first part of the console of MoRE-WS when triggering a dynamic evolution for a service operation that can negatively affect the `High Security` requirement. Line 1 shows a context observation carried out by the CONTEXT MONITOR. Afterwards, lines 3-6 show a fragment of the analysis carried out on context conditions. In this case, no predefined context condition is fulfilled. In line 9, MoRE-WS starts to look for unknown context events that may affect requirements. In line 11, MoRE-WS displays that it has found out that the `Barnes and Noble Books` service operation can affect the `High Security` requirement.

Listing B.16: Output in the console of MoRE-WS when observing the context and analyzing unknown context events.

```
WS [id=61, name=BarnesAndNobleBooksService, timestamp=Wed Jun 19 17:39:22
      CEST 2013, availability=true, exectime=5200] <ModelCom   {http://my
     .ontology#BarnesAndNobleBooksService @http://my.ontology#
     executionTime "5200"^^http://www.w3.org/2001/XMLSchema#integer; http
     ://my.ontology#BarnesAndNobleBooksService @http://my.ontology#
     isAvailable "true"} | >
...
B&NUnavailable? false
BNShoppingCartUnavailable? false
AmazonHiExecTime? false
UPSHiExecTime? false
...
=========================
Analyzing unknown context events...

The http://my.ontology#BarnesAndNobleBooksService Web service operation
     can affect the High Security requirement
...
```

Afterwards, the EVOLUTION PLANNER looks for a tactic in the requirements model to preserve the `High Security` requirement. Listing B.17 shows the console of MoRE-WS when the EVOLUTION PLANNER finds the `Deception` tactic with the highest positive contribution on the `High Security` requirement.

Listing B.17: Output in the console of MoRE-WS when the Evolution Planner looks for tactics to preserve a requirement.

```
Looking for tactis to preserve the High Security requirement...

    + Softgoal: openome_model.impl.SoftgoalImpl@7067ba82 (name:
         HighSecurity, system: true, boundary: false, exclusive: true,
         sequential: true, parallel: false,
         QualitativeReasoningCombinedLabel: None,
         QualitativeReasoningSatisfiedLabel: None,
         QualitativeReasoningDenialLabel: None,
         QuantitativeReasoningCombinedLabel: 0.0,
         QuantitativeReasoningDeniedLabel: 0.0,
         QuantitativeReasoningSatisfiedLabel: 0.0, initialEvalLabel: None
         ) (topic: null)
        + The Deception Tactic has the highest positive impact on the
             softgoal: openome_model.impl.TaskImpl@2e759230 (name:
             Deception, system: true, boundary: false, exclusive: true,
             sequential: true, parallel: false,
             QualitativeReasoningCombinedLabel: None,
             QualitativeReasoningSatisfiedLabel: None,
             QualitativeReasoningDenialLabel: None,
             QuantitativeReasoningCombinedLabel: 0.0,
             QuantitativeReasoningDeniedLabel: 0.0,
             QuantitativeReasoningSatisfiedLabel: 0.0, initialEvalLabel:
             None)
...
```

Listing B.18 shows a fragment of the requirements model in our case study.

Listing B.18: A fragment of the requirements model in XMI format.

```
<?xml version="1.0" encoding="UTF-8"?>
    <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:
        edu.toronto.cs.openome_model="http:///edu/toronto/cs/openome_
        model.ecore" xmlns:notation="http://www.eclipse.org/gmf/runtime
        /1.0.2/notation">
    <edu.toronto.cs.openome_model:Model xmi:id="_UzL0sPqMEeKDCPJR83H7bQ">
        <intentions xmi:type="edu.toronto.cs.openome_model:Goal" xmi:id
            ="_Vn2L0PqMEeKDCPJR83H7bQ" name="Keep Alive the Service
            Composition" parentDecompositions="_HVNXMPqNEeKDCPJR83H7bQ _
            H1SlcPqNEeKDCPJR83H7bQ _IUbYgPqNEeKDCPJR83H7bQ"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Softgoal" xmi:
            id="_X3JU8PqMEeKDCPJR83H7bQ" name="High Security"
            decompositions="_HVNXMPqNEeKDCPJR83H7bQ" contributesFrom="_q
            6bGsPq0EeKDCPJR83H7bQ _5voWIDdEEe057q-fith2Cw _6BejUDriEe00X
            _6KNXV0Dg"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Softgoal" xmi:
            id="_Y6lu8PqMEeKDCPJR83H7bQ" name="High Performance"
            decompositions="_H1SlcPqNEeKDCPJR83H7bQ" contributesFrom="_9
            DodgDdEEe057q-fith2Cw __7_Z8DriEe00X_6KNXV0Dg"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Softgoal" xmi:
            id="_ao2AgPqMEeKDCPJR83H7bQ" name="24/7 Availability"
            decompositions="_IUbYgPqNEeKDCPJR83H7bQ" contributesFrom="_
            sz5JwPq0EeKDCPJR83H7bQ _sS3gUPq0EeKDCPJR83H7bQ"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Task" xmi:id
            ="_dFoMAPqMEeKDCPJR83H7bQ" name="Temporal Separation"
            contributesTo="_q6bGsPq0EeKDCPJR83H7bQ _l8-OgDdFEe057q-fith2
            Cw"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Task" xmi:id
            ="_g_D7sPqMEeKDCPJR83H7bQ" name="Deception" contributesTo="_
            hfZUoDriEe00X_6KNXV0Dg _6BejUDriEe00X_6KNXV0Dg"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Task" xmi:id
            ="_iqSucPqMEeKDCPJR83H7bQ" name="Introduce Concurrency"
            contributesTo="_jJlg4DdFEe057q-fith2Cw __7_Z8DriEe00X_6KNXV0
            Dg"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Task" xmi:id
            ="_lLmNkPqMEeKDCPJR83H7bQ" name="Active Redundancy"
            contributesTo="_sS3gUPqOEeKDCPJR83H7bQ"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Task" xmi:id
            ="_nR3WgPqMEeKDCPJR83H7bQ" name="Shadow Operation"
            contributesTo="_sz5JwPqOEeKDCPJR83H7bQ"/>
        <intentions xmi:type="edu.toronto.cs.openome_model:Belief" xmi:id
            ="_f23d0DdDEe057q-fith2Cw" name="C1" contributesTo="_5
            voWIDdEEe057q-fith2Cw _9DodgDdEEe057q-fith2Cw"
            contributesFrom="_jJlg4DdFEe057q-fith2Cw _l8-OgDdFEe057q-
            fith2Cw _hfZUoDriEe00X_6KNXV0Dg"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HelpContribution" xmi:id="_q6bGsPqOEeKDCPJR83H7bQ" target="_
            X3JU8PqMEeKDCPJR83H7bQ" source="_dFoMAPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HelpContribution" xmi:id="_sS3gUPqOEeKDCPJR83H7bQ" target="_
            ao2AgPqMEeKDCPJR83H7bQ" source="_lLmNkPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HelpContribution" xmi:id="_sz5JwPqOEeKDCPJR83H7bQ" target="_
            ao2AgPqMEeKDCPJR83H7bQ" source="_nR3WgPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HelpContribution" xmi:id="_5voWIDdEEe057q-fith2Cw" target="_
            X3JU8PqMEeKDCPJR83H7bQ" source="_f23d0DdDEe057q-fith2Cw"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HelpContribution" xmi:id="_9DodgDdEEe057q-fith2Cw" target="_
            Y6lu8PqMEeKDCPJR83H7bQ" source="_f23d0DdDEe057q-fith2Cw"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            MakeContribution" xmi:id="_jJlg4DdFEe057q-fith2Cw" target="_
            f23d0DdDEe057q-fith2Cw" source="_iqSucPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HelpContribution" xmi:id="_l8-OgDdFEe057q-fith2Cw" target="_
            f23d0DdDEe057q-fith2Cw" source="_dFoMAPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            HurtContribution" xmi:id="_hfZUoDriEe00X_6KNXV0Dg" target="_
            f23d0DdDEe057q-fith2Cw" source="_g_D7sPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            MakeContribution" xmi:id="_6BejUDriEe00X_6KNXV0Dg" target="_
            X3JU8PqMEeKDCPJR83H7bQ" source="_g_D7sPqMEeKDCPJR83H7bQ"/>
        <contributions xmi:type="edu.toronto.cs.openome_model:
            MakeContribution" xmi:id="__7_Z8DriEe00X_6KNXV0Dg" target="_
            Y6lu8PqMEeKDCPJR83H7bQ" source="_iqSucPqMEeKDCPJR83H7bQ"/>
...
```

Listing B.19 shows the actions that are displayed in the console of MoRE-WS when it evolves the variability model. In line 2, MoRE-WS finds the variation point in which the tactic model will be merged. In line 4, MoRE-WS loads the tactic model for deception, which is expressed as a feature model. In line 7, MoRE-WS creates a mandatory feature between the LookForABook variation point and the root feature of the tactic model. In line 8, MoRE-WS activates the root feature of the deception tactic model. In lines 9-14, MoRE-WS merges into the variability model the relationships between the root feature of the tactic model and its subfeatures. These subfeatures are merged in the following lines. The subfeatures of the tactic model are activated in lines 17, 20, and 23.

Listing B.19: Output in the console of MoRE-WS when evolving the variability model.

```
1  Looking for the variation point in the variability model that needs to
        evolve...
2      + Found variation point: LookForABook
3
4  Loading the Deception model...
5
6      + Merged feature: Deception
7          - Created mandatory feature between the LookForABook VP and the
               root feature of the tactic model... FeatureModelPackage.impl
               .MandatoryImpl@17c9e523 (Name: tacticRel) (LowerBound: 0,
               UpperBound: 0)
8              - Feature activation: [FeatureModelPackage.impl.FeatureImpl@
                   9156782 (Name: Deception, featureNotation: With
                   Atributes, relationshipNotation: Both Graphic and
                   Cardinality, automaticConstraintResolution: true)]
9                  - From: FeatureModelPackage.impl.FeatureImpl@5c3e8c76 (
                       Name: Deception, featureNotation: With Atributes,
                       relationshipNotation: Both Graphic and Cardinality,
                       automaticConstraintResolution: true)
10                 - To: FeatureModelPackage.impl.FeatureImpl@e77c64a (Name:
                        LogIntrudersActivities, featureNotation: With
                       Atributes, relationshipNotation: Both Graphic and
                       Cardinality, automaticConstraintResolution: true)
11                 - From: FeatureModelPackage.impl.FeatureImpl@5c3e8c76 (
                       Name: Deception, featureNotation: With Atributes,
                       relationshipNotation: Both Graphic and Cardinality,
                       automaticConstraintResolution: true)
12                 - To: FeatureModelPackage.impl.FeatureImpl@597b2bb1 (Name
                       : ManageSensors, featureNotation: With Atributes,
                       relationshipNotation: Both Graphic and Cardinality,
                       automaticConstraintResolution: true)
13                 - From: FeatureModelPackage.impl.FeatureImpl@5c3e8c76 (
                       Name: Deception, featureNotation: With Atributes,
                       relationshipNotation: Both Graphic and Cardinality,
                       automaticConstraintResolution: true)
14                 - To: FeatureModelPackage.impl.FeatureImpl@25719f5c (Name
                       : SendEmailToSystemAdministrator, featureNotation:
                       With Atributes, relationshipNotation: Both Graphic
```

```
                  and Cardinality, automaticConstraintResolution: true
                  )

   + Merged feature: LogIntrudersActivities
       - Feature activation: [FeatureModelPackage.impl.FeatureImpl@3db
          0072d (Name: LogIntrudersActivities, featureNotation: With
          Atributes, relationshipNotation: Both Graphic and
          Cardinality, automaticConstraintResolution: true)]

   + Merged feature: ManageSensors
       - Feature activation: [FeatureModelPackage.impl.FeatureImpl@6fa73
          afe (Name: ManageSensors, featureNotation: With Atributes,
          relationshipNotation: Both Graphic and Cardinality,
          automaticConstraintResolution: true)]

   + Merged feature: SendEmailToSystemAdministrator
       - Feature activation: [FeatureModelPackage.impl.FeatureImpl@4c
          1232f6 (Name: SendEmailToSystemAdministrator,
          featureNotation: With Atributes, relationshipNotation: Both
          Graphic and Cardinality, automaticConstraintResolution: true
          )]
...
```

Listing B.20 shows the tactic model for deception expressed as a feature model in XMI format.

Listing B.20: Tactic model for deception expressed as a feature model in XMI format.

```
<?xml version="1.0" encoding="UTF-8"?>
<FeatureModelPackage:FeatureModel xmi:version="2.0" xmlns:xmi="http://www
    .omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:FeatureModelPackage="http://es.gvcase.featuremodelpackage"
    Name="DeceptionTacticModel">
   <Features Name="Deception">
       <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
          Mandatory" Name="Deception" LowerBound="1" UpperBound="1" To
          ="//@Features.1" From="//@Features.0"/>
       <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
          Mandatory" Name="Deception" LowerBound="1" UpperBound="1" To
          ="//@Features.2" From="//@Features.0"/>
       <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
          Mandatory" Name="Deception" LowerBound="1" UpperBound="1" To
          ="//@Features.3" From="//@Features.0"/>
   </Features>
   <Features Name="LogIntrudersActivities"/>
   <Features Name="ManageSensors"/>
   <Features Name="SendEmailToSystemAdministrator"/>
</FeatureModelPackage:FeatureModel>
```

Listing B.21 shows some key fragments of the evolved variability model in XMI format. Line 5 shows the mandatory relationship that was created between the `LookForABook` variation point and the root feature of the tactic model. Lines 8 to 24 shows the added relationships between the `Deception` feature and its subfeatures.

Listing B.21: Fragments of the evolved variability model in XMI format.

```
1  <Features Name="LookForABook">
2      <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Alternative" Name="LookForABook" LowerBound="1" UpperBound="1"/>
3      <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Optional" Name="LookForABook" UpperBound="1" To="//@Features.8"
           From="//@Features.6/@CardinalityBased_Relationships.0"/>
4      <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Optional" Name="LookForABook" UpperBound="1" To="//@Features.9"
           From="//@Features.6/@CardinalityBased_Relationships.0"/>
5      <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Mandatory" Name="tacticRel" To="//@Features.27" From="//
           @Features.6"/>
6  </Features>
7  ...
8  <Features Name="LogIntrudersActivities"/>
9  <Features Name="ManageSensors"/>
10 <Features Name="SendEmailToSystemAdministrator"/>
11 <Features Name="Deception">
12     <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Mandatory" Name="Deception">
13         <To href="DeceptionTacticModel.fm4spl#//@Features.1"/>
14         <From href="DeceptionTacticModel.fm4spl#//@Features.0"/>
15     </CardinalityBased_Relationships>
16     <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Mandatory" Name="Deception">
17         <To href="DeceptionTacticModel.fm4spl#//@Features.2"/>
18         <From href="DeceptionTacticModel.fm4spl#//@Features.0"/>
19     </CardinalityBased_Relationships>
20     <CardinalityBased_Relationships xsi:type="FeatureModelPackage:
           Mandatory" Name="Deception">
21         <To href="DeceptionTacticModel.fm4spl#//@Features.3"/>
22         <From href="DeceptionTacticModel.fm4spl#//@Features.0"/>
23     </CardinalityBased_Relationships>
24 </Features>
25 ...
```

Listing B.22 shows the output of the console of MoRE-WS with the actions that are carried out to evolve the running BPMN composition model. In line 1, MoRE-WS loads the tactic model for deception expressed as a BPMN composition model. In line 3, MoRE-WS creates a parallel relationship between the `Barnes and Noble Books` activity (which abstracts the problematic service) and the `Deception` tactic model. Lines 5-18 show the BPMN elements in the tactic model that are inserted into the running BPMN composition model.

Listing B.22: Output in the console of MoRE-WS when evolving the running composition model.

```
1  Loading the Deception BPMN composition model for merging...
2
3      - Creating... startEventForTactic, endEventForTactic, startGateway,
         endGateway, seqFromStartEventToStartGateway,
         seqFromStartGatewayToTacticSub, seqFromStartGatewayToAffectedSub
         , seqFromTacticSubToEndGateway, seqFromAffectedSubToEndGateway,
         seqFromEndGatewayToEndEvent
4
5      - Inserting... bpmn.impl.ActivityImpl@54776fc0 (iD: _ZnWVFnX3EeK__J2
         kDGu28A) (documentation: null, name: LogIntrudersActivities,
         ncname: null, orderedMessages: null, activityType: <unset>,
         looping: <unset>)
6      - Inserting... bpmn.impl.ActivityImpl@95e004c (iD: _d2aWhnX3EeK__J2
         kDGu28A) (documentation: null, name: ManageSensors, ncname: null
         , orderedMessages: null, activityType: <unset>, looping: <unset
         >)
7      - Inserting... bpmn.impl.ActivityImpl@1e52ed0e (iD: _7S1YRnX3EeK__J2
         kDGu28A) (documentation: null, name: start, ncname: null,
         orderedMessages: null, activityType: EventStartEmpty, looping: <
         unset>)
8      - Inserting... bpmn.impl.ActivityImpl@6377746a (iD: _AZ4KlnX4EeK__J2
         kDGu28A) (documentation: null, name: end, ncname: null,
         orderedMessages: null, activityType: EventEndEmpty, looping: <
         unset>)
9      - Inserting... bpmn.impl.ActivityImpl@318603db (iD: _LfoXBnX4EeK__J2
         kDGu28A) (documentation: , name: gateway1, ncname: null,
         orderedMessages: null, activityType: GatewayParallel, looping: <
         unset>)
10     - Inserting... bpmn.impl.ActivityImpl@6b9ee282 (iD: _i_6xhnX4EeK__J2
         kDGu28A) (documentation: null, name: gateway2, ncname: null,
         orderedMessages: null, activityType: GatewayParallel, looping: <
         unset>)
11     - Inserting... bpmn.impl.SequenceEdgeImpl@56b8abe3 (iD: _mqkeUnX4EeK
         __J2kDGu28A) (documentation: null, name: seq1, ncname: null,
         conditionType: <unset>, isDefault: <unset>)
12     - Inserting... bpmn.impl.SequenceEdgeImpl@65f578dd (iD: _vH53CHX4EeK
         __J2kDGu28A) (documentation: null, name: seq2, ncname: ,
         conditionType: <unset>, isDefault: <unset>)
```

```
    - Inserting... bpmn.impl.SequenceEdgeImpl@7fa02e81 (iD: _5Jq26HX4EeK
        __J2kDGu28A) (documentation: null, name: seq3, ncname: null,
        conditionType: <unset>, isDefault: <unset>)
    - Inserting... bpmn.impl.SequenceEdgeImpl@70f4a1d3 (iD: _BLYDiHX5EeK
        __J2kDGu28A) (documentation: null, name: seq4, ncname: ,
        conditionType: <unset>, isDefault: <unset>)
    - Inserting... bpmn.impl.SequenceEdgeImpl@4833c6df (iD: _GYbXOHX5EeK
        __J2kDGu28A) (documentation: null, name: seq5, ncname: null,
        conditionType: <unset>, isDefault: <unset>)
    - Inserting... bpmn.impl.SequenceEdgeImpl@6bb60e9a (iD: _Ldd-YnX5EeK
        __J2kDGu28A) (documentation: null, name: seq6, ncname: null,
        conditionType: <unset>, isDefault: <unset>)
    - Inserting... bpmn.impl.SequenceEdgeImpl@899aac7 (iD: _Tw9emHX5EeK__
        J2kDGu28A) (documentation: null, name: seq7, ncname: null,
        conditionType: <unset>, isDefault: <unset>)
    - Inserting... bpmn.impl.ActivityImpl@1cd6758f (iD: _fgU6qXX3EeK__J2
        kDGu28A) (documentation: null, name:
        SendEmailToSystemAdministrator, ncname: null, orderedMessages:
        null, activityType: <unset>, looping: <unset>)
...
```

Listing B.23 shows the tactic model for deception expressed as a composition model in XMI format.

Listing B.23: Tactic model for deception expressed as a composition model in XMI format.

```
<?xml version="1.0" encoding="UTF-8"?>
    <bpmn:BpmnDiagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI
        " xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
        bpmn="http://stp.eclipse.org/bpmn" iD="_i8Az0HX2EeK__J2kDGu28A"
        name="deceptionTactic">
        <pools iD="_w6WW83X2EeK__J2kDGu28A" name="DeceptionTacticPool">
            <vertices xsi:type="bpmn:SubProcess" iD="_BplUZXX3EeK__J2kDGu
                28A" name="Deception">
            <vertices xsi:type="bpmn:Activity" iD="_ZnWVFnX3EeK__J2kDGu28
                A" outgoingEdges="_BLYDiHX5EeK__J2kDGu28A" incomingEdges
                ="_vH53CHX4EeK__J2kDGu28A" name="LogIntrudersActivities
                "/>
            <vertices xsi:type="bpmn:Activity" iD="_d2aWhnX3EeK__J2kDGu28
                A" outgoingEdges="_GYbXOHX5EeK__J2kDGu28A" incomingEdges
                ="_5Jq26HX4EeK__J2kDGu28A" name="ManageSensors"/>
            <vertices xsi:type="bpmn:Activity" iD="_7S1YRnX3EeK__J2kDGu28
                A" outgoingEdges="_mqkeUnX4EeK__J2kDGu28A" name="start"
                activityType="EventStartEmpty"/>
            <vertices xsi:type="bpmn:Activity" iD="_AZ4KlnX4EeK__J2kDGu28
                A" incomingEdges="_Tw9emHX5EeK__J2kDGu28A" name="end"
                activityType="EventEndEmpty"/>
            <vertices xsi:type="bpmn:Activity" iD="_LfoXBnX4EeK__J2kDGu28
                A" outgoingEdges="_vH53CHX4EeK__J2kDGu28A _5Jq26HX4EeK__
                J2kDGu28A" incomingEdges="_mqkeUnX4EeK__J2kDGu28A"
                documentation="" name="gateway1" activityType="
                GatewayParallel"/>
            <vertices xsi:type="bpmn:Activity" iD="_i_6xhnX4EeK__J2kDGu28
                A" outgoingEdges="_Ldd-YnX5EeK__J2kDGu28A" incomingEdges
                ="_BLYDiHX5EeK__J2kDGu28A _GYbXOHX5EeK__J2kDGu28A" name
                ="gateway2" activityType="GatewayParallel"/>
                    <sequenceEdges iD="_mqkeUnX4EeK__J2kDGu28A" name="
                seq1"/>
            <sequenceEdges iD="_vH53CHX4EeK__J2kDGu28A" name="seq2"
                ncname=""/>
            <sequenceEdges iD="_5Jq26HX4EeK__J2kDGu28A" name="seq3"/>
            <sequenceEdges iD="_BLYDiHX5EeK__J2kDGu28A" name="seq4"
                ncname=""/>
            <sequenceEdges iD="_GYbXOHX5EeK__J2kDGu28A" name="seq5"/>
            <sequenceEdges iD="_Ldd-YnX5EeK__J2kDGu28A" name="seq6"/>
            <sequenceEdges iD="_Tw9emHX5EeK__J2kDGu28A" name="seq7"/>
            <eventHandlers iD="_fgU6qXX3EeK__J2kDGu28A" outgoingEdges="_
                Tw9emHX5EeK__J2kDGu28A" incomingEdges="_Ldd-YnX5EeK__J2
                kDGu28A" name="SendEmailToSystemAdministrator"/>
        </vertices>
    </pools>
</bpmn:BpmnDiagram>
```

Afterwards, the console of MoRE-WS shows the path of the evolved WS-BPEL composition schema. This composition schema is deployed in a deployment directory with an increasing version number. Listing B.24 shows the WS-BPEL fragment to invoke the service operation that implements the deception tactic. MoRE-WS merges this fragment into the evolved WS-BPEL composition schema.

Listing B.24: WS-BPEL fragment to invoke the deception tactic.

```
<bpel:sequence name="SequenceWithTactic">
    <bpel:assign validate="no" name="AssignDeceptionTactic">
        <bpel:copy>
            <bpel:from>
                <bpel:literal>
                    <tns:deceit xmlns:tns="http://www.example.org/
                        deceptiontactic/" xmlns:xsi="http://www.w3.org
                        /2001/XMLSchema-instance">
                        <deceitData>
                            <in>in</in>
                        </deceitData>
                    </tns:deceit>
                </bpel:literal>
            </bpel:from>
            <bpel:to variable="DeceptionTacticRequest" part="parameters
                "></bpel:to>
        </bpel:copy>
        <bpel:copy>
            <bpel:from part="parameters" variable="
                OnlineBookShoppingRequest">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                    sublang:xpath1.0">
                    <![CDATA[inTactics]]>
                </bpel:query>
            </bpel:from>
            <bpel:to part="parameters" variable="DeceptionTacticRequest">
                <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:
                    sublang:xpath1.0">
                    <![CDATA[deceitData]]>
                </bpel:query>
            </bpel:to>
        </bpel:copy>
    </bpel:assign>
    <bpel:invoke name="DeceptionTactic" partnerLink="DeceptionTacticPL"
        operation="deceit"  portType="ns17:deceptiontactic"
        inputVariable="DeceptionTacticRequest" outputVariable="
        DeceptionTacticResponse">
    </bpel:invoke>
</bpel:sequence>
```

Figure B.8 shows a fragment of the evolved WS-BPEL composition schema.

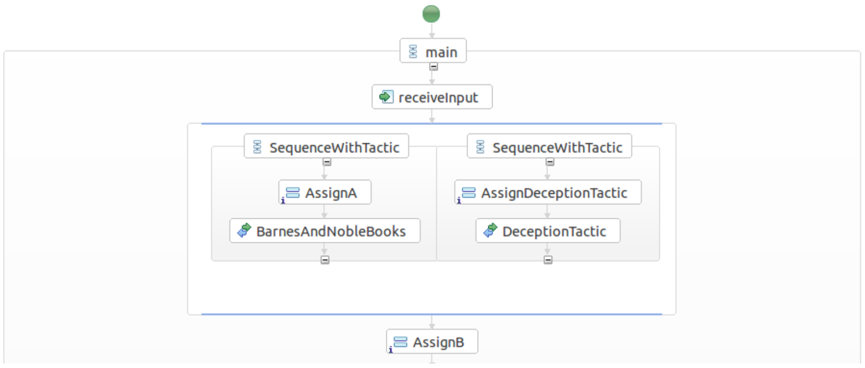


Figure B.8: A fragment of the evolved WS-BPEL composition schema.

Figure B.9 shows the deployment directory with an increased version number. MoRE-WS imports the WSDL file that is used to invoke the service operation for the deception tactic.

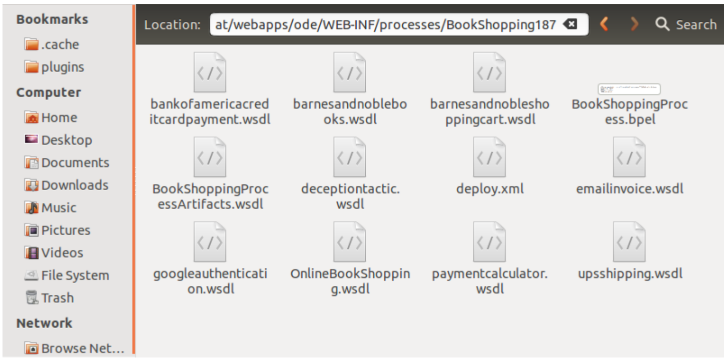


Figure B.9: Evolved deployment directory.

Finally, Listing B.25 shows a fragment of the output in the console of APACHE AXIS2 when the evolved WS-BPEL composition schema is invoked. Now, the service operation that implements the deception tactic is invoked in parallel every time the `Barnes and Noble Books` service operation is invoked in order to face the security attack (see line 2).

Listing B.25: Output in the console of APACHE AXIS2 when invoking the evolved WS-BPEL composition schema.

```
1  =========================================================
2  Deception tactic started...
3  =========================================================
4  <<Barnes and Noble Books Service>>: Web services:
5  Authors: Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju.
       Publisher: Springer; 2004 edition (September 15, 2009). Related
       titles:
6  - 1. Web Services Essentials.
7  - 2. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL
       and RESTful Web Services.
8  - 3. Web Services, Service-Oriented Architectures, and Cloud Computing:
       The Savvy Manager's Guide.
9  =========================================================
10 ...
```

BIBLIOGRAPHY

Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S., Tosetti, E., 2007. Developing eBusiness solutions with a model driven approach: the case of acer EMEA. In: Proceedings of the 7th International Conference on Web engineering. ICWE '07. Springer-Verlag, Berlin, Heidelberg, pp. 539–544.

Akkawi, F., Akkawi, K., Bader, A., Ayyash, M., Fletcher, D., Alzoubi, K., March 2007. Software adaptation: A conscious design for oblivious programmers. In: Proceedings of the IEEE Aerospace Conference. pp. 1–12.

Alférez, E. M., 2012. Derivation and consistency checking of models in early software product line engineering. Ph.D. thesis, Universidade Nova de Lisboa.
URL http://hdl.handle.net/10362/9370

Alférez, G. H., Pelechano, V., 2011a. Context-aware autonomous web services in software product lines. In: Proceedings of the 2011 15th International Software Product Line Conference. SPLC '11. IEEE Computer Society, Washington, DC, USA, pp. 100–109.

Alférez, G. H., Pelechano, V., 2011b. Systematic reuse of web services through software product line engineering. In: 2011 9th IEEE European Conference on Web Services. ECOWS '11. pp. 192–199.

Alférez, G. H., Pelechano, V., 2012a. Dynamic evolution of context-aware systems with models at runtime. In: France, R., Kazmeier, J., Breu, R., Atkinson, C. (Eds.), Model Driven Engineering Languages and Systems. Vol. 7590 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 70–86.

Alférez, G. H., Pelechano, V., November 2012b. Towards dynamic service compositions with models at runtime. Tech. Rep. ProS-TR-2012-05, Research Center on Software Production Methods.
URL http://pros.webs.upv.es/technicalreports/PROS-TR-2012-05.pdf

Alférez, G. H., Pelechano, V., 2013a. Achieving autonomic web service compositions: a model-driven approach.
URL http://www.harveyalferez.com/thesis

Alférez, G. H., Pelechano, V., 2013b. Facing uncertainty in web service compositions. In: Proceedings of the 20th IEEE International Conference on Web Services. ICWS '13. pp. 219–226.

Alférez, G. H., Pelechano, V., Mazo, R., Salinesi, C., Diaz, D., 2013. Dynamic adaptation of service compositions with variability models. Journal of Systems and Software.
URL http://www.sciencedirect.com/science/article/pii/S0164121213001465

Alférez, M., Lopez-Herrejon, R. E., Moreira, A., Amaral, V., Egyed, A., 2011. Supporting consistency checking between features and software product line use scenarios. In: Proceedings of the 12th International Conference on Top Productivity through Software Reuse. ICSR '11. Springer-Verlag, Berlin, Heidelberg, pp. 20–35.

Alonso, G., Casati, F., Kuno, H., Machiraju, V., 2004. Web Services: Concepts, Architectures and Applications, 1st Edition. Springer-Verlag Berlin Heidelberg.

Ameller, D., Franch, X., 2008. Service level agreement monitor (SALMon). In: Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008). IEEE Computer Society, Washington, DC, USA, pp. 224–227.

Amoui, M., Derakhshanmanesh, M., Ebert, J., Tahvildari, L., 2012. Achieving dynamic adaptation via management and interpretation of runtime models. Journal of Systems and Software 85 (12), 2720 – 2737.

Amoui Kalareh, M., 2012. Evolving software systems for self-adaptation. Ph.D. thesis, University of Waterloo.
URL http://hdl.handle.net/10012/6643

Andersson, J., Baresi, L., Bencomo, N., Lemos, R., Gorla, A., Inverardi, P., Vogel, T., 2013. Software engineering processes for self-adaptive systems. In: Lemos, R., Giese, H., Müller, H., Shaw, M. (Eds.), Software Engineering for Self-Adaptive Systems II. Vol. 7475 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 51–75.

Andersson, J., Lemos, R., Malek, S., Weyns, D., 2009. Software engineering for self-adaptive systems. Springer-Verlag, Berlin, Heidelberg, Ch. Modeling Dimensions of Self-Adaptive Software Systems, pp. 27–47.

Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P., Nov. 2007. PAWS: a framework for executing adaptive web-service processes. IEEE Softw. 24 (6), 39–46.

Ardagna, D., Pernici, B., 2007. Adaptive service composition in flexible processes. Software Engineering, IEEE Transactions on 33 (6), 369–384.

Asikainen, T., Mannisto, T., Soininen, T., 2006. A unified conceptual foundation for feature modelling. In: Proceedings of the 10th International on Software Product Line Conference. SPLC '06. IEEE Computer Society, Washington, DC, USA, pp. 31–40.

Aßmann, U., Bencomo, N., Cheng, B. H. C., France, R. B., 2011. Models@run.time (dagstuhl seminar 11481). Dagstuhl Reports 1 (11), 91–123.
URL http://drops.dagstuhl.de/opus/volltexte/2012/3379/

Ayora, C., Alférez, G. H., Torres, V., Pelechano, V., 2011. Procesos de negocio auto-adaptables al contexto. In: Álvarez, P., Del Arco, J., Luaces, M. (Eds.), Actas de las VII Jornadas de Ciencia e Ingeniería de Servicios. JCIS '11. Universidade da Coruña, pp. 147–160.
URL http://www.sistedes.es/ficheros/actas-conferencias/JCIS/2011.pdf

Ayora, C., Alférez, G. H., Torres, V., Pelechano, V., 2012. Applying CVL to business process variability management. In: Proceedings of VARiability for You. MODELS 2012. pp. 24–29.
URL http://vary2012.irisa.fr/VARY2012Proceedings.pdf

Bae, J., Kang, S., oct. 2007. A method to generate a feature model from a business process model for business applications. In: Computer and Information Technology. pp. 879–884.

Bandara, K. Y., Wang, M., Pahl, C., 2009. Context modeling and constraints binding in web service business processes. In: Proceedings of the 1st International Workshop on Context-Aware Software Technology and Applications. CASTA '09. ACM, New York, NY, USA,

pp. 29–32.
URL http://doras.dcu.ie/15960

Baresi, L., Di Nitto, E., Ghezzi, C., October 2006. Toward open-world software: Issue and challenges. Computer 39, 36–43.

Baresi, L., Guinea, S., March 2011. Self-supervising BPEL processes. IEEE Trans. Softw. Eng. 37, 247–263.

Baresi, L., Nitto, E., Ghezzi, C., Guinea, S., 2007. A framework for the deployment of adaptable web service compositions. Service Oriented Computing and Applications 1 (1), 75–91.

Basili, V. R., 1992. Software modeling and measurement: the goal/question/metric paradigm. Tech. rep., College Park, MD, USA.
URL http://www.cs.umd.edu/~basili/publications/technical/T78.pdf

Basili, V. R., Caldiera, G., Dieter, R. H., 1994. Goal question metric paradigm. In: Encyclopedia of Software Engineering. Vol. 2. John Wiley & Sons, Inc., pp. 528–532.
URL http://www.cs.umd.edu/~basili/publications/technical/T89.pdf

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, 3rd Edition. Addison-Wesley Professional.

Batory, D., 2005. Feature models, grammars, and propositional formulas. In: Software Product Lines Conference. Vol. 3714 of Lecture Notes in Computer Sciences. Springer–Verlag, p. 7–20.

Bayer, J., Gerard, S., Haugen, Ø., Mansell, J., MØller-Pedersen, B., Oldevik, J., Tessier, P., Thibault, J.-P., Widen, T., 2006. Consolidated product line variability modeling. In: Käkölä, T., Duenas, J. (Eds.), Software Product Lines. Springer Berlin Heidelberg, pp. 195–241.

Belady, L. A., Lehman, M. M., Sep. 1976. A model of large program development. IBM Syst. J. 15 (3), 225–252.

Benavides, D., Segura, S., Ruiz-Cortés, A., Sep. 2010. Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35 (6), 615–636.

Benavides, D., Trinidad, P., Ruiz-Cortés, A., 2005. Automated reasoning on feature models. In: Proceedings of the 17th international conference on Advanced Information Systems Engineering. CAiSE'05. Springer-Verlag, Berlin, Heidelberg, pp. 491–503.

Bencomo, N., Grace, P., Flores, C., Hughes, D., Blair, G., 2008. Genie: supporting the model driven development of reflective, component-based adaptive systems. In: Proceedings of the 30th International Conference on Software Engineering. ICSE '08. ACM, New York, NY, USA, pp. 811–814.

Bencomo, N., Hallsteinsen, S., Santana de Almeida, E., October 2012. A view of the dynamic software product line landscape. Computer 45 (10), 36 –41.

Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R., 2006. Heuristics for QoS-aware web service composition. In: Proceedings of the 2006 IEEE International Conference on Web Services. ICWS '06. IEEE Computer Society, Washington, DC, USA, pp. 72–82.

Berg, H. C., 2003. The rotary motor of bacterial flagella. Annual Review of Biochemistry 72 (1), 19–54.

Bergey, J., Fisher, M., Gallagher, B., Jones, L., Northrop, L., February 2000. Basic concepts of product line practice for the DoD. Tech. Rep. CMU/SEI-2000-TN-001, Software Engineering Institute, Carnegie Mellon University.
URL http://www.sei.cmu.edu/reports/00tn001.pdf

Bhola, S., Astley, M., Saccone, R., Ward, M., 2006. Utility-aware resource allocation in an event processing system. In: Proceedings of the 2006 IEEE International Conference on Autonomic Computing. IEEE Computer Society, Washington, DC, USA, pp. 55–64.

Biermann, E., Ermel, C., 2009. Transforming BPMN to BPEL with EMF Tiger. In: 5th International Workshop on Graph-Based Tools. Grabats 2009.
URL http://is.tm.tue.nl/staff/pvgorp/events/grabats2009/submissions/grabats2009_submission_25.pdf

BigLever Software, Inc., n.d. Basic software product line concepts.
URL http://www.softwareproductlines.com/introduction/concepts.html

Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., Mills, W. N., Diao, Y., Jul. 2002. ABLE: a toolkit for building multiagent autonomic systems. IBM Syst. J. 41 (3), 350–371.

Blair, G., Bencomo, N., France, R. B., October 2009. Models@ run.time. Computer 42, 22–27.

Bosch, J., 2000. Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Bosch, J., Capilla, R., October 2012. Dynamic variability in software-intensive embedded system families. Computer 45 (10), 28 –35.

Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, J. H., Pohl, K., 2002. Variability issues in software product lines. In: Revised Papers from the 4th International Workshop on Software Product-Family Engineering. PFE '01. Springer-Verlag, London, UK, UK, pp. 13–21.

Bosloper, I., Siljee, J., Nijhuis, J., Hammer, D., 2005. Creating self-adaptive service systems with DySOA. In: Proceedings of the 3rd European Conference on Web Services. ECOWS '05. IEEE Computer Society, Washington, DC, USA, pp. 95–104.

Bougaev, A., 2005. Pattern recognition based tools enabling autonomic computing. In: Proceedings of the 2nd IEEE International Conference on Autonomic Computing. ICAC 2005. pp. 313–314.

Brambilla, M., Cabot, J., Wimmer, M., September 2012. Model-Driven Software Engineering in Practice. Vol. 1 of Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers.

Brownsword, L., Clements, P. C., Olsson, U., 1996. Successful product line engineering: A case study. In: Software Technology Conference. URL http://130.203.133.150/viewdoc/download?doi=10.1.1.54.749&rep=rep1&type=pdf

Buckley, J., Mens, T., Zenger, M., Rashid, A., Kniesel, G., 2005. Towards a taxonomy of software change. Journal of Software Maintenance 17 (5), 309–332.

Byun, H. E., Cheverst, K., 2004. Utilizing context history to provide dynamic adaptations. Applied Artificial Intelligence 18 (6), 533–548.

Calinescu, R., 2009. General-purpose autonomic computing. In: Zhang, Y., Yang, L. T., Denko, M. K. (Eds.), Autonomic Computing and Networking. Springer US, pp. 3–30.

Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R., Sep. 2012. Self-adaptive software needs quantitative verification at runtime. Commun. ACM 55 (9), 69–77.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G., 2011. Dynamic QoS management and optimization in service-based systems. IEEE Transactions on Software Engineering 37, 387–409.

Canfora, G., Penta, M. D., Esposito, R., Villani, M. L., 2008. A framework for QoS-aware binding and re-binding of composite web services. Journal of Systems and Software 81 (10), 1754 – 1769.

Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Lo Presti, F., Mirandola, R., 2012. MOSES: A framework for QoS driven runtime adaptation of service-oriented systems. Software Engineering, IEEE Transactions on 38 (5), 1138–1159.

Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., 2010. Adaptive management of composite services under percentile-based service level agreements. In: Maglio, P., Weske, M., Yang, J., Fantinato, M. (Eds.), Service-Oriented Computing. Vol. 6470 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 381–395.

Cetina, C., 2010. Achieving autonomic computing through the use of variability models at run-time. Ph.D. thesis, Universitat Politécnica de Valéncia.
URL http://riunet.upv.es/handle/10251/7484

Cetina, C., Giner, P., Fons, J., Pelechano, V., October 2009. Autonomic computing through reuse of variability models at runtime: The case of smart homes. Computer 42, 37–43.

Chakraborty, D., Perich, F., Joshi, A., Finin, T. W., Yesha, Y., 2002. A reactive service composition architecture for pervasive computing environments. In: Proceedings of the IFIP TC6/WG6.8 Working Conference on Personal Wireless Communications. PWC '02. Kluwer, B.V., Deventer, The Netherlands, pp. 53–62.

Charfi, A., Mezini, M., Sep. 2007. AO4BPEL: An aspect-oriented extension to BPEL. World Wide Web 10 (3), 309–344.

Chen, F., Li, S., Yang, H., Wang, C.-H., Chu, W.-C., 2005. Feature analysis for service-oriented reengineering. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference. APSEC '05. pp. 201–208.

Cheng, B. H., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J., 2009a. Software engineering for self-adaptive systems. Springer-Verlag, Berlin, Heidelberg, Ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

Cheng, B. H., Sawyer, P., Bencomo, N., Whittle, J., 2009b. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. MODELS '09. Springer-Verlag, Berlin, Heidelberg, pp. 468–483.

Cheng, S.-W., Garlan, D., November 2007. Handling uncertainty in autonomic systems. In: Proceedings of the 2007 International Workshop on Living with Uncertainties. IWLU '07.
URL http://se.cs.toronto.edu/IWLU/papers/Autonomic_Cheng.pdf

Clements, P., Northrop, L., 2001. Software product lines: practices and patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Colombo, M., Di Nitto, E., Mauri, M., 2006. SCENE: A service composition execution environment supporting dynamic changes disciplined through rules. In: Dan, A., Lamersdorf, W. (Eds.), Service-Oriented Computing – ICSOC 2006. Vol. 4294 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 191–202.

Cotroneo, D., Gargiulo, M., Russo, S., Ventre, G., 2002. Improving the availability of web services. In: 22nd International Conference on Software Engineering. ICSE 2002. pp. 59–63.

Curtis, B., Kellner, M. I., Over, J., Sep. 1992. Process modeling. Commun. ACM 35 (9), 75–90.

Czarnecki, K., Antkiewicz, M., 2005. Mapping features to models: a template approach based on superimposed variants. In: Proceedings of the 4th international conference on Generative Programming and Component Engineering. GPCE'05. Springer-Verlag, Berlin, Heidelberg, pp. 422–437.

Czarnecki, K., Eisenecker, U. W., 2000. Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Czarnecki, K., Helsen, S., Eisenecker, U., 2004. Staged configuration using feature models. In: Nord, R. (Ed.), Software Product Lines. Vol. 3154 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 266–283.

De La Fuente, J., 2012. Herramienta para la generación y despliegue de composiciones de servicios web mediante modelos BPMN. Master's thesis, Universitat Politécnica de Valéncia.
URL http://riunet.upv.es/handle/10251/18022

Dean, M., Schreiber, G., February 2004. OWL web ontology language reference.
URL http://www.w3.org/TR/owl-ref

Deelstra, S., Sinnema, M., Bosch, J., January 2005. Product derivation in software product families: a case study. J. Syst. Softw. 74, 173–194.

Del Fabro, M. D., Bézivin, J., Valduriez, P., 2006. Weaving models with the Eclipse AMW plugin. In: Eclipse Modeling Symposium, Eclipse Summit Europe.
URL http://www.eclipsecon.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium2_WeavingModels.pdf

Dey, A. K., January 2001. Understanding and using context. Personal Ubiquitous Comput. 5, 4–7.

Differding, C., Hoisl, B., Lott, C. M., 1996. Technology package for the goal question metric paradigm. Tech. Rep. 281/96, Universität Kaiserslautern.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.5631&rep=rep1&type=pdf

Dinkelaker, T., Mitschke, R., Fetzer, K., Mezini, M., March 2010. A dynamic software product line approach using aspect models at runtime. In: Proceedings of the 1st Workshop on Composition and Variability.
URL http://www.stg.tu-darmstadt.de/media/st/publications/a_dynamic_software_product_line_approach_using_aspect_models_at_runtime.pdf

dos Santos Rocha, R., Fantinato, M., 2013. The use of software product lines for business process management: A systematic literature review. Information and Software Technology 55 (8), 1355 – 1373.

Doux, G., Jouault, F., Bézivin, J., 2009. Transforming BPMN process models to BPEL process definitions with ATL. In: 5th International Workshop on Graph-Based Tools. Grabats '09.
URL http://is.ieis.tue.nl/staff/pvgorp/events/grabats2009/submissions/grabats2009_submission_19-final.pdf

Duan, Y., 2012. A survey on service contract. In: Proceedings of the 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. SNPD '12. IEEE Computer Society, Washington, DC, USA, pp. 805–810.

Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G., 2009. Model evolution by run-time parameter adaptation. In: Proceedings of the 31st International Conference on Software Engineering. ICSE '09. IEEE Computer Society, Washington, DC, USA, pp. 111–121.

Epstein, R. A., 1984. In defense of the contract at will. The University of Chicago Law Review 51 (4), 947–982.
URL http://www.jstor.org/discover/10.2307/1599554?uid=3737952&uid=2134&uid=2&uid=70&uid=4&sid=21102568447597

Esfahani, N., Kouroshfar, E., Malek, S., 2011. Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. ESEC/FSE '11. ACM, New York, NY, USA, pp. 234–244.

Ezenwoye, O., Sadjadi, S., 2007. RobustBPEL2: transparent autonomization in business processes through dynamic proxies. In: Proceedings of the 8th International Symposium on Autonomous Decentralized Systems. ISADS '07. pp. 17–24.

Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Northrop, L., Schmidt, D., Sullivan, K., Wallnau, K., 2006. Ultra-large-scale systems: The software challenge of the future. Tech. Rep. 15213-3890, Software Engineering Institute. URL http://www.sei.cmu.edu/library/assets/ULS_Book20062.pdf

Fenson, E., Howard, R., 2004. Reinforcement learning for autonomic network repair. In: Proceedings of the 1st International Conference on Autonomic Computing. IEEE Computer Society, Washington, DC, USA, pp. 284–285.

Fleurey, F., Solberg, A., 2009. A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. MODELS '09. Springer-Verlag, Berlin, Heidelberg, pp. 606–621.

Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E., 2006. Using architecture models for runtime adaptability. IEEE Softw. 23 (2), 62–70.

Forgy, C. L., 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19 (1), 17 – 37.

Frakes, W. B., Isoda, S., September 1994. Success factors of systematic reuse. IEEE Softw. 11, 14–19.

Frakes, W. B., Kang, K., July 2005. Software reuse research: Status and future. IEEE Trans. Softw. Eng. 31, 529–536.

France, R., Rumpe, B., 2007. Model-driven development of complex software: A research roadmap. In: 2007 Future of Software Engineering. FOSE '07. IEEE Computer Society, Washington, DC, USA, pp. 37–54.

Franch, X., Grunbacher, P., Oriol, M., Burgstaller, B., Dhungana, D., Lopez, L., Marco, J., Pimentel, J., 2011. Goal-driven adaptation of service-based systems from runtime monitoring data. In: Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops. COMPSACW '11. IEEE Computer Society, Washington, DC, USA, pp. 458–463.

Ganek, A. G., Corbi, T. A., January 2003. The dawning of the autonomic computing era. IBM Syst. J. 42, 5–18.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P., October 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer 37 (10), 46–54.

Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjorven, E., Hallsteinsen, S., Horn, G., Khan, M. U., Mamelli, A., Papadopoulos, G. A., Paspallis, N., Reichle, R., Stav, E., March 2009. A comprehensive solution for application-level adaptation. Softw. Pract. Exper. 39 (4), 385–422.

Geyer, L., Becker, M., 2002. On the influence of variabilities on the application-engineering process of a product family. In: Proceedings of the 2nd International Conference on Software Product Lines. SPLC '02. Springer-Verlag, London, UK, pp. 1–14.

Goldsby, H. J., Cheng, B. H., 2008. Automatically generating behavioral models of adaptive systems to address uncertainty. In: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems. MODELS '08. Springer-Verlag, Berlin, Heidelberg, pp. 568–583.

Gottschalk, F., van der Aalst, W. M. P., Jansen-Vullers, M. H., Rosa, M. L., 2008. Configurable workflow models. Int. J. Cooperative Inf. Syst. 17 (2), 177–221.

Greenfield, J., Short, K., 2003. Software factories: assembling applications with patterns, models, frameworks and tools. In: Companion of the 18th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. OOPSLA '03. ACM, New York, NY, USA, pp. 16–27.

Gröner, G., Wende, C., Bošković, M., Parreiras, F. S., Walter, T., Heidenreich, F., Gašević, D., Staab, S., 2011. Validation of families of business processes. In: Proceedings of the 23rd International Conference on Advanced Information Systems Engineering. CAiSE'11. Springer-Verlag, Berlin, Heidelberg, pp. 551–565.

Grunske, L., 2008. Specification patterns for probabilistic quality properties. In: Proceedings of the 30th International Conference on Software Engineering. ICSE '08. ACM, New York, NY, USA, pp. 31–40.

Hadaytullah, H., Koskimies, K., Systa, T., 2009. Using model customization for variability management in service compositions. In:

Proceedings of the 2009 IEEE International Conference on Web Services. ICWS '09. pp. 687 –694.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., Papadopoulos, G., 2012. A development framework and methodology for self-adapting applications in ubiquitous computing environments. Journal of Systems and Software 85 (12), 2840 – 2859.

Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K., 2008. Dynamic software product lines. Computer 41, 93–95.

Hallsteinsen, S., Stav, E., Solberg, A., Floch, J., 2006. Using product line techniques to build adaptive systems. In: Proceedings of the 10th International Software Product Line Conference. SPLC '06. IEEE Computer Society, Washington, DC, USA, pp. 141–150.

Haugen, O., Møller-Pedersen, B., Oldevik, J., Olsen, G. K., Svendsen, A., 2008. Adding standardized variability to domain specific languages. In: Proceedings of the 12th International Software Product Line Conference. SPLC '08. IEEE Computer Society, Washington, DC, USA, pp. 139–148.

Heisenberg, W., 1927. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. Zeitschrift für Physik 43 (3-4), 172–198.

Hong, J.-Y., Suh, E.-H., Kim, S.-J., May 2009. Context-aware systems: A literature review and classification. Expert Syst. Appl. 36, 8509–8522.

Horkoff, J., Yu, E., 2011. Analyzing goal models: different approaches and how to choose among them. In: Proceedings of the 2011 ACM Symposium on Applied Computing. SAC '11. ACM, New York, NY, USA, pp. 675–682.

Horn, P., 2001. Autonomic computing: IBM's perspective on the state of information technology.
URL http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

Hummer, W., Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S., 2011. VRESCo - Vienna Runtime Environment for Service-oriented Computing. In: Dustdar, S., Li, F. (Eds.), Service Engineering: European Research Results. Springer, Ch. 11, pp. 299–324.

Hunt, G. R., January 1996. Manufacture and use of hook-tools by New Caledonian crows. Nature 379 (6562), 249–251.

IBM, 2005. Business processes.
URL http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.bpel.ui.doc/concepts/cunder.html

IBM, 2006. An architectural blueprint for autonomic computing. Tech. rep., IBM.
URL http://www.eecs.harvard.edu/~chaki/bib/papers/autonomic.pdf

IEEE, 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, 1–84.

Istoan, P., Nain, G., Perrouin, G., Jezequel, J.-M., 2009. Dynamic software product lines for service-based systems. In: Proceedings of the 2009 9th IEEE International Conference on Computer and Information Technology. Vol. 2 of CIT '09. IEEE Computer Society, Washington, DC, USA, pp. 193–198.

Jurič, M., Mathew, B., Sarang, P., 2004. Business Process Execution Language For Web Services. From technologies to solutions. Packt Publishing, Limited.

Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S., November 1990. Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University.
URL http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm

Karastoyanova, D., Leymann, F., 2009. BPEL'n'Aspects: Adapting service orchestration logic. In: Proceedings of the 2009 IEEE International Conference on Web Services. ICWS '09. pp. 222–229.

Keeney, J., 2004. Completely unanticipated dynamic adaptation of software. Ph.D. thesis, Trinity College Dublin.
URL http://www.tara.tcd.ie/bitstream/2262/30726/1/TCD-CS-2005-43.pdf

Kent, S., 2002. Model driven engineering. In: Proceedings of the 3rd International Conference on Integrated Formal Methods. IFM '02. Springer-Verlag, London, UK, pp. 286–298.

Kephart, J. O., Chess, D. M., January 2003. The vision of autonomic computing. Computer 36, 41–50.

Khadka, R., Sapkota, B., 2010. An evaluation of dynamic web service composition approaches. In: van, M. S., Sapkota, B. (Eds.), 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing. ACT4SOC 2010. SciTePress, pp. 67–79. URL http://doc.utwente.nl/72798/

Khan, M., Reichle, R., Geihs, K., July 2008. Architectural constraints in the model-driven development of self-adaptive applications. Distributed Systems Online, IEEE 9 (7), 1.

Kitchenham, B., July 2004. Procedures for performing systematic reviews. Tech. Rep. TR/SE-0401 and 0400011T.1, Keele University and NICTA.

Klusch, M., Kapahnke, P., 2010. iSeM: Approximated reasoning for adaptive hybrid selection of semantic services. In: Proceedings of the 2010 IEEE 4th International Conference on Semantic Computing. ICSC '10. IEEE Computer Society, Washington, DC, USA, pp. 184–191.

Kon, F., Costa, F., Blair, G., Campbell, R. H., Jun. 2002. The case for reflective middleware. Commun. ACM 45 (6), 33–38.

Koning, M., Sun, C.-a., Sinnema, M., Avgeriou, P., February 2009. VxBPEL: Supporting variability for web services in BPEL. Inf. Softw. Technol. 51, 258–269.

Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., Malm, E.-J., July 2003. Managing context information in mobile devices. IEEE Pervasive Computing 2 (3), 42–51.

Kotonya, G., Sommerville, I., 1998. Requirements Engineering: Processes and Techniques. John Wiley & Sons Ltd., Chichester, England.

Kramer, J., Magee, J., November 1990. The evolving philosophers problem: Dynamic change management. IEEE Trans. Softw. Eng. 16 (11), 1293–1306.

Kumar, A., Yao, W., 2012. Design and management of flexible process variants using templates and rules. Computers in Industry 63 (2), 112 – 130.

Kwiatkowska, M., Norman, G., Parker, D., August 2004. Probabilistic symbolic model checking with prism: a hybrid approach. Int. J. Softw. Tools Technol. Transf. 6 (2), 128–142.

Kwon, O. B., 2004. Modeling and generating context-aware agent-based applications with amended colored petri nets. Expert Systems with Applications 27 (4), 609 – 621.

Labhart, J., Rowe, M., Matney, S., Carrow, S., May 1990. Forward chaining parallel inference. In: Proceedings of the IEEE 1990 National Aerospace and Electronics Conference. Vol. 3 of NAECON 1990. pp. 1124 –1131.

Laddaga, R., 1998. Self adaptive software. Tech. Rep. BAA 98-12, DARPA BAA.
URL http://people.csail.mit.edu/rladdaga/BAA98-12excerpt.html

Laffont, J.-J., 1989. The Economics of Uncertainty and Information. The Massachusetts Institute of Technology.

Lee, J., Kotonya, G., May 2010. Combining service-orientation with product line engineering. IEEE Softw. 27, 35–41.

Lehman, M. M., 1996. Laws of software evolution revisited. In: Proceedings of the 5th European Workshop on Software Process Technology. EWSPT '96. Springer-Verlag, London, UK, UK, pp. 108–124.

Lehman, M. M., Ramil, J. F., November 2001. Rules and tools for software evolution planning and management. Ann. Softw. Eng. 11 (1), 15–44.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., Turski, W. M., 1997. Metrics and laws of software evolution - the nineties view. In: Proceedings of the 4th International Symposium on Software Metrics. METRICS '97. IEEE Computer Society, Washington, DC, USA, pp. 20–32.

Lemos, R., Giese, H., Müller, H., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D., Sousa, J., Tahvildari, L., Wong, K., Wuttke, J., 2013. Software engineering for self-adaptive systems: A second research roadmap. In: Lemos, R., Giese, H., Müller, H., Shaw, M. (Eds.), Software Engineering for Self-Adaptive Systems II. Vol. 7475 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–32.

Leymann, F., 2011. BPEL vs. BPMN 2.0: Should you care? In: Mendling, J., Weidlich, M., Weske, M. (Eds.), Business Process Modeling Notation. Vol. 67 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 8–13.

Leymann, F., Karastoyanova, D., Papazoglou, M. P., 2010. Business process management standards. In: vom Brocke, Jan, R. M. (Ed.), Handbook on Business Process Management 1. Springer-Verlag, pp. 513–542.

Little, M., October 2003. Transactions and web services. Commun. ACM 46, 49–54.

Liu, L., Yu, E., April 2004. Designing information systems in social context: a goal and scenario modelling approach. Inf. Syst. 29, 187–203.

Lora-Michiels, A., Salinesi, C., Mazo, R., January 2010. A Method Based on Association Rules to Construct Product Line Models. In: 4th International Workshop on Variability Modelling of Software-intensive Systems. pp. 147–150.
URL http://www.wi-inf.uni-duisburg-essen.de/FGFrank/download/icb/ICBReportNo37.pdf

Ludewig, J., 2003. Models in software engineering - an introduction. Software and Systems Modeling 2 (1), 5–14.

Maes, P., 1987. Computational reflection. Ph.D. thesis, Vrije Universiteit Brussel.

Maluf, D. A., Gawdiak, Y. O., Bell, D. G., 2005. On space exploration and human error - a paper on reliability and safety. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences. HICSS '05. IEEE Computer Society, Washington, DC, USA, p. 79.

March, S. T., Smith, G. F., Dec. 1995. Design and natural science research on information technology. Decis. Support Syst. 15 (4), 251–266.

Marconi, A., Pistore, M., Sirbu, A., Eberle, H., Leymann, F., Unger, T., 2009. Enabling adaptation of pervasive flows: Built-in contextual adaptation. In: Proceedings of the 7th International Joint Conference on Service-Oriented Computing. ICSOC-ServiceWave '09. Springer-Verlag, Berlin, Heidelberg, pp. 445–454.

Marples, D., Kriens, P., 2001. The Open Services Gateway Initiative: an introductory overview. Communications Magazine, IEEE 39 (12), 110–114.

Mazo, R., November 2011. A generic approach for automated verification of product line models. Ph.D. thesis, Université Paris 1 Panthéon - Sorbonne, Paris, France.
URL http://tel.archives-ouvertes.fr/tel-00707351

Mazo, R., Salinesi, C., Diaz, D., Lora-Michiels, A., 2011. Transforming attribute and clone-enabled feature models into constraint programs over finite domains. In: Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering. ENASE 2011. SciTePress, pp. 188–199.

Mazo, R., Salinesi, C., Djebbi, O., Diaz, D., Lora-Michiels, A., Apr. 2012. Constraints: the Heart of Domain and Application Engineering in the Product Lines Engineering Strategy. International Journal of Information System Modeling and Design IJISMD 3 (2), 50.

McIlroy, M. D., October 1969. Mass-produced Software Components. In: Buxton, J. M., Naur, P., Randell, B. (Eds.), Proceedings of Software Engineering Concepts and Techniques. NATO Science Committee.
URL http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

McKinley, P. K., Sadjadi, S. M., Kasten, E. P., Cheng, B. H. C., July 2004. Composing adaptive software. Computer 37, 56–64.

Mellor, S. J., Balcer, M., 2002. Executable UML: A Foundation for Model-Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Menasce, D., Gomaa, H., Malek, S., Sousa, J., 2011. SASSY: A framework for self-architecting service-oriented systems. IEEE Software 28, 78–85.

Menasce, D. A., November 2004. Composing web services: A QoS view. IEEE Internet Computing 8 (6), 88–90.

Mendonca, M., Branco, M., Cowan, D., 2009. S.P.L.O.T.: software product lines online tools. In: Proceedings of the 24th ACM SIGPLAN conference companion on object oriented programming systems languages and applications. OOPSLA '09. ACM, New York, NY, USA, pp. 761–762.

Mens, T., Demeyer, S. (Eds.), 2008. Software Evolution. Springer.

Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S., July 2010. End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo. IEEE Trans. Serv. Comput. 3, 193–205.

Michlmayr, A., Rosenberg, F., Platzer, C., Treiber, M., Dustdar, S., 2007. Towards recovering the broken SOA triangle: a software engineering perspective. In: 2nd International Workshop on Service Oriented Software Engineering. IW-SOSWE '07. ACM, New York, NY, USA, pp. 22–28.
URL http://www.infosys.tuwien.ac.at/Staff/sd/papers/Towards%20Recovering%20the%20Broken%20SOA%20Triangle.pdf

Milanovic, N., Malek, M., 2004. Current solutions for web service composition. Internet Computing, IEEE 8 (6), 51–59.

Miller, A., May 1989. Engineering design: its importance for software. Potentials, IEEE 8 (2), 14–16.

Mitchell, T. M., 1997. Machine Learning, 1st Edition. McGraw-Hill, Inc., New York, NY, USA.

Modica, S., Rustichini, A., 1994. Awareness and partitional information structures. Theory and Decision 37 (1), 107–124.

Montero, I., Peña, J., Ruiz-Cortes, A., 2008. From feature models to business processes. In: Proceedings of the 2008 IEEE International Conference on Services Computing. Vol. 2. IEEE Computer Society, Washington, DC, USA, pp. 605–608.

Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., Solberg, A., October 2009a. Models@ run.time to support dynamic adaptation. Computer 42, 44–51.

Morin, B., Barais, O., Nain, G., Jezequel, J.-M., 2009b. Taming dynamically adaptive systems using models and aspects. In: Proceedings of the 31st International Conference on Software Engineering. ICSE '09. IEEE Computer Society, Washington, DC, USA, pp. 122–132.

Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.-M., Solberg, A., Dehlen, V., Blair, G., 2008. An aspect-oriented and model-driven approach for managing dynamic variability. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems. MODELS '08. Springer-Verlag, Berlin, Heidelberg, pp. 782–796.

Moser, O., Rosenberg, F., Dustdar, S., 2008. Non-intrusive monitoring and service adaptation for WS-BPEL. In: Proceedings of the 17th international conference on World Wide Web. WWW '08. ACM, New York, NY, USA, pp. 815–824.

Mosincat, A., Binder, W., 2008. Transparent runtime adaptability for BPEL processes. In: Proceedings of the 6th International Conference on Service-Oriented Computing. ICSOC '08. Springer-Verlag, Berlin, Heidelberg, pp. 241–255.

Münch, J., Armbrust, O., Kowalczyk, M., Soto, M., 2012. Process modeling notations and tools. In: Software Process Definition and Management. The Fraunhofer IESE Series on Software and Systems Engineering. Springer Berlin Heidelberg, pp. 111–138.

Nguyen, T., Colman, A., Han, J., 2011. Modeling and managing variability in process-based service compositions. In: Proceedings of the 9th international conference on Service-Oriented Computing. ICSOC'11. Springer-Verlag, Berlin, Heidelberg, pp. 404–420.

Northrop, L. M., July 2002. SEI's software product line tenets. IEEE Softw. 19 (4), 32–40.

OASIS, March 2001. Web Services Description Language (WSDL). URL http://www.w3.org/TR/wsdl

OASIS, 2005. Universal Description, Discovery and Integration (UDDI).

URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec

OASIS, October 2006. Reference model for service oriented architecture.
URL http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

OASIS, April 2007. Web services business process execution language version 2.0.
URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

OASIS, 2012. ebXML Registry Services and Protocols.
URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep

OMG, 2008. Software & systems process engineering meta-model specification.
URL http://www.omg.org/spec/SPEM/2.0/PDF

OMG, 2011. Business Process Model and Notation (BPMN) Version 2.0.
URL http://www.omg.org/spec/BPMN/2.0/PDF/

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., Wolf, A. L., May 1999. An architecture-based approach to self-adaptive software. IEEE Intelligent Systems 14 (3), 54–62.

Ossher, H., Harrison, W., Tarr, P., 2000. Software engineering tools and environments: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ICSE '00. ACM, New York, NY, USA, pp. 261–277.

Osterweil, L., 1987. Software processes are software too. In: Proceedings of the 9th International Conference on Software Engineering. ICSE '87. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 2–13.

Ouyang, C., Dumas, M., Aalst, W. M. P. V. D., Hofstede, A. H. M. T., Mendling, J., August 2009. From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. 19, 2:1–2:37.

Ouyang, C., Dumas, M., ter Hofstede, A. H., van der Aalst, W. M., 2007. Pattern-based translation of BPMN process models to BPEL web services. International Journal of Web Services Research (JWSR) 5 (1), 42–62.
URL http://eprints.qut.edu.au/6810/

Ouyang, C., Dumas, M., ter Hofstede, A. H. M., van der Aalst, W. M. P., 2006. From BPMN process models to BPEL web services. In: Proceedings of the 2006 IEEE International Conference on Web Services. ICWS '06. IEEE Computer Society, Washington, DC, USA, pp. 285–292.

Pahl, C., Gacitua-Decar, V., Wang, M., Bandara, K., 2011. Ontology-based composition and matching for dynamic service coordination. In: Salinesi, C., Pastor, O. (Eds.), Advanced Information Systems Engineering Workshops. Vol. 83 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 464–477.

Paolucci, M., Sycara, K., Sep. 2003. Autonomous semantic web services. IEEE Internet Computing 7 (5), 34–41.

Papazoglou, M. P., Traverso, P., Dustdar, S., Leymann, F., November 2007. Service-oriented computing: State of the art and research challenges. Computer 40 (11), 38–45.

Parnas, D., 1976. On the design and development of program families. Software Engineering, IEEE Transactions on SE-2 (1), 1–9.

Parra, C., Blanc, X., Duchien, L., 2009. Context awareness for dynamic service-oriented product lines. In: Proceedings of the 13th International Software Product Line Conference. SPLC '09. Carnegie Mellon University, Pittsburgh, PA, USA, pp. 131–140.

Pelechano, V., Fons, J., Alférez, G. H., Gómez, M., 2013. Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas. RA-MA Editorial, Ch. Modelos en Tiempo de Ejecución, pp. 529 – 551.

Peltz, C., July 2003a. Web Service orchestration and choreography: a look at WSCI and BPEL4WS. Web Services Journal 3 (7), 1–5.

Peltz, C., Oct. 2003b. Web services orchestration and choreography. Computer 36 (10), 46–52.

Piechnick, C., Richly, S., Götz, S., Wilke, C., Aßmann, U., 2012. Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation – Smart Application Grids. In: Proceedings of the 4th International Conference on Adaptive and Self-adaptive Systems and Applications. ADAPTIVE 2012. pp. 93–102.

Pohl, K., Böckle, G., Linden, F. J. v. d., 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Puhlmann, F., Schnieders, A., Weiland, J., Weske, M., June 2005. Variability Mechanisms for Process Models. Tech. Rep. PESOA-Report No. TR 17/2005.
URL http://frapu.de/pdf/PESOA_TR_17-2005.pdf

Purves, D., Augustine, G. J., Fitzpatrick, D., Hall, W. C., LaMantia, A.-S., McNamara, J. O., White, L. E. (Eds.), 2008. Neuroscience, 4th Edition. Sinauer Associates.

Ramirez, A., Jensen, A., Cheng, B. H. C., 2012. A taxonomy of uncertainty for dynamically adaptive systems. In: 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 99–108.

Ranganathan, A., Al-Muhtadi, J., Campbell, R., Apr. 2004. Reasoning about uncertain contexts in pervasive computing environments. IEEE Pervasive Computing 3 (2), 62–70.

Raymond B., F., July-August 1984. Clausewitz and modern war gaming. Air & Space Power Journal.
URL http://www.airpower.maxwell.af.mil/airchronicles/aureview/1984/jul-aug/furlong.html

Razavian, M., Khosravi, R., 2008. Modeling variability in business process models using UML. In: Proceedings of the 5th International Conference on Information Technology: New Generations. ITNG '08. IEEE Computer Society, Washington, DC, USA, pp. 82–87.

Recker, J. C., Mendling, J., 2006. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In: Latour, T., Petit, M. (Eds.), 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortiums. Namur University Press,

pp. 521–532.
URL http://eprints.qut.edu.au/4637/

Rinderle, S., Reichert, M., Dadam, P., Jul. 2004. Correctness criteria for dynamic changes in workflow systems: a survey. Data Knowl. Eng. 50 (1), 9–34.

Rosemann, M., Van der Aalst, W. M. P., Mar. 2007. A configurable reference modelling language. Inf. Syst. 32 (1), 1–23.

Rossi, F., Van Beek, P., Walsh, T., 2006. Handbook of constraint programming. Elsevier Science.

Salehie, M., Tahvildari, L., May 2009. Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4 (2), 14:1–14:42.

Salinesi, C., Mazo, R., April 2012. Software Product Line - Advanced Topic. InTech, Ch. Defects in Product Line Models and how to Identify them, pp. 3–22.
URL http://www.intechopen.com/books/software-product-line-advanced-topic/defects-in-product-line-models-and-how-to-identify-them

Salinesi, C., Mazo, R., Diaz, D., Djebbi, O., 2010. Using integer constraint solving in reuse based requirements engineering. In: Proceedings of the 2010 18th IEEE International Requirements Engineering Conference. RE '10. IEEE Computer Society, Washington, DC, USA, pp. 243–251.

Sawyer, P., Mazo, R., Diaz, D., Salinesi, C., Hughes, D., October 2012. Using constraint programming to manage configurations in self-adaptive systems. Computer 45 (10), 56 –63.

Schmidt, A., November 2002. Ubiquitous computing - computing in context. Ph.D. thesis, Lancaster University.
URL http://www.comp.lancs.ac.uk/~albrecht/phd/

Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., Bontemps, Y., February 2007. Generic semantics of feature diagrams. Comput. Netw. 51 (2), 456–479.

Schumm, D., Karastoyanova, D., Leymann, F., Nitzsche, J., 2009. On visualizing and modelling BPEL with BPMN. In: Workshops at the

Grid and Pervasive Computing Conference, 2009. GPC '09. pp. 80–87.

SEI, n.d. Framework for software product line practice.
URL http://www.sei.cmu.edu/productlines/tools/framework/

Selic, B., Sep. 2003. The pragmatics of model-driven development. IEEE Softw. 20 (5), 19–25.

Shivam, P., Babu, S., Chase, J. S., 2006. Learning application models for utility resource planning. In: IEEE International Conference on Autonomic Computing, 2006. ICAC '06. pp. 255–264.

Siljee, J., Bosloper, I., Nijhuis, J., Hammer, D., 2005. DySOA: making service systems self-adaptive. In: Proceedings of the 3rd international conference on service-oriented computing. ICSOC '05. Springer-Verlag, Berlin, Heidelberg, pp. 255–268.

Sonntag, M., Karastoyanova, D., August 2011. Compensation of adapted service orchestration logic in BPEL'n'aspects. In: Proceedings of the 9th International Conference on Business Process Management. BPM '11. Springer-Verlag, pp. 1–16.

Srinivasan, N., Paolucci, M., Sycara, K., 2005. An efficient algorithm for OWL-S based semantic search in UDDI. In: Proceedings of the 1st International Conference on Semantic Web Services and Web Process Composition. SWSWPC '04. Springer-Verlag, Berlin, Heidelberg, pp. 96–110.

Stuart-Fox, D., Moussalli, A., Whiting, M. J., 2008. Predator-specific camouflage in chameleons. Biology Letters 4 (4), 326–329.

Sun, C., Rossing, R., Sinnema, M., Bulanov, P., Aiello, M., 2010. Modeling and managing the variability of web service-based systems. Journal of Systems and Software 83 (3), 502 – 516.

Taylor, B. N., Kuyatt, C. E., September 1994. Guidelines for evaluating and expressing the uncertainty of NIST measurement results. Tech. Rep. 1297, National Institute of Standards and Technology, Gaithersburg, MD 20899-0001.
URL http://physics.nist.gov/Pubs/guidelines/TN1297/tn1297s.pdf

Tesauro, G., Kephart, J. O., 2004. Utility functions in autonomic systems. In: Proceedings of the 1st International Conference on Autonomic Computing. ICAC '04. IEEE Computer Society, Washington, DC, USA, pp. 70–77.

Torres, M. V., 2008. A web engineering approach for the development of business process-driven web applications. Ph.D. thesis, Universitat Politécnica de Valéncia.
URL http://riunet.upv.es/handle/10251/2933

Torres, V., Giner, P., Pelechano, V., 2012. Developing BP-driven web applications through the use of MDE techniques. Software and Systems Modeling 11, 609–631.

Tseng, M. M., Jiao, J., 2001. Mass customization. In: Salvendy, G. (Ed.), Handbook of Industrial Engineering: Technology and Operations Management. John Wiley & Sons, Ch. 25, pp. 684–709.

Tversky, A., Kahneman, D., 1974. Judgment under uncertainty: Heuristics and biases. Science 185 (4157), 1124–1131.

Vaishnavi, V., Kuechler, W., January 2004. Design science research in information systems.
URL http://desrist.org/design-research-in-information-systems/

Van der Aalst, W. M. P., Dumas, M., Gottschalk, F., ter Hofstede, A. H. M., Rosa, M. L., Mendling, J., May 2010. Preserving correctness during business process model configuration. Form. Asp. Comput. 22 (3-4), 459–482.

Van Gurp, J., 2000. Variability in software systems: the key to software reuse. Ph.D. thesis, University of Groningen.
URL http://www.jillesvangurp.com/static/Lic/licentiatethesis.pdf

W3C, October 2002. Web services description requirements. Tech. rep., W3C.
URL http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028/ws-desc-reqs.pdf

W3C, February 2004. Resource Description Framework (RDF).
URL http://www.w3.org/RDF

W3C, April 2007. SOAP Version 1.2.
URL http://www.w3.org/TR/soap/

Weber, B., Reichert, M., Rinderle-Ma, S., September 2008. Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. 66, 438–466.

Webster, J., Watson, R. T., Jun. 2002. Analyzing the past to prepare for the future: writing a literature review. MIS Q. 26 (2), xiii–xxiii.

Welsh, K., Sawyer, P., Bencomo, N., nov 2011. Towards requirements aware systems: Run-time resolution of design-time assumptions. In: 26th IEEE/ACM International Conference on Automated Software Engineering. ASE '11. pp. 560 –563.

White, J., Schmidt, D. C., Wuchner, E., Nechypurenko, A., 2007. Automating product-line variant selection for mobile devices. In: Proceedings of the 11th International Software Product Line Conference. IEEE Computer Society, Washington, DC, USA, pp. 129–140.

White, S. A., May 2004. Business process modeling notation, v. 1.0. URL http://www.omg.org/bpmn/Documents/BPMN_V1-0_May_3_2004.pdf

White, S. A., March 2005. Using BPMN to model a BPEL process. BPTrends 3 (3), 1–18.
URL http://bptrends.com/publicationfiles/03-05%20WP%20Mapping%20BPMN%20to%20BPEL-%20White.pdf

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., 2012. Experimentation in Software Engineering. Springer.

Yu, E. S., 2009. Conceptual modeling: Foundations and applications. Springer-Verlag, Berlin, Heidelberg, Ch. Social Modeling and i*, pp. 99–121.
URL http://dx.doi.org/10.1007/978-3-642-02463-4_7

Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J. C. S. P., 2008. From goals to high-variability software design. In: Proceedings of the 17th International Conference on Foundations of Intelligent Systems. ISMIS '08. Springer-Verlag, Berlin, Heidelberg, pp. 1–16.

Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J., Chang, H., May 2004. QoS-aware middleware for web services composition. IEEE Trans. Softw. Eng. 30 (5), 311–327.

Zenmyo, T., Yoshida, H., Kimura, T., 2006. A self-healing technique based on encapsulated operation knowledge. In: Proceedings of the 2006 IEEE International Conference on Autonomic Computing. IEEE Computer Society, Washington, DC, USA, pp. 25–32.

Zhang, J., Cheng, B. H. C., 2006. Model-based development of dynamically adaptive software. In: Proceedings of the 28th international conference on Software engineering. ICSE '06. ACM, New York, NY, USA, pp. 371–380.

Zhao, C., Duan, Z., Zhang, M., 2009. A model-driven approach for dynamic web service composition. In: Proceedings of the 2009 WRI World Congress on Software Engineering - Volume 04. WCSE '09. IEEE Computer Society, Washington, DC, USA, pp. 273–277.

Ziv, H., Richardson, D. J., Klösch, R., August 1997. The uncertainty principle in software engineering. In: Proceedings of the 19th International Conference on Software Engineering (ICSE).

DECLARATION

Germán Harvey Alférez Salinas

This thesis provides a tool-supported framework to guide autonomic adjustments of context-aware service compositions using models at runtime. This framework spans over design time and runtime to face arising known and unknown context events (i.e., foreseen and unforeseen at design time) in the closed and open worlds, respectively.

December 2013

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA