Achieving Autonomic Web Service Compositions with Models at Runtime

Germán H. Alférez[a,*], Vicente Pelechano[b]

[a] *Facultad de Ingeniería y Tecnología, Universidad de Montemorelos, Apartado 16-5, Montemorelos N.L., Mexico*
[b] *Centro de Investigación en Métodos de Producción de Software (ProS), Universitat Politècnica de València, Camí de Vera s/n, E-46022, Spain*

Abstract

Several exceptional situations may arise in the complex, heterogeneous, and changing contexts where Web service operations run. For instance, a Web service operation may have greatly increased its execution time or may have become unavailable. The contribution of this article is to provide a tool-supported framework to guide autonomic adjustments of context-aware service compositions using models at runtime. At runtime, when problematic events arise in the context, models are used by an autonomic architecture to guide changes of the service composition. Under the closed-world assumption, the possible context events are fully known at design time. 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. An evaluation demonstrates that our framework is efficient during dynamic adjustments.

Keywords: Web service compositions, models at runtime, autonomic computing, dynamic software product lines, dynamic adaptation, dynamic evolution

1. Introduction

In nature, adaptation contributes to the survival of individuals to cope with the weather, enemies, or any hazards. 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

*Corresponding author.
Email addresses: harveyalferez@um.edu.mx (Germán H. Alférez), pele@dsic.upv.es (Vicente Pelechano)

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 fact, adaptability is emerging as a necessary underlying capability of highly-dynamic context-aware systems [1]. The *context* is any information that can be used to characterize the situation of an entity [2]. Context-aware systems are concerned with the acquisition of context, the abstraction and understanding of context, and application behavior based on the recognized context [3]. Adaptability is specially relevant in systems built upon Web service operations. 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 Web service compositions (or simply called service compositions). However, implementing dynamic adaptations with variability constructs at the language level can become complex and error-prone, specially in large systems [4].

This article provides a framework based on models at runtime to guide dynamic adjustments of context-aware service compositions. *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 [5]. Our framework tries to solve the following gaps found in related work on autonomic service compositions: 1) need for designing the autonomic behavior of context-aware service compositions by means of easy-to-understand abstractions; 2) need for facing unanticipated context events (unseen at design time) in the open world; 3) need for transparent solutions that do not require changes in the orchestration engine; and 4) need for means of verifying autonomic adjustments to achieve safe reconfigurations.

In our approach, in response to changes in the context, the system itself can query models during execution to determine the necessary modifications in the underlying service composition. When a problematic event arises in the context, models are leveraged for decision-making. The activation and deactivation of features in a variability model result in changes in a composition model that abstracts the service composition. Our approach verifies new possible configurations before they are applied to the running service composition. Changes in the models 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. Our solution does not require the modification of the orchestration engine. If model adaptations are not enough to solve uncertainty at runtime, we provide a solution to guide the dynamic evolution of the supporting models to preserve high-level requirements. We present novel evaluation results of our framework that demonstrate its potential.

The rest of this article is organized as follows: Section 2 presents the state of the art. In this section, a taxonomy is proposed to facilitate the analysis of related work. Section 3 describes the main building blocks of our framework. Section 4 presents our framework for autonomic service compositions. Section 5 presents the evaluation of our framework. Section 6 presents conclusions and

future work.

2. Research on Autonomic Service Compositions

This section examines related work proposed during a period of thirteen years to achieve autonomic service compositions in the following areas: variability constructs at the language level, brokers, models at runtime, and dynamic software product lines (DSPLs). These areas are covered in this section because of their relevance in the state of the art of autonomic service compositions and their closed relationship with our approach. The aim of studying these approaches is to answer the following questions:

- `Question 1 (Q1):` Which approaches have been proposed in recent years to achieve autonomic service compositions by means of variability constructs at the language level, brokers, models at runtime, and DSPLs?
- `Question 2 (Q2):` How to characterize the approaches for autonomic service compositions according to desirable properties of Self-Adaptive Systems (SAS)?
- `Question 3 (Q3):` What are the gaps to be faced in the field of autonomic service compositions?

In order to answer `Q1`, we identified **31 approaches** on autonomic service compositions classified in the four studied areas of research (see Table 1). 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: 1) WS-BPEL has an inherent static nature; and 2) WS-BPEL does not provide any means for monitoring the context. Therefore, research works in the first 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.

In another area of research, a system works as a broker 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 Quality of Service (QoS) attributes) to accomplish a task.

In the third area of research, models at runtime extend the applicability of models produced in Model-driven Engineering (MDE) approaches to execution time [40]. Since models at runtime provide up-to-date and exact information about the runtime system, they can offer a rich semantic base for runtime decision-making in order to achieve system adaptation. Thus, the system itself

Year	Variability Constructs at the Language Level	Broker	Models at Runtime	DSPL
2016		Multi-Agent Reinforcement Learning [6], and Trustworthy Stigmergic-based Self-Organization[7]		
2015		MoDAR [8], LIOM[9], and Dynamic Evolution [10]	Requirements-Driven Self-Optimization [11]	
2014		QoS-Gasp[12]		
2013		Chemistry-Inspired Middleware [13]	Runtime Evolution [14]	FM-DAMASCo [15]
2012			MUSIC [16] and MOSES [17]	
2011	Self-Supervising WS-BPEL Processes [18]		SASSY [19], MAESoS [20], and QoSMOS [21]	
2010		Percentile-Based SLAs [22]		SOPL [23]
2009	VxBPEL [24]	BPEL'n'Aspects [25, 26]		CAPucine [27]
2008		VieDAME [28], Transparent Runtime Adaptability [29], and QoS-Aware Binding [30]		
2007	AO4BPEL [31]	Robust-BPEL2 [32], and Paws [33, 34]		
2006	SCENE [35]	WSQoSX [36]		
2005			DySOA [37, 38]	
2004		AgFlow [39]		

Table 1: Compendium of primary research works on autonomic service compositions from 2004 to 2016.

can query the models at runtime to make adaptation decisions, to choose the adaptation strategy, and to control the adaptation process.

Also, Dynamic Software Product Line Engineering (DSPLE) has been proposed as a way to achieve autonomic service compositions. 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 [41].

According to Table 1, brokers have been the most widely used mechanism to achieve autonomic service compositions among the areas covered in this section. Also, it is interesting to see that variability constructs at the language level were a popular way to achieve autonomic service compositions during several years (from 2006 to 2011). Nevertheless, this interest has moved in recent years towards models at runtime or even DSPLE.

In order to answer `Q2`, we propose a taxonomy to facilitate the analysis of research works. This taxonomy has a set of dimensions that describe several expected facets of autonomic service compositions (see Table 2). 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. Several dimensions were taken from previous taxonomies for SAS [42, 43] 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, change level, and the dimensions that are related to the maturity of the approaches for autonomic service compositions. For details on how these research works have been organized according to the proposed taxonomy, see supplementary material [1].

Table 3 presents a summary of the research works in the four areas presented in this section. This table shows that adaptations are caused by exclusively external factors in the 54.84% of works, by exclusively internal factors in the 6.45% of works, and by both external and internal factors in the 22.58% of works. The nature of change is related to non-functional requirements in the 54.84% of works, to functional requirements in the 9.67% of works, and to both functional and non-functional requirements in the 29.03% of works.

The vast majority of the presented works propose fully automated solutions for service compositions (83.87%). Also, the most of the approaches localize the necessary adaptations (77.41% of works have local scope and 9.67% of works have both local and global scopes). Moreover, the 70.96% of works trigger adaptations to face events (event-triggered) and 6.45% of works are both event-triggered and time-triggered. 51.61% of works have a decentralized architecture for adaptation. There are several works that do not provide information about the frequency of changes (38.71%). Nevertheless, the 45.16% of works show a high tendency to carry out adaptations at arbitrary intervals. The adaptation time of the presented solutions is short in the 41.93% of works. However, the 35.48% of works do not provide any information about adaptation time. The majority of research works deal with autonomic adjustments of service compositions that are implemented with Web service operations – non-generic SOA-related solutions (70.96%).

According to the maturity-related dimensions, the 61.29% 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 54.83% of studied approaches do not specify the enterprise orchestration engines that were used to execute the adjusted service composition; and 2) the 93.54% of works do not provide demonstrability means (such as source code or video demonstration).

[1] http://www.harveyalferez.com/StateOfTheArt_AchievingAutonomicWebServiceCompositionsWithModelsAtRuntime/

However, we cannot conclude it is a sign of immaturity. Instead, it may be caused by other facts, such as protection of intellectual property. Finally, although the 51.61% of works provide evaluation results that are uniquely based on examples, there is a significant percentage of works (45.16%) that use strong evaluation mechanisms (such as simulations or industrial case studies).

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	Service-Oriented Architecture (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 2: Taxonomy to classify research works on autonomic service compositions.

The information in Table 3 allows to answer the following question: `Q3`, *What are the gaps to be faced in the field of autonomic service compositions?* Our analysis is as follows:

- **Need for transparency:** Only 35.48% of the studied research works offer a transparent solution (they do not require changes in the orchestration engine). The 58.06% of the studied research works do not provide

Dimension	Summary of Research Works
WHAT change is the cause for adaptation	
Source	17 external, 2 internal, 7 external and internal, 4 N/S, and 1 N/A
Type	17 non-functional, 3 functional, 9 functional and non-functional, 1 N/S, and 1 N/A
HOW the service composition faces changes	
Autonomy	26 automated, 2 manual, 1 automated or manual, and 2 N/S
Organization	16 decentralized, 7 centralized, 7 N/S, and 1 N/A
Scope	24 local, 2 global, 3 local and global, and 2 N/S
Triggering	22 event-trigger, 1 time-trigger, 2 event- and time-trigger, 1 manual, 4 N/S, and 1 N/A
Safety	6 verified and 25 unverified
Transparency	11 transparent, 2 no transparent, and 18 N/S
WHEN changes are carried out	
Frequency	14 at arbitrary intervals, 4 periodically, 12 N/S, and 1 N/A
Adaptation time	13 short, 1 medium, 5 long, 11 N/S, and 1 N/A
Anticipation	25 foreseen, 3 unforeseen, 1 foreseen and unforeseen, 1 N/S, and 1 N/A
WHERE changes are carried out	
Generality	7 SOA, 22 Web services, 1 Web services and API recommenders, and 1 services
Change level	4 at the language level, 16 at the service message level, 8 at the abstract level and in underlying mechanisms (e.g. service message), 1 aspect injection, 1 bricks (components), and 1 bind/unbind components
MATURITY of the approach	
Applicability scenarios	1 PXE, 1 BPWS4J, 7 ActiveBPEL, 1 OSGi, 1 Apache ServiceMix and XTEAM, 1 Riftsaw, 1 WSO2 Enterprise Service Bus, 1 Oracle BPEL Process Manager, and 17 N/S
Demonstrability	1 has available code and demonstrations, 1 has available source code, and 29 N/S
Life-cycle support	16 cover design time and runtime, 3 cover implementation and runtime, and 12 cover runtime
Evaluation	16 use examples, 2 use simulations, 2 case studies, 1 uses an example and industrial experiments, 2 use simulations and examples, 6 experiments, 1 experiments and case study, and 1 N/S

Table 3: Summary of the approaches that support autonomic service compositions (N/A means No Applicable, N/S means Not Specified).

information about transparency. In any case, there is a need for transparent solutions to guide dynamic adjustments. A transparent solution can result in higher flexibility to adopt autonomic computing in service compositions.

- **Need for facing unanticipated context events:** The vast majority of research works foresee the possible context events at design time (80.64%). 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 safe reconfigurations:** The vast majority of related work does not provide means to verify autonomic adjustments (80.64%). As a result,

there is a need for verifying autonomic adjustments to achieve safe configurations at runtime. Otherwise, the benefits of autonomic computing will be diminished with erroneous adjustments.

- **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 [4]. On the other hand, brokers have been widely used to implement QoS-aware Web service compositions. Although QoS-aware Web service compositions have proven to be effective, the selection of Web services maximizing the QoS of the overall service composition leads to an optimization problem that is NP-hard [30].

  In order to reach an easier-to-understand and more flexible solution for autonomic service compositions, several approaches based on models at runtime have appeared in recent years. In these research works, the knowledge in models created at design time is used to reason about the problem and solution domains during execution. Nevertheless, in order to materialize the industrial adoption of models at runtime as a feasible option to achieve autonomic service compositions, we believe that it is important to consider the following challenges found in the presented eight approaches based on models at runtime: 1) among the set of research works, just one of them describes a transparent solution, which does not require the orchestration engine to be modified; 2) models at runtime are causally connected to the underlying service composition (i.e., if this model changes, then the service composition changes and vice versa). Therefore, a logical benefit of using models at runtime is to verify new configurations of the service composition first at the model level to ensure safe configurations. Nevertheless, just the half of the presented approaches offer mechanisms for verifying new configurations of the service composition; 3) just two approaches offer mechanisms for facing unanticipated context events at runtime. Most of them are focused on the closed world; 4) three research works were applied on industrial orchestration engines and none of them offer demonstrations. These facts may negatively impact the adoption of models at runtime in industry; and 5) half of the research works do not provide information about adaptation time. This fact implies the need for stronger evaluations that show the feasibility of models at runtime to solve self-adjustments of the service composition in a reasonable time.

3. Main Building Blocks of Our Framework

The feature model in Figure 1 presents the main building blocks of our framework for achieving autonomic service compositions from a conceptual point of view. 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.

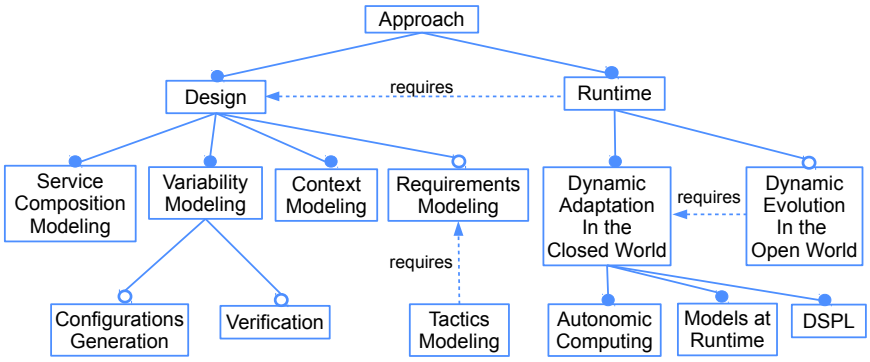

Figure 1: Main building blocks of the approach from a conceptual point of view.

3.1. 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. To this end, we propose the creation of a composition model that is causally connected to the underlying service composition. The Business Process Model and Notation (BPMN) 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. Atomic Web service operations are abstracted as BPMN tasks. Composite service operations are abstracted as BPMN subprocesses. The workflow that is followed by the service composition is abstracted by BPMN gateways and sequence flows that connect BPMN tasks and subprocesses.

- **Variability Modeling:** The composition model by itself lacks semantics for variability. Therefore, it is necessary to count on feasible, semantically-rich, and coarse-grained variability representations of service compositions. A *feature model* is 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 this model to determine the necessary modifications in the service composition. We propose to appoint certain features in the 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 [44]. The *current configuration* 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.

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, we propose the creation of the following models [44]: 1) a *BPMN base composition model* extends the composition model with semantics for variability and preserves the main workflow during adaptations; and 2) a set of *variant BPMN models* to be bound into the variation points of the base composition model during execution.

There are two optional sub-blocks related to variability modeling:

- **Configurations Generation:** The *configuration* of a service-based system is the set of all active features in its related 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.
- **Verification:** Even though it is possible to use an unverified variability model 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, expressed as a feature 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. The following datatype properties are used in the context model [44]: 1) the isAvailable data type property indicates whether the service operation is currently available (it is a Boolean value); and 2) the *hasExecutionTime* datatype property observes the current execution time in milliseconds that a service operation takes to execute a job.

  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 c*ontext conditions* that may trigger adaptations. 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. Each context condition is represented as a Resource Description Framework (RDF) triple in the form of (subject, predicate, object) [44].

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).

We propose to use the *resolution* concept to represent the set of changes in a feature model triggered by a context condition [44, 45]. 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.

- **Requirements Modeling:** 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. Since the requirements model is leveraged during dynamic evolutions in the open world, it is considered an optional block. We are particularly interested in keeping non-functional requirements at runtime. Therefore, the Goal-oriented Requirements Language (GRL) [46] has been used for requirements modeling because it is focused on non-functional requirements [47, 48, 49].

- **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, highly-abstract tactic models can be used to express the tactical functionality to be triggered on the underlying service composition to preserve affected requirements [47, 48, 49].

  Since tactics inject new variant functionalities into the service composition to preserve requirements, tactic models are 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. 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.

  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.

3.2. Runtime-Related Building Blocks

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. 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? The Oxford dictionary[2] defines these terms as follows: *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. 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 [50]. These laws essentially characterize the software evolution process as a self-stabilizing and self-regulating system, subject to continuing growth and change.

Figure 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.

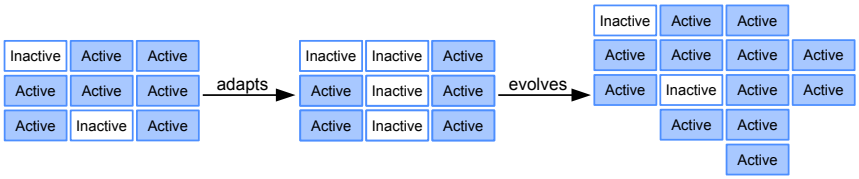

Figure 2: Dynamic adaptation vs. Dynamic evolution of software.

[2] `http://oxforddictionaries.com`

In our approach, 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 a variability model guide adaptations in the service composition according to known (or foreseen at design time) context events. The following sub-blocks are the underpinnings for the dynamic adaptation of service compositions: 1) **Autonomic Computing:** In order to support the dynamic adaptation of service composition through autonomic computing, we propose a computing infrastructure that implements the components of IBM's Monitor, Analyze, Plan, Execute, and Knowledge (MAPE-K) loop [51]; 2) **Models at Runtime:** In our approach, the set of models that are created under the **Design Building Block** are used at runtime to automatically determine how the service composition should be adjusted; and 3) **DSPL:** DSPLE goes a step further from traditional software product lines 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. Despite the recognized need for handling unexpected events in SAS [52, 53], the dynamic evolution of service compositions in the open world 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:

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

4. 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 3. This framework consists of three phases: *Design*, *Dynamic Adaptation*, and *Dynamic Evolution*.

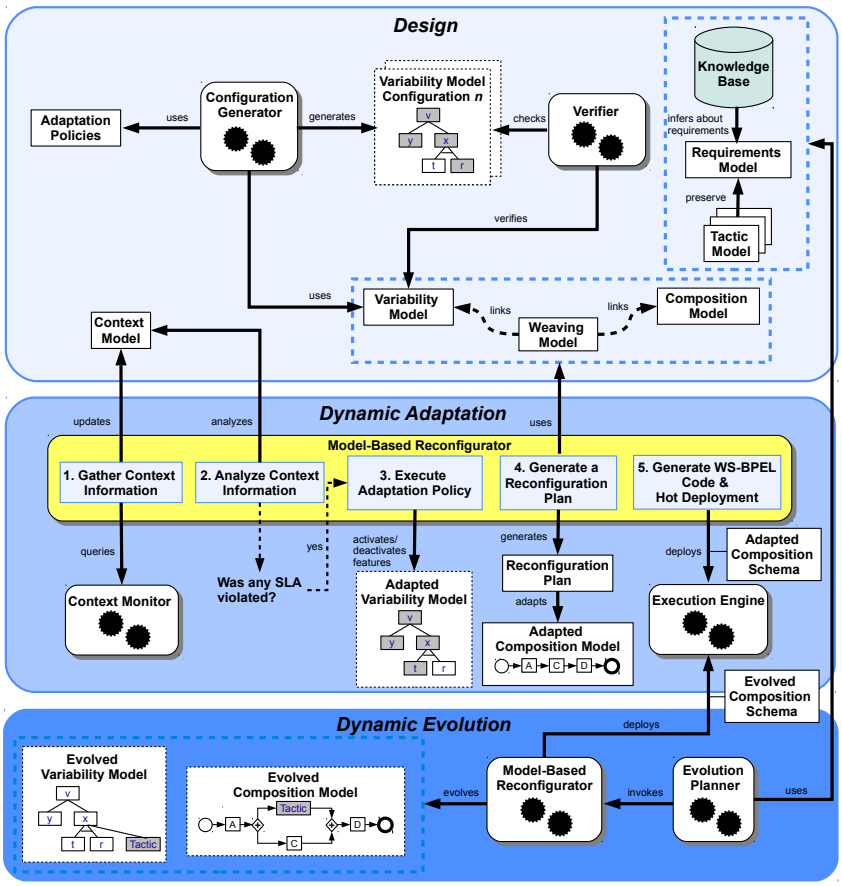

Figure 3: A framework for the dynamic adjustment of service compositions.

4.1. 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. To this end, the *Design Phase* covers the creation of a set of models and other supporting artifacts. From a methodological point of view, in order to organize the work of requirements engineers, systems analysts, and developers in the creation of these assets for dynamic adaptation and dynamic

evolution, we propose two software process models. These process models are supported by two method contents, which are based on the Software and Systems Process Engineering Meta-Model (SPEM) 2.0 [54]. In order to conserve space, the method contents are available online[3].

The models to be created at design time are summarized as follows. The *Systems Analyst* role creates a *composition model* that describes the workflow in the service composition and a *variability model* that 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 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 [44]. We also propose the creation of two additional supporting models by the *Systems Analyst* role. First, a *context model* formalizes the 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 [44, 45].

Two tools provide variability reasoning at design time, the Configuration Generator and the Verifier. The Configuration Generator uses the variability model and the set of adaptation policies in terms of *resolutions* to automatically 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. For example, the right-hand side of Figure 4 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} (i.e., the resolution R to face the context condition C_1), results in thirteen transitions in the adaptation space. The Configuration Generator is implemented with our MOSKitt4SPL tool[4].

The Verifier uses constraint programming 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,

[3] `http://www.harveyalferez.com/thesis/method_contents.html`
[4] `https://tatami.dsic.upv.es/moskitt4spl/`

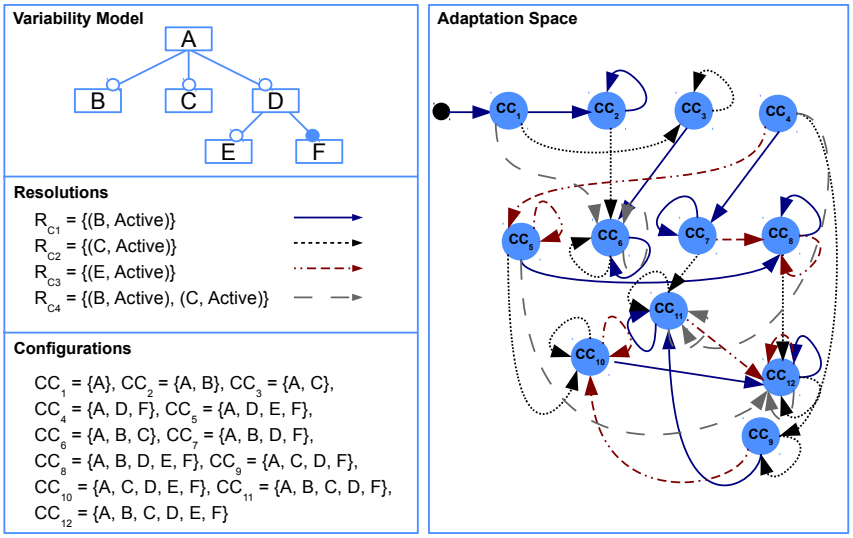

Figure 4: Adaptation space with a simple variability model and a set of resolutions.

which can drastically diminish the quality and outcome of the entire adaptation. This VERIFIER is not attached to any specific tool. Therefore, we have chosen FAMA-FW[5] and GNU PROLOG[6] to implement the VERIFIER depending on the desired verification operations. Specifically, four verification operations are implemented in GNU PROLOG: accuracy of the variability model, non-existence of dead features, stability, and semi-aliveness [44].

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*. Specifically, the *Systems Analyst* role specifies tactics that 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 artificial intelligence mechanism looks for the requirements that can be affected by an unknown context event. Therefore, the *Systems Analyst* role also defines 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*

[5] `http://www.isa.us.es/fama`
[6] `http://gprolog.univ-paris1.fr`

[47, 48, 49].

4.2. 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. 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. In our approach, the Model-Based Reconfigurator is materialized by our Model-based Reconfiguration Engine for Web Services (MoRE-WS) tool, which implements the components of the MAPE-K loop [55]. 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.

The Context Monitor counts on sensors to monitor the context and to get the measures for basic metrics of quality attributes. Specifically, our prototype has a sensor for *Availability* that measures the availability of a Web service operation. 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. The sensors implement the ping/echo approach by sending requests to service operations and waiting for a response. Requests are sent continually, periodically, and sequentially to each service operation to be observed. Several sensors can be chosen at the same time. Also, they can be extended according to particular needs (e.g. a sensor to monitor security). As soon as the Context Monitor starts running, it creates an XML file to store the observed context information. Each record in this document keeps an increasing identification number for each context observation, the name of the observed service operation, and the timestamp of the observation.

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 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 fulfilled. If a context condition is fulfilled, then an adaptation is triggered on the service composition to deal with the arising situation. We have implemented the operations to insert context information into the context model and reason about

context conditions by means of SPARQL[7].

As soon as an adaptation has been requested (i.e., after a context condition has been fulfilled), MoRE-WS carries out three steps to plan the adaptation of the service composition:

1. MoRE-WS triggers a resolution associated to a context condition, which has occurred. To this end, MoRE-WS carries out the following actions (see Figure 5):

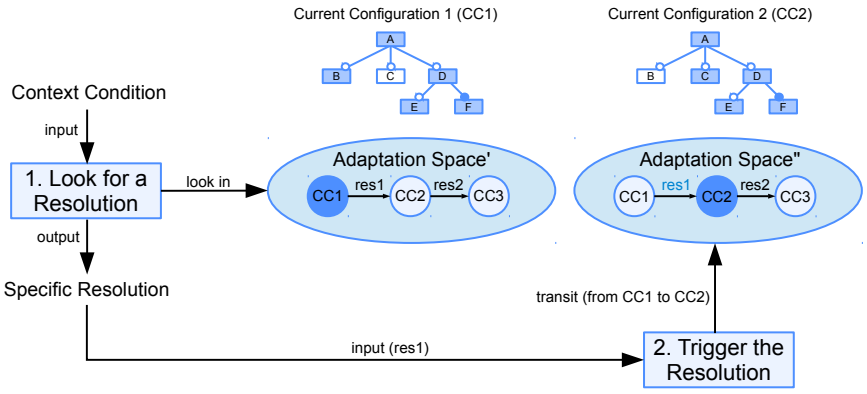

Figure 5: Execute a resolution.

(a) *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.

(b) *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.

[7] `http://www.w3.org/TR/rdf-sparql-query/`

2. 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 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 6).

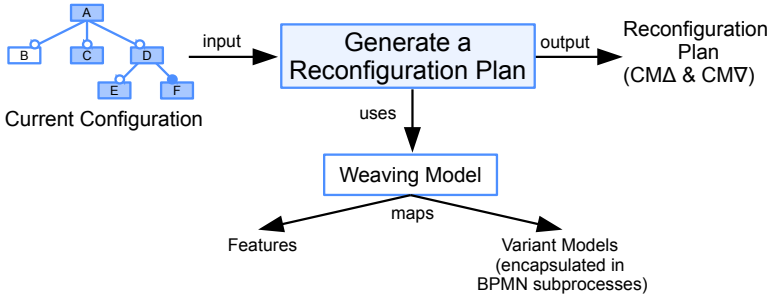

Figure 6: Generate a reconfiguration plan.

In order to generate reconfiguration actions, MoRE-WS queries a weaving model to realize the mappings between the features that are active in the new configuration of the feature model and a set of related BPMN subprocesses, which abstract BPMN variant models that can be injected into a BPMN base composition model [44]. 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.

3. 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:
   (a) 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 intact the information of the variation points.
   (b) MoRE-WS deletes all the modeling elements in the variation points of the running composition model that are affected by $CM\nabla$ actions.
   (c) MoRE-WS loads the file in XML Metadata Interchange (XMI) format that contains the BPMN base composition model and the BPMN

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.

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

(e) MoRE-WS saves the new version of the running composition model.

Finally, 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 transparently on the Execution Engine. In turn, the Execution Engine uses the adapted WS-BPEL composition schema to orchestrate the service composition.

4.3. Dynamic Evolution Phase

In the *Dynamic Evolution Phase*, the knowledge in models is used to guide the dynamic evolution of the service composition in the open world. To this end, the Evolution Planner queries the context information in the context model to find out if a requirement in the requirements model can be negatively impacted by an unknown context event. In order to find the requirement(s) that can be affected by unknown context events, the Evolution Planner uses forward chaining [47, 48, 49]. 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.

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* [56]. In order to preserve affected requirements, the Evolution Planner chooses surviving tactics. According to the chosen tactics, MoRE-WS evolves the variability and composition models by means of tactic models. The following steps are carried out to reflect the changes in the evolved composition model into the WS-BPEL composition schema [49]: 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 Web Services Description Language (WSDL) file, 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. In each evolution, MoRE-WS puts the evolved composition schema and other required artifacts into a deployment directory. This directory is hot deployed by the Execution Engine.

5. Evaluation

This section presents a set of experiments to measure the following: 1) the performance of the Context Monitor, which was not evaluated in our previous work. We believe that the efficient context observation is a requirement to carry out self-adjustments on time; and 2) the performance of all the internal operations that are carried out in MoRE-WS and in the Evolution Planner during dynamic adaptations and dynamic evolutions. We argue that a comprehensive evaluation of these operations can help to demonstrate that models at runtime are a feasible way to guide dynamic adjustments of service compositions. We chose to evaluate execution time because SAS are expected to offer prompt self-adjustments in response to arising context events. Also, we evaluated memory consumption because models are loaded into memory to make decisions.

In order to develop the evaluation metrics, we used the Goal/Question/Metric (GQM) paradigm [57]. Each GQM model supports key aspects of our contribution at the *Dynamic Adaptation* and *Dynamic Evolution* phases of our framework. The GQM models in Section 5.1 present three GQM models that are used to evaluate the Context Monitor and MoRE-WS during dynamic adaptations. The GQM model in Section 5.2 presents one GQM model that is used to evaluate the Evolution Planner and MoRE-WS to carry out dynamic evolutions. In the experiments, we used the online-book-shopping service composition presented in our previous work [44]. Two video demonstrations of our framework in action, one for dynamic adaptation and another for dynamic evolution, are available online[8].

The Web services in the experiments ran on Apache Axis2[9] version 1.6.1, which is deployed as a WAR distribution on Apache Tomcat[10] version 7.0.8. Hot deployment is carried out by MoRE-WS on Apache ODE[11] 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 Open Services Gateway Initiative (OSGi) bundles. The aforementioned pieces run on a 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.

5.1. Validation in the Dynamic Adaptation Phase

This section presents three GQM models, which are used to validate key aspects of our implemented Context Monitor and MoRE-WS. These two tools are used to support the *Dynamic Adaptation Phase* of our framework.

[8] `http://www.harveyalferez.com/thesis/videos.html`
[9] `http://axis.apache.org/axis2/java/core`
[10] `http://tomcat.apache.org`
[11] `http://ode.apache.org`

5.1.1. Context Observation Efficiency

Table 4 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	**Q1**	Is the Context Monitor efficient to observe service operations?
Metrics	**M1 - M5**	(**M1**) execution time for observing service operations, (**M2**) memory consumption when observing service operations, (**M3**) number of observed service operations, (**M4**) observation period, and (**M5**) elapsed time

Table 4: GQM model for the "efficient context observation of service operations from the Context Monitor's viewpoint" goal.

In order to answer **Q1**, we measured the execution time (**M1**) and the memory consumption (**M2**) 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 (**M3**) measured in a period of time (**M4**) during an elapsed time (**M5**). In our case, it observed the 15 service operations of our case study (common and variant operations) in sequence every five seconds (**M4**) during one hour (**M5**). In every observation, the Context Monitor saved the observed data in a file. Figure 7 shows the resulting execution time 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.

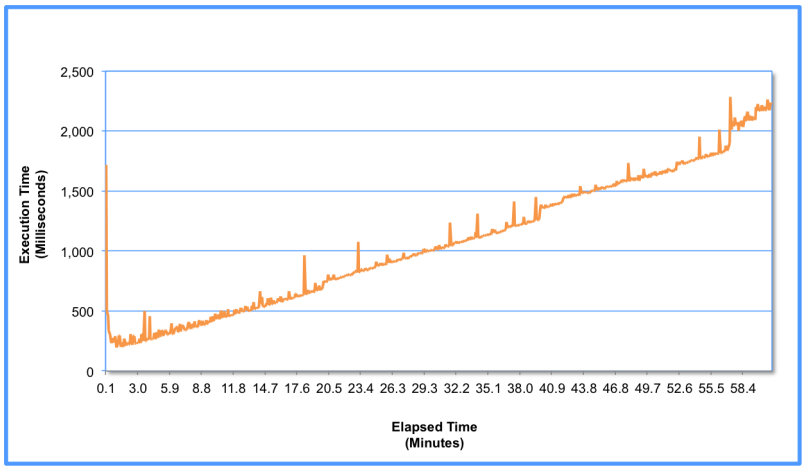

Figure 7: Resulting execution time for the experiment to answer **Q1**.

Figure 8 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 increases the amount of free memory. Therefore, this figure shows periodic and sudden memory improvements.

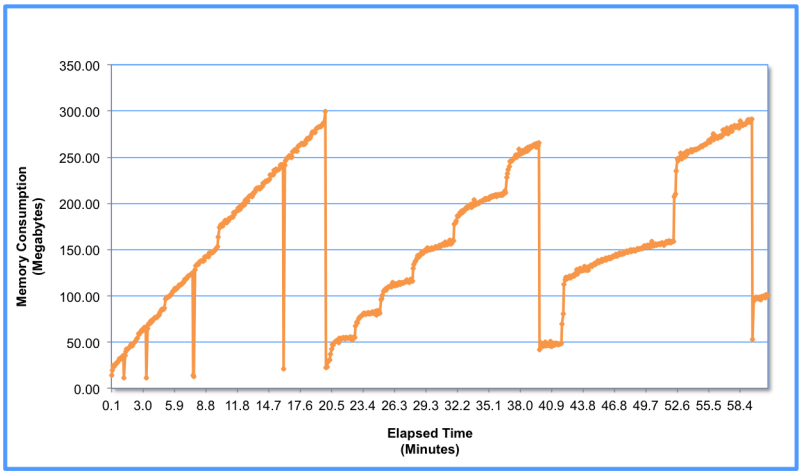

Figure 8: Resulting memory consumption for the experiment to answer **Q1**.

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. This situation

may require a further alternative approach, such as managing input data in a stream database, which can handle continuous data streams.

5.1.2. Dynamic Adaptation Efficiency

Table 5 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	**Q2**	Is MoRE-WS efficient to carry out the dynamic adaptation of service compositions?
Metrics	**M6 - M10**	(**M6**) average execution time of the operations that are carried out by MoRE-WS for dynamic adaptation, (**M7**) average memory consumption of the operations that are carried out by MoRE-WS for dynamic adaptation, (**M8**) CPU consumption in a time frame, (**M9**) memory consumption in a time frame, and (**M10**) time frame

Table 5: GQM model for the "efficient dynamic adaptation of service compositions from MoRE-WS's viewpoint" goal.

In order to answer **Q2**, we carried out two experiments. In the first experiment, we measured the average execution time (**M6**) and the average memory consumption (**M7**) of the operations that are carried out by MoRE-WS to dynamically adapt the service composition to face one context condition. This context condition is triggered when a composite service is currently unavailable. In order to face this context condition, a resolution deactivates two Web service functionalities. In turn, this resolution activates three Web service functionalities. In the second experiment, we measured the CPU consumption (**M8**) and the memory consumption (**M9**) during dynamic adaptations for four context conditions. The problematic context events happened sequentially in a time frame of less than a minute (**M10**). In both experiments, we used the following files with models at runtime: a file with a feature model (23.0 kB); a file with a BPMN base composition model and BPMN 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 6 shows the summary of the average execution time in milliseconds and the average memory consumption in megabytes 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.

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 6: Summary of the dynamic adaptation results to answer **Q2**.

In the *Analyzing the Context* section of Table 6, 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 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)[12].

In the *Planning the Adaptation* section of Table 6, 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 increments and decrements in the composition model (these operations take

[12] `http://stax.codehaus.org`

a new configuration of the variability model as input, and they calculate the modifications to the composition model by adding or removing variant models). Finally, the "adapting the composition model" operation looks for the variation points that have to be rebound and does the necessary rebindings with BPMN 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 6, 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 recent servers.

Overall, the resulting execution times and memory consumptions in Table 6 demonstrate that MoRE-WS is efficient to carry out dynamic adaptations.

Results of the Second Experiment

Figure 9 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[13].

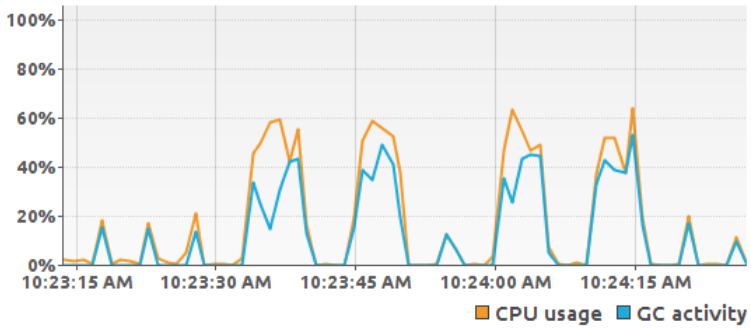

Figure 9: CPU consumption of MoRE-WS during four dynamic adaptations (line in orange).

The line in orange indicates the percentage of CPU consumption. The line in blue is for the Garbage Collector (GC). 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

[13] `http://visualvm.java.net`

peaks. During dynamic adaptations, the CPU consumption grows up to 60% for a few seconds.

In Figure 10, 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 Java virtual machine. Memory consumption is constant and low even during adaptations.

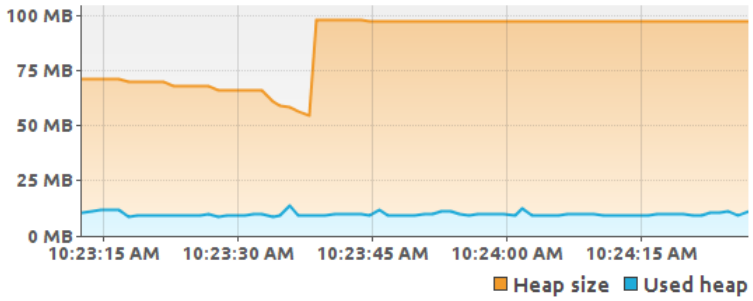

Figure 10: Memory consumption of MoRE-WS during four dynamic adaptations (line in blue).

5.1.3. Operability under Stress

Table 7 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 **Q3**, we manually injected four problematic context events into the file with the observations log (**M11**). These events are separated by very small time frames of less than one second (**M12**).

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 (**M13**). Then, MoRE-WS evaluated four context conditions that could be affected by these events (**M14**). Table 8 shows execution time (**M15**) and memory consumption (**M16**) 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.

Goal	**Purpose**	**Avoid**
	Issue	**saturation under stress circumstances of**
	Object	**MoRE-WS**
	Viewpoint	**from MoRE-WS's viewpoint**
Question	**Q3**	Does MoRE-WS have a good performance and memory consumption under stress circumstances?
Metrics	**M11 - M16**	(**M11**) number of problematic context events, (**M12**) time frame between problematic context events, (**M13**) frequency to observe the context, (**M14**) number of context conditions that could be affected by arising context events, (**M15**) execution time under stress circumstances, and (**M16**) memory consumption under stress circumstances

Table 7: GQM model for the "avoid saturation under stress circumstances of MoRE-WS from MoRE-WS's viewpoint" goal.

Context Condition	**Execution Time (ms)**	**Memory Consumption (MB)**
Unavailable Composite Service Operation δ	6,645	11.8
High Execution Time of Web Service Operation ϑ $\bigwedge$ Execution Time of Web Service Operation Φ Lower than the Execution Time of a Variant Web Service Operation	4,858	10.7
High Execution Time of Web Service Operation ψ $\bigwedge$ Execution Time of Web Service Operation Ω Lower than the Execution Time of a Variant Web Service Operation	5,138	10.7
High Execution Time of Web Service Operation Ψ	4,571	10.7

Table 8: Execution time and memory consumption during dynamic adaptations for four context events in a very tight time frame.

5.2. Validation in the Dynamic Evolution Phase

Table 9 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 **Q4**, 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	**Q4**	Are the Evolution Planner and MoRE-WS efficient to carry out dynamic evolutions?
Metrics	**M17 - M22**	(**M17**) average execution time of the operations that are carried out by the Evolution Planner to plan an evolution, (**M18**) average memory consumption of the operations that are carried out by the Evolution Planner to plan an evolution, (**M19**) average execution time of the operations that are carried out by MoRE-WS for dynamic evolution, (**M20**) average memory consumption of the operations that are carried out by MoRE-WS for dynamic evolution, (**M21**) overall CPU consumption, and (**M22**) overall memory consumption

Table 9: 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 (**M17**) and the average memory consumption (**M18**) of the operations that are carried out by the Evolution Planner to plan an evolution; and 2) the average execution time (**M19**) and the average memory consumption (**M20**) of the operations that are carried out by MoRE-WS to evolve the service composition. In this experiment, we triggered an unknown context event (non previously defined at design time). In the second experiment, we measured the overall CPU consumption (**M21**) and the overall memory consumption (**M22**) 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 a tactic implemented as a feature model (0.96 kB); a file with a tactic implemented as a composition model (2.18 kB); a file with the weaving model between the tactic implemented as a feature model, and the 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 shows the summary of the average execution time in milliseconds and the average memory consumption in megabytes for the operations that are carried out by the Evolution Planner and MoRE-WS during a dynamic evolution. This dynamic evolution is triggered by an injected event not previously defined at design time. We ran the same evolution three times and calculated the average of the measures in order to give results as accurate as possible.

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: Summary of the dynamic evolution results to answer **Q4**.

In Table 10, the results of memory consumption in all the operations were similar and very low. In the section about the Evolution Planner, 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 Eclipse Modeling Framework (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 [58]. This algorithm reduces the number 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, 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 composition model increments ($CM\triangle$) and composition model decrements ($CM\nabla$). 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 demonstrate that our proposed computer infrastructure is efficient to carry out dynamic evolutions.

Results of the Second Experiment

Figure 11 shows the percentage of CPU consumption of MoRE-WS during the dynamic evolution for an unknown context event. 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.

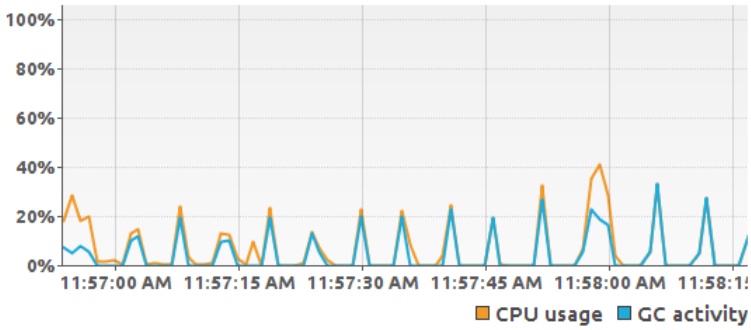

Figure 11: CPU consumption of MoRE-WS during a dynamic evolution around 11:58 AM (line in orange).

In Figure 12, 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.

6. Conclusions and Future Work

The present work has described a tool-supported framework to guide autonomic adjustments of context-aware service compositions in the closed and open worlds using models at runtime. We carried out the analysis of research works on autonomic service compositions. In this analysis we found a set of gaps: need for abstract mechanisms to guide dynamic adjustments, need for facing unanticipated context events in the open world, need for transparency, and need for

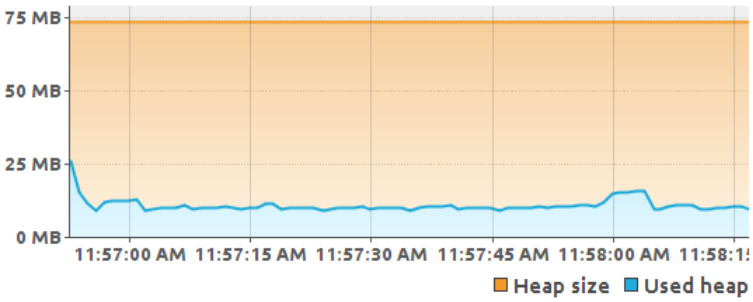

Figure 12: Memory consumption of MoRE-WS during a dynamic evolution around 11:58 AM (line in blue).

safe reconfigurations. Our approach tries to solve these gaps by means of the application of building blocks that are the basis for the *Design Phase*, the *Dynamic Adaptation Phase*, and the *Dynamic Evolution Phase* of our framework. The evaluation results show that the Context Monitor, MoRE-WS and the Evolution Planner, which are key components at runtime, are efficient during dynamic adaptations and dynamic evolutions.

As future work, we would like to use models at runtime to migrate running instances of the service composition. 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).

Also, 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. 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 data to make further decisions.

Acknowledgements

This work has been developed with the support of MINECO under the project SMART ADAPT TIN2013-42981-P and co-financed with ERDF.

References

[1] J.-Y. Hong, E.-H. Suh, S.-J. Kim, Context-aware systems: A literature review and classification, Expert Syst. Appl. 36 (2009) 8509–8522. doi:10.1016/j.eswa.2008.10.071.

[2] A. K. Dey, Understanding and using context, Personal Ubiquitous Comput. 5 (2001) 4–7.

[3] A. Schmidt, Ubiquitous computing - computing in context, Ph.D. thesis, Lancaster University (November 2002).
URL http://www.comp.lancs.ac.uk/~albrecht/phd/

[4] F. Fleurey, A. Solberg, 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, 2009, pp. 606–621.

[5] G. Blair, N. Bencomo, R. B. France, Models@ run.time, Computer 42 (2009) 22–27. doi:http://dx.doi.org/10.1109/MC.2009.326.

[6] H. Wang, X. Wang, X. Hu, X. Zhang, M. Gu, A multi-agent reinforcement learning approach to dynamic service composition, Information Sciences 363 (2016) 96 – 119. doi:http://dx.doi.org/10.1016/j.ins.2016.05.002.

[7] A. Moustafa, M. Zhang, Q. Bai, Trustworthy stigmergic service compositionand adaptation in decentralized environments, IEEE Transactions on Services Computing 9 (2) (2016) 317–329. doi:10.1109/TSC.2014.2298873.

[8] J. Yu, Q. Z. Sheng, J. K. Swee, J. Han, C. Liu, T. H. Noor, Model-driven development of adaptive web service processes with aspects and rules, Journal of Computer and System Sciences 81 (3) (2015) 533 – 552, special Issue on selected papers from the 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013). doi:http://dx.doi.org/10.1016/j.jcss.2014.11.008.

[9] P. Xie, Y. Song, Y. Wang, Y. Luo, Y. Zhang, A solution for web service composition based on logic-interface orchestration, in: Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on, 2015, pp. 555–560. doi:10.1109/CSCWD.2015.7231019.

[10] C. Lv, W. Jiang, S. Hu, J. Wang, G. Lu, Z. Liu, Efficient dynamic evolution of service composition, IEEE Transactions on Services Computing PP (99) (2015) 1–1. doi:10.1109/TSC.2015.2466544.

[11] B. Chen, X. Peng, Y. Yu, W. Zhao, Requirements-driven self-optimization of composite services using feedback control, IEEE Transactions on Services Computing 8 (1) (2015) 107–120. doi:10.1109/TSC.2014.2298866.

[12] J. A. Parejo, S. Segura, P. Fernandez, A. Ruiz-Cortés, Qos-aware web services composition using GRASP with path relinking, Expert Systems with Applications 41 (9) (2014) 4211 – 4223. doi:http://dx.doi.org/10.1016/j.eswa.2013.12.036.

[13] C. Wang, J. L. Pazat, A chemistry-inspired middleware for self-adaptive service orchestration and choreography, in: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, 2013, pp. 426–433. doi:10.1109/CCGrid.2013.51.

[14] M. Hussein, J. Han, J. Yu, A. Colman, Enabling runtime evolution of context-aware adaptive services, in: Services Computing (SCC), 2013 IEEE International Conference on, 2013, pp. 248–255. doi:10.1109/SCC.2013.77.

[15] J. Cubo, N. Gamez, L. Fuentes, E. Pimentel, Composition and Self-Adaptation of Service-Based Systems with Feature Models, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 326–342.

[16] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, G. Papadopoulos, A development framework and methodology for self-adapting applications in ubiquitous computing environments, Journal of Systems and Software 85 (12) (2012) 2840 – 2859.

[17] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, R. Mirandola, MOSES: A framework for QoS driven runtime adaptation of service-oriented systems, Software Engineering, IEEE Transactions on 38 (5) (2012) 1138–1159.

[18] L. Baresi, S. Guinea, Self-supervising BPEL processes, IEEE Trans. Softw. Eng. 37 (2011) 247–263.

[19] D. Menasce, H. Gomaa, S. Malek, J. Sousa, SASSY: A framework for self-architecting service-oriented systems, IEEE Software 28 (2011) 78–85.

[20] X. Franch, P. Grunbacher, M. Oriol, B. Burgstaller, D. Dhungana, L. Lopez, J. Marco, J. Pimentel, 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, 2011, pp. 458–463.

[21] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, G. Tamburrelli, Dynamic QoS management and optimization in service-based systems, IEEE Transactions on Software Engineering 37 (2011) 387–409.

[22] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, Adaptive management of composite services under percentile-based service level agreements, in: P. Maglio, M. Weske, J. Yang, M. Fantinato (Eds.), Service-Oriented Computing, Vol. 6470 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 381–395.

[23] J. Lee, G. Kotonya, Combining service-orientation with product line engineering, IEEE Softw. 27 (2010) 35–41.

[24] M. Koning, C.-a. Sun, M. Sinnema, P. Avgeriou, VxBPEL: Supporting variability for web services in BPEL, Inf. Softw. Technol. 51 (2009) 258–269.

[25] D. Karastoyanova, F. Leymann, BPEL'n'Aspects: Adapting service orchestration logic, in: Proceedings of the 2009 IEEE International Conference on Web Services, ICWS '09, 2009, pp. 222–229.

[26] M. Sonntag, D. Karastoyanova, 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, 2011, pp. 1–16.

[27] C. Parra, X. Blanc, L. Duchien, 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, 2009, pp. 131–140.

[28] O. Moser, F. Rosenberg, S. Dustdar, 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, 2008, pp. 815–824.

[29] A. Mosincat, W. Binder, Transparent runtime adaptability for BPEL processes, in: Proceedings of the 6th International Conference on Service-Oriented Computing, ICSOC '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 241–255.

[30] G. Canfora, M. D. Penta, R. Esposito, M. L. Villani, A framework for QoS-aware binding and re-binding of composite web services, Journal of Systems and Software 81 (10) (2008) 1754 – 1769.

[31] A. Charfi, M. Mezini, AO4BPEL: An aspect-oriented extension to BPEL, World Wide Web 10 (3) (2007) 309–344.

[32] O. Ezenwoye, S. Sadjadi, RobustBPEL2: transparent autonomization in business processes through dynamic proxies, in: Proceedings of the 8th International Symposium on Autonomous Decentralized Systems, ISADS '07, 2007, pp. 17–24.

[33] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, P. Plebani, PAWS: a framework for executing adaptive web-service processes, IEEE Softw. 24 (6) (2007) 39–46.

[34] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, Software Engineering, IEEE Transactions on 33 (6) (2007) 369–384.

[35] M. Colombo, E. Di Nitto, M. Mauri, SCENE: A service composition execution environment supporting dynamic changes disciplined through rules, in: A. Dan, W. Lamersdorf (Eds.), Service-Oriented Computing – ICSOC 2006, Vol. 4294 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 191–202.

[36] R. Berbner, M. Spahn, N. Repp, O. Heckmann, R. Steinmetz, 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, 2006, pp. 72–82.

[37] I. Bosloper, J. Siljee, J. Nijhuis, D. Hammer, 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, 2005, pp. 95–104.

[38] J. Siljee, I. Bosloper, J. Nijhuis, D. Hammer, DySOA: making service systems self-adaptive, in: Proceedings of the 3rd international conference on service-oriented computing, ICSOC '05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 255–268.

[39] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, QoS-aware middleware for web services composition, IEEE Trans. Softw. Eng. 30 (5) (2004) 311–327.

[40] U. Aßmann, N. Bencomo, B. H. C. Cheng, R. B. France, Models@run.time (dagstuhl seminar 11481), Dagstuhl Reports 1 (11) (2011) 91–123.
URL `http://drops.dagstuhl.de/opus/volltexte/2012/3379/`

[41] S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid, Dynamic software product lines, Computer 41 (2008) 93–95. doi:http://doi.ieeecomputersociety.org/10.1109/MC.2008.123.

[42] J. Buckley, T. Mens, M. Zenger, A. Rashid, G. Kniesel, Towards a taxonomy of software change, Journal of Software Maintenance 17 (5) (2005) 309–332.

[43] J. Andersson, R. Lemos, S. Malek, D. Weyns, Software engineering for self-adaptive systems, Springer-Verlag, Berlin, Heidelberg, 2009, Ch. Modeling Dimensions of Self-Adaptive Software Systems, pp. 27–47.

[44] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, D. Diaz, Dynamic adaptation of service compositions with variability models, Journal of Systems and Software 91 (2014) 24 – 47.

[45] G. H. Alférez, V. Pelechano, 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, 2011, pp. 100–109.

[46] L. Liu, E. Yu, Designing information systems in social context: a goal and scenario modelling approach, Inf. Syst. 29 (2004) 187–203.

[47] G. H. Alférez, V. Pelechano, Dynamic evolution of context-aware systems with models at runtime, in: R. France, J. Kazmeier, R. Breu, C. Atkinson (Eds.), Model Driven Engineering Languages and Systems, Vol. 7590 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2012, pp. 70–86.

[48] G. H. Alférez, V. Pelechano, Facing uncertainty in web service compositions, in: Proceedings of the 20th IEEE International Conference on Web Services, ICWS '13, 2013, pp. 219–226. doi:10.1109/ICWS.2013.38.

[49] G. H. Alférez, V. Pelechano, Facing uncertainty in web service compositions, International Journal of Services Computing 2 (2) (2014) 1–16.

[50] L. A. Belady, M. M. Lehman, A model of large program development, IBM Syst. J. 15 (3) (1976) 225–252.

[51] P. Horn, Autonomic computing: IBM's perspective on the state of information technology (2001).
URL `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf`

[52] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, J. Whittle, Software engineering for self-adaptive systems, Springer-Verlag, Berlin, Heidelberg, 2009, Ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

[53] R. Calinescu, C. Ghezzi, M. Kwiatkowska, R. Mirandola, Self-adaptive software needs quantitative verification at runtime, Commun. ACM 55 (9) (2012) 69–77.

[54] OMG, Software & systems process engineering meta-model specification (2008).
URL http://www.omg.org/spec/SPEM/2.0/PDF

[55] IBM, An architectural blueprint for autonomic computing, Tech. rep., IBM (2006).
URL http://www.eecs.harvard.edu/~chaki/bib/papers/autonomic.pdf

[56] N. Esfahani, E. Kouroshfar, S. Malek, 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, 2011, pp. 234–244.

[57] V. R. Basili, G. Caldiera, R. H. Dieter, Goal question metric paradigm, in: Encyclopedia of Software Engineering, Vol. 2, John Wiley & Sons, Inc., 1994, pp. 528–532.
URL http://www.cs.umd.edu/~basili/publications/technical/T89.pdf

[58] C. L. Forgy, Rete: A fast algorithm for the many pattern/many object pattern match problem, Artificial Intelligence 19 (1) (1982) 17 – 37.