# Reconfiguración Dinámica e Incremental de Arquitecturas de Servicios Cloud Dirigida por Modelos

*Miguel Ángel Zúñiga Prieto*

**Director:**
Dr. Emilio Insfrán Pelozo

Julio 2017

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

**Departamento de Sistemas Informáticos y Computación**


UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

Tesis Doctoral depositada en cumplimiento parcial de los requisitos para el grado de Doctor en Informática

# Reconfiguración Dinámica e Incremental de Arquitecturas de Servicios Cloud Dirigida por Modelos

**Miguel Ángel Zúñiga Prieto**

Director:

**Dr. Emilio Insfrán Pelozo**

Valencia, julio de 2017

**Diseño de la Portada:** Angel Humberto Guapisaca Vargas

| | |
|---|---|
| Título: | Reconfiguración Dinámica e Incremental de Arquitecturas de Servicios Cloud Dirigida por Modelos |
| Presentado por: | Miguel Ángel Zúñiga Prieto |
| Director: | Dr. Emilio Insfrán Pelozo |
| Institución: | Universitat Politècnica de València (UPV) |
| Departamento: | Sistemas Informáticos y Computación (DSIC) |
| Programa de doctorado: | Doctor en Informática |
| Fecha de envío: | 28 de abril de 2017 |
| Fecha de defensa: | 25 de julio de 2017 |
| Revisores externos: | Dr. Juan Manuel Murillo Rodríguez (Universidad de Extremadura)<br>Dra. Jennifer Pérez Benedí (Universidad Politécnica de Madrid)<br>Dra. María José Escalona Cuaresma (Universidad de Sevilla) |
| Tribunal: | Presidente:<br>Dr. Juan Manuel Murillo Rodríguez (Universidad de Extremadura)<br>Secretario:<br>Dr. Carlos Cuesta Quintero (Universidad Rey Juan Carlos)<br>Vocal:<br>Dr. Juan de Lara Jaramillo (Universidad Autónoma de Madrid) |

# Agradecimientos

Este es el final de un largo camino lleno de obstáculos, no siendo posible recorrerlo solo. A continuación, se citan a todas las personas que contribuyeron de alguna forma en este trabajo.

A Emilio por su confianza, esfuerzo y apoyo durante todos estos años.

A Silvia por sus siempre sabios consejos y colaboración en la validación de este trabajo.

Al equipo de trabajo del proyecto Value@Cloud: Marta, Fernando, Silvia, Emilio, Ana, Joao, Eric, Javier[2], Carlos, y Raphael, quienes han aportado sus puntos de vista para mejorar este trabajo.

A Javier por sus contribuciones aportadas a esta tesis.

A Carlos, Daniel y Juan por sus contribuciones en la construcción de la infraestructura software.

A los compañeros del grupo ISSI, por todos los momentos que hicieron ameno todo este tiempo y por su inestimable colaboración en los experimentos (David, Fernando, Vicente, Julio, Kamil, Patricia, Abel).

A todos los integrantes de los grupos de investigación que participaron en las sesiones experimentales.

A mi esposa Priscila, a nuestras hijas Alejandrita y María Emilia, y a mis padres por todos sus sacrificios y por ser mi apoyo constante.

A mi país, al Programa de becas de la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación (SENESCYT) de Ecuador y a la Universidad de Cuenca-Ecuador.

Y por último a todos aquellos que de un modo u otro han contribuido a hacer posible este proyecto.

# Resumen

La computación cloud representa un cambio fundamental en la manera en la que las organizaciones adquieren recursos tecnológicos (p. ej., hardware, entornos de desarrollo y ejecución, aplicaciones); en donde, en lugar de comprarlos adquieren acceso remoto a ellos en forma de servicios cloud suministrados a través de Internet. Entre las principales características de la computación cloud está la asignación de recursos de manera ágil y elástica, reservados o liberados dependiendo de la demanda de los usuarios o aplicaciones, posibilitando el modelo de pago basado en métricas de consumo. Con respecto a las aplicaciones cloud, éstas se componen de servicios (generalmente Servicios Web) desplegados en uno o varios proveedores cloud, los cuales son seleccionados dependiendo de las necesidades específicas de cada servicio, haciendo uso de sus recursos.

El desarrollo de aplicaciones cloud sigue mayoritariamente un enfoque incremental, en donde la entrega incremental de funcionalidades al cliente cambia – o reconfigura – sucesivamente la arquitectura actual de la aplicación. Los proveedores cloud tienen sus propios estándares tanto para las tecnologías de implementación como para los mecanismos de gestión de servicios, requiriéndose soluciones que faciliten: la construcción, integración y despliegue de servicios portables; la interoperabilidad entre servicios desplegados en diferentes proveedores cloud; y la continuidad en la ejecución de la aplicación mientras su arquitectura es reconfigurada producto de la integración de los sucesivos incrementos.

Los principios del enfoque de desarrollo dirigido por modelos, del estilo arquitectónico de arquitecturas orientadas a servicios y de la reconfiguración dinámica cumplen un papel importante en este contexto. Estos principios ofrecen mecanismos que permiten afrontar los retos de integración, portabilidad, interoperabilidad y continuidad de servicios desde un nivel de abstracción más elevado, permitiendo la sistematización y automatización del proceso de desarrollo.

La hipótesis de esta tesis doctoral es que los métodos de desarrollo dirigido por modelos brindan a los desarrolladores de servicios cloud mecanismos de abstracción y automatización para la aplicación sistemática de los principios de la ingeniería de modelos durante el diseño, implementación y despliegue incremental de servicios cloud, facilitando la reconfiguración dinámica de la arquitectura orientada a servicios de las aplicaciones cloud.

El objetivo principal de esta tesis doctoral es por tanto definir y validar empíricamente DIARy, un método de reconfiguración dinámica e incremental de arquitecturas orientadas a servicios. Este método permitirá especificar la integra-

ción arquitectónica del incremento con la aplicación cloud actual, y con esta información automatizar la derivación de los artefactos de implementación que faciliten la integración y reconfiguración dinámica de la arquitectura de servicios de la aplicación cloud. Esta reconfiguración dinámica se consigue al ejecutar los artefactos de reconfiguración que no solo despliegan/repliegan los servicios del incremento y servicios de orquestación entre los servicios del incremento con los servicios de la aplicación cloud actual; sino también, cambian en tiempo de ejecución los enlaces entre servicios.

También se ha diseñado e implementado una infraestructura software que soporta las actividades del método propuesto e incluye los siguientes componentes: i) un conjunto de DSLs, con sus respectivos editores gráficos, que permiten describir aspectos relacionados a la integración arquitectónica, implementación y aprovisionamiento de incrementos en entornos cloud; ii) transformaciones que generan modelos de implementación y aprovisionamiento específicos de la plataforma a partir de los modelos de integración de alto nivel; iii) transformaciones que generan artefactos que implementan la lógica de integración y orquestación de servicios, y scripts de aprovisionamiento, despliegue y reconfiguración dinámica específicos para distintos proveedores cloud (Microsoft Azure©, Google Cloud Platform, y Cloudify), tanto en un modelo de servicio IaaS como PaaS.

El método DIARy ha sido validado empíricamente mediante un experimento para evaluar la facilidad de uso, utilidad percibida, intensión de uso y rendimiento con resultados satisfactorios y prometedores.

Esta tesis doctoral contribuye al campo de las arquitecturas orientadas a servicios y en particular a la reconfiguración dinámica de arquitecturas de servicios cloud en contextos de desarrollo iterativo e incremental. El principal aporte es un método bien definido, basado en los principios del desarrollo dirigido por modelos, que facilita elevar el nivel de abstracción y automatizar por medio de transformaciones la generación de artefactos que realizan la reconfiguración dinámica de aplicaciones cloud.

# Abstract

Cloud computing represents a fundamental change in the way organizations acquire technological resources (e.g., hardware, development and execution environments, applications); where, instead of buying them, they acquire remote access to them in the form of cloud services supplied through the Internet. Among the main characteristics of cloud computing is the allocation of resources in an agile and elastic way, reserved or released depending on the demand of the users or applications, enabling the payment model based on consumption metrics. With regard to cloud applications, these are composed of services (usually Web Services) deployed in one or several cloud providers, which are chosen depending on the specific needs of each service and making use of its resources.

The development of cloud applications mostly follows an incremental approach, where the incremental delivery of functionalities to the client changes - or reconfigures - successively the current architecture of the application. Cloud providers have their own standards for both implementation technologies and service management mechanisms, requiring solutions that facilitate: building, integrating and deploying portable services; interoperability between services deployed across different cloud providers; and continuity In the execution of the application while its architecture is reconfigured product of the integration of the successive increments.

The principles of the model-driven development approach, the architectural style *service-oriented architectures*, and the dynamic reconfiguration play an important role in this context. These principles offer mechanisms that allow addressing the integration, portability, interoperability, and continuity of services from a higher level of abstraction, allowing the systematization and automation of the development process.

The hypothesis of this doctoral thesis is that model-driven development methods provide cloud service developers with abstraction and automation mechanisms for the systematic application of the principles of model engineering during the design, implementation, and incremental deployment of cloud services, facilitating the dynamic reconfiguration of the service-oriented architecture of cloud applications.

The main objective of this doctoral thesis is therefore to define and validate empirically DIARy, a method of dynamic and incremental reconfiguration of service-oriented architectures for cloud applications. This method will allow specifying the architectural integration of the increment with the current cloud

application, and with this information to automate the derivation of implementation artifacts that facilitate the integration and dynamic reconfiguration of the service architecture of the cloud application. This dynamic reconfiguration is achieved by running reconfiguration artifacts that not only deploy / un-deploy increment's services and orchestration services between services of the increment with the services of the current cloud application; but also, they change the links between services at runtime.

A software infrastructure that supports the activities of the proposed method has also been designed and implemented. The software infrastructure includes the following components: i) a set of DSLs, with their respective graphical editors, that allow to describe aspects related to the architectural integration, implementation and provisioning of increments in cloud environments; ii) transformations that generate platform-specific implementation and provisioning models based on high-level integration models; (iii) transformations that generate artifacts that implement integration logic and orchestration of services, and scripts of provisioning, deployment, and dynamic reconfiguration for different cloud vendors (Microsoft Azure ©, Google Cloud Platform, and Cloudify), both in a IaaS and PaaS service models.

The DIARy method has been empirically validated through an experiment to evaluate the perceived ease of use, perceived usefulness, intention to use and performance with satisfactory and promising results.

This doctoral thesis contributes to the field of service-oriented architectures and in particular to the dynamic reconfiguration of cloud services architectures in an iterative and incremental development context. The main contribution is a well-defined method, based on the principles of model-driven development, which makes it easy to raise the level of abstraction and automate, through transformations, the generation of artifacts that perform the dynamic reconfiguration of cloud applications.

# Resum

La computació cloud representa un canvi fonamental en la manera en què les organitzacions adquirixen recursos tecnològics (ej., maquinari, entorns de desplegament i execució, aplicacions) ; on, en compte de comprar-los adquirixen accés remot a ells en forma de servicis cloud subministrats a través d'Internet. Entre les principals característiques de la computació cloud els recursos cloud són assignats de manera àgil i elàstica, reservats o alliberats depenent de la demanda dels usuaris o aplicacions, possibilitant el model de pagament basat en mètriques de consum. Respecte a les aplicacions cloud, estan compostes per servicis, generalment Servicis Web, desplegats en un o més proveïdors cloud seleccionats depenent de les necessitats específiques de cada servici, fent ús dels seus recursos.

El desenrotllament d'aplicacions cloud seguix majoritàriament un enfocament incremental, on l'entrega incremental de funcionalitats al client canvia - o reconfigura - successivament l'arquitectura actual de l'aplicació. Els proveïdors cloud tenen els seus propis estàndards tant per a les tecnologies d'implementació com per als mecanismes de gestió de servicis, requerint-se solucions que faciliten: la construcció, integració i desplegament de servicis portables; la interoperabilitat entre servicis desplegats en diferents proveïdors cloud; i la continuïtat en l'execució de l'aplicació mentres la seua arquitectura és reconfigurada producte de la integració dels successius increments.

Els principis de l'enfocament de desenrotllament dirigit per models, de l'estil arquitectònic d'arquitectures orientades a servicis i de la reconfiguració dinàmica complixen un paper important en este context. Aquests principis oferixen mecanismes que permeten afrontar els reptes d'integració, portabilitat, interoperabilitat i continuïtat de servicis des d'un nivell d'abstracció més elevat, permetent la sistematització i automatització del procés de desenrotllament.

La hipòtesi d'esta tesi doctoral és que els mètodes de desenrotllament dirigit per models brinden als desenvolupadors de servicis cloud mecanismes d'abstracció i automatització per a l'aplicació sistemàtica dels principis de l'enginyeria de models durant el disseny, implementació i desplegament incremental de servicis cloud, facilitant la reconfiguració dinàmica de l'arquitectura orientada a servicis de les aplicacions cloud.

L'objectiu principal d'esta tesi doctoral és per tant de definir i validar empíricamente DIARy, un mètode de reconfiguració dinàmica i incremental d'arquitectures orientades a servicis per a aplicacions cloud. Este mètode permetrà especificar la integració arquitectònica de l'increment amb l'aplicació

cloud actual, i amb esta informació automatitzar la derivació dels artefactes d'implementació que faciliten la integració i reconfiguració dinàmica de l'arquitectura de servicis de l'aplicació cloud. Esta reconfi-guración dinàmica s'aconseguix a l'executar els artefactes de reconfiguració que no sols despleguen/repleguen els servicis de l'increment i servicis d'orquestració entre els servicis de l'increment amb els servicis de l'aplicació cloud actual; sinó també, canvien en temps d'execució els enllaços entre servicis.

També s'ha dissenyat i implementat una infraestructura programari que suporta les activitats del mètode proposat i inclou els següents components: i) un conjunt de DSLs, amb els seus respectius editors gràfics, que permeten descriure aspectes relacionats a la integració arquitectònica, implementació i aprovisionament en entorns cloud dels increments; ii) transformacions que generen models d'implementació i aprovisionament específics de la plataforma a partir dels models d'integració d'alt nivell; iii) transformacions que generen artefactes que implementen la lògica d'integració i orquestració de servicis, i scripts d'aprovisionament, desplegament i reconfiguració dinàmica específics per a distints proveïdors cloud (Microsoft Azure©, Google Cloud Platform, i Cloudify; tant en un model de servici IaaS com PaaS).

El mètode DIARy ha sigut validat empíricament per mitjà d'un experiment per a avaluar la facilitat d'ús, utilitat percebuda, intensió d'ús i rendiment amb resultats satisfactoris i prometedors.

Esta tesi doctoral contribuïx al camp de les arquitectures orientades a servicis i en particular a la reconfiguració dinàmica d'arquitectures de servicis cloud en contextos de desenrotllament iteratiu i incremental. La principal aportació és un mètode ben definit, basat en els principis del desenrotllament dirigit per models, que facilita elevar el nivell d'abstracció i automatitzar per mitjà de transformacions la generació d'artefactes que realitzen la reconfiguració dinàmica d'aplicacions cloud.

# Palabras Clave

**Palabras Clave:** Arquitecturas Orientada a Servicios, Reconfiguración Dinámica de Arquitecturas, Arquitecturas Cloud, Servicios Cloud, Software como Servicio, Desarrollo Incremental, Desarrollo de Software Dirigido por Modelos, Lenguajes de Descripción de Arquitecturas, Despliegue Multi-cloud, Orquestación de Servicios.

**Key Words:** Service Oriented Architectures, Dynamic Architecture Reconfiguration, Cloud Architectures, Cloud Services, Software as a Service, Incremental Development, Model-Driven Software Development, Architecture Description Languages, Multi-cloud Deployment, Service Orchestration.

**Paraules Clau:** Arquitectures Orientades a Servicis, Reconfiguració Dinàmica d'Arquitectures, Arquitectures Cloud, Servicis Cloud, Programari com a Servici, Desenrotllament Incremental, Desenrotllament de Programari Dirigit per Models, Llenguatges de Descripció d'Arquitectures, Desplegament Multi-cloud, Orquestració de Servicis.

# Contenido

# Índice de Figuras

# Índice de Tablas

# Índice de Listados

# Acrónimos

| | |
|---|---|
| ADL | Architecture Description Language / Lenguaje de Descripción Arquitectónica |
| ATL | Atlas Transformation Language |
| API | Application Program Interface / Interfaces de Programación de Aplicaciones |
| AWS | Amazon Web Services |
| BPEL | Business Process Execution Language |
| CD | Continuous Delivery / Entrega Continua |
| CIM | Computation Independent Model / Modelo Independiente de Computación |
| CPU | Central Processing Unit |
| DSDM | Desarrollo de Software Dirigido por Modelos |
| DSL | Domain Specific Language / Lenguaje Específico de Dominio |
| EMF | Eclipse Modeling Framework |
| GPL | General Purpose Languages / Lenguajes de Propósito General |
| IaaS | Infrastructure as a Service / Infraestructura como Servicio |
| IDE | Integrated Development Environment / Entorno de Desarrollo Integrado |
| IDL | Interface Definition Language / Lenguaje de Definición de Interface |
| IT | Information Technology |
| JAR | Java Archive |
| MEM | Method Evaluation Model / Método para la Evaluación de Modelos |
| MDA | Model Driven Architecture / Arquitectura Dirigida por Modelos |
| MDE | Model Driven Engineering / Ingeniería Basada en Modelos |
| MOF | Meta Object Facility |
| M2M | Model to Model / Modelo a Modelo |
| M2T | Model to Text / Modelo a Texto |
| NIST | National Institute of Standards and Technology |
| OCL | Object Constraint Language |
| OMG | Object Management Group |
| PaaS | Platform as a Service / Plataforma como Servicio |
| PHP | Hypertext Preprocessor |
| PIM | Platform Independent Model / Modelo Independiente de Plataforma |
| PSM | Platform Specific Model / Modelo Específico de Plataforma |

| | |
|---|---|
| QoS | Quality of Service / Calidad de Servicio |
| QVT | Query/View/Transformation |
| RPC | Remote Procedure Call / Llamada a Procedimiento Remoto |
| SaaS | Software as a Service / Software como Servicio |
| SBA | Service-based Applications / Aplicaciones Basadas en Servicios |
| SLA | Service Level Agreement / Acuerdo de Nivel de Servicio |
| SOA | Service Oriented Architecture / Arquitectura Orientada a Servicios. |
| SoaML | Service oriented architecture Modeling Language |
| SOAP | Simple Object Access Protocol / Protocolo Simple de Acceso a Objetos |
| SOC | Service Oriented Computing / Computación Orientada a Servicios |
| SPEM | Software & Systems Process Engineering Metamodel |
| TAM | Technology Acceptance Model / Modelo de Aceptación Tecnológica |
| UUDI | Universal Description, Discovery, and Integration / Descripción Universal, Descubrimiento e Integración |
| UML | Unified Modeling Language / Lenguaje Unificado de Modelado |
| URI | Unique Resource Identifier / Identificador de Recursos Uniforme |
| URL | Uniform Resource Locator |
| WCF | Windows Communication Foundation |
| WSDL | Web Service Description Language / Lenguaje de Descripción de Servicios Web |
| W3C | World Wide Web Consortium |
| XDT | XML Document Transform |
| XMI | XML Metadata Interchange |
| XML | eXtensible Markup Language / Lenguaje de Marcas Extensible |

# Capítulo 1. Introducción

En este capítulo, en primer lugar, se contextualiza el trabajo de investigación desarrollado en esta tesis. En segundo lugar, se plantea el problema a resolver, se establecen los objetivos, se recogen las contribuciones de la misma y, por último, se describe la metodología de investigación:

La sección 1.1 introduce al desarrollo incremental de servicios cloud.

La sección 1.2 introduce los aspectos relacionados con la reconfiguración dinámica de arquitecturas en el desarrollo incremental de servicios cloud.

La sección 1.3 plantea el problema que se aborda en la presente tesis doctoral: la definición y evaluación empírica de un método para la reconfiguración dinámica e incremental de arquitecturas de servicios cloud.

La sección 1.4 describe los objetivos a desarrollar, así como las hipótesis que se pretende contrastar.

La sección 1.5 describe el contexto en el que se ha desarrollado el trabajo de investigación que constituye esta tesis doctoral.

La sección 1.6 describe la metodología de investigación aplicada y las técnicas previstas para validar empíricamente la aplicabilidad del método propuesto.

Por último, la sección 1.7 describe la estructura del documento.

## 1.1 Desarrollo incremental de servicios cloud

Con el fin de ser competitivos en la economía actual las organizaciones requieren ofrecer servicios de negocio de forma rápida y ágil, además de garantizar un nivel apropiado de calidad, coste y flexibilidad que satisfaga las cambiantes necesidades de sus clientes. La computación cloud es un modelo de negocio que representa un cambio fundamental en la manera en la que las organizaciones adquieren recursos tecnológicos, en donde en lugar de comprar hardware o software adquieren acceso remoto a ellos a través de Internet. En este modelo de negocio los recursos tecnológicos son asignados de manera ágil y elástica (reservados dependiendo de la demanda y liberados cuando no son necesarios); reduciendo el exceso o el bajo aprovisionamiento de recursos y posibilitando el modelo de pago basado en métricas de consumo (pago por uso).

A pesar de que existen diferentes definiciones de computación cloud, la definición provista por el National Institute of Standards and Technology (NIST[1]) (Mell and Grance, 2011) es considerada la más completa y precisa, ésta describe formalmente la computación cloud como:

> "*Un modelo que permite el acceso ubicuo, adaptado y bajo demanda a través de una red a un conjunto compartido de recursos de computación configurables (p. ej., redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser aprovisionados o liberados rápidamente, con un mínimo esfuerzo de gestión o interacción por parte del prestador del servicio*".

El paradigma de computación cloud se basa en servicios, por lo tanto, los recursos tecnológicos son suministrados como servicios: *servicios cloud*. Los proveedores de computación cloud ofrecen servicios cloud en tres niveles. i) *Infraestructura como Servicio* (Infrastructure as a Service – IaaS) que hace referencia al suministro de hardware virtualizado: servidores, recursos de red, almacenamiento, u otros recursos computacionales. ii) *Plataforma como Servicio* (Platform as a Service – PaaS) que hace referencia al suministro de plataformas de desarrollo y entornos de ejecución sobre los cuales los consumidores del servicio despliegan su propio software. y iii) *Software como Servicio* (Software as a Service – SaaS) que hace referencia al suministro de una aplicación como un servicio sobre Internet, en donde los usuarios compran el acceso a la aplicación en lugar de comprar licencias para

[1] NIST es una agencia de la Administración de Tecnología del Departamento de Comercio de los Estados Unidos que promueve la innovación y la competencia industrial mediante avances en metrología, normas y tecnología.

usarla localmente. Desde el punto de vista de ingeniería de software, un software provisto como servicio es una aplicación cloud (Hamdaqa y otros, 2011).

Las aplicaciones cloud son sistemas de software basados en servicios, generalmente Servicios Web. En la actualidad las aplicaciones empresariales están constituidas por servicios desplegados ya sea en la intranet de las organizaciones o en soluciones PaaS o IaaS, resultando en entornos cloud híbridos (Hajjat y otros, 2010). De acuerdo a un reporte especial de Gartner (Rivera and Meulen, 2013), para finales del 2017 la mitad de las organizaciones grandes adoptarán el uso de modelos cloud híbridos; lo cual no solo implica que para ese año la mitad de organizaciones grandes habrán construido aplicaciones cloud, sino que refleja la importancia que está tomando este tipo de aplicaciones.

Desde el punto de vista del proceso de desarrollo de software, los enfoques de desarrollo agiles se alinean al paradigma de computación cloud debido a que combinan entregas rápidas de aplicaciones con frecuentes retroalimentaciones por parte del usuario (Talreja, 2010). Siguiendo estos enfoques las aplicaciones cloud se organizan en un conjunto de incrementos compuestos por uno o más servicios que entregan nuevas funcionalidades al cliente o actualizan funcionalidades existentes. Cada incremento es diseñado, construido, desplegado e integrado separadamente, de acuerdo a una priorización previamente establecida; evitando los largos tiempos que podría requerir desarrollar una aplicación completa y favoreciendo a la agilidad del negocio. Adicionalmente, durante el despliegue de los servicios incluidos en un incremento, los desarrolladores seleccionan un conjunto de recursos tecnológicos ofrecidos por proveedores de soluciones IaaS o PaaS, sobre los cuales se desplegarán y ejecutarán.

En el contexto de este trabajo utilizamos: el término *servicios cloud* para referirnos a aplicaciones cloud, el término *recursos cloud* para referirnos a los recursos tecnológicos que un servicio cloud (aplicación) consume de los proveedores de soluciones IaaS y PaaS, y el término *proveedores cloud* para referirnos a los proveedores de soluciones IaaS y PaaS sobre los cuales se desplegará y ejecutará un servicio cloud.

## 1.2 Reconfiguración dinámica de arquitecturas en el desarrollo incremental de servicios cloud

Las decisiones de diseño establecen restricciones para las actividades de desarrollo de software posteriores, conduciendo a la producción de aplicaciones que satisfacen tanto requisitos funcionales como de calidad; en donde, la arquitectura de la aplicación es el artefacto de diseño que facilita la evaluación temprana de

la calidad de la aplicación a ser desarrollada. La arquitectura de una aplicación representa los conceptos o propiedades de un sistema en su entorno, plasmadas en su estructura (elementos y conexiones fundamentales), y en los principios de su diseño y evolución (IEEE., 2000).

Con respecto al término arquitectura, en el contexto de computación en la nube, debemos diferenciar la *Arquitectura de Entornos Cloud* de la *Arquitectura de Aplicaciones Cloud* (Arquitectura de Servicios Cloud). Las primeras se refieren a la organización conceptual de alto nivel de la plataforma de computación en la nube, sin tener en cuenta las tecnologías subyacentes; mientras que las segundas se refieren a la estructura de la aplicación que se ejecutará haciendo uso de recursos cloud. En este trabajo de tesis el término arquitectura se refiere a las Arquitecturas de Servicios Cloud. Las Arquitecturas de Servicios Cloud generalmente siguen los principios de diseño propuestos en el estilo arquitectónico *Arquitectura Orientada a Servicios* (Erl, 2005) (Service Oriented Architecture – SOA). Este estilo arquitectónico aboga por una mayor flexibilidad, agilidad y reutilización de las aplicaciones; sus principios de diseño proveen a los entornos cloud de un modelo de servicios ágil y escalable con la habilidad para adaptar eficientemente el aprovisionamiento de recursos cloud a las demandas dinámicas de los usuarios en Internet (Kyriazis y otros, 2014)(Infosys, 2011). SOA es un enfoque clave en el desarrollo de aplicaciones cloud, facilitando el diseño, construcción, despliegue e integración servicios de manera independiente de la plataforma computacional en la que se ejecutan (Laplante y otros, 2008), (Tsai y otros, 2010). Una aplicación cloud que sigue los principios SOA está organizada en servicios desplegados en diferentes proveedores cloud cuya interoperabilidad se logra mediante la orquestación de servicios (Parameswaran y Chaddha, 2009).

Las aplicaciones necesitan evolucionar, los cambios pueden ocurrir en el contexto en el cual están siendo utilizadas, en componentes externos provistos por terceros, o en sus requisitos. De acuerdo al estándar ISO/IEC 14764 y a (Lehman, 1996) el propósito de los cambios de software puede ser categorizado en 4 tipos: *correctivo*, *perfectivo*, *adaptativo* y *preventivo*. Según (Zhang y otros, 2012), desde el punto de vista arquitectónico se tiene que: los cambios *correctivos* corregirán errores identificados en la arquitectura; los cambios *perfectivos* alteran el código de elementos arquitectónicos (p. ej., componentes/servicios, conectores, etc.) para mejorar atributos no funcionales; los cambios *adaptativos* adaptan los elementos arquitectónicos para satisfacer cambios ya sea en el entorno o requisitos; y finalmente en los cambios *preventivos* previenen los efectos secundarios de otros cambios. De lo expuesto anteriormente se puede deducir que, en el desarrollo incremental de aplicaciones, la integración de incrementos de software en una aplicación producirá en su arquitectura cambios de tipo adaptativo. Los

cambios adaptativos pueden estar relacionados con la adición de nuevas funcionalidades, con la actualización de funcionalidades existentes, o con la mejora en la calidad de servicio (Quality of Service – QoS), (La Manna, 2011).

Cuando un incremento de software es integrado en una aplicación, sus elementos arquitectónicos son incorporados en la arquitectura actual de la aplicación, cambiando su estructura y comportamiento (reconfigurándola). Si la integración se gestiona fuera de línea, toda la aplicación debe ser detenida, reconfigurada, y reiniciada. Con el objetivo de gestionar la integración mientras la aplicación se encuentra en ejecución se aplican enfoques de reconfiguración dinámica de arquitecturas, los cuales actualizan ya sea la orquestación de los servicios que conforman la aplicación o las conexiones entre ellos, minimizando las interrupciones de la aplicación y evitando que ésta requiera ser detenida. La reconfiguración dinámica es especialmente importante debido a que interrupciones en una aplicación podrían afectar a un gran número de usuarios(Oliva y otros, 2012). A diferencia de la reconfiguración dinámica de arquitecturas en aplicaciones tradicionales basadas en servicios, la reconfiguración dinámica en entornos cloud debe tener en cuenta que los servicios que conforman una aplicación podrían estar desplegados en diferentes proveedores cloud seleccionados dependiendo de las características de los recursos cloud ofertados. Las características de los recursos cloud ofertados deben satisfacer requisitos de QoS, términos de Acuerdos de Nivel de Servicio (Service Level Agreement - SLA) u otros criterios de negocio.

## 1.3 Planteamiento del problema

La computación cloud representa un cambio fundamental en la manera en la cual las organizaciones adquieren sus recursos, brindándoles flexibilidad y agilidad. Sin embargo, en la actualidad no existe estandarización en las tecnologías cloud por lo que los proveedores cloud ofrecen recursos similares pero de diferente manera (Armbrust y otros, 2010). Esto genera una estrecha dependencia entre la implementación de servicios con la tecnología del proveedor cloud debido a que es prácticamente imposible para los desarrolladores utilizar la misma implementación de servicios para desplegarlos en otros proveedores (Loutas, 2011). El problema de dependencia con la tecnología del proveedor cloud, en algunos casos, es afrontado con la aplicación del paradigma de desarrollo dirigido por modelos. El Desarrollo de Software Dirigido por Modelos (DSDM) promueve el uso de modelos a lo largo de todo el proceso de desarrollo de software, elevando el nivel de abstracción y permitiendo que estos modelos puedan ser transformados sucesivamente hasta la obtención del producto final.

Existen propuestas que aplican principios DSDM como soporte a las actividades de modelado de servicios cloud; tal es el caso de MULTICLAPP (Guillén y otros, 2013a) y CloudML (Brandtzæg y otros, 2012b). Estas proponen lenguajes de modelado para describir topologías de despliegue, aplicaciones como una composición de artefactos de despliegue (p. ej., archivos WAR, paquetes Phyton, imágenes de máquinas virtuales, etc.) a ser desplegados en diferentes clouds, o recursos cloud que un artefacto de despliegue requiere del proveedor cloud. Sin embargo, en el ámbito del desarrollo de aplicaciones cloud, estos modelos se centran, en la mayoría de los casos, en la representación de la topología de despliegue de servicios o en facilitar la migración de servicios entre plataformas. A pesar de que abstraen las decisiones de despliegue de la tecnología específica de proveedores cloud, no establecen una relación entre los artefactos de despliegue y la arquitectura de la aplicación cloud ni dan soporte a la interoperación entre ellos.

Con respecto a propuestas que toman en cuenta aspectos relacionados con la arquitectura de la aplicación, cloudADL (Perez y Rumpe, 2013) propone un Lenguaje de Descripción Arquitectónica (Architecture Description Language – ADL) para el cloud como el elemento central de una metodología dirigida por modelos. Esta metodología describe a los servicios cloud como sistemas interactivos; es decir, como una composición de componentes independientes (p. ej., software, hardware destino, dispositivos externos, etc.) que interactúan unos con otros para alcanzar un objetivo común. cloudADL describe de manera textual la arquitectura de alto nivel de un servicio cloud, pero los requisitos tecnológicos específicos de un entorno cloud deben ser descritos utilizando otros lenguajes, lo que plantea nuevos retos a los arquitectos que tienen que lidiar con la integración de diferentes lenguajes. Por otro lado, StratusML (Hamdaqa and Tahvildari, 2015) propone un Lenguaje Específico de Dominio (Domain Specific Language – DSL) para aplicaciones cloud. Este provee múltiples vistas y diferentes capas para hacer frente a las diferentes necesidades de las partes interesadas, facilita el modelamiento visual y facilita la generación de código. Sin embargo, estas propuestas no hacen frente explícitamente a la naturaleza incremental del desarrollo de servicios cloud y no brindan soluciones a la evolución de las arquitecturas de los servicios cloud generada por la integración de incrementos. No consideran a la arquitectura del incremento a integrar como un modelo arquitectónico independiente cuya integración modificará el modelo arquitectónico actual de la aplicación, sino asumen que se cuenta con el modelo arquitectónico de toda la aplicación. El asumir que se cuenta con un modelo de toda la aplicación dificulta, en etapas posteriores del desarrollo, identificar los cambios arquitectónicos producidos por la integración del incremento. Si bien la información de cambios

arquitectónicos (p. ej., servicios adicionados, servicios eliminados, servicios actualizados, nuevas relaciones entre servicios, etc.) podría estar detallada en la documentación adjunta a los modelos arquitectónicos, generalmente se encuentra descrita en lenguaje natural. Esto evita que la información de cambios sea procesada automáticamente, impidiendo la propagación de los cambios especificados durante el diseño hacia otros artefactos de software en etapas de desarrollo posteriores. Adicionalmente, la falta de documentación de la interacción entre los servicios a ser integrados con los servicios existentes en la aplicación cloud actual dificulta la implementación de la orquestación entre servicios desplegados en diferentes plataformas.

Finalmente, en el contexto de reconfiguración dinámica de arquitecturas por propósitos adaptativos, a pesar de que la reconfiguración arquitectónica se produce en tiempo de ejecución, las acciones de cambio deberían ser planificadas desde las etapas iniciales y soportadas durante las diferentes etapas de desarrollo. Soportadas mediante la toma de decisiones que satisfagan la planificación inicial, así como mediante la aplicación de principios que permitan diseñar, implementar y desplegar arquitecturas de aplicaciones cloud fáciles de reconfigurar. Sin embargo, esfuerzos actuales tales como el proyecto europeo SeaClouds (Brogi y otros, 2014) o el proyecto MODAClouds (Ardagna y otros, 2012) se centran en la reconfiguración dinámica de las arquitecturas por necesidades correctivas o perfectivas, identificadas como resultado de actividades de monitorización, mas no con propósitos adaptativos provocados por la integración de incrementos de software que implementan nuevas funcionalidades.

La reconfiguración de las arquitecturas de servicios cloud producida por la integración de nuevos servicios introduce algunos retos; en particular la alta disponibilidad requerida de los servicios cloud hace necesario que su reconfiguración se produzca mientras la aplicación está aún en ejecución. Teniendo en cuenta las restricciones de los enfoques de reconfiguración dinámica de arquitecturas de servicios cloud, se requiere de soluciones que permitan: i) tratar a la arquitectura de los servicios cloud a integrar como una descripción arquitectónica parcial e independiente (del modelo arquitectónico de la aplicación actual), la cual complementará la arquitectura actual de la aplicación en la que se integrarán; ii) especificar la integración de incrementos a través de describir tanto la interoperabilidad entre servicios desplegados en diferentes proveedores cloud, así como los cambios arquitectónicos (impacto arquitectónico) que se producirán en la arquitectura actual de la aplicación cloud luego de la integración, iii) describir los recursos cloud necesarios para satisfacer los requisitos no funcionales de cada servicio incluido en un incremento; y iv) propagar los cambios especificados en las descripciones arquitectónicas hacia artefactos de software que soporten las

actividades de implementación, despliegue de servicios y reconfiguración arquitectónica.

## 1.4 Objetivos e hipótesis

El objetivo principal de esta tesis doctoral es definir y validar una aproximación de reconfiguración dinámica e incremental en el contexto de arquitecturas de servicios cloud, que permita derivar, a partir del diseño arquitectónico de incrementos de software, artefactos de implementación que faciliten la reconfiguración dinámica de arquitecturas de servicios cloud.

Este objetivo principal puede descomponerse en las siguientes metas:

1. Definición de un método de reconfiguración dinámica de arquitecturas de servicios cloud. Este método debe soportar la integración incremental de servicios mediante mecanismos que permitan: especificar la integración de incrementos de software y su impacto arquitectónico sobre la aplicación actual, documentar las decisiones de implementación y despliegue de incrementos de forma independiente de la tecnología y ofertas específicas de proveedores cloud, propagar la especificación del impacto arquitectónico hacia artefactos de implementación, despliegue y reconfiguración dinámica.
2. Definición de una aproximación tecnológica de soporte. Esta aproximación debe proveer un entorno para gestionar y especificar arquitecturas de servicios cloud, así como especificar el impacto de la integración de incrementos sobre la arquitectura de servicios cloud actual. Adicionalmente, mediante una estrategia de DSDM derivar los artefactos de implementación, despliegue de servicios y reconfiguración dinámica de acuerdo al impacto arquitectónico descrito en la especificación. Se definirán modelos para describir: la integración de incrementos, los artefactos requeridos para implementar la integración, y los recursos cloud necesarios para satisfacer los requisitos no funcionales de los servicios incluidos en los incrementos. Por último, se implementará una herramienta que soporte dicha aproximación tecnológica basada en el marco tecnológico Eclipse.
3. Validación del método mediante experimentos. Se definirá y ejecutará un experimento con el fin de proporcionar evidencia empírica sobre la facilidad de uso percibida, la utilidad de uso percibida y la intención de uso en el futuro del proceso de reconfiguración dinámica e incremental de arquitecturas de aplicaciones cloud.

El cumplimiento del objetivo propuesto pretende cubrir las siguientes hipótesis de investigación:

- Un método que considere la arquitectura del incremento como una arquitectura independiente, pero relacionada a la arquitectura de la aplicación cloud con la cual debe integrarse, *proporcionará* independencia al equipo de desarrollo para diseñar y desarrollar los incrementos de software y *facilitará* la identificación, especificación y trazabilidad de los cambios arquitectónicos producidos por la integración de los incrementos.
- Un método que brinde a los desarrolladores mecanismos que les permitan aplicar principios de arquitecturas orientadas a servicios *favorecerá* la evolución durante el diseño, implementación y despliegue de incrementos *facilitando* la reconfiguración dinámica e incremental en entornos cloud.
- Una estrategia DSDM *permitirá* a los desarrolladores elevar el nivel de abstracción y hacer frente a la portabilidad requerida en entornos cloud heterogéneos. Además, *facilitará* automatizar la propagación de las decisiones de integración, especificadas en modelos independientes de la tecnología, hacia artefactos de implementación, despliegue y reconfiguración dinámica específicos de las plataformas cloud.
- Usar y adaptar lenguajes de modelado ampliamente utilizados y específicos para aplicaciones basadas en servicios *permitirá* especificar las arquitecturas de servicios cloud y las necesidades de reconfiguración dinámica como parte de una planificación de acciones de cambio a un nivel de abstracción independiente de la tecnología.

## 1.5 Contexto de la investigación

Esta tesis doctoral se ha desarrollado en el contexto del grupo de investigación de Ingeniería del Software y Sistemas de Información (ISSI) del Departamento de Sistemas Informáticos y Computación de la Universitat Politècnica de València (UPV).

Los trabajos que han hecho posible el desarrollo de esta tesis, se engloban en proyectos de I+D financiados con fondos públicos. Los proyectos y ayudas son los siguientes:

- Proyecto Value@Cloud: "Desarrollo Incremental de Servicios Cloud Dirigido por Modelos y Orientado al Valor del Cliente" de la convocatoria de ayudas a proyectos de I+D+i. Financiado por el Ministerio de Economía y Competitividad (España). Participantes: Universitat Politècnica de València y Universidade Nova de Lisboa. IP: Silvia Abrahão. De Enero de 2013 a Diciembre de 2017.

- Proyecto "Aplicación de técnicas de desarrollo ágil de software para mejorar la productividad del proceso de despliegue incremental de servicios cloud en plataformas híbridas", del XIV Concurso 2015 convocado por la Dirección de Investigación de la Universidad de Cuenca. Financiado por la Universidad de Cuenca (Ecuador). IP: Miguel Angel Zúñiga Prieto. De Junio 2016 a Septiembre 2017.
- Proyecto "Model-Driven Incremental Development of Cloud Services", Microsoft Azure Research Award, Microsoft Research. Valor estimado de la ayuda: USD $ 40,000 por año. IP: Emilio Insfran. Julio de 2014 a Julio de 2016.
- Programa de becas de la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación (SENESCYT) de Ecuador, y Universidad de Cuenca-Ecuador.

## 1.6 Metodología de investigación

Se ha utilizado una extensión de la metodología propuesta por Gorschek y otros (2006) donde se incluyen actividades de evaluación y observación tanto en el ámbito académico como en la industria. Este modelo nos propone ocho actividades (ver Figura 1-1) relacionadas en un proceso iterativo donde se formulan soluciones candidatas que se evalúan de una forma empírica con el objetivo de alcanzar una solución realista. A continuación, se enumeran las actividades:

1. *Identificación del problema*. Se pretende entender el problema que el socio industrial pretende resolver.
2. *Formulación del problema*. Una vez que el problema está identificado, tras las reuniones necesarias, es preciso acotarlo y definirlo de una manera más precisa. Se deben de especificar los factores contextuales claramente.
3. *Revisión del estado del arte*. Se realiza una revisión de la literatura de forma crítica con el objetivo de identificar hasta qué punto las soluciones actuales están cubiertas y cuáles son los problemas abiertos a resolver con la investigación a desarrollar.
4. *Solución candidata*. Se idean soluciones posibles que serán depuradas en las distintas etapas de refinamiento (etapas 6 y 7).
5. *Entrenamiento*. Esta etapa se realiza de forma incremental. Las primeras fases del entrenamiento se centran en crear el conocimiento necesario para que los profesionales puedan opinar sobre la aplicabilidad de la propuesta. En

fases más tardías el entrenamiento sirve para crear guías y pasos metodológicos detallados para aplicar las soluciones.

6. *Validación inicial*. Se lleva a cabo una evaluación preliminar de las soluciones en el contexto de investigación o en una configuración industrial limitada. Pueden realizarse experimentos controlados o casos de estudio con alumnos o profesionales en el ámbito académico. En el ámbito académico podrían realizarse seminarios, talleres prácticos y encuestas.
7. *Validación realista*. Se llevan a cabo casos de estudio en configuraciones industriales. Es esta fase se definirán guías prácticas y se desarrollarán herramientas que soporten la propuesta.
8. *Lanzamiento de la solución*. En este último paso se valoran los resultados obtenidos, y las herramientas y el material de entrenamiento se preparan para el uso en contextos industriales.

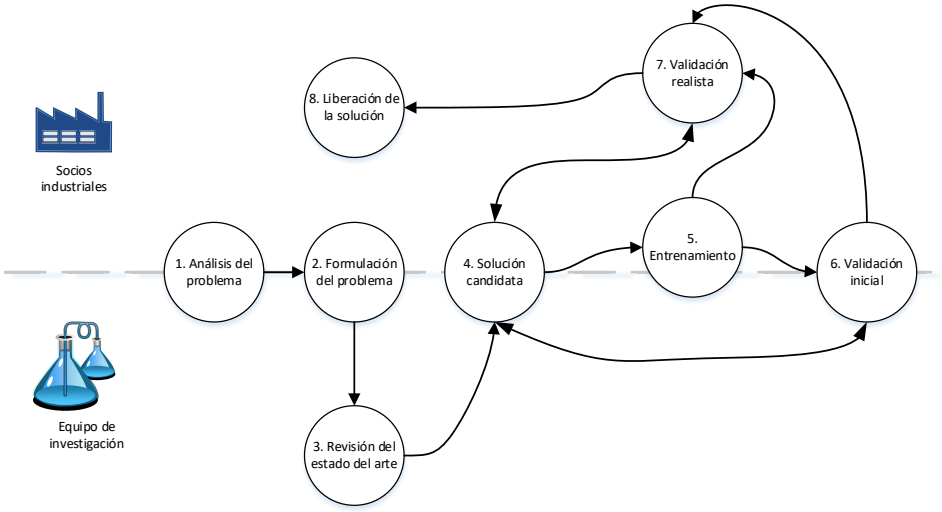



**Figura 1-1 Modelo de transferencia tecnológica**

Durante este trabajo de investigación se han cubierto las seis primeras etapas de la investigación. En las primeras fases de la tesis doctoral y en el contexto del proyecto de investigación Value@Cloud se identificó el problema de proveer una aproximación de reconfiguración dinámica e incremental de servicios cloud. Una vez diseñada una solución candidata, ésta se mejoró mediante la etapa de entrenamiento y se realizó una validación inicial, mediante un experimento, en el contexto de la Universitat Politècnica de València (España). En la siguiente subsección se presenta la metodología de experimentación, basada en experimentos que se ha utilizado en el contexto de esta tesis doctoral.

### 1.6.1 Experimento

La experimentación es una fase crucial de la validación y puede ayudar a determinar si los métodos utilizados se ajustan a una teoría particular. Los experimentos controlados son apropiados para investigar diferentes aspectos, como la confirmación o prueba de teorías existentes, la evaluación de la precisión de los modelos, etc. De acuerdo con Campbell y Stanley (1982) los experimentos pueden dividirse en dos categorías: los que aplicaron una asignación aleatoria de grupos de control, y los que no. En el primer caso se trata de un experimento mientras que el segundo grupo se llama cuasi-experimento. Autores como Laitenberger y Rombach (2003) consideran los cuasi-experimentos como aproximaciones prometedoras para incrementar la cantidad de estudios empíricos en la industria de la ingeniería del software. En este trabajo de tesis se utilizará la modalidad de cuasi-experimento para realizar la validación de los procesos definidos. Se ha diseñado un cuasi-experimentos de acuerdo a las guías propuestas por Wohlin y otros (2000). Las principales actividades de este proceso pueden observarse en la Figura 1-2.

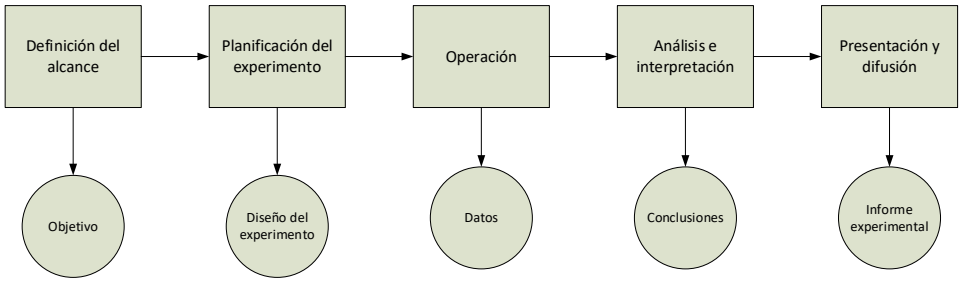



**Figura 1-2 Proceso experimental con los artefactos generados en las distintas actividades**

La primera actividad es la *definición del alcance.* Aunque inicialmente no estén definidas formalmente, las hipótesis del experimento deberán definirse en esta actividad junto con los objetivos y las metas. Las metas del experimento se formulan a partir del problema a resolver siguiendo el marco de trabajo llamado en *Goal/Question/Metric* (GQM) (Basili y otros 1998)con el esquema siguiente:

- *Analizar*: qué se analiza, cuál es el objeto de estudio.
- *Con el propósito*: cuál es la intención.
- *Con respecto a su*: cuál es su efecto estudiado.
- *Desde el punto de vista*: qué vista.

- *En el contexto de*: dónde tiene lugar el estudio, sobre qué artefactos y con qué tipo de sujetos.

La siguiente actividad es la *planificación del experimento*. En esta actividad se define en profundidad el contexto, que incluye tanto la componente personal (qué perfiles tendrán los sujetos) como el entorno en el que tendrá lugar. También se especifican formalmente las hipótesis experimentales, incluyendo las hipótesis nulas e hipótesis alternativas, así como las variables, tanto las independientes (entradas) como las variables dependientes (salidas). Además, se seleccionará un diseño experimental y se identificará y preparará la instrumentación a utilizar. El diseño experimental describe, por ejemplo, como se llevarán a cabo los ensayos (*online* vs *offline*) o la aleatorización de los sujetos. Por último, en la fase de planificación, es donde se ha de considerar la cuestión de la validez de los resultados que cabe esperar.

Durante la *operación* del experimento se van a preparar, ejecutar y validar los datos recogidos. Durante la ejecución se debe asegurar que el experimento se lleva a cabo siguiendo el diseño definido en la fase de planificación y se recogen los datos experimentales. Por último, se validan los datos recogidos para asegurar que son correctos y proveen una imagen real del experimento.

En el *análisis e interpretación* se emplean los datos recogidos y validados en la fase de operación. Se emplean estadísticos descriptivos con el fin de entender los datos de manera informal. El siguiente paso es analizar si el conjunto de datos a considerar debe ser reducido, bien eliminando puntos o bien eliminando variables, tras analizar si hay variables redundantes que nos ofrecen la misma información. Una vez reducido el conjunto de datos, se llevarán a cabo las pruebas de las hipótesis. Se seleccionarán las pruebas en función del tipo de escala, variables de entrada y tipo de resultados que se buscan. La interpretación se centra en determinar si, en base a las pruebas, se pueden aceptar o rechazar las distintas hipótesis, esto es, determinar la influencia de las variables independientes sobre las variables dependientes en el caso en que se rechacen las hipótesis nulas.

Por último, la actividad de *presentación y difusión* está relacionada con la preparación de la documentación, ya sea un artículo de investigación para difundir los resultados o un paquete de laboratorio con el fin de llevar a cabo replicaciones del experimento.

## 1.7 Estructura de la tesis

En este capítulo se han presentado: las motivaciones de la investigación, los ob-

jetivos y metas a resolver, el contexto de investigación y el método de investigación aplicado. El resto de la tesis de organiza en los siguientes capítulos:

- Capítulo 2. Marcos y espacios tecnológicos.

Este capítulo presenta el marco tecnológico en el que se desarrolla esta tesis: ingeniería basada en modelos, arquitectura de software orientada a servicios y reconfiguración dinámica, y computación cloud y desarrollo incremental de aplicaciones cloud. Adicionalmente se presenta el espacio tecnológico relacionado con este trabajo de tesis.

- Capítulo 3. Estado del arte.

En este capítulo de analizan las propuestas existentes en la literatura en el ámbito de la reconfiguración dinámica de aplicaciones cloud.

- Capítulo 4. Reconfiguración dinámica e incremental de arquitecturas orientadas a servicios.

En este capítulo se presenta la contribución metodológica de esta tesis: el método DIARy, integrado por la definición de un lenguaje para la descripción o diseño del impacto arquitectónico de la integración de incrementos en aplicaciones cloud, un proceso dirigido por transformaciones para la generación de artefactos de implementación que agilizan la integración, despliegue de servicios y la reconfiguración dinámica, y modelos que permiten documentar las decisiones tomadas en las diferentes actividades del proceso.

- Capítulo 5. Herramientas de soporte a DIARy.

En este capítulo se presenta la aproximación tecnológica definida para soportar la reconfiguración dinámica e incremental de arquitecturas de aplicaciones cloud. Se presenta una infraestructura software que incluye la implementación del lenguaje de especificación del impacto arquitectónico de la integración, los DSLs y respectivos metamodelos que dan soporte al método y las transformaciones de modelos utilizadas para agilitar la integración, despliegue de servicios y reconfiguración dinámica.

- Capítulo 6. Aplicación del método DIARy.

En ese capítulo se presenta un escenario de desarrollo incremental y de reconfiguración dinámica de una aplicación cloud provocada por la integración de un incremento de software. Tanto la aplicación actual como el incremento de software son desplegados en las plataformas Microsoft Azure© y Google Cloud Platform. Se describe los resultados de utilizar el método de reconfiguración dinámica.

- Capítulo 7. Validación empírica del método DIARy.

En este capítulo se presenta la validación empírica de la propuesta mediante un experimento.

- Capítulo 8. Conclusiones y trabajos futuros.

En este capítulo se presentan las contribuciones de esta tesis, así como las líneas de investigación presentes y futuras, junto con las publicaciones que se originaron a partir de este trabajo de investigación.

- Apéndice A – Guías del método DIARy.

Este apéndice contiene las guías propuestas en DIARy durante la especificación del impacto de la integración.

- Apéndice B – Material del experimento.

Este apéndice contiene el material experimental utilizado durante el experimento que evalúa el método DIARy.

# Capítulo 2. Marcos y espacios tecnológicos

Este capítulo presenta los tres pilares sobre los que se asienta este trabajo de investigación: ingeniería basada en modelos, arquitectura de software orientada a servicios y reconfiguración dinámica, computación cloud y desarrollo incremental de aplicaciones cloud. El objetivo del capítulo es introducir el marco conceptual que permite entender el trabajo de investigación en su totalidad. En las siguientes secciones se van a introducir y discutir cada uno de estos campos, así como las interacciones que aparecen entre ellos. La estructura del capítulo es la siguiente:

La sección 2.1 presenta el enfoque de desarrollo dirigido por modelos, el estándar MDA, las transformaciones de modelos, y los lenguajes de dominio específico.

La sección 2.2 introduce a las arquitecturas de software orientadas a servicios, su papel en el desarrollo de aplicaciones basadas en servicios y su impacto en la reconfiguración dinámica de aplicaciones basadas en servicios.

La sección 2.3 introduce conceptos de computación cloud, el desarrollo de aplicaciones cloud y su relación con las arquitecturas de software orientadas a servicios.

La sección 2.4 presenta el lenguaje SPEM 2.0 el cual se utiliza en la definición de los procesos de software de este trabajo de tesis.

La sección 2.5 describe el espacio tecnológico que se utilizará durante la implementación de la aproximación tecnológica propuesta en este trabajo de tesis.

La sección 2.6 presenta un breve resumen de los conceptos introducidos durante este capítulo.

## 2.1 Ingeniería basada en modelos

La ingeniería basada en modelos (Model Driven Engineering – MDE) es un enfoque de la ingeniería del software que considera a los modelos como artefactos de primera clase y que tiene como objetivo desarrollar, mantener y evolucionar software por medio de transformaciones de modelos. El proceso de desarrollo que sigue un enfoque MDE es llamado Desarrollo de Software Dirigido por Modelos (DSDM) (Brambilla y otros 2012).

En MDE los modelos son partes activas del proceso de desarrollo de software; siendo éstos abstractos y formales, al mismo tiempo; en donde, los modelos dejan de ser documentos de diseño y pasan a ser partes del software, lo que constituye un factor decisivo para aumentar la velocidad y calidad de desarrollo de software (Stahl y otros 2006). Los modelos MDE tienen exactamente el mismo significado y tratamiento que el código de la aplicación, en el sentido de que la mayor parte de la aplicación final, no solo las clases y los esqueletos de los métodos pueden ser generados a partir de ellos. Un enfoque basado en modelos requiere lenguajes para la especificación de modelos, definición de transformación y descripción del metamodelo. MDE propone el uso de transformaciones de modelos con el fin de transformar un modelo en otro, y también para producir el producto final.

Existen cinco elementos que una infraestructura de soporte MDE debe definir:

- Conceptos para la creación de modelos y reglas claras que rigen su uso.
- La notación a utilizar para describir los modelos.
- Claridad de cómo los elementos del modelo representan los elementos del mundo real y los artefactos de software.
- Mecanismos para facilitar las extensiones de usuarios.
- Mecanismos para facilitar el intercambio de conceptos.

### 2.1.1 Meta-Object Facility

El Meta-Object Facility (MOF) es un estándar propuesto por el Object Management Group Inc. (2013). MOF es el vocabulario básico, llamado meta-metamodelo, provisto por el estándar MDA para describir meta-modelos, y por lo tanto nuevos vocabularios. Un metamodelo puede ser considerado como la sintaxis abstracta de un lenguaje. Modelos y metamodelos son generalmente organizados en una estructura de N capas, M3-M0 (ver Figura 2-1), con la siguiente distribución:

- *Nivel M3: Meta-metamodelado.* En esta capa reside el meta-metamodelo, un lenguaje usado para definir los metamodelos del nivel M2. Dentro de OMG, MOF es el lenguaje estándar utilizado en esta capa. Además, este lenguaje MOF se define a sí mismo. La relación Instance-of (o interfaces de reflexión) permite el recorrido a través de cualquier número de meta niveles recursivamente.
- *Nivel M2: Metamodelo.* Los metamodelos de la capa M2 se utilizan para describir los modelos del nivel M1. Por ejemplo el meta-modelo de UML (Object Management Group Inc., 2001) se define en este nivel.
- *Nivel M1: Modelo.* En este nivel se definen los modelos. Un modelo es una instancia de un metamodelo del nivel M2. Por ejemplo, si en el M2 residiera el metamodelo de UML, en el nivel M1 podríamos tener uno de sus modelos: modelo de clases de facturación, diagrama de actividad de ventas, etc.
- *Nivel M0: Instancias.* En esta capa se describen objetos que son instancia de los modelos.

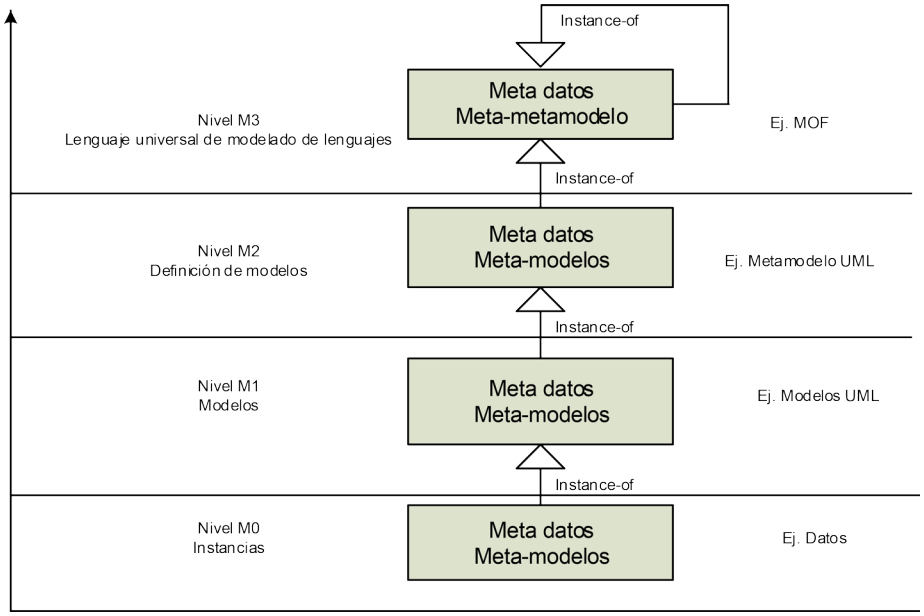



**Figura 2-1 Niveles de la arquitectura MOF**

MOF se considera una arquitectura de meta-modelado cerrada. Esto quiere decir que el último nivel, el MN, se pueden definir con instancias de elementos de MN. Esto implica que MOF se puede definir a sí mismo.

Además, MOF provee los siguientes conceptos para definir un lenguaje:

- *Clases*, las cuales modelan los meta-objetos MOF.
- *Tipos de datos*, que modelan la necesidad de datos descriptivos.
- *Asociaciones*, que modelan relaciones binarias entre meta-objetos.
- *Paquetes*, que permiten organizar y agrupar las clases.

### 2.1.2 Desarrollo de software dirigido por modelos

El Desarrollo de Software Dirigido por Modelos (DSDM) es un enfoque de desarrollo de software basado en modelos. Un modelo puede definir la funcionalidad, estructura o comportamiento de un sistema. Los modelos permiten trabajar con un nivel de abstracción más cercano a los conceptos de dominio, en lugar de centrarse en conceptos orientados a la plataforma como el desarrollo del software tradicional. El objetivo de esta propuesta es maximizar la productividad e incrementar la interoperabilidad entre sistemas, facilitando la reutilización y adaptación del cambio tecnológico.

Además, cualquier artefacto software es considerado un modelo o un elemento del modelo. Mientras que los enfoques tradicionales utilizan modelos para documentación o la comunicación de ideas, en DSDM son entidades de primera clase durante todo el ciclo de vida del desarrollo (Bézivin, 2005). El paso de desarrollar software haciendo uso de modelos con el propósito de documentar las decisiones de diseño, a desarrollar el software a partir de modelos, significa un salto significativo en pos de elevar el nivel de abstracción. Éste es el objetivo que persigue el DSDM, en el que se aborda el desarrollo de sistemas software mediante el refinamiento o transformación sucesiva de modelos. El sistema se modela en términos del dominio del problema y cada nuevo modelo, desciende en el nivel de abstracción, añadiendo nuevos detalles de la plataforma destino a los modelos definidos en niveles de abstracción superiores, hasta llegar a la obtención del código del sistema en desarrollo.

El uso de transformaciones automáticas de modelos a lo largo del ciclo de vida tiene múltiples beneficios:

- Fomenta la reutilización una vez definida la cadena de transformaciones que pasan de los modelos en un alto nivel de abstracción al código de la solución final. Éstas transformaciones pueden ser utilizadas una y otra vez para construir sistemas similares.

- Minimiza los errores en tareas repetitivas de creación de un modelo a partir de otro.
- Mejora la calidad de los modelos y código resultantes al encapsular las buenas prácticas en la propia definición de la transformación.
- Encapsula el conocimiento de los expertos del dominio en las transformaciones, una vez una transformación ha sido definida, estás pueden ser empleadas por desarrolladores que no necesitan conocer el proceso de transformación en detalle.
- El DSDM es una aproximación abierta e integradora no vinculada a una norma especial, por lo tanto, existen diferentes implementaciones.

#### *2.1.2.1 Arquitectura dirigida por modelos*

La iniciativa Arquitectura de Software Dirigida por Modelos (Model-Driven Architecture – MDA) fue propuesta por el Object Management Group (OMG) (Belaunde et al., 2003). El objetivo de MDA es solventar el problema de heterogeneidad de tecnologías e integración. La idea principal es utilizar modelos de tal manera que las propiedades y características de los sistemas están reflejadas en descripciones abstractas; por lo tanto, no se vean afectados por cambios tecnológicos. Por otro lado, técnicas de generación de código permiten obtener código conforme a diferentes tecnologías; ésto no solo ayuda a solucionar problemas de interoperabilidad, sino que permite el reúso de modelos.

Algunos conceptos básicos definidos en el estándar MDA de acuerdo al Object Management Group Inc. (2003) son:

- *Plataforma*: conjunto de subsistemas y tecnologías que proveen un conjunto coherente de funcionalidad a través de interfaces y el uso de patrones de uso, que cualquier aplicación soportada por la plataforma podrá usar sin prestar atención a cómo la funcionalidad ofrecida está implementada.
- *Modelo Independiente de Computación (Computation Independent Model – CIM)*: Vista del sistema desde un punto de vista independiente de la computación. Se centra en el entorno del sistema y los requisitos para el sistema; los detalles de la estructura y procesamiento del sistema están ocultos o aún no se han determinado. Es un modelo del sistema que muestra el sistema en su entorno de operación, y muestra cuál será su cometido. Para su especificación se emplean términos del vocabulario común entre los profesionales del dominio de interés.

- *Modelo Independiente de Plataforma (Platform Independent Model – PIM)*: Vista del sistema en el que se ocultan los detalles de la plataforma de ejecución y que permite describir el sistema de manera que, esta descripción, pueda ser utilizada con varias plataformas cuyo tipo sea similar.
- *Modelo de Plataforma (Platform Model)*: Conjunto de conceptos técnicos que representan las distintas partes que conforman una plataforma y los servicios que ésta ofrece. También provee los conceptos que representan los distintos elementos para ser usados en la especificación del uso de la plataforma por una aplicación.
- *Modelo Específico de Plataforma (Platform Specific Model – PSM)*: Vista del sistema en el que se combina la especificación del PIM con los detalles de cómo el sistema hace uso de plataforma específica.

MDA se basa en una gran cantidad de los estándares OMG, algunos de ellos son los siguientes:

- *Meta Object Facility (MOF)*: Es el lenguaje común (meta-meta-modelo) utilizado para describir metamodelos en el enfoque MDA. Lenguajes como UML son descritos utilizando MOF.
- *Unified Modeling Language (UML)*: Provee un lenguaje para describir diferentes sistemas. UML es un lenguaje independiente del dominio, aunque sus orígenes están en el modelado orientado a objetos.
- *Object Constraint Language (OCL)*: Es un lenguaje declarativo y sin efectos colaterales, que permite la definición de restricciones como reglas en los modelos basados en MOF.
- *XML Metadata Interchange (XMI)*: Es el mecanismo de persistencia para el almacenamiento e intercambio de modelos entre herramientas compatibles con MOF.

MDA define, además, su propio proceso de desarrollo para producir el código ejecutable de un sistema a partir de la especificación, en términos de modelos, con una sintaxis y semántica formales. El proceso de desarrollo parte de una especificación del sistema en modelos PIM, que será transformado mediante una o varias transformaciones en uno o varios modelos PSM. El código final de la aplicación se obtiene mediante una transformación de modelos PSM a código. La separación entre PIM y PSM, es clave en el marco de trabajo MDA y es lo que permite la definición de sistemas portables, al abstraer en la primera definición del sistema los detalles de la plataforma. La definición de transformaciones de PIMs a PSMs de distintas plataformas es el mecanismo mediante el cual se

podrán obtener versiones del mismo sistema para distintas plataformas. La operación fundamental de MDA es la transformación de PIMs a PSMs. La guía de MDA (Belaunde et al., 2003) establece que modelos PIM y PSM son expresados como modelos MOF, y que la transformación debe ser automatizada tanto como sea posible.

#### *2.1.2.2 Lenguajes específicos de dominio*

De acuerdo a Van Deursen y otros (2000), un Lenguaje Específico de Dominio (Domain Specific Language – DSL), es "un lenguaje de programación o lenguaje de especificación que ofrece, mediante notaciones y abstracciones apropiadas, poder expresivo enfocado en, y usualmente restringido a, un problema de dominio específico". Adicionalmente, mediante una revisión literaria, identificaron que entre los beneficios del uso de DSLs se tiene:

- Permiten que las soluciones sean expresadas en el idioma y nivel de abstracción del problema de dominio.
- Mejoran la productividad, confiabilidad, mantenibilidad, y portabilidad.
- Incorporan el conocimiento del dominio y, por lo tanto, permiten la conservación y reutilización de este conocimiento.
- Permiten la validación y optimización en el nivel de dominio.
- Permiten que usuarios sin experiencia en programación creen modelos o programas en base a su conocimiento del dominio.

Los DSLs son utilizados en la ingeniería de software con el fin de mejorar la calidad, flexibilidad y tiempo de entrega de sistemas de software. Estos lenguajes pueden ser textuales, como la mayoría de lenguajes de programación, o gráficos. A diferencia de los Lenguajes de Propósito General (General Purpose Languages – GPL), tales como C++, Java, o Python, los DSLs contienen usualmente un conjunto restringido de notaciones y abstracciones.

Un DSL está compuesto esencialmente de tres componentes (Clark y otros, 2008)(Kelly y Tolvanen, 2007):

- *Sintaxis abstracta*: El conjunto de conceptos del lenguaje y sus relaciones, conjuntamente con reglas para combinarlos.
- *Sintaxis concreta*: Define la notación que el usuario final utilizará para especificar programas o modelos conforme a la sintaxis abstracta. Las notaciones más utilizadas son textuales o gráficas.
- *Semántica*. Describe el significado de los constructores del lenguaje.

MDE proporciona las bases para construir DSLs mediante metamodelos, los cuales proveen un mecanismo unificado y expresivo para definir los conceptos del dominio; en donde, MOF es utilizado para definir la sintaxis abstracta del dominio.

### 2.1.3 Object Constraint Language

El Object Constraint Language *(OCL)* (Object Management Group Inc., 2006) es un lenguaje declarativo para el análisis y diseño de sistemas de software. OCL es definido como un lenguaje estándar que complementa UML o cualquier lenguaje de modelado MOF. Proporciona soporte para la especificación de restricciones y consultas sobre modelos, permitiendo definir y documentar modelos MOF con mayor precisión.

Por ejemplo, un modelo UML, que usa un diagrama de clases o un diagrama de colaboración no tiene la expresividad de modelado suficiente para ser una especificación precisa y sin ambigüedad de un sistema (o un aspecto de un sistema). Las expresiones OCL completan esta especificación, proveyendo información adicional; en donde, usualmente esta información no puede ser expresada por medio de elementos del diagrama. Las expresiones OCL siempre están ligadas a un elemento de modelado, por ejemplo, una clase, un método, un estado de un diagrama de estados. Adicionalmente, la evaluación de una expresión OCL no produce efectos secundarios; es decir, su evaluación no puede alterar el estado del sistema correspondiente.

Las expresiones OCL se pueden aplicar tanto a modelos UML, modelos MOF así como a cualquier meta-modelo MOF. Por lo tanto, expresiones OCL pueden usarse para restringir la semántica de los modelos y metamodelos. En el contexto MDA, OCL puede utilizarse: para construir modelos más precisos, en el nivel M1 de MOF (ver Figura 2-1), metamodelos más precisos en el nivel M2 (definición de Lenguajes de Modelado) y para especificar transfor-maciones.

### 2.1.4 Transformaciones de modelos

Transformación de modelos es el proceso de conversión de un modelo a otro modelo del mismo sistema (Belaunde et al., 2003). Un modelo puede ser transformado a varios modelos alternativos que pueden mantener la semántica, pero con diferente sintaxis y nivel de detalle.

Una definición de transformación (ver Figura 2-2) consiste en una colección de reglas de transformación que son especificaciones inequívocas de la manera que (parte de) un modelo puede utilizarse para crear una pieza de otro modelo. Las transformaciones son definidas en términos de los metamodelos involucrados

en el proceso de transformación. Una transformación, una definición de transformación y sus reglas de transformación pueden definirse de la siguiente manera (Hurwitz y otros 2009):

- Una transformación es la generación automática de un modelo destino desde un modelo de fuente, según una definición de transformación.
- Una definición de transformación es un conjunto de reglas de transformación que juntas describen cómo un modelo origen puede ser transformado en un modelo destino.
- Una regla de transformación es una descripción de cómo una o más constructores en el lenguaje origen pueden ser transformados en uno o más constructores de un lenguaje destino.

La característica más importante de una transformación es el hecho de que una transformación de modelos debe mantener el significado entre el modelo origen y el modelo destino. En este punto hay que decir que el significado del modelo sólo puede ser preservado y expresado en el modelo origen y destino. Parte de la información pueden perderse si el lenguaje de destino es menos expresivo que el lenguaje de origen.

Miller y otros (2004) clasifican las transformaciones en dos tipos: vertical y horizontal. Una transformación es horizontal si es que el modelo origen y el modelo destino se encuentran al mismo nivel de abstracción. En MDA esto puede ser una transformación PIM a PIM, o una transformación PSM a PSM. Una transformación es vertical cuando el modelo origen y el modelo destino se encuentran en diferentes niveles de abstracción. En MDA esto puede ser una transformación de PIM a PSM, o de PSM a código. En MDA, la forma más aceptada de definir modelos es a través de técnicas de metamodelado; mientras que las transformaciones son definidas utilizando lenguajes de transformación de modelos.

Por otro lado Mens y Van Gorp (2006) presentan una taxonomía que permite clasificar los distintos tipos de transformaciones, basándose en el objeto de la transformación. Establecen cuatro criterios principales:

- *Transformaciones de programa y transformaciones de modelos*: Se clasifican en: Transformaciones de programa (código-a-código), cuando el objeto de la transformación son programas (código fuente), Transformaciones modelo-a-modelo (M2M) o transformaciones modelo-a-texto (M2T), cuando al menos uno de los elementos de la transformación es un modelo.

- *Nivel de abstracción*: Se clasifican en transformaciones horizontales y transformaciones verticales.
- *Lenguajes involucrados*. Se clasifican en: Transformaciones exógenas, cuando las transformaciones se dan entre modelos creados conforme a distintos metamodelos. Transformaciones endógenas, cuando los modelos involucrados en la transformación han sido creados conforme a un mismo metamodelo.
- *Direccionalidad*: Se clasifican en: Transformación bidireccional, cuando la transformación trabaja en ambas direcciones; es decir, la transformación de un modelo origen a un modelo destino, y la transformación del modelo destino al modelo origen utilizan la misma definición de transformación. Transformación unidireccional, cuando se requieren dos definiciones de transformación para transformaciones inversas.

En la propuesta original de MDA, el metamodelado no fue una condición necesaria; solo en versiones posteriores de MDA se incorporaron las ideas definidas en MDE, lo que llevó a la definición de estándares como MOF, el cual propone cuatro niveles de arquitectura de metamodelos (M0 – M3), ver Figura 2-2.

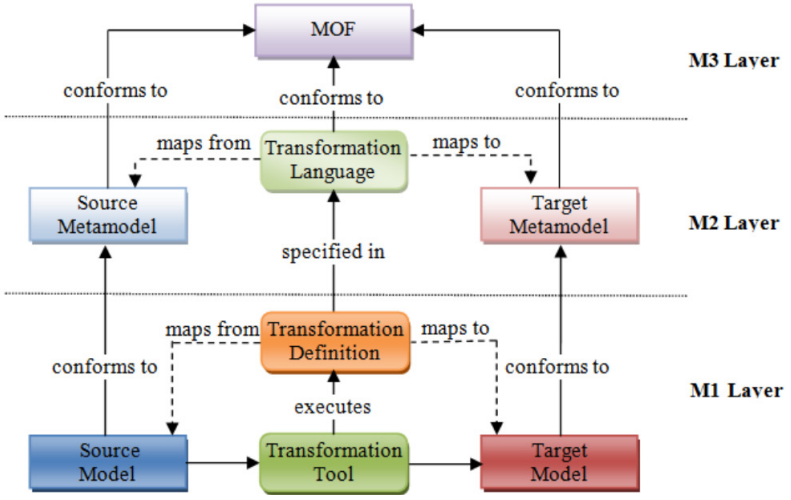



**Figura 2-2 Definición de una transformación de modelos**

## 2.2 Arquitectura de software orientada a servicios

El objetivo del diseño de arquitectura de software es definir las restricciones para las actividades subsecuentes de diseño e implementación que resulten en el desarrollo de un sistema que cumpla con sus objetivos funcionales y de calidad. Una

arquitectura de software es a la vez permisiva y restrictiva con respecto a las decisiones adoptadas en actividades posteriores (Perry y Wolf 1992). Las decisiones de diseño afectan a los atributos de calidad del software. Una arquitectura de software facilita la evaluación temprana de la calidad del sistema a ser desarrollado; de hecho, la arquitectura de software es el primer artefacto de diseño creado en un proyecto de software que permite tal evaluación.

La computación orientada a servicios (Service Oriented Computing – SOC) es una nueva generación de plataformas para la computación distribuida. Esta incluye muchos conceptos, como su propio paradigma de diseño (orientación a servicios) que ofrece normas y directrices para realizar ciertas características de diseño en una aplicación. SOC tiene sus propios principios y patrones de diseño, un propio modelo de arquitectura que es conocida como Arquitectura Orientada a Servicios (Service Oriented Architecture – SOA), junto a muchos conceptos, tecnologías y marcos de trabajo asociados. Los términos SOA y SOC no son sinónimos, pero la literatura muy a menudo los trata como uno (Erl, 2007).

SOA es un estilo arquitectónico que tiene como objetivo aumentar la eficiencia, agilidad y productividad de una empresa, haciendo hincapié en los servicios como el principal medio, a través del cual, la lógica del negocio es representada en soporte de la realización de los objetivos estratégicos, asociados con la computación orientada a servicios (SOC). En donde, un servicio es realizado por una aplicación o sistema, y representa el comportamiento externo de la aplicación o sistemas (Silva y otros, 2008). Adicionalmente, un servicio debería tener la capacidad de realizar un trabajo, una especificación del trabajo ofrecido, y la oferta para realizar un trabajo (Brown y otros 2006), (Brown y otros, 2006). Una implementación de SOA puede consistir en una combinación de tecnologías, productos, Interfaces de Programación de Aplicaciones (Application Programming Interface – API), extensiones de infraestructura, y otras partes (Erl, 2007).

De acuerdo con Linthicum (2009) SOA es:

> *"Un marco estratégico de tecnología que permite a todos los sistemas interesados, dentro y fuera de una organización, exponer y tener acceso a servicios bien definidos, e información ligada a aquellos servicios, además que pueden ser abstraídos y procesados por capas y aplicaciones compuestas para el desarrollo de soluciones".*

El autor de esta definición indica que SOA añade el aspecto de la agilidad con la arquitectura, lo que permite a los desarrolladores tratar con cambios en el sistema utilizando una capa de configuración en vez de constantemente tener que volver a desarrollar estos sistemas. Por otro lado, OASIS (2008) define SOA como:

> *"Un paradigma para organizar y utilizar capacidades distribuidas que pueden estar bajo el control de diferentes dominios de propiedad. Proporciona un medio uniforme para*

> *ofrecer, descubrir, interactuar y utilizar las capacidades para producir los efectos deseados en consonancia con las condiciones previas y expectativas medibles"*

Y por último Josuttis (2008) define SOA como:

> *"Un paradigma arquitectónico para hacer frente a los procesos de negocio distribuidos en un amplio panorama de los sistemas heterogéneos existentes y las nuevos que se encuentran bajo el control de diferentes propietarios".*

De las definiciones presentadas se deduce que SOA, debido a su enlace con los procesos de negocio, es un estilo arquitectónico que guía a los negocios a crear, organizar y reutilizar sus componentes computacionales. En SOA, los componentes se comunican a través de una única interfaz y los sistemas normalmente sirven para múltiples usuarios remotos. Para establecer SOA con éxito se necesitan tres componentes (Josuttis 2008): la infraestructura, la arquitectura y los procesos. Con respecto a la infraestructura, los servicios Web son un enfoque tecnológico específico para la implementación de SOA.

### 2.2.1 Elementos

Para la aplicación del estilo arquitectónico SOA, requiere que una aplicación sea dividida en unidades más pequeños, en unidades individuales de la lógica conocida como servicios. Un servicio es una entidad independiente que encapsula una lógica relacionada con una determinada tarea de negocio u otra agrupación lógica, e intercambia información con otros servicios para lograr la meta deseada. Un servicio puede combinarse con otros servicios con el fin de crear composiciones de servicios. Los servicios pueden ser distribuidos, lo que significa que ellos pueden residir dentro o fuera de la organización. En donde, a diferencia de los componentes, los recursos no se compran como un paquete de software, sino son puestos a disposición como servicios independientes a otros participantes en la red (Haines y otros 2010). Sin embargo, a pesar de existir de manera autónoma, mantienen un grado de uniformidad y son accedidos de forma estandarizada (Erl, 2007).

La comunicación entre servicios requiere que los servicios tengan conocimiento de otros servicios; lo cual se logra mediante descripciones de los servicios, la cual contienen el nombre y la ubicación del servicio, así como entradas y salidas de datos. Los servicios son combinados libremente a través de sus descripciones; requiriéndose un marco de comunicación para establecer la interacción ellos. Este marco de comunicación define la estructura de los mensajes, las políticas y protocolos que son usados en la comunicación. Como los servicios los mensajes también son autónomos, por lo que contienen suficiente inteligencia para gobernar la lógica de procesamiento que les corresponde.

Un servicio es un componente de software de significado funcional distintivo que típicamente encapsula un concepto de negocio de alto nivel. De acuerdo con Krafzig y otros (2004), un servicio consiste en: una implementación que provee la lógica de negocios y datos, un contrato de servicio que especifica su funcionalidad, uso y restricciones para un cliente del servicio, y una interface de servicio que expone físicamente la funcionalidad.

- *Implementación*: La implementación del servicio provee la lógica de negocio y los datos requeridos. Esta es la realización técnica que cumple con el contrato de servicio, y consiste de uno o más artefactos tales como programas, datos de configuración, y bases de datos. La lógica de negocio es disponible mediante las interfaces del servicio.
- *Contrato de Servicio*: Proporcionan una especificación informal de la finalidad, funcionalidad, restricciones y el uso del servicio. La forma de esta especificación puede variar, dependiendo del tipo de servicio. Un elemento no obligatorio de los servicios de contrato es una definición de interfaz formal basada en lenguajes como son el lenguaje de definición de Interface (IDL) o el o Lenguaje de Descripción de Servicios Web (WSDL). Estos elementos proporcionan abstracción e independencia de tecnología, incluyendo el lenguaje de programación, el protocolo de middleware de la red y su entorno de ejecución.
- *Interface*: Expone la funcionalidad del servicio a los clientes que estan conectados al servicio utilizando una red. A pesar de que la interface es parte del contrato de servicio, la implementación física de la interface consiste de partes del servicio que son incorporados en los clientes del servicio (aplicaciones u otros servicios).

### 2.2.2 Principios de diseño

Varios principios de diseño pueden ser seguidos en el diseño de una solución orientada a servicios. Al diseñar un servicio orientado a la solución y construyendo SOA, algunas preguntas importantes son: ¿Cómo los servicios deben ser diseñados?, ¿Cómo debe ser definida la relación entre servicios?, ¿Cómo deben diseñarse las descripciones del servicio? y ¿Cómo deben diseñarse los mensajes? Ocho principios de diseño de SOA se han definido (Erl, 2007). No todos los principios de diseño pueden realizarse simultáneamente, porque los principios se interrelacionan. Esto significa que para lograr un principio se puede requerir la realización de otro principio. Por ejemplo, para alcanzar el acoplamiento flexible, el servicio de contrato debe estar presente. Cinco principios establecen las

características de diseño servicio concreto y los tres restantes actúan más como influencias reguladoras.

### 2.2.3 Abstracción de la capa de servicios

Con el fin de lograr la reutilización, no es posible crear servicios ágiles que se adaptan tanto para negocios como para aplicaciones de consideraciones lógicas simultáneamente. Por lo tanto, las capas especializadas de servicios tienen que ser construidas. Estas capas son: la capa de servicios de aplicación, la capa de servicios de negocio y la capa de servicios de orquestación. La Figura 2-3 muestra estas capas entre la capa de procesos de negocio y la capa de aplicación.

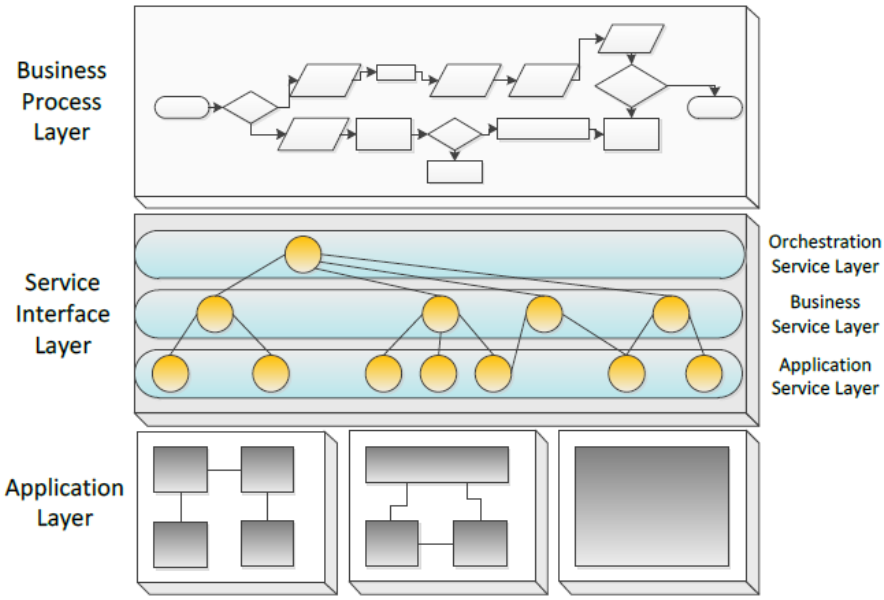



**Figura 2-3 Capas de servicios en SOA (Erl, 2007)**

De acuerdo a Erl (2007) la capa de *servicios de aplicación* proporciona funciones reutilizables que están relacionadas con el procesamiento de datos en un entorno de aplicaciones. Los servicios de aplicación son altamente específicos de tecnología. Los tipos de servicios de aplicaciones más comunes son servicios de utilidad y servicios de envoltura. Los servicios de utilidad son servicios genéricos que proporcionan las operaciones reutilizables para servicios de negocio permitiendo completar las tareas centrada en el negocio. Los servicios de envoltura se utilizan típicamente para los propósitos de integración. Ellos encapsulan la lógica de los sistemas heredados. Los servicios que contienen tanto aplicación como lógica del negocio se denominan servicios híbridos.

El propósito de la capa de procesos de negocio es introducir servicios que se concentran en la representación de la lógica de negocio. Los servicios de negocios siempre son implementaciones del modelo del servicio de negocio. Sin embargo, los servicios de negocio también pueden clasificarse como un servicio de controlador o un servicio de utilidad. Es muy probable que los servicios empresariales actúen como reguladores que se componen de varios servicios de aplicaciones para ejecutar su lógica de negocio.

Los modelos de *servicios de negocio* incluyen los modelos de negocio centrados en las tareas y los modelos de negocio centrados en la entidad. Los servicios de negocio centrados en las tareas encapsulan una lógica de negocio que se específica de un determinado proceso de trabajo o negocio. Los servicios de negocios centrados en la entidad encapsulan una entidad de negocios específicos (por ejemplo, una hoja o una factura) teniendo así un mayor potencial de reutilización.

La capa de *servicios de orquestación* consiste en uno o más servicios de proceso que componen los servicios en los niveles inferiores (servicios de negocio y aplicaciones) según las reglas de negocio y la lógica empresarial que está alojada dentro de las definiciones del proceso.

La capa de *servicios de orquestación* evita la necesidad de otros servicios, administra los detalles relacionados con la interacción que se requieren para garantizar la correcta ejecución de las operaciones de servicio. Los servicios de procesos residen en la capa de orquestación y otros servicios que proporcionan conjuntos específicos de funciones, independientes de las reglas de negocio y la lógica específica del escenario a componer. Los servicios de procesos pueden clasificarse como servicios de control porque componen otros servicios para ejecutar la lógica del proceso de negocio. Los servicios de procesos también pueden convertirse en servicios públicos si el proceso que se ejecuta puede considerarse reutilizable. Las reglas de negocio y servicios de ejecución de la lógica de la secuencia se abstraen de otros servicios de orquestación.

### 2.2.4 Beneficios de SOA

Una correcta implementación de SOA resulta en una arquitectura empresarial ágil y reutilizable; lo cual significa que la lógica de aplicación será reutilizable, y que los procesos de negocio pueden cambiarse sin requerir cambios en todos los sistemas (Erl, 2007). La Figura 2-4, muestra el diseño de una aplicación, que automatiza los procesos de negocio, siguiendo tanto un enfoque tradicional así como un enfoque SOA. Siguiendo un enfoque tradicional, la automatización del proceso de negocio "D" requerirá de la integración de dos aplicaciones a través

de varias conexiones punto a punto; lo que producirá un sistema difícil de mantener y adaptar a nuevas necesidades del negocio. Por otro lado, siguiendo un enfoque basado en SOA, la automatización del proceso de negocio "E" (igual al "D") requiere de la creación de una composición de servicios mediante añadir un nuevo servicio y reutilzar servicios de un inventario de servicios; lo cual resulta en un sistema más fácil de mantener y gestionar. Adicionalmente, un enfoque basado en SOA generalmente requiere de menos código, el necesario para la integración de los servicios a reutilizar.

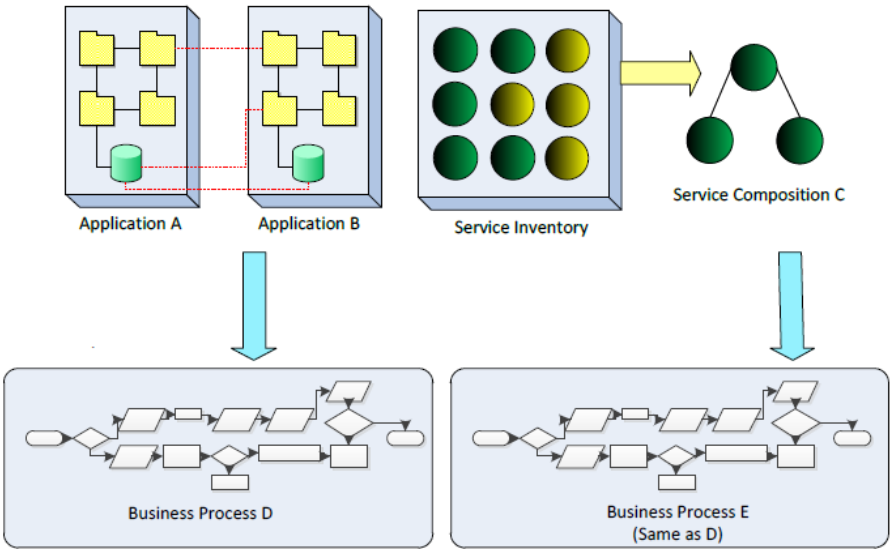



**Figura 2-4 Modelo Tradicional vs. Modelo SOA (Chou, 2010)**

Según (Erl, 2007) los objetivos estratégicos de SOA ) están relacionados con los beneficios resultantes (ver Figura 2-5); en donde los segundos (p. ej., incremento del retorno de la inversión o ROI, aumento de la agilidad organizacional, y reducción de la carga de TI) dependen del logro de los primeros (p. ej., aumento de la federación, aumento de la interoperabilidad intrínseca, incremento de las opciones de diversidad de proveedores, y aumento de la alineación negocio-tecnología).

Por otro lado, de acuerdo con Endrei y otros (2004) SOA brinda, con el fin de tener éxito en los escenarios dinámicos de negocio, los siguientes beneficios a las organizaciones:

- *Aprovechar los activos actuales*: SOA provee una capa de abstracción que permite a las organizaciones continuar usando sus activos actuales: servicios de software que proveen funcionalidades de negocio.

- *Facilidad de integrar y gestionar complejidad*: El punto de integración en SOA es la especificación del servicio, no la implementación. Esto provee transparencia en la implementación y minimiza el impacto de cambios en infraestructura e implementación.

- *Mayor capacidad de respuesta y tiempo de salida al mercado más rápido*: La habilidad de componer nuevos servicios a partir de servicios existentes provee una ventaja distintiva a las organizaciones que requieren ser ágiles para atender las necesidades del negocio. Aprovechar los componentes y servicios actuales reduce el tiempo necesario para ejecutar las actividades del ciclo de vida de desarrollo, tales como obtención de requisitos, diseño, desarrollo y pruebas.

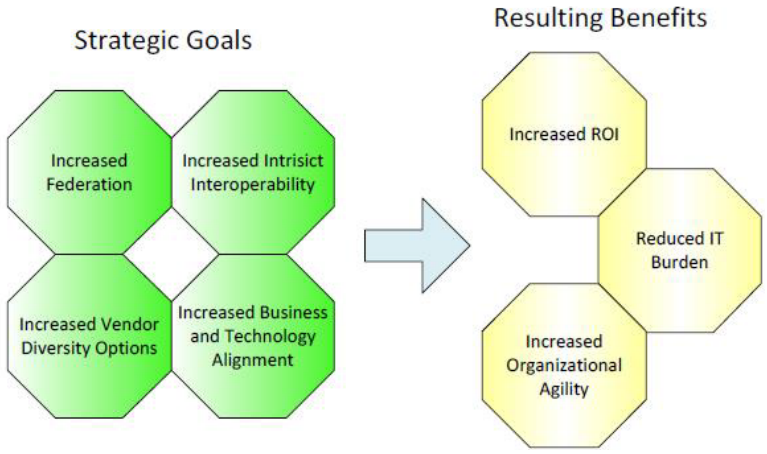



**Figura 2-5 Objetivos estratégicos y beneficios de SOA (Chou 2010)**

- *Reduce costos e incrementa el reúso*: Con los servicios esenciales del negocio expuestos de manera débilmente acoplada, éstos pueden ser más fácilmente utilizados y combinados en base a las necesidades. Esto significa menor duplicación de recursos, mayor potencial para el reúso, y menores costos.

- *Estar listo para lo que vendrá*: SOA permite a las organizaciones estar listas para el futuro. Los procesos del negocio, los cuales están compuestos por una serie de servicios de negocio, pueden ser creados, cambiados y gestionados más fácilmente para cumplir las necesidades actuales.

Desde el punto de vista de gestión, SOA requiere un cambio significativo del clima organizacional y de la forma de trabajo de los equipos de desarrollo; no siendo conveniente para todas las situaciones (Josuttis, 2007). Por ejemplo, no es una buena alternativa para aplicaciones con alto nivel de transferencia de datos

o para aplicaciones de tiempo de vida corto. Aplicar SOA implica tener un claro conocimiento de los procesos de negocio, clasificarlos y extraer ya sea las funcionalidades propias de un proceso de negocio o las funcionalidades comunes para diferentes procesos. Adicionalmente, requiere del establecimiento de estándares que faciliten la comunicación entre servicios.

### 2.2.5 Servicios Web

Los Servicios Web han recibido una amplia aceptación como enfoque tecnológico para la implementación de SOA, en done el canal de comunicación es la Web. Esto es porque los servicios Web proveen un enfoque de computación distribuida para la integración sobre Internet de aplicaciones heterogéneas (Endrei y otros 2004). Los servicios Web no necesariamente se traducen a SOA, y no todo SOA es basada en servicios Web pero la relación entre estas dos tecnologías es importante y son mutuamente influyentes: el impulso de los servicios Web llevó a SOA a los usuarios comunes y la arquitectura de mejores prácticas de SOA ayuda a que las iniciativas de servicios Web sean exitosas (Gartner Inc., 2001).

Las especificaciones de servicios Web son completamente independientes del lenguaje de programación, sistema operativo, y hardware con el fin de promover el acoplamiento débil entre el consumidor del servicio y el proveedor del servicio. Un servicio Web es un componente de software distribuido y accesible a través de Internet; siendo además auto-contenido, autónomo y reusable, el cual encapsula funcionalidad discreta. Por lo tanto, el término servicio Web se refiere a un enfoque tecnológico específico. El W3C's Web Services Architecture Working Group (W3c, 2004) define un servicio Web como:

> *"Una aplicación de software identificada por un Identificador de Recursos Uniforme (Unique Resource Identifier – URI), sus interfaces y enlaces pueden definirse, describirse y descubrirse como artefactos XML. Un servicio Web soporta interacciones directas con otros agentes de software utilizando mensajes basados en XML intercambiados vía protocolos basados en Internet"*

#### *2.2.5.1 Características de los servicios Web*

De acuerdo a Endrei y otros (2004) los servicios Web:

- *Son auto-contenidos*: No se requiere de software adicional al lado del cliente. Por ejemplo, un lenguaje de programación con soporte de clientes XML y HTTP es suficiente para iniciar a trabajar con servicios Web. Al lado del servidor se requiere únicamente un servidor Web y un motor de servlet.

- *Se auto-describen*: Ni el cliente ni el servidor conocen o se preocupan por nada aparte del formato y el contenido de los mensajes de solicitud y respuesta (integración de aplicaciones débilmente acopladas). La definición del formato de mensaje viaja con el mensaje. No se requieren repositorios de meta-datos externos o herramientas de generación de código.
- *Son modulares*: Los servicios Web son una tecnología para implementar y proporcionar acceso a funciones empresariales a través de la Web; J2EE, CORBA, y otros estándares son tecnologías para la implementación de servicios Web.
- *Se pueden publicar, localizar e invocar en la Web*: Los estándares requeridos son:
    - Protocolo Simple de Acceso a Objetos (Simple Object Access Protocol – SOAP), también conocido como protocolo de arquitectura orientada a servicios, una Llamada a Procedimiento Remoto (Remote Procedure Call –RPC) basado en XML y protocolo de mensajes.
    - Lenguaje de Descripción de Servicios Web (Web Service Description Language – WSDL) es una interfaz descriptiva y un lenguaje de enlace de protocolo.
    - Descripción Universal, Descubrimiento e Integración (Universal Description, Discovery, and Integration –UDDI), un mecanismo de registro que puede ser usado para buscar descripciones de servicios Web.
- *Son independientes del lenguaje e interoperables*: La interacción entre un proveedor del servicio y un solicitante del servicio es diseñada para ser completamente independiente de la plataforma y lenguaje. Esta interacción requiere un documento WSDL para definir la interface y describir el servicio, conjuntamente con un protocolo de red (usualmente HTTP). Dado que el proveedor de servicios y el solicitante del servicio no tienen idea de qué plataformas o lenguajes utiliza el otro, la interoperabilidad es un hecho.
- *Son inherentemente abiertos y basados en estándares*: XML y HTTP son la base técnica para los servicios Web. Una gran parte de la tecnología de servicios Web se ha construido utilizando proyectos de código abierto. Por lo tanto, la independencia de los proveedores y la interoperabilidad son objetivos realistas.

- *Son dinámicos*: El e-business dinámico puede convertirse en una realidad utilizando servicios Web porque, con UDDI y WSDL, la descripción y el descubrimiento de servicios Web pueden ser automatizados.
- *Son componibles*: Los servicios Web simples se pueden agregar a los más complejos, ya sea utilizando técnicas de flujo de trabajo o llamando a servicios Web de capa inferior desde una implementación de servicios Web.

Los servicios Web tienen una interface descrita en un formato procesable por el computador (específicamente WSDL). Otros sistemas interactúan con un servicio Web en una manera prescrita por su descripción utilizando mensajes SOAP, normalmente transmitidos utilizando HTTP con una serialización XML junto con otros estándares relacionados con la Web. El uso de estándares abiertos provee amplia interoperabilidad entre soluciones de diferentes vendedores. Estos principios significan que las empresas pueden implementar servicios Web sin tener ningún conocimiento de los consumidores de servicios, y viceversa. Esto facilita la integración justo a tiempo y permite a las empresas establecer nuevos asociaciones de forma fácil y dinámica (Endrei y otros, 2004).

#### *2.2.5.2 Interfaz e implementación*

De acuerdo a (Erl, 2007), la interfaz del servicio es visible para los usuarios del servicio y define la funcionalidad del servicio y cómo acceder a ella. Consiste en la definición del WSDL y la definición del esquema XML, y puede incluir definición de políticas del servicio Web. Los detalles de implementación del servicio están ocultos a los usuarios del servicio, y se divide en lógica de programación lógica y el procesamiento del mensaje. La lógica de programación puede ser lógica heredada que es envuelta por un servicio Web o puede ser lógica especialmente desarrollada para el servicio Web. En este último caso, la lógica es generalmente creada como un componente y es referida como lógica de servicio núcleo (core service logic) o lógica de negocios. Los analizadores, procesadores y agentes de servicio componen la lógica de procesamiento de mensajes de un servicio Web. Ellos pueden manejar los mensajes enviados o recibidos por los servicios Web. Algunas de estas lógicas son proporcionadas por el entorno de ejecución, pero también puede ser construidas a medida. La Figura 2-6 representa los componentes de un servicio.

Una interfaz de servicio puede implementarse por diferentes prestadores de servicios mediante el uso de cualquier lenguaje de programación; cada una podría implementar una lógica de negocio diferentes. Adicionalmente, pueden utilizar una combinación de otros servicios para proporcionar la lógica de negocio requerida. Diferentes proveedores de SOA han desarrollado sus plataformas para

utilizar la tecnología de los servicios Web. Estas tecnologías incluyen IBM Websphere Toolkit, Sun's Open Net Environment y JINITM technology, Microsoft .NET y Novell's One Initiatives, HP e-speak y BEA's WebLogic Integration (Hull y Su, 2005).

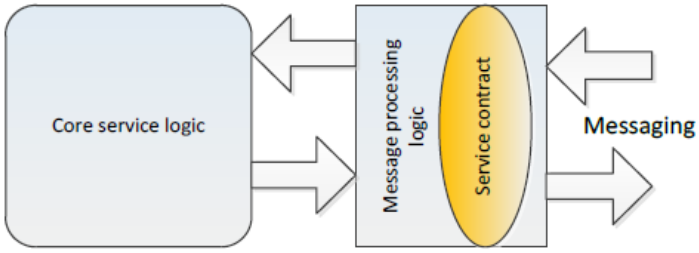



**Figura 2-6 Componentes de un Servicio Web (Chou, 2010)**

### 2.2.6 Service-Oriented Architecture Modeling Language (SoaML)

Las decisiones relacionadas a la arquitectura de software deben ser documentadas de manera útil con el fin de utilizarlas eficientemente (p. ej., para evaluación o mantenimiento). Por un lado, es generalmente aceptado documentar las arquitecturas de software mediante el uso de múltiples vistas (Kruchten, 1995), (Hofmeister y otros 2005); en donde, UML es utilizado para modelar estas vistas. Por otro lado, lenguajes de descripción de arquitecturas especializados (ADLs) han sido desarrollados (principalmente en la comunidad científica). Típicamente, los ADLs ofrecen una sintaxis formal y una semántica para la descripción de arquitecturas de software en términos de componentes en tiempo de ejecución (elementos computacionales) y conectores (abstracciones para interacción entre componentes), y configuraciones (dependencias estructurales entre componentes) (Medvidovic y Taylor, 1997).

Los primeros ADLs se enfocaron en la formalización de modelos arquitectónicos de software, lo cual los hace difíciles de entender y utilizar. Estos modelos requieren que los arquitectos aprendan modelos matemáticos de software (Kandé y otros, 2002), ignorando asuntos del dominio o negocio, sin ser ampliamente utilizados en la industria (Clements y otros, 2003). Por otro lado, una segunda/próxima generación de ADLs, basados en UML, han ganado popularidad y amplia adopción; en donde su notación semiformal permite la inclusión de constructores relacionados al dominio de la aplicación y negocio, y han sido señalados como posibles sucesores de ADLs existentes (Malavolta y otros, 2013).

El Lenguaje de Modelado de la Arquitectura Orientada a Servicio SoaML (Object Management Group Inc., 2012) es un lenguaje, estándar de la OMG, y

ha sido específicamente diseñado para modelar SOA. Este consiste de un perfil UML y un metamodelo que extienden UML ampliándolo con nuevas notaciones y semántica. Las extensiones de SoaML soportan: i) la especificación de servicios y requisitos que deben ser cumplidos, ii) la especificación de servicios a través de sus capacidades funcionales, iii) la definición de consumidores y proveedores de servicios a través de los servicios que consumen y proveen, y iv) la definición de políticas para consumir y proveer servicios (Todoran y otros, 2011). A continuación, se provee una breve descripción de los elementos principales de SoaML. De acuerdo a lo descrito en (Object Management Group Inc., 2012) SoaML extiende a UML en las siguientes áreas principales:

- *Services Architecture*: Una Arquitectura de Servicios define la manera en la que un grupo de participantes trabaja en conjunto proveyendo y consumiendo servicios para un propósito de negocio particular. Sus componentes internos son: participantes (Participant), contratos de servicio (Service Contract), y enlaces (Role Binding).

- *Participant*: Un participante puede cumplir un rol (Role) de proveedor de servicio, consumidor de servicio, o ambos.

- *Service Contract*: Representa un acuerdo entre los participantes (Participant) involucrados acerca de cómo se supone que el servicio será provisto y consumido (interoperación). La definición de un Service Contract incluye los siguientes elementos internos: i) Roles que deben cumplir los participantes (Participants) involucrados en un servicio con el fin de interoperar; ii) interfaces (Interface) provistas y requeridas que modelan explícitamente las operaciones provistas y requeridas para cumplir la funcionalidad del servicio, las cuales deben ser implementadas por los Participants involucrados en el Service Contract con el propósito de cumplir un Role; y iii) protocolo de interacción (Interaction Protocol) que especifica la interacción entre Participants sin definir sus procesos internos.

- *Role Binding*: Enlaza cada uno de los Role a una parte de la Services Architecture para indicar el rol que cada parte cumple en la Services Architecture a la que pertenecen.

La especificación de SoaML define tres enfoques diferentes para especificar servicios. Estos resultan en la definición de diferentes constructores en el perfil de SoaML, los enfoques son (Elvesæter y otros 2011), (Object Management Group Inc., 2012):

- *Basado en Interfaz Única (Simple Interface)*: Utiliza las interfaces UML para especificar interacciones en un solo sentido provista por un Participant.

El Participant recibe operaciones en su puerto y puede proveer resultados al consumidor del servicio. Esta clase de interfaces en un solo sentido puede ser utilizada con consumidores de servicio anónimos y el Participan no hace asunciones a cerca del cliente o la coreografía del servicio.

- *Basado en Contrato de Servicios (Service Contract)*: Extiende una colaboración de UML para especificar interacciones de servicios entre dos o más partes. Este enfoque define especificaciones de servicios (Service Contracts) que especifican cómo los proveedores, consumidores y (potencialmente) otros roles trabajan conjuntamente para intercambiar valor. Service Contracts son generalmente parte de una o más Services Architectures. Este enfoque es más aplicable cuando: i) se define una arquitectura SOA de empresa, una arquitectura de comunidad o una coreografía, y luego se construyen o adaptan servicios para trabajar dentro de esa arquitectura; o ii) cuando hay más de dos Participant involucrados en el servicio.
- *Basado en Service Interface (Interface de Servicios)*: Extiende una clase en UML para especificar interacciones de servicios entre dos o más partes; en donde existen respuestas al consumidor del servicio (callbacks) como parte de la interacción entre ellos. Una Service Interface es definida en términos de proveedor del servicio y especifica la interfaz que el proveedor ofrece, así como la interfaz que éste espera del consumidor. Las interfaces del proveedor y consumidor deben ser las mismas o compatibles.

La diferencia fundamental entre los enfoques basados en Service Contract y los basados en interfaz es que: la interacción entre Participants es definida separadamente de los Participants en un Service Contract; mientras que en los enfoques basados en interfaz la interacción es definida individualmente en el servicio o solicitud de cada Participant.

### 2.2.7 Reconfiguración dinámica

De acuerdo a Buckley y otros (2005), la evolución es un factor crítico en el ciclo de vida de los sistemas de software. Los cambios en los sistemas de software son inevitables; pueden ocurrir en el entorno de la aplicación, en componentes externos provistos por terceros, o en los requisitos. La estrategia para hacer frente a los cambios depende de varios factores. Cuando la actualización del sistema es gestionada fuera de línea, todo el sistema debe ser detenido, actualizado, y reiniciado. Por otro lado, la reconfiguración dinámica tiene el objetivo de gestionar

dichas actualizaciones de manera automática, por lo tanto, minimizando las interrupciones del sistema y evitando que el sistema se detenga.

El propósito de los cambios de software puede ser categorizado en 4 tipos (ISO/IEC 14764 ), (Lehman, 1996): correctivo, perfectivo, adaptativo y preventivo. Según Zhang y otros (2012) desde el punto de vista arquitectónico se tiene que: los cambios *correctivos* corrigen errores identificados en la arquitectura; los cambios *perfectivos* alteran el código de elementos arquitectónicos (p. ej., Componentes/servicios, conectores, etc.) para mejorar atributos no funcionales (p. ej., rendimiento); los cambios *adaptativos* adaptan los elementos arquitectónicos para satisfacer cambios ya sea en el entorno o requisitos (p. ej., Incrementos de software debido a requisitos de nuevas funcionalidades); y finalmente los cambios *preventivos* previenen los efectos secundarios de otros cambios (p. ej., al añadir un nuevo servicio se podría requerir adaptar el servicio con el que interactuará).

Las arquitecturas de software dinámicas pueden ser modificadas durante la ejecución del sistema después de que el sistema ha sido construido, (Medvidovic y Taylor, 2000). Este comportamiento es más comúnmente conocido como evolución en tiempo de ejecución o reconfiguración dinámica. Por lo tanto, la reconfiguración dinámica de arquitecturas tiene como objetivo minimizar las interrupciones del servicio facilitando la aplicación de cambios arquitectónicos mientras el sistema se encuentra en ejecución; evitando que el sistema deba ser detenido, actualizado, y reiniciado para aplicar los cambios. En donde, "ser capaz de hacerlo con un alto nivel de abstracción proporciona a los desarrolladores mucho poder y proporciona extensibilidad, personalización y evolutividad de sistemas de software de gran tamaño"(Medvidovic, 1996). Adicionalmente de acuerdo a Costa-soria (2011) los enfoques de reconfiguración dinámica deberían facilitar la mantenibilidad de los sistemas. La reconfiguración dinámica debería estar aislada de aspectos relacionados a la funcionalidad de los elementos arquitectónicos; esto con el fin de mejorar el mantenimiento y reuso tanto del código de reconfiguración como del código funcional (Costa-Soria et al., 2009).

De acuerdo con Li y otros (2010) los sistemas de software basados en SOA pueden estar compuestos de algunos servicios Web; pudiendo ser: piezas prefabricadas, a lo mejor desarrolladas en diferentes plataformas, publicadas por diferentes proveedores, y posiblemente con diferentes usos en mente. Los sistemas de software basados en SOA son por naturaleza heterogéneos, dinámicos y colaborativos; por lo tanto, pueden soportar fácilmente la reconfiguración dinámica. Adicionalmente, Li y otros (2010) indican que la reconfiguración dinámica puede ayudar a las aplicaciones basadas en SOA a actualizar, modificar, añadir y remover sus funciones, mejorar su rendimiento, mejorar su fiabilidad y robustez mediante la modificación, adición y composición de servicios dinámicamente en

tiempo de ejecución.

Algunos investigadores están de acuerdo en la definición de los siguientes patrones de reconfiguración dinámica (Yin y otros, 2009):

- la desconexión de servicios.
- la creación de nuevas conexiones.
- la modificación de conexiones existentes.
- la adición o eliminación de servicios.
- el reemplazo de servicios o composición de servicios.

## 2.3 Computación cloud

Computación cloud es un modelo de negocio de entrega de recursos de tecnologías de la información y aplicaciones (recursos IT) como servicios accesibles remotamente, bajo demanda y a través de Internet (Leavitt, 2009). En donde, la unión de centros de datos, hardware, software, y almacenamiento es lo que se conoce como una cloud (o nube). En este modelo de negocio los usuarios compran el acceso remoto a recursos IT. La diferencia de este modelo de negocio con el modelo de entrega de recursos tradicional es que en un modelo tradicional los recursos son entregados en forma de productos vendidos o licenciados a usuarios, para luego ser explotados localmente en la infraestructura tecnológica del usuario.

En el dominio de computación en la nube, el término *recursos cloud* hace referencia a cualquier tipo de recurso IT, sea éste virtualizado o real (p. ej., servidores, almacenamiento, redes, sistemas operativos, aplicaciones o middlewares) que se pueda encontrar en un centro de datos. Los *servicios cloud* pueden estar formados por uno o más recursos cloud. Cuando un conjunto de recursos cloud es agregado como un servicio cloud no solo se reduce dramáticamente el número de recursos que un administrador debe gestionar (p. ej., aprovisionar, monitorizar medir, mantener alta disponibilidad, y administrar la carga de trabajo); sino que además oculta sus complejidades internas al administrador. Tratar un servicio cloud como una única unidad de gestión no solo significa que los recursos respectivos son agregados de manera bien definida, sino que las actividades de gestión son tratadas como una parte integral del contexto general en el que se ejecuta el servicio en la nube (Breiter y Behrendt, 2009).

Las ventajas de la computación en la nube sobre el modelo tradicional de entrega de recursos IT incluyen (Mell y Grance, 2011) (Tsai y otros, 2010):

- Modelo de pago pago-por-uso: La facturación es en base al uso real de los recursos; por lo que los costos iniciales son bajos o nulos.
- Independencia de la ubicación: Los usuarios pueden acceder a los recursos desde cualquier ubicación.
- Elasticidad: Capacidad de atender fuertes cambios en la demanda, agregando o disminuyendo dinámicamente el uso de recursos en respuesta a variaciones en la carga de trabajo de servicios o aplicaciones.
- Alta disponibilidad, agilidad, tolerancia a fallos e independencia de los dispositivos.

### 2.3.1 Modelos de servicio cloud

La mayoría de los entornos cloud actuales están construidos haciendo uso de centros de datos modernos en donde los proveedores cloud ofrecen servicios cloud de acuerdo a modelos de servicio. Dependiendo del modelo de servicio cloud, existirán diferentes niveles de control entre proveedores cloud y consumidores cloud; en donde proveedores cloud ofrecen a los consumidores diferentes tipos de operaciones de gestión de servicios cloud. Existen tres modelos de servicio cloud, la Figura 2-7 muestra una vista jerárquica de computación en la nube:

- *Infraestructura como Servicio (Infrastructure as a Service – IaaS)*: Hace referencia al suministro de hardware virtualizado: servidores, recursos de red, o almacenamiento. Desde el punto de vista del consumidor, o usuario, la infraestructura opera como si se tratase de una infraestructura estándar; sin embargo, es uno de muchos entornos virtuales hospedados simultáneamente en la misma infraestructura física.
- *Plataforma como Servicio (Platform as a Service – PaaS)*: Hace referencia al suministro de entornos de ejecución o entornos de desarrollo de aplicaciones sobre los cuales los consumidores cloud despliegan sus propias aplicaciones o componentes. Lo entornos de desarrollo soportan todo el ciclo de vida de diseño, implementación, pruebas y despliegue de aplicaciones Web y servicios; en donde, los desarrolladores no requieren descargar o instalar software de desarrollo localmente en sus ordenadores. Los desarrolladores aprovechan varios beneficios para desarrollas sus aplicaciones en entornos de programación provistos por proveedores PaaS. Beneficios tales como escalamiento automático y balanceo de carga, así como la integración con otros servicios de la plataforma (p. ej.,

correo electrónico, autentificación). Los entornos de desarrollo y ejecución ofrecidos por PaaS hacen menos complicadas las tareas de desarrollo de aplicaciones cloud, aceleran los tiempos de despliegue y minimizan las fallas lógicas en la aplicación(Muppalla y otros 2013).

- *Software como Servicio (Software as a Service – SaaS)*: Hace referencia al suministro de una aplicación como un servicio sobre Internet, en donde los usuarios compran el acceso a la aplicación en lugar de comprar licencias para usarla localmente.

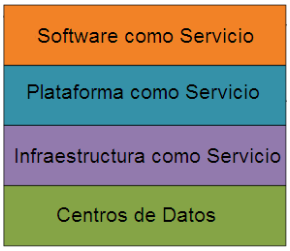



**Figura 2-7 Vista jerárquica de Cloud Computing**

### 2.3.2 Modelos de despliegue de servicios cloud

Los modelos de despliegue describen cómo la infraestructura computacional que entrega los servicios cloud será compartida, y cómo los servicios cloud serán desplegados. Los modelos de despliegue son:

- *Cloud Publica*: Oferta o combinación de ofertas IaaS, PaaS o SaaS que son compartidas entre diferentes compañías (consumidores cloud) con el fin de reducir costos y permitir el uso dinámico de recursos siguiendo un modelo de servicios de pago-por-uso.

- *Cloud Privada*. Oferta o combinación de ofertas IaaS, PaaS o SaaS que son utilizadas internamente por una compañía debido a que requisitos de privacidad u otras restricciones hacen imposible el uso de cloud públicas.

- *Cloud Comunitaria*: Oferta o combinación de ofertas IaaS, PaaS o SaaS confidenciales que son compartidas entre diferentes consumidores cloud que tienen una relación de confianza y se juntan para reducir costos de la oferta cloud, compartir recurso y permitir el uso dinámico de recursos.

- *Cloud Hibrida*: Oferta o combinación de ofertas IaaS, PaaS o SaaS ofrecidas a consumidores cloud con requisitos de privacidad y seguridad altos. Se combina con ofertas compartidas por otros consumidores; ésto,

con el fin de reducir costos de la oferta y permitir el uso dinámico de recursos siguiendo un modelo de pago pago-por-uso.

### 2.3.3 Plataformas de computación cloud

Antes del 2007, las grandes compañías necesitaban mantener su propia infraestructura tecnológica para satisfacer sus necesidades y las de sus clientes; situación que ha cambiado con la aparición de la computación cloud, existiendo mayor aceptación por comprar acceso a infraestructura virtualizada en lugar de comprar hardware propio (Muppalla y otros 2013).

La computación en la nube introduce una nueva forma de pensar a cerca de la arquitectura y diseño; en donde, los desarrolladores no controlan la infraestructura directamente. El desarrollo de aplicaciones cloud es ayudado significativamente con la oferta de entornos de desarrollo utilizados para producir aplicaciones guiadas por abstracciones útiles. Estos entornos de desarrollo han probado su utilidad reduciendo los tiempos de desarrollo, recibiendo amplia aceptación. Por otro lado, las aplicaciones reciben soporte para hacer frente a las variaciones en la demanda de sus servicios mediante entornos de ejecución que escalan los recursos empleados por las aplicaciones añadiendo nuevos recursos, permitiendo obtener aplicaciones efectivas en el costo de operación. La Tabla 2-1 identifica algunos de los mayores proveedores en la industria (Vecchiola y otros, 2009) y la clase de servicios que ellos ofrecen; esta tabla utiliza como fuentes de información los trabajos de Amini y Sadat (2013), y de Xu (2010).

#### *2.3.3.1 Amazon Web Services (AWS)*

Amazon Elastic Compute Cloud (EC2) opera al nivel más bajo de la jerarquía de computación en la nube. Este provee una gran infraestructura de computación y servicios basados en virtualización de hardware. Los usuarios utilizan Amazon Web Services para crear Amazon Machine Images (AMIs) y grabarlas como plantillas desde las cuales ejecutar múltiples instancias. Es posible ejecutar máquinas virtuales ya sea de Windows o Linux y el usuario es facturado por hora y por cada una de las instancias en ejecución. Amazon también provee servicios de almacenamiento mediante Amazon Simple Storage Service (S3); en donde, los usuarios toman ventaja de Amazon S3 para mover archivos grandes a la infraestructura y acceder a ellos desde instancias de máquinas virtuales.

#### *2.3.3.2 Microsoft Azure©.*

La plataforma Microsoft Azure de Microsoft Corporation provee abstracciones de hardware mediante virtualización. Cada aplicación que es desplegada en Mi-

crosoft Azure se ejecuta en una o más máquinas virtuales. La aplicación se comporta como si se tratase de un ordenador dedicado, aunque éste puede compartir recursos físicos como espacio en disco, entradas y salidas de red, o núcleos de procesador con otras máquinas virtuales en el mismo servidor físico; esta abstracción es posible desacoplando la infraestructura de las aplicaciones. Un beneficio clave de la capa de abstracción por encima del hardware físico es la posibilidad de escalabilidad. La virtualización de un servicio permite que éste sea movido a cualquier número de servidores físicos en un centro de datos. Los usuarios son facturados tomando en consideración el tiempo de uso del procesador, el ancho de banda y el almacenamiento utilizado, el número de transacciones ejecutadas por servicio, y el uso de servicios específicos como SQL o servicios .NET.

**Tabla 2-1 Plataformas cloud representativas**

| *Propiedades* | *Amazon EC2* | *Google Cloud Platform* | *Microsoft Azure©* |
|---|---|---|---|
| *Modelo de Servicios* | -IaaS | -IaaS (Google Compute Engine)<br>- PaaS (Google App Engine) | -IaaS<br>-PaaS |
| *Aplicación* | -Elastic Compute Cloud (EC2)<br>-Simple Storage Services (S3) | -Gmail<br>-Google Email Security<br>-Google Docs | -Windows<br>-.NET services<br>-SQL services |
| *Uso* | -Hospedaje de aplicaciones Web<br>-Respaldo y almacenamiento<br>-Computación de alto rendimiento | -Mensajería<br>-Seguridad de sistemas de correo electrónico existentes<br>-Colaboración | -Ofrecer aplicaciones a organizaciones como servicios SaaS<br>-Desarrollo de aplicaciones |
| *Modelo de almacenamiento* | -Simple DB S3 | -Mega Store<br>-BigTable | -SQL Data Services<br>-Servicio de almacenamiento de Azure |
| *Interfaces de acceso al usuario* | -Web APIs y herramientas de líneas de comando | -Web APIs y herramientas de líneas de comando | -Azure Web Portal |
| *Modelo de red* | -Dirección IP elástica | -Estructura de aplicaciones Web de 3 capas | -Componentes de aplicación automáticos |

### *2.3.3.3 Google App Engine*

Es una plataforma que es parte de Google Cloud Platform provista para el desarrollo de aplicaciones Web escalables que se ejecutan sobre la infraestructura de

servidor de Google, el cual ofrece escalado automático de aplicaciones Web según aumenta el número de solicitudes. Google App Engine[2] provee un conjunto de APIs y un modelo de aplicación que permite a los desarrolladores tomar ventaja de servicios adicionales ofrecidos por Google, tales como: correo electrónico, almacenamiento de datos, caché y otros. A través de proveer un modelo de aplicación, los desarrolladores pueden crear aplicaciones en Java, Python, y JRuby. Estas aplicaciones se ejecutan dentro de un entorno computacional sin afectar las aplicaciones en las que se ejecutan y AppEngine se hace cargo de escalarla automáticamente cuando sea necesario. Google provee un servicio gratis limitado y utiliza cuotas diarias y por minuto para medir y cuantificar aplicaciones que requieren un servicio profesional.

#### *2.3.3.4 Criterios para la selección de plataformas*

De acuerdo a Muppalla y otros (2013) las razones para seleccionar un proveedor cloud, de entre los incluidos en esta sección, son:

Razones para utilizar Google App Engine:

- Los plug-in de los servicios de Google como Gmail y Calendar son fáciles de utilizar.
- Es una buena elección si es que se utilizan los lenguajes Java y Python.
- Las aplicaciones probadas localmente se ejecutan al igual que en Google App Engine.
- Permite la ejecución de varias versiones del mismo almacenamiento de datos.

Razones para utilizar Microsoft Azure©:

- Más adecuado para aplicaciones basadas en SOA (Service-Oriented Architecture).
- Provee un entorno de pruebas (staging) que ayuda durante la implementación de aplicaciones.
- Ofrece dos soluciones de almacenamiento – SQL Azure (relacional) y Azure Storage (no relacional).
- Más adecuado para aplicaciones basadas en .NET.

[2] https://cloud.google.com

Razones para utilizar Amazon Web Services:

- Soporta Windows y varias distribuciones de Linux.
- Soporta varios lenguajes, tales como: C#, PHP, ASP.NET, Python, y Ruby.
- Provee balanceadores de carga listos para utilizar, varios tamaños de almacenaje, y permite instalar software personalizado.

### 2.3.4 Soporte al aprovisionamiento y despliegue

Existen proveedores de soluciones o frameworks que facilitan la automatización del aprovisionamiento o despliegue de entornos cloud; por ejemplo Chef[3] y Cloudify[4].

#### *2.3.4.1 Chef*

Es una plataforma de código abierto para la automatización. Proporciona poder y flexibilidad tanto para el aprovisionamiento y el despliegue de servidores como para el despliegue de aplicaciones y servicios a cualquier escala. Utiliza scripts denominados *recipes* (recetas) escritos en un lenguaje específico de dominio basado en Ruby. Las recetas son creadas por desarrolladores con el fin de permitir el despliegue, configuración y aprovisionamiento de componentes. Conjuntos predefinidos de recetas, llamados *cookbooks* (libros de cocina), son creados por los desarrolladores. Un mismo libro de cocina pude ejecutarse en diferentes sistemas operativos debido a que el lenguaje especifico de dominio utilizado para escribirlas es independiente de la plataforma.

#### *2.3.4.2 Cloudify*

Es una solución PaaS que se sitúa entre las aplicaciones y un cloud cualquiera seleccionado (IaaS). Es un proyecto de código abierto que se enfoca en el despliegue y ejecución de aplicaciones en el cloud. Funciona sobre varios proveedores cloud y ofrece características de escalabilidad básicas. Cloudify utiliza recetas para describir una aplicación, sus servicios y sus interdependencias, cómo monitorizarla, y escalar sus servicios y recursos. Para el despliegue de aplicaciones Cloudify propone un modelo inspirado en Chef que involucra los siguientes conceptos.

---

[3] https://www.chef.io/chef/

[4] http://docs.getcloudify.org/

- Receta de Servicios: describe información general a cerca de servicios incluyendo la infraestructura que requieren, cómo deberían ser utilizados y las pruebas para monitorizarlo.
- Servicio: es un grupo de instancias de servicios que conforman una capa de la aplicación.
- Receta de aplicación: describe la configuración, (incluyendo aprovisionamiento y reglas de escalado) de una aplicación y los servicios que la conforman.
- Aplicación: una aplicación es un conjunto de servicios trabajando conjuntamente la cual es descrita en una receta de aplicación. El Cloudify manager desplegado en un cloud permite a los operadores cloud gestionar varias aplicaciones en la misma infraestructura.
- Prueba: son utilizadas para monitorizar el estado del sistema.

### 2.3.5 Aplicaciones cloud *(servicios cloud)*

Las aplicaciones cloud son aplicaciones basadas en servicios que "desde el punto de vista de la ingeniería del software es un software provisto como servicio" (Hamdaqa y otros, 2011); las cuales están conformadas por: un paquete de servicios interrelacionados, la definición de los servicios, y archivos de configuración que contienen información dinámica empleada por los servicios en tiempo de ejecución.

Nguyen (2013) se refiere a estas aplicaciones como "*Aplicaciones basadas en servicios cloud*" y las define como:

> "*Un tipo de aplicación basada en servicios que es desarrollada reutilizando y componiendo servicios cloud a través de las tres capas de la jerarquía cloud (p. ej., IaaS, PaaS, SaaS)*"

En el contexto de esta tesis doctoral, consideramos que un *incremento de software* está constituido por servicios desarrollados para ser integrados en la aplicación cloud en construcción (o para ser reutilizados en otras aplicaciones), así como por servicios ofrecidos por terceros y reutilizados en la aplicación cloud en construcción. Por lo tanto, nuestra definición de aplicación cloud es:

> "*Un tipo de aplicación basada en servicios que es desarrollada creando, reutilizando y componiendo servicios cloud a través de las tres capas de la jerarquía cloud (p. ej., IaaS, PaaS, SaaS)*"

Adicionalmente, en el contexto de este trabajo de tesis nos referimos a las aplicaciones basadas en servicios cloud de manera indiferente como: aplicaciones cloud o servicios cloud.

Las aplicaciones cloud:

- Son aplicaciones distribuidas, generalmente compuestas por servicios Web.
- Se ejecutan consumiendo recursos obtenidos de proveedores cloud. Utilizan el entorno de ejecución provisto por proveedores cloud bajo el modelo de servicios PaaS, o el entorno de ejecución creado sobre la infraestructura provista bajo el modelo de servicios IaaS.
- Sus propietarios compran recursos cloud a proveedores cloud y venden las aplicaciones a sus usuarios
- Su modelo de servicios pago-por-uso se basa generalmente en cálculos que involucran el uso de recursos computacionales y de almacenamiento, así como en métricas de consumo de otros recursos.
- Se acceden ya sea a través de interfaces gráficas de usuario, o desde otras aplicaciones del usuario cuyo código de implementación o API son suministrados por el proveedor de la aplicación cloud.

#### *2.3.5.1 Desarrollo de aplicaciones cloud*

Desde el punto de vista del proceso de desarrollo de software, los enfoques de desarrollo ágiles se alinean al paradigma de computación cloud debido a que combinan entregas rápidas de aplicaciones con retroalimentaciones frecuentes por parte del usuario (Talreja, 2010). Siguiendo estos enfoques las aplicaciones cloud se organizan en un conjunto de incrementos compuestos por uno o más servicios que entregan nuevas funcionalidades al cliente o actualizan funcionalidades existentes. Cada incremento es diseñado, construido, desplegado e integrado separadamente, de acuerdo a una priorización previamente establecida; evitando los largos tiempos que podría requerir desarrollar una aplicación completa y favoreciendo a la agilidad del negocio. Adicionalmente, durante el despliegue de los servicios incluidos en un incremento, los desarrolladores seleccionan un conjunto de recursos IT, ofrecidos por proveedores soluciones IaaS o PaaS, sobre los cuales se desplegarán y ejecutarán los servicios. Sin embargo, en un enfoque de desarrollo incremental, mientras más componentes se añaden al sistema, la integración se vuelve más compleja de construcción a construcción; en donde, entre las clausas de esta coplejidad están las deficiencias en el diseño para la integración y la ruptura de código (Tan y otros, 2009).

### 2.3.6 SOA y computación cloud

SOA ha emergido como un enfoque para la entrega de software siguiendo un modelo de servicios SaaS (Guha, 2013). SOA facilita el diseño, construcción, despliegue e integración de servicios de manera independiente de las aplicaciones y de las plataformas en las cuales se ejecutan. Adicionalmente, SOA organiza y gestiona servicios Web en el cloud (Vouk, 2008). SOA y la computación cloud se complementan, SOA es un estilo arquitectónico que propone principios a cerca de como diseñar las aplicaciones empresariales (Krill, 2009); mientras que la computación cloud es una arquitectura de despliegue. Por un lado, SOA promueve una manera de diseñar, desarrollar, desplegar y gestionar aplicaciones caracterizadas por servicios de grano grueso que representan una funcionalidad reusable. Por otro lado, los principios de SOA se ven reflejados en el cloud el cual adapta eficientemente el aprovisionamiento de recursos a la demanda dinámica de servicios por parte del usuario. Esto lo hacen los entornos cloud a través de la agilidad, escalabilidad, elasticidad, aprovisionamiento rápido de recursos IT bajo demanda, y virtualización de hardware (Kyriazis y otros, 2014).

La combinación de SOA y la computación cloud introduce un nuevo enfoque para la construcción de aplicaciones distribuidas y basadas en servicios. La composición de aplicaciones utilizando servicios basados en cloud (SaaS) permite a los desarrolladores de aplicaciones tomar ventaja de sus diferentes funcionalidades y calidades para construir una solución a la medida que satisfaga requisitos de negocio específicos (incluyendo requisitos de calidad y términos SLA). Sin, embargo el desarrollo de servicios cloud introduce retos relacionados con la heterogeneidad de las tecnologías de proveedores cloud: mecanismos de acceso y aprovisionamiento de recursos cloud, así como requisitos de interoperación entre servicios desplegados en diferentes proveedores cloud. Finalmente , SOA implementa integraciones débilmente acopladas, lo que facilita la integración y reduce el costo en entornos de computación distribuidos (Moinuddin, 2007); favoreciendo a los procesos de actualización de productos de software y reconfiguración dinámica en entornos cloud.

## 2.4 El lenguaje SPEM2 para definir procesos software

Software & Systems Process Engineering Metamodel Specification (SPEM2.0) es un estándar de la OMG dedicado al modelado de procesos de software. Su objetivo es proporcionar a las organizaciones con mecanismos para modelar, intercambiar, documentar, gestionar y presentar sus métodos y procesos de desarrollo (Object Management Group Inc., 2008a).

SPEM 2.0 se basa en un metamodelo construido conforme MOF que reutiliza la infraestructura de UML2.0, cuyo objetivo es proveer de una solución de compromiso que permita modelar métodos y procesos de ingeniería de software sin añadir características específicas a un dominio o disciplina; y que al mismo tiempo permita el modelado en diferentes estilos, culturas, nivel de formalismo o paradigmas de ciclo de vida. La idea principal de SPEM 2.0, descrita en la Figura 2-8, es representar los procesos en base a elementos básicos que permiten especificar un proceso en los términos de "Quien (rol) realiza qué (tarea) con el fin de obtener, a partir de una serie de entradas (productos de trabajo), un resultado (productos de trabajo)" (Ruiz y Verdugo, 2008). En consecuencia, los conceptos básicos de SPEM 2.0 son:

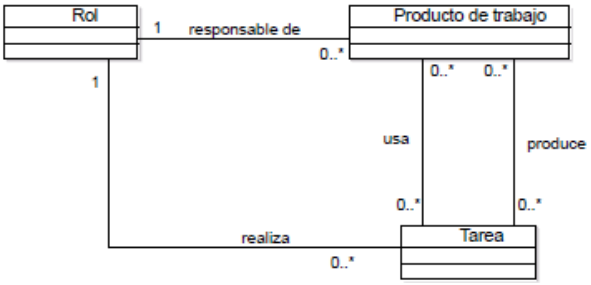



**Figura 2-8 Conceptos utilizados para la descripción de procesos con SPEM 2.0**

- *Rol*: Representan la responsabilidad de realización de las tareas. Quién realiza una tarea para obtener, a partir de unas entradas (productos de trabajo) un resultado (productos de trabajo).
- *Producto de trabajo*: Representan las entradas requeridas o las salidas que se producen en una tarea.
- *Tarea*: Representan el esfuerzo que se realiza.

En este trabajo de tesis se utilizará la notación de SPEM 2.0 dado que es un estándar conocido en el campo de la ingeniería del software. La Figura 2-9 provee una vista general de cómo los componentes clave definidos en SPEM 2.0 son posicionados para representar el contenido de un método o proceso. El Contenido del Método es expresado usando definiciones de: productos de trabajo, roles y tareas; adicionalmente utilizan guías. Las guías o elementos de orientación (p. ej., documentos técnicos, listas de verificación, ejemplos o planes de trabajo) están ubicados en la intersección entre Contenido del Método y Procesos debido a que estas pueden ser definidas para proveer orientación a cerca del contenido del método, así como para procesos específicos. Por otro lado, los elementos utilizados para representar Procesos son actividades que, a más de ser

el elemento principal de los procesos, pueden ser anidadas para definir estructuras de desglose, así como relacionarse una con otra para definir un flujo de trabajo. Las actividades gestionan referencias al contenido del método, las cuales son representadas mediante conceptos de correspondencia "uso". La Tabla 2-2 resume las primitivas, utilizadas en este trabajo de tesis, para el modelado de contenidos del método y procesos definidos en el estándar SPEM v2.0.

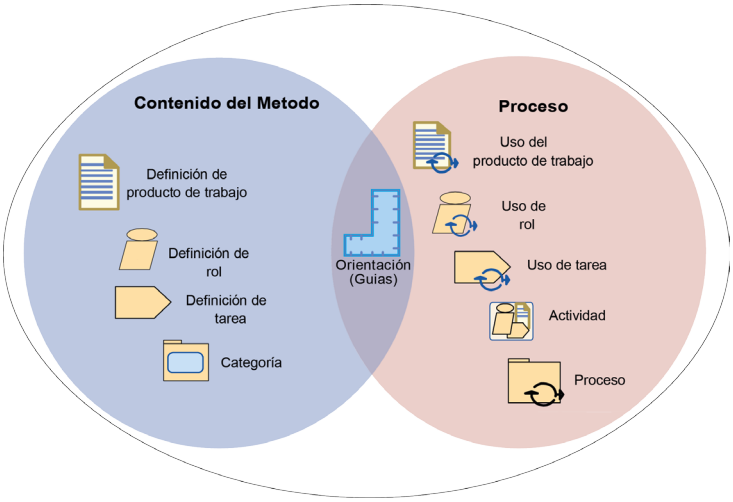



**Figura 2-9 Terminología clave de SPEM 2.0 y correspondencias entre contenido del método y procesos**

## 2.5 Espacios tecnológicos

La tendencia MDA está alineada con los principios DSDM, el cual considera a los modelos como el principal activo en el proceso de desarrollo de software. Los modelos recogen las propiedades que describen los sistemas de información a un alto nivel de abstracción, lo cual permite el desarrollo de las aplicaciones de manera automática mediante técnicas de generación de código.

En este proceso los modelos constituyen artefactos de software que experimentan refinamientos desde el espacio del problema (modelos que capturan los requisitos de la aplicación) al espacio de la solución (modelos que especifican el diseño, desarrollo y despliegue del producto software final). En este contexto es esencial tener el soporte apropiado de herramientas que permitan definir metamodelos y modelos, y que provean la implementación de estándares y lenguajes necesarios. Sin el soporte adecuado de herramientas el enfoque de DSDM no puede ser implementado en ningún proceso de desarrollo de software.

**Tabla 2-2 Primitivas de modelado de SPEM v2.0**

| Icono | Nombre | Descripción |
|---|---|---|
|  | Definición de rol | Conjunto de habilidades, competencias y responsabilidades de un individuo o grupo |
|  | Definición de tarea | Describen una unidad de trabajo a ser asignada o manejada. Identifica el trabajo que se lleva a cabo por los roles. Una tarea puede ser desglosada en pasos. |
|  | Definición de producto de trabajo | Definen productos usados o producidos por las tareas. Pueden ser: artefactos de naturaleza tangible o artefactos entregables. Pueden asociarse entre ellos con relaciones de agregación, composición o impacto. |
|  | Categoría | Clasifican elementos como tareas, roles o productos basándose en criterios establecidos por el ingeniero de procesos. Hay diferentes tipos de categorías: grupos de roles, disciplinas (para tareas) o dominios (para productos). |
|  | Guías | Proveen información adicional relacionada a otros elementos. Existen sub-tipos de guías: activos reusables, guías o plantillas de documentación, entre otros. |
|  | Uso de rol | Representan un rol que lleva a cabo una tarea o actividad de un proceso. Hacen referencia a la definición de un rol (Contenido del Método). |
|  | Uso de tarea | Representan una tarea en un proceso definido. Hacen referencia a la definición de una tarea (Contenido del Método). |
|  | Uso de producto de trabajo | Representan una entrada o salida de una actividad o tarea. Hacen referencia a la definición de un producto (Contenido del Método). |
|  | Actividad | Representan un conjunto de tareas que se ejecutan dentro de un proceso con sus respectivos roles y productos de trabajo. |
|  | Paquete de proceso | Representan un paquete que contiene todos los elementos de un proceso. |
|  | Paso | Describen una parte significativa y consistente del trabajo general descrito para una Definición de Tarea. Representa todo el trabajo que se debe hacer para lograr el objetivo de desarrollo de la Definición de Tarea. |

Tradicionalmente, las herramientas que proveen soporte para las tareas de modelado fueron ya sea específicas de contexto, por ejemplo, específicas de meta modelado (ontologías, base de datos relacionales, etc.) o específicas de tecnología (Gómez, 2012). Con la llegada de MDA y MOF, nuevas herramientas de modelado están siendo diseñadas para ser fácilmente personalizadas, proveyendo soporte para definir metamodelos y modelos individualizados. La adopción de MOF está también incrementado la importancia de DSLs. Estas herramientas de modelado están siendo diseñadas siguiendo un paradigma en el cual la modularidad y extensibilidad son atributos clave a maximizar. En este contexto, la plataforma Eclipse (Eclipse Foundation, 2015), y sus proyectos relacionados brindan el soporte necesario para el DSDM y sus estándares asociados (MDA, MOF, QVT, OCL, etc.), y son utilizados en la implementación de las herramientas de soporte generadas en este trabajo de tesis.

### 2.5.1 Eclipse

Eclipse es entorno de desarrollo de código abierto que ofrece además un entorno de ejecución genérico que implementa un modelo dinámico y completo de componentes. Su modelo de componentes permite la construcción de aplicaciones de software de manera modular. La plataforma Eclipse provee soporte mediante un conjunto de herramientas (componentes o plug-ins) que son incorporadas y definen IDEs y configuraciones que extienden las funcionalidades de la plataforma. Las herramientas de soporte construidas en este trabajo de tesis son implementadas como proyectos de Eclipse utilizando los compontes que éste provee. Los proyectos de Eclipse son implementados en Java y se ejecutan en diferentes sistemas operativos, incluyendo Windows, Linux, o Mac OS X.

La naturaleza de esta herramienta es un IDE abierto y ampliable para múltiples propósitos. En este trabajo será de particular interés el Eclipse Modeling Framework (EMF), un marco para la gestión de modelos y generación de código de modelos que se describen en XMI. EMF es descrito con más detalle en el apartado siguiente.

### 2.5.2 Eclipse Modeling Framework (EMF)

El proyecto EMF es un marco de modelado y generación de código que facilita la creación de aplicaciones basadas en modelos de datos estructurados. El desarrollo de aplicaciones inicia con la especificación de su modelo descrito en XMI; para luego producir un conjunto de clases Java que representan el modelo, EMF proporciona las herramientas necesarias para la generación de estas clases.

Los modelos se pueden especificar utilizando Java anotado, documentos XML, o herramientas de modelado como Rational Rose (luego ser importados a EMF). Lo más importante es que EMF proporciona la base para establecer la interoperabilidad con otras herramientas y aplicaciones basadas en EMF. Con respecto a la relación de EMF con la OMG y MOF, EMF empezó como una implementación de la especificación madura de MOF de la experiencia adquirida en el desarrollo de herramientas por los desarrolladores de Eclipse. EMF puede verse como una implementación eficiente de la utilización conjunta de la API de MOF. Sin embargo, para evitar confusión, el metamodelo de EMF está basado en el núcleo de MOF y es llamado Ecore (Eclipse Foundation, 2015).

### 2.5.3 El lenguaje de transformación ATL

ATLAS Transformation Language (ATL) (ATL, 2014) es un lenguaje de dominio específico para la especificación de transformaciones de un modelo a otro. ATL está inspirado en los requisitos del estándar QVT (Object Management Group Inc., 2008b), y se basa en el formalismo de OCL (Object Management Group Inc., 2006). ATL es un lenguaje de transformación M2M que es publicado bajo los términos de la licencia de código abierto Eclipse Public License (EPL).

Aunque ATL está alineado con el estándar QVT, éste no provee la misma arquitectura ni un lenguaje puramente declarativo, en su lugar este provee un lenguaje hibrido (imperativo / declarativo) para realizar transformaciones de modelos conforme a MOF. El estilo de especificación de transformaciones declarativo tiene ventajas; generalmente está basado en la especificación de las relaciones entre los patrones origen y destino, y por lo tanto tiende a estar más cerca de la forma en que los desarrolladores perciben intuitivamente una transformación (Jouault y otros, 2008).

Las transformaciones ATL son unidireccionales: es decir, durante la ejecución de una transformación, el modelo origen puede ser navegado pero no actualizado; mientras que el modelo destino no puede ser navegado (Jouault y otros, 2008). Para implementar una transformación bidireccional ATL requiere un par de transformaciones, una para cada dirección. Los modelos origen y destino de ATL pueden expresarse en el formato de XMI, mientras que los metamodelos origen y destino pueden expresarse tanto en formato XMI como en la notación más conveniente.

Las definiciones de transformación en ATL se realizan en módulos (module). Un módulo contiene una sección obligatoria de cabecera (header), una sección

de importación (import), y un número de ayudantes (helpers), y reglas de transformación (transformation rules). La sección de cabecera da el nombre al módulo de transformación y declara los modelos origen y destino.

### 2.5.4 Acceleo

Acceleo (Obeo, 2005) es una implementación pragmática del estándar MOF Model to Text Language (MTL) del OMG. Este plug-ing se inició en la empresa francesa Obeo, la unión entre el estándar MTL, y la experiencia de su equipo con la generación de código industrial y los últimos avances en el campo de investigación en el campo de M2T. Acceleo permite crear transformaciones M2T y ofrece ventajas excepcionales: alta capacidad personalización, interoperabilidad, gestión de la trazabilidad, generación incremental (el código puede ser generado, modificado, y regenerado), y otras.

### 2.5.5 Microsoft XML Document Transformation

Microsoft Azure© emplea documentos en formato XML para las configuraciones (Microsoft, 2017a). Para poder modificar estos archivos de configuración, Microsoft permite a los desarrolladores el empleo de ficheros XDT (Microsoft, 2017b).

Los ficheros XDT utilizan el metalenguaje XML pero con un namespace propio (XDT) que entiende el motor que utiliza Azure. Este namespace es muy simple ya que únicamente agrega dos atributos Locator y Transform.

El atributo Locator permite indicar que parte del archivo de configuración va a modificarse, para ello Locator provee de varias funciones:

- Condition: Especifica una expresión XPath que se combina con la expresión XPath del elemento actual. Aquellos elementos que coinciden con la expresión XPath combinada son seleccionados.
- Match: Selecciona aquellos elementos que tienen el mismo valor indicado en uno de los atributos indicados.
- XPath: Permite especificar una expresión XPath absoluta, a diferencia de Condition esta expresión no se combina con la expresión XPath actual.

El atributo Transform permite a los desarrolladores indicar cuál va a ser la acción a realizar sobre el elemento seleccionado (mediante Locator). Para ello el atributo Transform tiene varias opciones: Replace, Insert, InsertAfter, InsertBefore, Remove, RemoveAll, RemoveAttributes, SetAttributes.

## 2.6 Resumen

En este capítulo se ha proporcionado una breve introducción a los marcos tecnológicos que se utilizan en este trabajo de tesis: ingeniería basada en modelos, arquitectura de software orientada a servicios y reconfiguración dinámica, computación cloud y desarrollo incremental de aplicaciones cloud, los lenguajes SPEM para la definición de procesos software, y marcos tecnológicos.

Hemos presentado una vista general de los estándares más importantes del DSDM propuestos por la OMG y los hemos incluido con el propósito de hacer esta tesis un documento auto contenido. Un objetivo de este capítulo es brindar también una introducción concisa al DSDM y los estándares y lenguajes relacionados. Por ejemplo: MOF para describir metamodelos, modelos e instancias, y OCL para definir restricciones en los modelos.

Dentro de la sección de computación cloud, un modelo de negocio de entrega de recursos de tecnologías de la información y aplicaciones, hemos presentado una introducción a sus principales características. En el desarrollo de aplicaciones a ser desplegadas en entornos cloud no solo es importante sacar ventaja de las facilidades y flexibilidad en cuanto al aprovisionamiento de recursos tecnológicos requeridos por las aplicaciones; sino proveer mecanismos que permitan hacer frente a la heterogeneidad en las tecnologías (p. ej., lenguajes de programación, APIs, protocolos, etc.) impuestas por los diferentes proveedores. Para hacer frente a esta heterogeneidad la aplicación de un enfoque DSDM permite abstraer las características de las ofertas de recursos y mecanismos ofertados por los diferentes proveedores cloud, y en base a estas abstracciones documentadas en modelos, generar el código requerido en las diferentes actividades del desarrollo e integración de servicios.

Por otra parte, hemos presentado una introducción a las arquitecturas de software basadas en servicios (SOA). Un estilo arquitectónico que ha tomado gran importancia en la construcción de sistemas de software debido a que sus principios promueven la creación de aplicaciones con características de flexibilidad, agilidad y reutilización. Adicionalmente, la aplicación de sus principios no solo facilita la organización, escalabilidad, y gestión de servicios de aplicaciones cloud; sino que permite adaptar eficientemente el aprovisionamiento de recursos a las demandas dinámicas de usuarios de servicios cloud, y favorece la reconfiguración arquitectónica dinámica.

Además, se ha presentado el lenguaje SPEM 2.0 el cual se utiliza en la definición de los procesos de software de este trabajo de tesis. SPEM tiene la ventaja de ser un estándar de la OMG.

Por último, dentro de los espacios tecnológicos relacionados a este trabajo de tesis se han presentado los espacios tecnológicos de Eclipse y EMF. Se incluye una breve descripción de las herramientas disponibles en el mercado para llevar a cabo un proceso complejo de DSDM. En este caso, las herramientas que son de interés para los propósitos de este trabajo de tesis deben ser interoperables y deben cumplir con los estándares para el metamodelado, modelado, transformación de modelo a modelo y de modelo a código.

Con todo esto, se ha definido el marco conceptual sobre el que se sustenta esta tesis y que permite entender con mayor detalle, el problema que se pretende resolver.

# Capítulo 3. Estado del arte

En este capítulo se analizan los métodos, técnicas y enfoques existentes para el soporte a la reconfiguración dinámica e incremental de servicios cloud. El diseño o implementación incorrectos de software, conjuntamente con el uso inadecuado de los recursos que una aplicación comparte con otras, podría resultar en aplicaciones que no cumplen con sus requisitos (Oriaku y Lami, 2012). Por lo tanto, a pesar de que la reconfiguración dinámica se produce en tiempo de ejecución es importante conocer el soporte que se brinda al desarrollo de aplicaciones cloud durante las actividades de diseño e implementación. En este contexto, la estructura de este capítulo es el siguiente:

La sección 3.1 analiza aproximaciones que brindan soporte tanto al desarrollo de aplicaciones cloud como a la migración de aplicaciones a entornos cloud. Además, se identifica la manera en la que las propuestas hacen frente a la integración incremental de servicios.

La sección 3.2 analiza aproximaciones para la descripción de arquitecturas de servicios cloud y los lenguajes que soportan la especificación de diferentes aspectos en el desarrollo de aplicaciones cloud.

La sección 3.3 se analizan las propuestas que hacen frente a la reconfiguración dinámica de arquitecturas de servicios cloud producida por la integración de incrementos de software.

## 3.1 Enfoques de desarrollo y migración

### 3.1.1 Enfoques basados en middleware

Estas propuestas tienen como objetivo proveer una plataforma para el desarrollo de aplicaciones multi-cloud que pueden interoperar entre ellas y con los servicios ofrecidos por el entorno cloud. Permiten a los desarrolladores construir aplicaciones multi-cloud abstrayendo su implementación de las plataformas cloud en las que serán desplegadas. Para ello, definen su propias APIs, las cuales son invocadas desde las aplicaciones desarrolladas. Las invocaciones a las APIs son transformadas internamente a llamadas a servicios específicos de una plataforma. A continuación, se presentan algunos enfoques middleware que soportan procesos de desarrollo y migración.

La plataforma de código abierto MoSAIC (Di Martino y otros, 2011) permite a los desarrolladores construir aplicaciones cloud de manera flexible e independiente del proveedor cloud. El principal beneficio de utilizar esta plataforma (Di Martino y otros, 2011) es el acceso transparente y simple a recursos cloud heterogéneos. Para ello, la plataforma middleware crea una capa de abstracción entre las aplicaciones desarrolladas y la plataforma cloud en las que serán desplegadas; esta capa abstrae la manera en que los recursos son accedidos y conectados entre sí. Las aplicaciones son construidas haciendo uso de APIs cuyas invocaciones son internamente transformadas a llamadas de servicios específicos de la plataforma en la que los servicios son desplegados. La plataforma de código abierto permite a las aplicaciones negociar servicios cloud de acuerdo a los requisitos del usuario. Las aplicaciones utilizan una ontología cloud para especificar sus requisitos de servicios y comunicarlos a la plataforma mediante APIs. La plataforma implementa un mecanismo de intermediación multi-agente que gestiona tanto negociaciones SLA con cada plataforma cloud, así como el despliegue de cada aplicación. Sin embargo, esto se alcanza al definir APIs propias sobre las cuales se desarrollan las aplicaciones, por lo tanto, las invocaciones a las APIs son internamente transformadas a llamadas de servicios específicas de plataforma. A pesar de que ofrecen como beneficio el acceso a recursos cloud heterogéneos, evitando la dependencia con soluciones específicas de los proveedores cloud, los desarrolladores estarán atados al uso del API.

Cloud4SOA (Kamateri y otros, 2013) provee un marco de referencia interoperable de código abierto para desarrolladores y proveedores PaaS. Cloud4SOA tiene como objetivo dar soporte a los desarrolladores para desplegar y monitorizar sus aplicaciones con el propósito de reducir el riesgo de dependencia con un proveedor. Propone un sistema de gestión y monitorización entre PaaSs para

aplicaciones hospedados en múltiples clouds a fin de asegurar que su rendimiento cumple consistentemente con los requisitos y que los recursos cloud están siendo utilizados efectivamente. Ofrece adaptadores que conectan múltiples plataformas cloud y un servicio unificado de monitorización que es la base de un sistema de gestión de SLA orientado a las aplicaciones. La monitorización está basada en métricas unificadas y se realiza por separado para cada aplicación. Cloud4SOA ofrece una API armonizada que cubre funcionalidades ofrecidas por una variedad de proveedores PaaSs. Por otro lado, adaptadores específicos de PaaS son responsables de implementar la lógica de conexión entre la API homogénea y las APIs de las plataformas PaaS. Las tareas de adaptación también permiten que los servicios cloud sean invocados de manera uniforme. Sin embargo, no tiene en cuenta la interoperabilidad entre los servicios que conforman la aplicación, los cuales podrían estar distribuidos en diferentes proveedores cloud.

Service Oriented Cloud Computing Architecture (SOCCA) (Tsai y otros, 2010) es una arquitectura de cuatro capas que soporta tanto SOA como cloud computing. Ésta permite que una aplicación se ejecute en diferentes proveedores e interoperen entre sí. Facilita la migración desde un proveedor cloud a otro y el re-despliegue en diferentes proveedores cloud mediante separar los roles de proveedor de lógica de servicios y proveedor de servicios de hosting / cloud. En SOCCA los servicios son publicados como paquetes de servicios re-desplegables. Un paquete de servicios puede tener la siguiente información: código compilado, código fuente y archivo de configuración. Los recursos son componentizados, estandarizados y combinados a fin de construir un ordenador virtual multiplataforma. Una capa de mapeo ontológico es configurada en los servicios como un mecanismo para enmascarar la diferencia entre proveedores cloud. Por otro lado, intermediarios cloud interactúan con la capa de mapeo ontológico para desplegar aplicaciones en una u otra cloud dependiendo de parámetros (p. ej., presupuesto, términos SLA y requisitos de QoS negociados con cada proveedor cloud). Las aplicaciones SOCCA pueden ser desarrolladas utilizando las interfaces estándar provistas por la arquitectura o las APIs específicas de plataforma definidas por los proveedores cloud. Sin embargo, a pesar de que SOCCA soporta SOA, cuyos principios facilitan la evolución arquitectónica, éste proyecto no provee mecanismos para gestionar los cambios arquitectónicos que se producirán durante el despliegue/re-despliegue de servicios.

### 3.1.2 Infraestructuras cloud computing

El proyecto RESERVOIR (Rochwerger y otros, 2009), tiene como objetivo facilitar el despliegue, gestión y ejecución de servicios en múltiples dominios de

administración. RESERVOIR soporta la descripción y configuración de infraestructuras virtualizadas. La configuración es aplicada en tiempo de ejecución en base a información provista por una infraestructura que monitoriza las aplicaciones desplegadas. Para la descripción propone un lenguaje basado en Open Virtualization Format (OVF) (DMTF, 2013), el cual principalmente incluye primitivas para describir una aplicación en términos de componentes y reglas de escalabilidad que controlan las configuraciones de máquinas virtuales en un entorno cloud OpenNebula. Sin embargo, esta propuesta no toma en cuenta aspectos a la arquitectura de la aplicación cloud y, a pesar de que cambia las características o capacidad de los recursos desplegados a fin de satisfacer reglas de elasticidad, no reconfigura la arquitectura de la aplicación como resultado de la monitorización.

### 3.1.3 Enfoques para el desarrollo de aplicaciones cloud

El marco de referencia MULTICLAPP (Guillén y otros, 2013b) presenta un proceso de desarrollo de tres etapas que permite el desarrollo de aplicaciones multicloud sin acoplarlas a un vendedor específico. Una etapa de modelado inicial en la que se modelan las aplicaciones, una etapa de codificación de comportamiento funcional en la que se implementa la funcionalidad de la aplicación y una etapa final de generación y despliegue de artefactos cloud en la que se generan artefactos compatibles a una plataforma cloud específica (ver Figura 3-1). Los desarrolladores definen un conjunto de artefactos cloud que interoperan entre ellos, así como con servicios ofrecidos por la plataforma en la que serán desplegados. Un perfil UML permite llevar a cabo esta tarea en donde las clases y componentes que implementan la funcionalidad de la aplicación son etiquetados como elementos de artefactos cloud, lo que permite asignarle a uno o más artefactos cloud; siendo posible asignar a más de un artefacto y plataforma de despliegue. El perfil también permite modelar requisitos no funcionales de los artefactos cloud involucrados. Las aplicaciones pueden ser asignadas a plataformas cloud sin que exista impacto en los modelos independientes de plataforma o en el código fuente. Separa la información relacionada a entornos cloud del código fuente de la aplicación, y utiliza técnicas de adaptación que permite a los artefactos interoperar con servicios específicos de plataforma. MULTICLAPP provee un motor de generación de código que facilita la obtención de adaptadores que son inyectados en los artefactos cloud, y planes de despliegue. Genera adaptadores que permiten que artefactos remotos interoperen entre ellos, así como con servicios provistos por su entorno cloud. A pesar de que la arquitectura de la aplicación podría ser modelada mediante UML, no se menciona el modelado de aspectos relacionados al impacto de la integración de los artefactos de despliegue en la arquitectura de la aplicación. Adicionalmente, no genera código que soporte

la reconfiguración de la arquitectura de la aplicación producida por la integración de artefactos de despliegue.

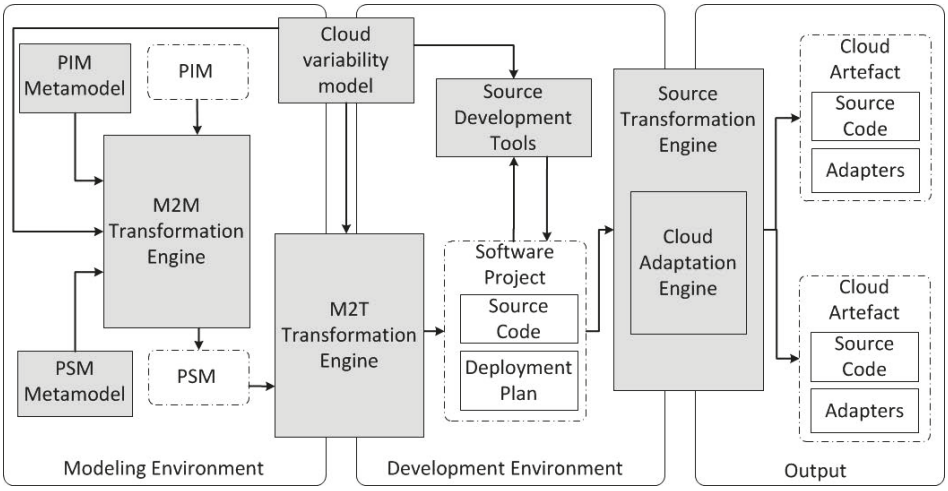



**Figura 3-1 Fases de MULTICLAPP – (Guillén y otros, 2013b)**

### 3.1.4 Enfoques de migración de sistemas a entornos cloud

CloudMIG (Frey y Hasselbring, 2011) tiene como objetivo soportar a los proveedores SaaS en la migración semiautomática de sistemas de software existentes a aplicaciones basadas en PaaS y IaaS escalables y eficientes en uso de recursos. CloudMIG es un enfoque dirigido por modelos para la migración de un sistema a entornos cloud y está compuesto por seis actividades que producen la generación semiautomática de partes de una arquitectura destino que hace uso eficiente de entornos cloud, utilizando para ello heurísticas basadas en reglas. Se enfoca principalmente en la ingeniería inversa de aplicaciones hacia representaciones conforme el Knowledge Discovery Model (KDM)(Object Management Group Inc., 2011) y en su despliegue basado en cloud. CloudMIG provee un modelo basado en metamodelos Ecore, el Cloud Environment Model (CEM). Éste cubre las perspectivas de aplicaciones cloud y de entornos cloud, conteniendo propiedades y restricciones (CloudEnvironment Constraints – CECs) de diferentes entornos cloud; en donde las restricciones impuestas por entornos cloud condicionan las actividades de reingeniería. CloudMIG provee además una herramienta que sugiere el entorno de despliegue óptimo (Soren Frey y otros, 2013) de acuerdo a la conformidad de los elementos arquitectónicos con la oferta proveedores cloud potenciales (Sören Frey y otros, 2013). Para determinar la conformidad utiliza información estadística a cerca del comportamiento de los elementos arquitectónicos obtenidas automáticamente en un modelo de utilización; por ejemplo, invocaciones de un servicio en un periodo de tiempo, o promedio

de paquetes de datos enviados. Sin embargo, no brinda mecanismos que faciliten la interacción de los servicios que conforman la arquitectura una vez que éstos han sido migrados e integrados a la aplicación. Tampoco ofrece mecanismos que faciliten la evolución arquitectónica luego de la integración de los servicios migrados.

El enfoque MOve to Clouds for Composite Applications (MOCCA) (Leymann y otros 2011) propone un método para migrar sistemas heredados a entornos cloud. MOCCA incluye un metamodelo conforme al cual se crean modelos que describen la arquitectura de la aplicación (mediante describir componentes y sus relaciones) y el despliegue de sistemas heredados. La idea principal del método propuesto es que un modelo arquitectónico de la aplicación a ser migrada al cloud es enriquecido con información adicional, y ese modelo enriquecido es la base para reorganizar automáticamente la aplicación y aprovisionar la aplicación reordenada en distintas nubes. La información con la que se enriquece el modelo arquitectónico de la aplicación corresponde a: información de despliegue, información de la distribución de componentes que define grupos de componentes que serán hospedados en un mismo entorno cloud, unidades de implementación tales como imágenes virtuales de la aplicación que son asociadas con los componentes del modelo combinado, datos asociados a los componentes arquitectónicos e interacción entre ellos. Sin embargo, no considera la naturaleza incremental de aplicaciones cloud y no provee mecanismos que soporten la evolución arquitectónica que se producirá al integrar la migración de cada elemento. A pesar de que permite describir información de las interacciones entre componentes, considera únicamente información de métricas de la interacción más no el protocolo de interacción u orquestación entre los componentes que serán migrados e integrados. La Figura 3-2 muestra los artefactos principales creados siguiendo el método MOCCA.

### 3.1.5 Discusión

El desarrollo incremental, conjuntamente con estrategias interactivas permiten a los desarrolladores descubrir y corregir errores de manera ordenada. En donde, la integración de un incremento de software en la aplicación actual no solo provee a los clientes con nuevas funcionalidades, sino provee de actualizaciones y correcciones. Los desarrolladores de software no solo necesitan desplegar aplicaciones en una plataforma cloud específica, requiriendo también enfoques que soporten el desarrollo, la migración de aplicaciones desde una plataforma cloud a otra, y la gestión de aplicaciones distribuidas que abarcan múltiples proveedores IaaS o PaaS. La Tabla 3-1 presenta el resumen de los proyectos o propuestas

para el desarrollo de aplicaciones cloud presentados en esta sección. En esta tabla se describe el enfoque o aproximación con el que cada proyecto hace frente al desarrollo/migración de aplicaciones cloud y el soporte que brindan.

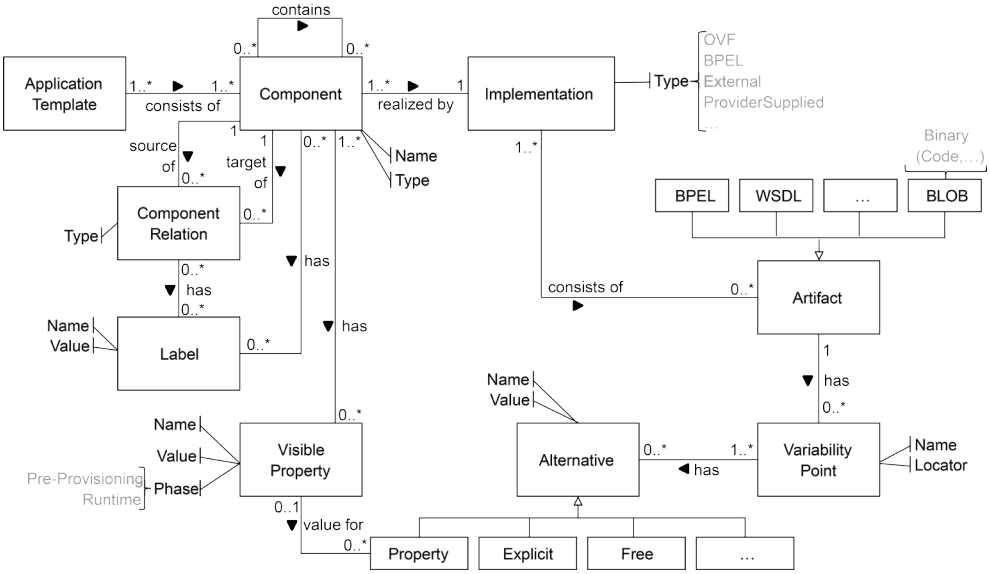



**Figura 3-2 Metamodelo de MOCCA**

Adicionalmente se analiza si es que los proyectos brindan mecanismos ya sea para facilitar la evolución arquitectónica o el desarrollo incremental. En dónde; el soporte al desarrollo incremental hace referencia a la oferta de mecanismos que permitan planificar la integración, propagar el impacto arquitectónico de la integración hacia artefactos de software en diferentes etapas del ciclo de vida de un servicio, o faciliten la integración incremental de incrementos de software.

Los proyectos presentados en la Tabla 3-1 proponen enfoques que permiten abstraer las decisiones de plataformas cloud específicas, haciendo frente a la dependencia de la tecnología de estas plataformas. Sin embargo, algunos de ellos (p. ej., los que proponen enfoques basados en APIs) crean dependencias con la solución que proponen. Adicionalmente, los proyectos analizados, proveen soluciones para facilitar el aprovisionamiento o despliegue. Éstos proveen mecanismos que soportan la especificación o descripción automática de entornos cloud que satisfacen los requisitos de las aplicaciones o de los servicios a ser desplegados; así como mecanismos de negociación, monitorización, verificación de conformidad, optimización, o adaptación de la interacción entre la aplicación y los servicios cloud que utilizan. Sin embargo, a pesar de que el diseño de la arquitectura de software define las restricciones para las actividades de diseño e

implementación subsecuentes, los proyectos no brindan soporte al diseño arquitectónico de la aplicación.

**Tabla 3-1 Enfoques que soportan el desarrollo/migración de aplicaciones cloud**

| *Proyecto* | *Capa de Jerarquía Cloud* | *Propone un enfoque que soporta* | *Soporta Evolución Arquitectónica* | *Soporta Desarrollo Incremental* |
|---|---|---|---|---|
| *mOSAIC* | IaaS/PaaS | ***Propone un enfoque:*** Middleware/Semántica/API ***Soporta:*** Negociación SLA de servicios Cloud/Despliegue | No | No |
| *Cloud4SOA* | PaaS | ***Propone un enfoque:*** Middleware/Semántica/API ***Soporta:*** Despliegue/Monitorización Adaptadores de conexión entre servicios/ Gestión SLA | No | No |
| *SOCCA* | IaaS/PaaS | ***Propone un enfoque:*** Middleware/Semántica/API ***Soporta:*** Migración/Despliegue | No | No |
| *RESERVOIR* | IaaS | ***Propone un enfoque:*** Infraestructura de monitorización ***Soporta:*** Monitorización/Despliegue / Cambio dinámico de características de recursos cloud | No | No |
| *MULTICLAPP* | IaaS/PaaS | ***Propone un enfoque:*** MDE/Técnicas de Adaptación ***Soporta:*** Genera adaptadores que facilitan la interoperación/ Planes de despliegue | Si | Si |
| *CloudMIG* | IaaS/PaaS | ***Propone un enfoque:*** MDE/Heurísticas de reglas ***Soporta:*** Sugiere entorno de despliegue optimo/Despliegue/Optimización de espliegue/Verificación de Conformidad | No | No |
| *MOCCA* | IaaS/PaaS | ***Propone un enfoque:*** MDE ***Soporta:*** Despliegue de aplicaciones/ Optimización de despliegue | No | No |

Los proyectos analizados no promueven durante la implementación la aplicación de principios que satisfagan el diseño arquitectónico, ni promueven la construcción de servicios cloud que interoperan entre sí y sean fáciles de distribuir (desplegar/re-desplegar) en diferentes proveedores cloud. Consideran únicamente la interoperación con los recursos ofrecidos por los proveedores cloud más no el protocolo de interacción u orquestación entre los servicios que serán construidos/migrados e integrados.

Los proyectos analizados aplican enfoques que agilitan el desarrollo y migración de aplicaciones cloud; en done, "conseguir la integración correcta es el aspecto más importante de la tecnología asociada a los enfoques ágiles" (Newman, 2015). Sin embargo, a pesar de que el desarrollo de aplicaciones cloud generalmente sigue una aproximación incremental, estos proyectos no proporcionan mecanismos que faciliten la especificación o implementación de las decisiones arquitectónicas relativas a la integración.

## 3.2 Lenguajes de especificación en entornos cloud

A fin de aprovechar las oportunidades de optimización en el uso de recursos tecnológicos que ofrece la computación cloud, y soportar el proceso de desarrollo de aplicaciones cloud se requieren de soluciones que soporten la toma de decisiones en las diferentes actividades de desarrollo. En esta sección investigamos los enfoques actuales para lenguajes de modelado que soportan el desarrollo de aplicaciones cloud.

### 3.2.1 Lenguajes de especificación

De acuerdo a una revisión sistemática de lenguajes de descripción de servicios y sus problemas en cloud computing (Sun y otros, 2012), existe una carencia de lenguajes de especificación de servicios cloud que permitan especificarlos desde diferentes perspectivas. Adicionalmente, entre los lenguajes que son considerados en esta revisión sistemática se puede evidenciar que: i) la mayoría de lenguajes tienen como propósito la selección, descubrimiento y composición de servicios; ii) únicamente SoaML tiene como propósito el modelado y diseño de la arquitectura de servicios; y iii) únicamente SoaML sigue una aproximación MDE. Adicionalmente, Breivold y otros (2014) llevaron a cabo una revisión sistemática en la cual categorizaron estudios que describen enfoques arquitectónicos para el diseño en el dominio cloud, identificando la necesidad de enfoques arquitectónicos y de diseño para la evolución de servicios cloud. A continuación, se describen los lenguajes que han sido identificados como relacionados a este trabajo de tesis, y que soportan el desarrollo de aplicaciones cloud.

El estándar Topology and Orchestration Specification for Cloud Applications (TOSCA) desarrollado por OASIS (Palman y Spatzier 2013) provee un lenguaje para especificar los componentes que forman la topología de aplicaciones cloud, conjuntamente con el procesos para su orquestación. TOSCA tienen como objetivo habilitar la interoperabilidad y reusabilidad de componentes de aplicaciones tales como servidores Web, sistemas operativos, máquinas virtuales, etc.; permitiendo combinarlos en una aplicación compuesta. Conceptualmente consiste de dos partes diferentes: la descripción estructural de la aplicación, llamada topología; y la descripción estandarizada de la gestión de la aplicación. TOSCA está basada en XML; proveyendo además una notación gráfica para la descripción de plantillas de servicios (Vino4TOSCA) (Breitenbücher y otros, 2012). Las plantillas de servicios pueden ser operacionalizadas con planes de gestión con la cual las operaciones (p. ej., aprovisionamiento) pueden ser ejecutadas. Adicionalmente, permite la definición de plantes de gestión basados en flujos de trabajo descritos mediante Business Process Model Notation (BPMN)[5] que pueden ser vistos como el mapeo entre la aplicación y entornos cloud. Sin embargo, no tiene en consideración la arquitectura lógica del software de la aplicación ni brinda soportan la naturaleza incremental de aplicaciones cloud.

El lenguaje Cloud Application Modeling Language (CAML) (Bergmayr y otros 2014), permite la representación de topologías de despliegue basadas en cloud mediante UML, refinándolos con ofertas cloud capturadas por perfiles UML dedicados. Los perfiles UML capturan ofertas de proveedores cloud desde una perspectiva técnica (p. ej., rendimiento), así como una no técnica (p.ej., precio). El refinamiento mediante perfiles UML permite separar modelos de despliegue independientes de proveedor cloud de modelos de despliegue específicos a un proveedor cloud. CAML fomenta la reutilización de soluciones de implementación mediante la definición de plantillas. Sin embargo, no ofrece soporte para que los desarrolladores seleccionen la topología de despliegue que mejor satisfaga los requisitos de los componentes. Por ejemplo, no existe información acerca de los requisitos relacionados a la naturaleza de trabajo o demanda de los componentes, información que permita a los desarrolladores seleccionar las plantillas o características de los elementos de despliegue que satisfagan los requisitos de los componentes. La Figura 3-3(a) muestra los componentes de una aplicación, mientras que la Figura 3-3(b) muestra el posible despliegue de los componentes de la aplicación, en donde sus elementos son instancias de los tipos definidos en la vista de componentes Figura 3-3(a) y de la vista de despliegue Figura 3-3(c).

[5] http://www.omg.org/spec/BPMN/2.0

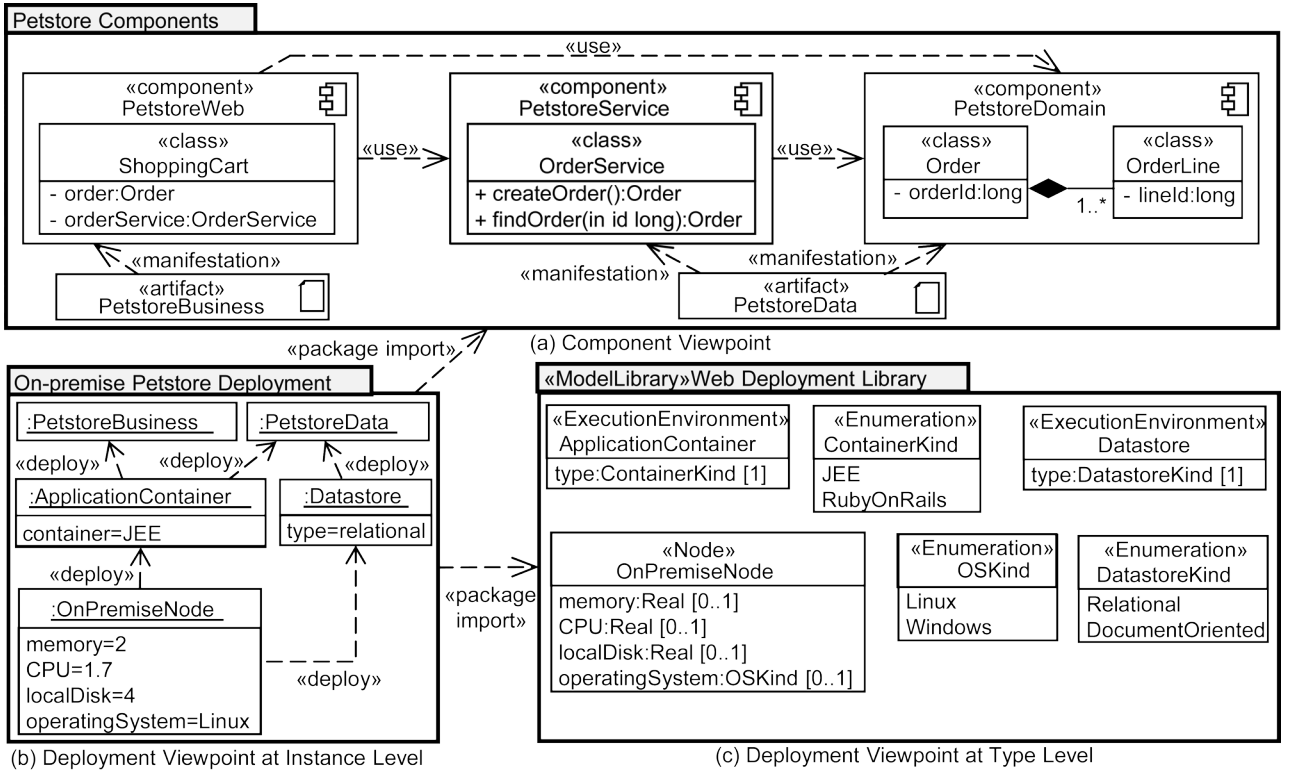



**Figura 3-3 CAML Caso de Uso (Bergmayr y otros 2014)**

Por otro lado, el lenguaje Cloud Modelling Language (CloudML) (Brandtzæg y otros 2012) permite a los usuarios modelar los recursos que esperan utilizar de entornos cloud. Los sistemas son definidos como un conjunto de nodos con diferentes requisitos de hardware que pueden ser conectados a ofertas cloud de diferentes proveedores. Este lenguaje es parte del proyecto FP7 REMICS (Mohagheghi y otros 2010) y propone una extensión de SoaML que incorpora información acerca de los recursos de hardware y red requeridos por una aplicación de las plataformas correspondientes. Los modelos CloudML son creados conforme al metamodelo de la Figura 3-4 y son utilizados como entrada para un motor de aprovisionamiento (CloudML-engine) que analiza automáticamente los requisitos y aprovisiona los recursos en el proveedor cloud mediante la configuración y creación de nodos. El motor de aprovisionamiento retorna un modelo en tiempo de ejecución, de acuerdo al enfoque models@run.time, correspondiente a los recursos aprovisionados, el cual puede ser utilizado por el usuario para interactuar con los recursos aprovisionados y desplegar la aplicación. Sin embargo, esta propuesta no enlaza el aprovisionamiento a aspectos de la arquitectura de la aplicación, lo que permitiría modelar los recursos que satisfagan requisitos de los componentes de la aplicación y automatizar el despliegue.

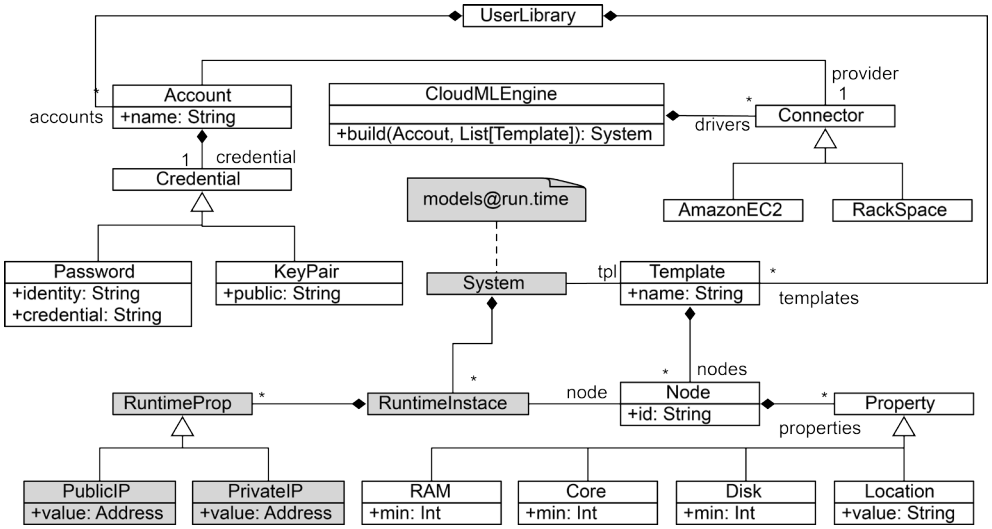



**Figura 3-4 Arquitectura de CloudML (Brandtzæg y otros 2012)**

El lenguaje cloudADL (Perez y Rumpe, 2013) es el elemento central de un metodología basada en modelos para la ingeniería de servicios cloud. Esta metodología describe a las aplicaciones cloud como sistemas interactivos. CloudADL es un ADL textual de componentes y conectores que extiende MontiArc (Haber y otros, 2012) y describe arquitecturas mediante componentes interconectados

descompuestos jerárquicamente. Sin embargo, los requisitos tecnológicos específicos para entornos cloud deben ser especificados utilizando otros lenguajes, lo que genera nuevos retos a los arquitectos que tienen que lidiar con la integración de diferentes lenguajes.

El DSL Platform Independent Model dedicated to Cloud (Pim4Cloud) (Brandtzæg y otros, 2012a) provee soporte para el despliegue de aplicaciones en entornos cloud. El diseñador de aplicaciones utiliza este DSL para modelar el despliegue de aplicaciones cloud siguiendo un enfoque basado en componentes. En paralelo, el proveedor de infraestructura describe la infraestructura disponible que puede ser utilizada por la aplicación. Es decir, el diseñador requiere nodos de cómputo (p. ej., máquinas virtuales) de entornos cloud, y el proveedor de infraestructura describe esos nodos de acuerdo a su catálogo. El DSL de Pim4Cloud es implementado en Scala el cual permite describir diferentes topologías de aplicaciones cloud; adicionalmente, la plataforma provee un intérprete el cual es utilizado para establecer la correspondencia entre los requisitos de la aplicación y los recursos disponibles. Sin embargo, no provee soporte al aprovisionamiento de entornos cloud.

El enfoque Blueprint (Nguyen y otros 2011) describe aplicaciones basadas en servicios (Service-based Applications – SBA) en términos de artefactos de despliegue de grano grueso que proveen una representación uniforme de SBAs conectados con las ofertas de recursos cloud requeridas. Provee un lenguaje de especificación bien definido, el Blueprint Specification Language (BSL), que brinda a los proveedores cloud un mecanismo para especificar servicios cloud de manera abstracta y sin ambigüedades. Los modelos de Blueprint son codificados en XML y representados en términos de una topología de la arquitectura virtual (Virtual Architecture Topology – VAT) para lo cual sugiere una notación gráfica. La idea es publicar los modelos en un repositorio público (Nguyen y otros, 2012) para establecer un mercado de servicios. Por otro lado, Blueprint asiste a los desarrolladores de aplicaciones a seleccionar ofertas de servicios cloud (p. ej., software, plataforma e infraestructura) de múltiples proveedores y configurarlas dinámicamente a fin de satisfacer los requisitos de las aplicaciones. Sin embargo, no provee mecanismos para especificar la arquitectura de la aplicación ni los requisitos de recursos cloud específicos a los servicios que la conforman.

CloudML-UFPE (Gonçalves y otros, 2011) propone un enfoque basado en XML para la descripción servicios cloud de infraestructura y recursos cloud desde una perspectiva de proveedor cloud; permitiendo a los proveedores cloud modelar sus ofertas de entornos cloud. Provee conceptos de modelado para re-

presentar ofertas cloud que se conectan con los recursos cloud internos gestionados en un entorno cloud. La perspectiva del consumidor es soportada por requisitos de servicio. Los servicios cloud y recursos son representados en términos de nodos y enlaces entre ellos. Los nodos tienen propiedades para CPU, almacenamiento, y memoria, mientras los enlaces tienen propiedades para retraso y tasa. Los requisitos de servicios contienen los nodos requeridos y enlaces de acuerdo a los servicios cloud especificados, lo que resulta esencialmente en un mapeo manual basado en identificadores. Sin embargo, éste lenguaje se enfoca únicamente en la descripción de recursos desde la perspectiva del proveedor cloud, sin considerar los requisitos de los componentes o servicios que harán uso de los recursos cloud.

La propuesta Heat Orchestration Template (HOT) soporta la descripción de plantillas de aprovisionamiento principalmente de entornos cloud que operan a nivel de infraestructura. Esta propuesta se desarrolla en el contexto de OpenStack[6] y es construida alrededor notaciones de recursos que describen los artefactos principales (p. ej., instancias de unidades de computo o redes) de un despliegue. Los recursos pueden recibir parámetros y se pueden utilizar funciones que realizan tareas específicas al interior de las plantillas (p. ej., obtener en tiempo de ejecución el valor de un parámetro de recurso). Sin embargo, HOT permite especificar la oferta de proveedores cloud, sin proveer mecanismos que permitan especificar los requisitos que deben satisfacer las ofertas.

### 3.2.2 Discusión

La Tabla 3-2 presenta el resumen de los lenguajes de especificación descritos en esta sección, así como los lenguajes que conforman los enfoques de desarrollo y migración descritos en la Sección 3.1. La Tabla 3-2 muestra información que describe cada uno de los lenguajes analizados; permite identificar aquellos lenguajes que describen la arquitectura de la aplicación cloud, y aquellos lenguajes que describen la interacción entre los servicios que conforman la aplicación cloud. Adicionalmente, la Tabla 3-2 permite identificar los lenguajes que ofrecen mecanismos para describir la integración o impacto arquitectónico que producirá el despliegue de servicios en la arquitectura actual de la aplicación cloud.

Se han identificado lenguajes que permiten documentar decisiones de diseño en entornos cloud. Algunos de estos lenguajes son DSLs cuya semántica y sintaxis ha sido construida desde cero, mientras otros extienden el lenguaje UML. Estos lenguajes permiten describir topologías de despliegue, aplicaciones como una

[6] http://www.openstack.org

composición de artefactos de software a ser desplegado en múltiples plataformas cloud, o recursos cloud que una aplicación puede requerir de entornos cloud. Estos lenguajes hacen frente a la dependencia con la tecnología u ofertas específicas de proveedores cloud. Algunas propuestas permiten refinar descripciones independientes de un proveedor cloud a descripciones conforme la oferta concreta de un proveedor cloud.

**Tabla 3-2 Resultados del análisis de lenguajes propuestos como soporte a la construcción de aplicaciones cloud**

| *Lenguaje* | *Describe* | *Describe Arquitecturas* | *Describe Interacción* | *Soporte Incremental* |
|---|---|---|---|---|
| *RESERVOIR ML* | Infraestructura / Despliegue / Reglas de escalabilidad de imágenes de máquinas virtuales | Si | No | No |
| *MULTICLAPP* | Despliegue / Requisitos no funcionales | Si | Si | No |
| *CloudMIG* | Propiedades y restricciones de entornos cloud | Si | No | No |
| *MOCCA* | Arquitectura / Despliegue / Distribución de componentes en entornos cloud | Si | No | No |
| *CAML* | Topología de despliegue / Refinamiento de descripciones de despliegue a ofertas concretas | Si | No | No |
| *Blueprint* | Servicios cloud (IaaS, PaaS, Saas) / Composición de servicios cloud | No | No | No |
| *CloudML-UFPE* | Recursos cloud de infraestructura | No | No | No |
| *HOT* | Aprovisionamiento de recursos | No | No | No |
| *cloudADL* | Arquitectura de Software | No | No | No |
| *CloudML* | Despliegue / Aprovisionamiento de recursos | Si | No | No |
| *Pim4Cloud* | Despliegue | No | No | No |
| *TOSCA* | Despliegue / Aprovisionamiento de recursos | No | No | No |

Las aplicaciones basadas en servicios, tales como las aplicaciones cloud, son desarrolladas de manera incremental a través de construir servicios reutilizables capaces de interactuar entre ellos, en donde no existe una definición de todo el sistema. Desde un punto de vista arquitectónico, para soportar la especificación incremental de servicios cloud, las descripciones arquitectónicas deberían permitir la definición parcial del sistema, sin restringir a los arquitectos a modelarlo en su totalidad (Malavolta y otros, 2013). Sin embargo, las propuestas analizadas

no aíslan la arquitectura de software del incremento a ser integrado, en su lugar promueven la especificación declarativa de una arquitectura global que incluye los elementos arquitectónicos del incremento. Adicionalmente, las propuestas que describen la arquitectura de la aplicación, generalmente lo hacen en términos de clases, componentes y sus relaciones. Sin embargo, a pesar de que las aplicaciones cloud son aplicaciones basadas en servicios, no se proponen descripciones arquitectónicas específicas para arquitecturas orientadas a servicios, salvo CloudML que extiende SoaML. Por otro lado, no consideran aspectos relacionados a la evolución arquitectónica que se produce en el desarrollo incremental de aplicaciones cloud, sin soportar el razonamiento sistemático acerca del impacto de la integración de incrementos en la arquitectura de la aplicación actual.

A fin de dotar a los arquitectos con las herramientas que le permitan especificar arquitecturas de servicios cloud de manera sistemática, un lenguaje debería tener la expresividad de capturar características específicas de entornos cloud (p. ej., elasticidad, escalabilidad). Algunos de los lenguajes permiten describir requisitos no funcionales; sin embargo, en general no existe información acerca de los requisitos relacionados a la naturaleza de trabajo o demanda esperada de los servicios. Esta información permitiría a los desarrolladores tomar decisiones de implementación o seleccionar recursos cloud cuyas características satisfagan los requisitos de los servicios de la aplicación.

## 3.3 Reconfiguración dinámica de arquitecturas de software

Las arquitecturas de software proporcionan una base para la evolución sistemática del software en tiempo de ejecución; permitiendo a los desarrolladores ir desde un punto de vista de las líneas de código a componentes de grano grueso y su estructura de interconexión. La integración o actualización de incrementos de servicios puede requerir la reconfiguración de la arquitectura actual de servicios cloud. En esta sección describimos los enfoques de reconfiguración dinámica de arquitecturas de aplicaciones cloud. Las aplicaciones cloud son aplicaciones basadas en servicios, por lo tanto, partimos desde la reconfiguración dinámica de aplicaciones basadas en servicios.

### 3.3.1 Reconfiguración dinámica de aplicaciones basadas en servicios

Existen pocos trabajos que se centran en la reconfiguración dinámica de aplicaciones basadas en SOA. Estas propuestas son divididas en dos categorías: los enfoques de reconfiguración por razones de adaptación y los enfoques de reconfiguración por razones de perfectivas o correctivas (ver Sección 1.2). La reconfiguración dinámica se realiza utilizando técnicas que modifican los enlaces entre

puntos de acceso a los servicios, y estructuras de control. Por ejemplo, Halima y otros (2008) proponen un marco de trabajo que provee propiedades de autocorrección para la gestión de QoS en aplicaciones interactivas distribuidas. La plataforma SIROCO (Fredj y otros, 2008) hace frente a la reconfiguración de servicios de orquestaciones para servicios no disponibles. Por otro lado, el enfoque Dynamic Reconfigurable ESB Service Routing (DRESR) (Bai y otros, 2007) permite que la tabla de rutas sea cambiada en tiempo de ejecución; en donde el proveedor de servicio de cada nodo, la lógica de composición y la lógica de integración pueden ser cambiadas. Sin embargo, a pesar de que las propuestas son para la reconfiguración dinámica de aplicaciones basadas en servicios, y hacen uso de técnicas para la gestión de componentes o enlaces entre puntos de acceso, hasta donde sabemos, no existen propuestas que brinden mecanismos para especificar la integración ni los cambios arquitectónicos que ésta producirá.

### 3.3.2 Reconfiguración dinámica de aplicaciones cloud

El enfoque MOdel-Driven Approach for the design and execution of applications on multiple Clouds (MODAClouds) (Ardagna y otros 2012) tiene como objetivo ayudar a los desarrolladores y operadores a explotar múltiples plataformas y migrar componentes de un sistema desde una plataforma a otra dependiendo de requisitos de calidad. MODAClouds (ver Figura 3-5) propone un enfoque DSDM a través de un modelo independiente de computación (CIM) que permite, durante el diseño, descomponer una aplicación en componentes e incluir información de requisitos no funcionales y restricciones que permitirán hacer un match entre plataformas cloud y características de los componentes. Luego en un segundo modelo se introducen conceptos cloud, pero de manera independiente de una plataforma cloud específica (CPIM), y finalmente un modelo especifico del proveedor cloud (CPSM) en el que se describen los artefactos requeridos por cada plataforma. Provee transformaciones totalmente automatizadas (p. ej., la generación de scripts de despliegue) o transformaciones que generan esqueletos. Adicionalmente ofrece un sistema de soporte a las decisiones que permite el análisis de riesgos para la selección de proveedores cloud, un entorno de ejecución que permite a los desarrolladores monitorizar y adaptar el sistema (p. ej., los desarrolladores migran algunos componentes del sistema desde un IaaS a otro que ofrece un mejor rendimiento en un determinado momento) como resultado de fluctuaciones de rendimiento. MODAClouds provee además herramientas que dan soporte a todas las fases del ciclo de vida de un sistema de software basado en cloud. Sin embargo, no indica cómo se gestiona la interoperabilidad y orquestación entre los componentes de la aplicación distribuidos en diferentes proveedores cloud. La reconfiguración se logra mediante re-desplegar los componentes de la aplicación; sin embargo, no brinda soporte

al desarrollo incremental en el cual la integración de los componentes producirá cambios en la arquitectura.

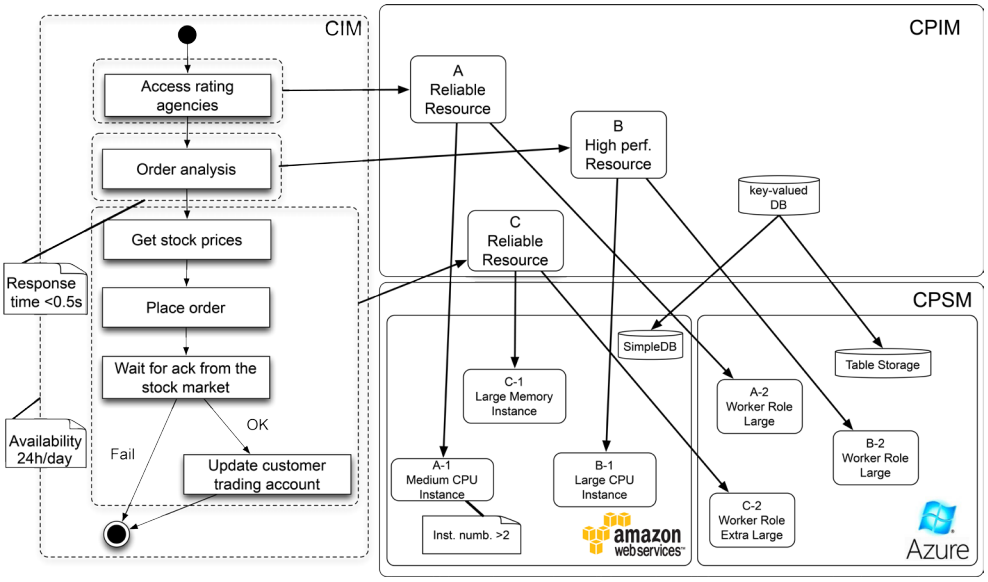



**Figura 3-5 Enfoque MODAClouds – Extracto de: (Ardagna y otros 2012)**

El proyecto Seamless adaptive multi-cloud management of service-based applications (SeaClouds) (Brogi y otros 2014) provee un marco de trabajo que permite hacer frente al despliegue, gestión y reconfiguración de aplicaciones basadas en servicios en múltiples plataformas cloud heterogéneas. Este enfoque se basa en una perspectiva de orquestación de servicios distribuidos en diferentes proveedores cloud y es diseñado para satisfacer propiedades funcionales y no funcionales de la aplicación. Las aplicaciones son reconfiguradas dinámicamente cambiando la orquestación de los servicios como resultado de procesos de monitorización que detectan desviaciones en las propiedades y la necesidad de re-distribuir servicios en varios proveedores cloud. La reconfiguración puede implicar actualizar un servicio, remplazar dinámicamente servicios que funcionan incorrectamente, o migrar servicios a un proveedor diferente a fin de beneficiarse de su oferta o evitar las deficiencias en la oferta del proveedor actual. Una aplicación compleja, la cual consiste de módulos y requisitos tecnológicos y de QoS, es provista como entrada al planificador SeaClouds.

SeaClouds genera una orquestación mediante asignar módulos a diferentes plataformas PaaS. Luego la orquestación es desplegada y monitorizada de acuerdo a métricas estándar. Si los requisitos no se satisfacen el analizador de SeaClouds genera información de reconfiguración, la cual es utilizada para la creación de una orquestación diferente para la aplicación (ver Figura 3-6). Sin embargo, la

reconfiguración arquitectónica que producirá la integración de incrementos en la aplicación actual no es tomada en cuenta. Por ejemplo, no contempla el remplazar servicios u orquestaciones de servicios a causa de cambios en los requisitos funcionales o protocolos de interacción que han sido implementados en un incremento de software a ser integrado.

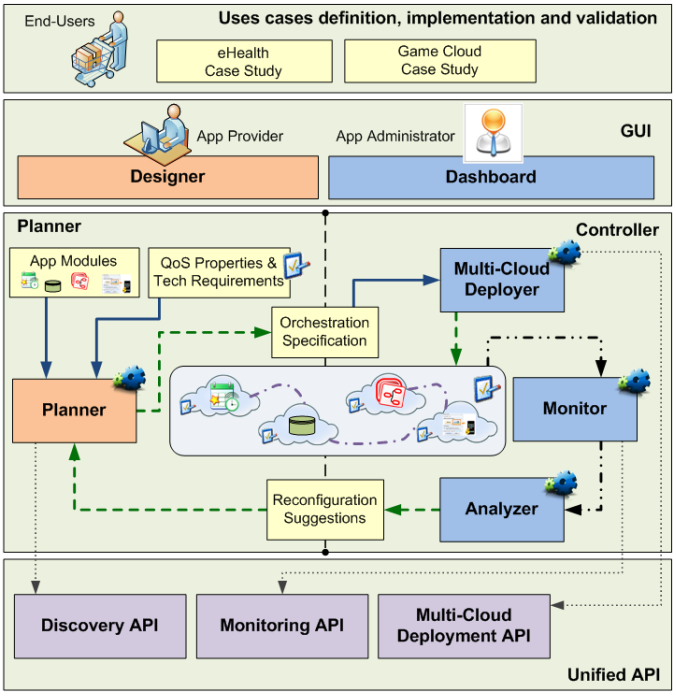



**Figura 3-6 Arquitectura SeaClouds – (Brogi y otros 2014)**

### 3.3.3 Discusión

A pesar de que la reconfiguración dinámica se produce en tiempo de ejecución, las acciones de cambio deberían ser planificadas desde las etapas iniciales y soportadas durante las diferentes etapas de desarrollo mediante la toma de decisiones que satisfagan la planificación inicial, así como mediante la aplicación de principios que permitan diseñar, implementar y desplegar arquitecturas de aplicaciones cloud fáciles de reconfigurar. Las propuestas que proveen soluciones para la reconfiguración dinámica de arquitecturas aplican técnicas que reemplazan los servicios, la orquestación de servicios, o enlaces entre puntos de acceso, cubriendo aspectos relacionados al aprovisionamiento y despliegue de servicios. Sin embargo, no se enfocan en aspectos relacionados a la arquitectura del software ni toman en cuenta alternativas de implementación que facilitan la escalabilidad y reconfiguración dinámica. Por ejemplo, no brindan soporte para aplicar

principios que promuevan el desacoplamiento entre servicios durante la implementación. Adicionalmente, la reconfiguración o redespliegue en las propuestas actuales se inician como resultado de actividades de monitorización y proponen enfoques de reconfiguración arquitectónica cuyo fin es mejorar requisitos no funcionales (satisfacer criterios de calidad). Sin embargo, los cambios adaptativos (p. ej., la integración de incrementos de software resultantes de nuevas funcionalidades) que requieren soporte para la reconfiguración dinámica no son tomados explícitamente en cuenta.

Con respecto a la validación empírica de las propuestas, de acuerdo a Jamshidi y otros (2013), los enfoques de reconfiguración en aplicaciones basadas en servicios incluyen algunas pruebas como parte de la evolución; en donde la evidencia es frecuentemente obtenida mediante la aplicación de casos de estudio pequeños y ejemplos. Adicionalmente, indican que pocos estudios empíricos han sido ejecutados; sin embargo, ninguno de ellos se ha ejecutado en el contexto de aplicaciones cloud. Los enfoques analizados en esta sección, en base a nuestro conocimiento, no presentan validaciones empíricas, por lo que no existe un marco de referencia o criterios para la comparación de enfoques de reconfiguración dinámica de arquitecturas producida por la integración de incrementos.

# Capítulo 4. Reconfiguración dinámica e incremental de arquitecturas orientadas a servicios

En este capítulo se presenta el método DIARy (Dynamic Incremental Architectural Reconfiguration), un método para la reconfiguración dinámica e incremental de arquitecturas de aplicaciones cloud producida por la integración de servicios. DIARy hace uso de un enfoque DSDM, cuyos modelos permiten capturar diferentes aspectos relacionados con la integración de incrementos de software conformados por uno o más servicios y la consecuente reconfiguración dinámica de arquitecturas de servicios cloud que se producen como resultado del despliegue de incrementos. Los modelos y transformaciones de modelo consideran a la arquitectura del incremento como una arquitectura de software parcial que será integrada en la arquitectura de la aplicación, facilitando por lo tanto la propagación de cambios arquitectónicos hacia artefactos que implementan la reconfiguración e interoperación, y la trazabilidad de los cambios producidos por la integración.

La sección 4.1 presenta un lenguaje de especificación de integración. Elemento vertebrador del proceso, el cual, a un alto nivel de abstracción, no solo permite planificar la reconfiguración sino especificar los requisitos de recursos cloud de los servicios incluidos en un incremento de software.

La sección 4.2 presenta la vista global del método. Se describen las principales actividades, y sus artefactos de entrada y salida.

La sección 4.3 describe las actividades que se contemplan dentro del proceso de reconfiguración dinámica, describiéndolas en detalle en las sub-secciones correspondientes.

La sección 4.4 presenta las conclusiones del capítulo.

## 4.1 Lenguaje de especificación de arquitecturas para la integración incremental de servicios cloud

En esta sección se presenta un ADL que permite especificar la lógica de integración y el impacto arquitectónico relacionados con la integración de los servicios incluidos en un incremento en la aplicación cloud existente.

### 4.1.1 Requisitos

Los ADLs facilitan el modelado de arquitecturas de software por medio de notaciones para describir la estructura y comportamiento de un sistema (Medvidovic y Taylor, 2000). Típicamente las descripciones arquitectónicas se realizan en términos de componentes (elementos computacionales), conectores (interacción entre componentes), y configuraciones (dependencias estructurales entre componentes). Aunque históricamente se han propuesto varios ADLs, para el propósito de soportar de forma satisfactoria la integración incremental de servicios cloud, un ADL debería satisfacer los siguientes requisitos.

#### *4.1.1.1 Requisito 1: Especificación parcial de arquitecturas de software*

El desarrollo incremental es un factor crítico de éxito para proyectos de software modernos, en donde partes de aplicaciones de software (llamadas incrementos de software) son desarrolladas a diferentes ritmos o tiempos y luego liberadas e integradas (Cockburn y Highsmith, 2001). El desarrollo incremental, conjuntamente con estrategias iterativas permiten a los desarrolladores descubrir y corregir errores de manera ordenada. En donde, la integración de un incremento de software en la aplicación actual, no solo provee a los clientes con nuevas funcionalidades, sino también actualizaciones y correcciones.

Desde un punto de vista arquitectónico, para soportar la especificación incremental de servicios cloud, las descripciones arquitectónicas deberían permitir la definición parcial del sistema, sin restringir a los arquitectos a modelarlo en su totalidad (Malavolta y otros 2013). Las descripciones actuales de sistemas basados en servicios no aíslan la arquitectura de software del incremento a ser integrado, en su lugar promueven la especificación declarativa de una arquitectura global que incluye los elementos arquitectónicos del incremento.

#### *4.1.1.2 Requisito 2: Especificación del impacto de la integración*

Debido a que la evolución es un factor crítico en el ciclo de vida de sistemas de software, los mecanismos de actualización facilitan la incorporación de cambios

de software en el sistema actual (Buckley y otros, 2005). Diferentes artefactos de software son sujeto de cambios. Los cambios *estáticos* afectan a artefactos que van desde artefactos de requisitos hasta artefactos de arquitectura y diseño; mientras que cambios *dinámicos* afectan a artefactos que pueden ser modificados en tiempo de ejecución. Los cambios estáticos y dinámicos no son excluyentes; artefactos modificados estáticamente pueden ser la base de cambios dinámicos de otros artefactos. Por lo tanto, un ADL para especificar la integración incremental debería permitir a los arquitectos especificar el impacto de integrar cada elemento arquitectónico de un incremento de software en la arquitectura actual de una aplicación cloud (planificar los cambios estáticos), a la vez que facilitar la generación o actualización de los artefactos requeridos para reconfigurar dinámicamente la arquitectura actual de la aplicación en la cual se producirá la integración.

#### *4.1.1.3 Requisito 3: Especificación de requisitos de recursos cloud*

Los enfoques SOA proporcionan beneficios como el uso de elementos que permiten tratar con la interoperabilidad entre servicios (Clements y otros, 2011), soportando escenarios en los que proveedores y consumidores de servicios son desplegados independientemente en Internet, tal como sucede con los servicios cloud. Adicionalmente, las decisiones tomadas en etapas tempranas del ciclo de desarrollo tienen influencia sobre las decisiones de implementación, aprovisionamiento o despliegue tomadas en etapas posteriores; decisiones relacionadas no solo al uso adecuado de recursos cloud o la selección de mecanismos que faciliten la reconfiguración arquitectónica, sino decisiones que afectan a la calidad o al costo total de poseer aplicaciones desplegadas en plataformas cloud (Homer y otros, 2014). Por ejemplo, la consolidación de múltiples servicios en una única unidad computacional de acuerdo a similitudes en su perfil de requisitos de escalabilidad o capacidad de procesamiento, permite que todos los servicios escalen como una única unidad, ayudando a reducir costos de propiedad, incrementando la velocidad de comunicación y aliviando los esfuerzos de gestión.

La elasticidad es una de las características más distinguibles de los entornos cloud (Motta y otros, 2012), la cual es capaz de atender fuertes cambios en la demanda, agregando o disminuyendo el uso de recursos en respuesta a variaciones en la carga de trabajo, sin que esto suponga un incremento en los costos de gestión. Un ADL para la especificación incremental de la integración de servicios cloud debería por lo tanto no solo promover la aplicación de principios SOA, sino también permitir a los arquitectos especificar requisitos relacionados a los recursos cloud requeridos por los servicios incluidos en un incremento. Esto ayudará a los desarrolladores a tomar decisiones (p. ej., consolidar servicios, seleccionar

recursos cloud, etc.) en etapas posteriores de acuerdo a términos de SLAs, QoS, restricciones de precio, u otros criterios. En la siguiente subsección se propone un ADL que toma en consideración los requisitos aquí identificados.

### 4.1.2 Definición

En esta tesis se propone un ADL específico para dar soporte a la especificación incremental de la integración de servicios cloud. Este ADL aprovecha los beneficios tanto de SoaML como de UML y los extiende para satisfacer los requisitos indicados en la sección anterior.

SoaML (Object Management Group Inc., 2012) es un lenguaje basado en UML específicamente diseñado para describir arquitecturas basadas en servicios; sin embargo, en un contexto de integración incremental, este lenguaje no provee mecanismos para: i) aislar la arquitectura de los incrementos a ser integrados, ii) establecer una relación (interoperación) entre la arquitectura del incremento con la arquitectura actual de la aplicación en la que se integrará, iii) especificar el impacto que producirá la integración en la arquitectura actual de la aplicación, y iv) especificar los recursos cloud requeridos para la correcta ejecución de los servicios incluidos en un incremento.

Se ha elegido extender SoaML principalmente debido a que: i) es un lenguaje que se basa en los principios de SOA y sigue un enfoque DSDM, lo cual permitirá hacer frente al problema de interoperabilidad y portabilidad entre diferentes proveedores cloud; y ii) proporciona una vista arquitectónica de alto nivel, lo cual permitirá a los arquitectos especificar, a un alto nivel de abstracción, la integración de incrementos y la interoperabilidad entre los servicios incluidos en un incremento con los servicios pertenecientes a la aplicación en la cual serán integrados. A continuación, se describen las extensiones propuestas a SoaML.

#### 4.1.2.1 ADL para la integración incremental de servicios cloud: Elementos de modelado

Esta sección introduce los elementos de modelado del *ADL para la integración incremental de servicios cloud*; éstos extienden los elementos de SoaML y UML con el propósito de satisfacer los requisitos identificados previamente (ver 4.1.1). SoaML provee varios enfoques para especificar un sistema como un conjunto de componentes que trabajan en conjunto proveyendo y consumiendo servicios. Este trabajo de tesis sigue el enfoque basado en *Contrato de Servicios* porque este es el más adecuado para escenarios en los que más de dos partes están involucradas en un servicio (Object Management Group Inc., 2012). El diagrama de

perfiles UML de la Figura 4-1 es utilizado para describir los elementos de modelado del ADL propuesto; los estereotipos y valores etiquetados definidos con este propósito son:

- El estereotipo «*Extended Increment Architecture*», cuya semántica es similar a la semántica del elemento arquitectónico de SoaML *Services Architecture*, permite especificar la integración de incrementos. Sin embargo, éste no solo incluye una descripción de alto nivel de los elementos arquitectónicos de un incremento, sino también, la especificación del impacto arquitectónico de su integración, la demanda esperada y naturaleza del trabajo de los servicios cloud incluidos en un incremento. Estas últimas permiten, en etapas de desarrollo posteriores, determinar las características de los recursos cloud requeridas por los servicios cloud incluidos en un incremento. Los elementos arquitectónicos interiores de elementos «*Extended Increment Architecture*» son: «*Participant*», «*Service Contract*» y «*Role Binding*».

- El estereotipo «*Participant*» describe: i) un servicio cloud a ser integrado; ii) un servicio cloud o componente existente en la arquitectura de la aplicación cloud actual con el cual un servicio cloud o componente del incremento interactuará; iii) un servicio externo o componente que provee o consume servicios a (o de) otros servicios cloud del incremento; o iv) un recurso cloud consumido por un servicio cloud incluido en el incremento.

- El estereotipo «*Service Contract*» describe: i) la lógica de integración entre la arquitectura del incremento con la arquitectura de la aplicación cloud actual, en donde la lógica de integración corresponde a la interacción entre uno o más elementos de tipo «*Participant*» que pertenece(n) al incremento con uno o más elementos de tipo «*Participant*» que pertenece(n) a la arquitectura de la aplicación cloud actual; y ii) la interacción entre elementos de tipo «*Participant*». La interacción entre elementos de tipo «*Participant*» se describe a través de diagramas de interacción, en este ADL se emplean diagramas de secuencia.

- El estereotipo «*Role Binding*» describe enlaces entre los *Roles* definidos en un «*Service Contract*» con elementos de tipo «*Participant*» incluidos en «*Extended Increment Architecture*», con el propósito de indicar el rol

- que cada «*Participant*» cumple en el «*Extended Increment Architecture*» al que pertenece.

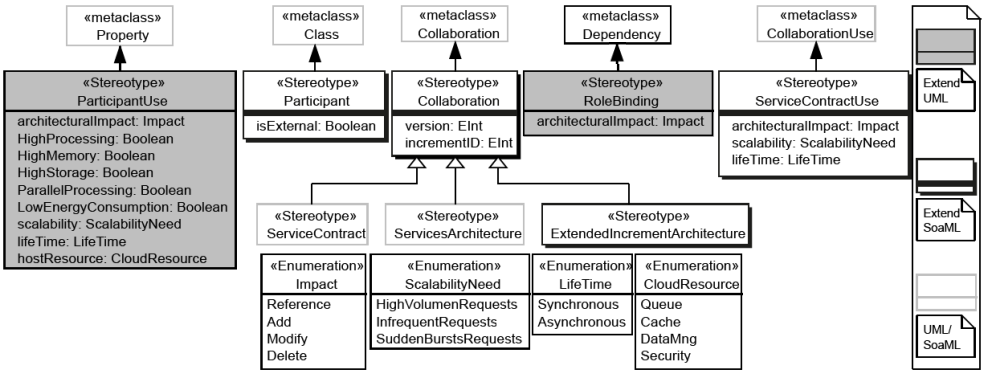



**Figura 4-1 Extensiones a SoaML y UML para la integración incremental**

- El valor etiquetado *architecturalImpact*, con el que se etiquetan los elementos arquitectónicos internos de un «*Extended Increment Architecture*», permite a los arquitectos especificar el impacto de integrar los elementos arquitectónicos del incremento en la arquitectura de la aplicación cloud actual. Los posibles valores son: i) *Reference*, utilizado para etiquetar elementos arquitectónicos que existen en la arquitectura de la aplicación cloud actual que interactuarán con elementos arquitectónicos del incremento, y no serán afectados o cambiarán a causa de la integración; ii) *Add*, utilizado para etiquetar elementos arquitectónicos que se incorporarán en la arquitectura de la aplicación cloud actual; iii) *Modify*, utilizado para etiquetar elementos arquitectónicos que existen en la arquitectura de la aplicación cloud actual y cuya implementación cambiará a causa de la integración; iv) *Delete*, utilizado para etiquetar elementos arquitectónicos que serán eliminados de la arquitectura de la aplicación cloud actual a causa de la integración. Ver la enumeración *Impact* en Figura 4-1.

- Los valores etiquetados *scalability* y *lifetime* permiten a los arquitectos especificar requisitos relacionados a la demanda esperada de elementos de tipo «*Participant*» o «*Service Contract*».

- Los valores etiquetados *High Processing*, *High Memory*, *High Storage*, *Static Content*, *Parallel Processing*, y *Low Energy Consumption* permiten a los arquitectos especificar requisitos relacionados a la naturaleza del trabajo de elementos de tipo «*Participant*». Estos requisitos son utilizados en etapas de desarrollo posteriores para determinar las características de los recursos cloud que elementos de tipo «*Participant*» requieren de entornos cloud para cumplir con términos de SLA o criterios de QoS.

El valor etiquetado *host Resource* permite a los arquitectos describir el tipo de recurso cloud de un elemento de tipo «*Participant*» que representa un recurso cloud. Finalmente, los valores etiquetados *version* e *incrementID* definidos en elementos de tipo «*Collaboration*» facilitan la gestión de versiones, la preservación del orden del proceso de evolución, y la coherencia de interacciones entre instancias de servicios.

## 4.2 Vista general del método DIARy

El método DIARy sigue un enfoque DSDM y está compuesto por un proceso que define las actividades que soportan las diferentes fases del desarrollo y un conjunto de herramientas que facilitan su ejecución. En la Figura 4-2 se muestra una visión global del método DIARy, con el flujo de acción y las principales actividades, que se describen haciendo uso del lenguaje de modelado de procesos SPEM 2.0 (Software & Systems Process Engineering). En la Sección 4.3 se describirán con detalle cada una de las actividades incluyendo los artefactos de entrada y salida. A diferencia de otras propuestas para la construcción incremental de arquitecturas (Díaz y otros, 2014), (Pérez y otros, 2010), que permiten construir arquitecturas diseñadas iterativa e incrementalmente con el producto (conocidas como *working architectures*), DIARy considera a la arquitectura del incremento como un artefacto de diseño independiente de la arquitectura global de la aplicación, el cual será integrado con esta arquitectura global y la reconfigurará. Considerar la arquitectura del incremento como un artefacto independiente facilita no solo la especificación de los cambios que producirá la integración, sino su propagación a otros artefactos de software en actividades posteriores de la integración.

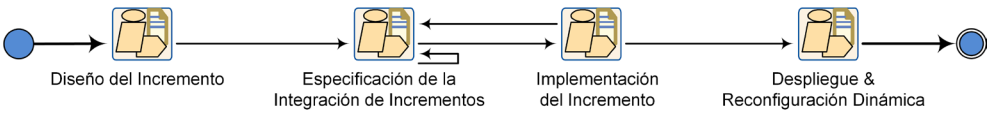



**Figura 4-2 El método DIARy**

A continuación, se describen cada una de estas actividades indicando los roles que participan y los artefactos que se producen como entrada o salida.

- *Diseño del Incremento*. DIARy no ofrece soporte específico a esta actividad; sin embargo, durante las demás actividades del proceso se emplean artefactos resultantes de ésta. Los arquitectos de la aplicación cloud producen el diseño arquitectónico del incremento a ser integrado, el cual lo documentan en el *Modelo de Arquitectura del Incremento*. Este modelo des-

cribe la estructura y comportamiento del incremento; es decir los servicios que conforman el incremento, y relaciones e interoperación entre ellos. Adicionalmente, documentan los términos relacionados al acuerdo de nivel de servicio en el artefacto *SLA*.

- *Especificar la integración del Incremento*. En esta actividad los arquitectos de la aplicación especifican la integración de incrementos a través del razonamiento sistemático a cerca de la lógica de integración, del impacto que la integración de los servicios incluidos en un incremento producirá sobre la arquitectura actual de la aplicación cloud, y de la naturaleza del trabajo que realizarán los servicios incluidos en el incremento. El resultado de esta actividad es utilizado en actividades posteriores: i) como artefacto de requisitos para seleccionar los recursos cloud que soportaran la naturaleza de trabajo de los servicios incluidos en un incremento, y ii) como entrada de transformaciones de modelos que propagan el impacto arquitectónico hacia artefactos de software de implementación y reconfiguración dinámica. Por ejemplo, soporta la generación de scripts de reconfiguración dinámica que cambian los enlaces entre servicios de acuerdo al impacto arquitectónico descrito en la especificación.

- *Implementación del incremento*. En esta actividad los desarrolladores utilizan transformaciones de modelos que toman como entrada el resultado de la actividad *Especificar la integración del incremento* y generan descripciones de los artefactos de software requeridos para implementar la arquitectura del incremento, incluyendo descripciones de los artefactos que implementan la lógica de integración. Luego, toman decisiones acerca de la estructura de empaquetado de los artefactos de software; permitiendo especificar si es que múltiples servicios serán consolidados en un único artefacto de despliegue a ser desplegado en una unidad computacional (recurso cloud). Finalmente ejecutan transformaciones de modelos que generan artefactos de software que: i) implementan la lógica de integración (p. ej., orquestación o coreografía de servicios) de acuerdo al impacto arquitectónico y la estructura de empaquetado definida, y ii) empaquetan los artefactos de implementación. Estos artefactos son generados de acuerdo a la tecnología de implementación del proveedor cloud en el que se desplegarán los servicios.

- *Despliegue y reconfiguración dinámica*. En esta actividad, los especialistas en operaciones toman decisiones acerca de los recursos cloud necesarios para satisfacer los requisitos descritos durante la actividad *Especificar la integración del incremento*. Luego, usan transformaciones de modelos para generar artefactos de despliegue y reconfiguración dinámica específicos

de la plataforma cloud en la que los servicios incluidos en el incremento serán desplegados. La reconfiguración dinámica se realiza mediante: i) el despliegue de los servicios incluidos en el incremento, generados y empaquetados durante la actividad *Implementación del incremento*; ii) el despliegue (como otro servicio) de los artefactos que implementan la lógica de integración, generados durante la actividad *Implementación del incremento*; y iii) la ejecución de scripts de reconfiguración, que reemplazan en tiempo de ejecución los enlaces entre servicios incluidos en el incremento con los servicios incluidos en la aplicación actual. Finalmente, los desarrolladores ejecutan transformaciones que actualizan los artefactos de diseño que describen la arquitectura de la aplicación en la que se produjo la integración.

## 4.3 Descripción de las actividades del método

### 4.3.1 Especificación de la integración del incremento

La especificación de la integración de incrementos se lleva a cabo de una manera iterativa e incremental, identificando los servicios de la arquitectura actual de la aplicación que serán afectados por la integración de los servicios incluidos en el incremento hasta completar la especificación de la integración. En esta actividad los arquitectos especifican la lógica de integración y el impacto arquitectónico de la integración de los servicios incluidos en el incremento sobre la aplicación cloud actual. A continuación se describen los artefactos y tareas de esta actividad (ver Figura 4-3).

Esta actividad toma como entrada los artefactos:

- *Modelo de Arquitectura del Incremento*. Este modelo es expresado utilizando SoaML, describe la estructura y comportamiento del incremento a ser integrado. Incluye elementos que representan los componentes que proveen o consumen servicios, a los cuales se refiere de manera general como *servicios* en este trabajo de tesis, lo representamos en este modelo como elementos *Participant*. Adicionalmente incluye elementos que describen el protocolo de interacción entre servicios, representados como elementos *ServiceContract*.

- *SLA*. Este artefacto describe los términos de nivel de servicio relacionados a los servicios incluidos en un incremento. La definición y gestión de este artefacto esta fuera del alcance de este trabajo de tesis; sin embargo, es utilizado como insumo para que arquitectos, desarrolladores y

especialistas en operaciones tomen decisiones durante la ejecución de las diferentes actividades del método DIARy.

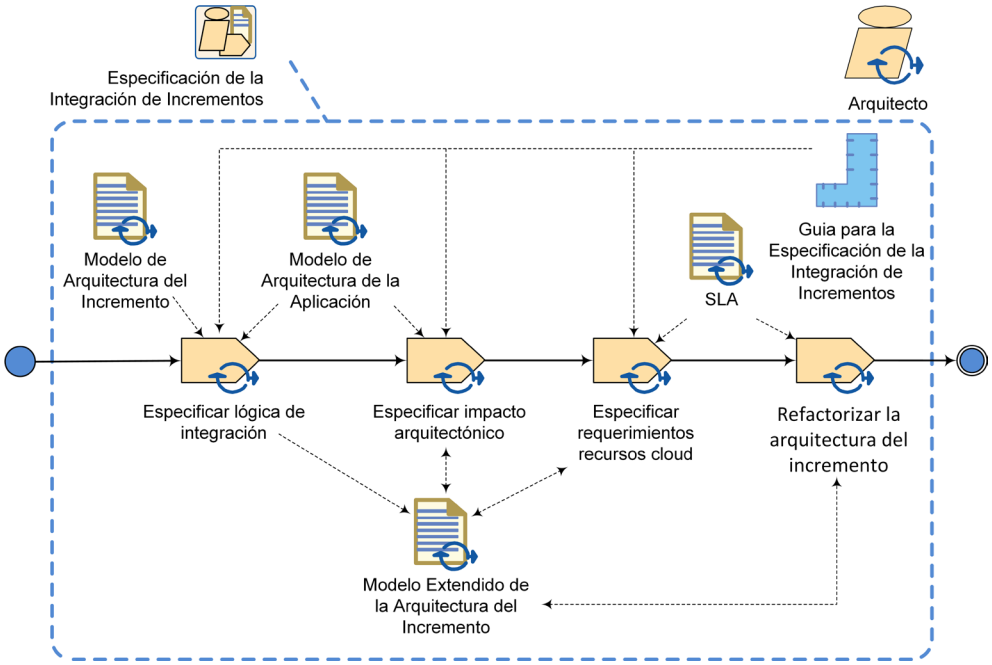



**Figura 4-3 Actividad Especificación de la integración del incremento**

- *Modelo de Arquitectura de la Aplicación*. Representa la estructura y comportamiento actuales de la aplicación cloud en la que se integrará el incremento, la cual será dinámicamente reconfigurada luego de la integración de cada incremento. Es decir, los servicios que conforman la aplicación ya desplegada, relaciones e interoperación entre ellos, y requisitos de recursos cloud de los servicios que conforman la aplicación. Este modelo es expresado utilizando el *ADL para la integración incremental de servicios cloud*.

Con el propósito de asistir al arquitecto, en esta actividad se provee la *Guía para la Especificación de la Integración de Incrementos*, presentada en el apéndice A.1. El artefacto resultante de esta actividad es:

- *Modelo Extendido de la Arquitectura del Incremento*. Este modelo se produce al complementar el *Modelo de Arquitectura del Incremento* con información que describe: i) como los elementos arquitectónicos del incremento colaboran para reconfigurar la arquitectura actual de la aplicación cloud, lo cual describe el impacto de la integración; ii) los recursos requeridos por los servicios incluidos en un incremento para satisfacer los términos de

SLA relacionados; y iii) la naturaleza y demanda esperada de cada servicio. El arquitecto completa sistemáticamente el *Modelo de Arquitectura del Incremento* con información relacionada a la especificación de la integración durante las tareas de esta actividad. Para ello utiliza el *ADL para la integración incremental de servicios cloud.*

Los modelos arquitectónicos utilizados como artefactos en esta actividad emplean SoaML o el *ADL para la integración incremental de servicios cloud* para su descripción. Estos lenguajes representan servicios como elementos de tipo *Participant*, y protocolos de interacción entre servicios como elementos *Service Contract*; por lo tanto, en adelante servicios y protocolos de interacción son referidos como elementos *Participant* y *Service Contract* respectivamente. La Sección 4.1.2 explica estos elementos.

#### *4.3.1.1 Especificación de la lógica de la integración*

El arquitecto especifica la lógica de integración a través de describir el protocolo de interacción entre los elementos *Participant* incluidos en el *Modelo de Arquitectura del Incremento* con elementos *Participant* del *Modelo de Arquitectura de la Aplicación;* la Figura 4-4 muestra la descomposición en pasos de esta tarea.

El arquitecto hace una copia del *Modelo de Arquitectura del Incremento*, creando el *Modelo Extendido de la Arquitectura del Incremento*, el cual será complementado utilizando el *ADL para la integración incremental de servicios cloud*. Por lo tanto, en adelante, al referirnos a los elementos del *Modelo Extendido de la Arquitectura del Incremento* haremos referencia tanto a los elementos que forman parte de la arquitectura del incremento (entrada de esta actividad), así como a los elementos incluidos en la arquitectura del incremento con el propósito de especificar la integración.

En el primer paso de esta actividad, el arquitecto, por cada uno de los roles que cumplen los elementos *Participant* definidos en el *Modelo Extendido de la Arquitectura del Incremento*, identifica en el *Modelo de Arquitectura de la Aplicación* (actual) los elementos *Participant* que cumplen esos roles. Luego, asigna el nombre de instancia del elemento *Participant* del *Modelo de Arquitectura de la Aplicación* al correspondiente en el *Modelo Extendido de la Arquitectura del Incremento*. Los elementos *Participant* identificados son los *Puntos de Integración* entre los modelos.

En el siguiente paso, el arquitecto determina si es que las interfaces definidas para los elementos *Participant* identificados en el paso anterior son compatibles con las interfaces definidas en el *Modelo Extendido de la Arquitectura del Incremento*. Si es que las interfaces no son compatibles, el arquitecto diseña *Adaptadores* que

corrigen las incompatibilidades detectadas y los incluye como operaciones en el diseño de las interfaces del elemento *Participant* incompatible.

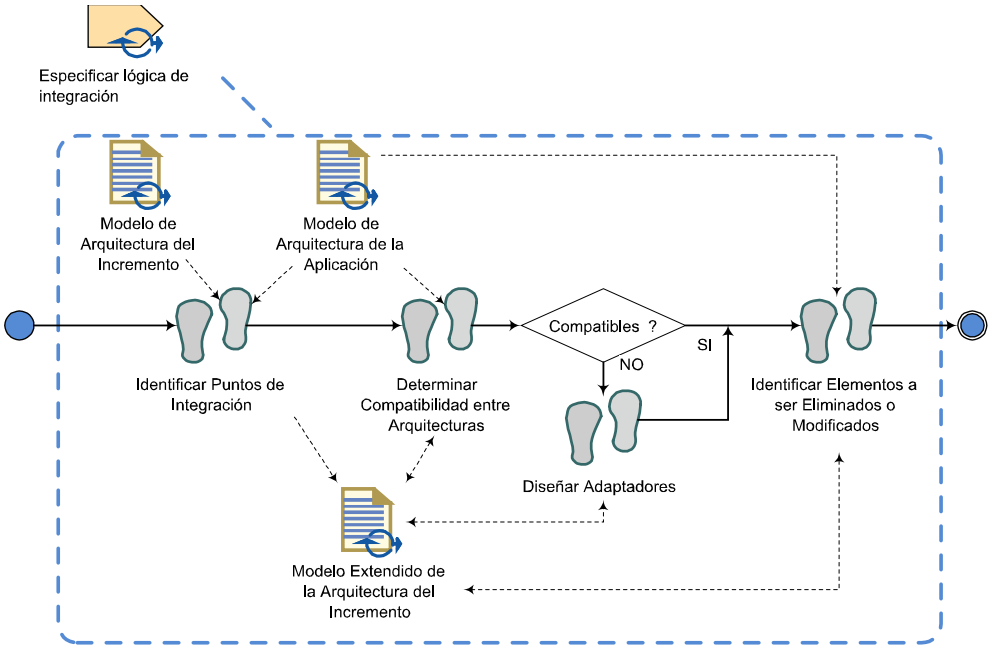



**Figura 4-4 Descomposición en pasos de la tarea Especificación de la lógica de integración**

En el tercer paso de esta actividad el arquitecto identifica en el *Modelo de Arquitectura de la Aplicación* (actual) los elementos *Participant* o *Service Contract* que serán eliminados o actualizados a causa de la integración. Luego, debido a que elementos a ser eliminados no forman parte del *Modelo Extendido de la Arquitectura del Incremento*, el arquitecto los copia desde el *Modelo de Arquitectura de la Aplicación*, incluyendo sus tipos, nombres de instancia y enlaces con otros elementos. Esto permitirá especificar que estos elementos deberán ser eliminados de la aplicación actual. En caso de los elementos a ser modificados el arquitecto copia únicamente el nombre de instancia. El *ADL para la integración incremental de servicios cloud* describe enlaces como elementos *RoleBinding*.

### *4.3.1.2 Especificación del impacto arquitectónico*

En esta actividad, una vez que la lógica de integración ha sido descrita, el arquitecto especifica cómo cada uno de sus elementos arquitectónicos incluidos en el *Modelo Extendido de la Arquitectura del Incremento* cambiará el *Modelo de Arquitectura de la Aplicación*. Para esto asignan el valor correspondiente (*Add*, *Modify*, *Delete* o *Reference*) al atributo *architectural Impact* de cada elemento del *Modelo Extendido de la*

*Arquitectura del Incremento*, tal como lo describe la *Guía para la Especificación de la Integración de Incrementos* (ver apéndice A.1) y el *ADL para la integración incremental de servicios cloud* (ver Sección 4.1.2.1).

#### *4.3.1.3 Especificación de requisitos de recursos cloud*

En esta actividad, una vez que el arquitecto ha especificado el impacto arquitectónico de la integración procede a analizar la naturaleza del trabajo de cada uno de los elementos *Participant* incluidos en el *Modelo Extendido de la Arquitectura del Incremento* y específica, a un alto nivel de abstracción, las características de los recursos cloud necesarias para satisfacer los términos SLA relacionados. Esta actividad incluye también el análisis de la naturaleza de trabajo de los elementos *Service Contract*, los cuales describen el protocolo de interacción entre elementos *Participant* y serán desplegados como servicios en etapas posteriores. Como resultado de éste análisis, los arquitectos especifican: i) la demanda esperada, mediante los atributos de *scalability* y *lifetime* en los elementos *Participant* y *Service Contract*; y ii) requisitos relacionados a la naturaleza del trabajo, mediante los atributos *High Processing*, *High Memory*, *High Storage*, *Static Content*, *Parallel Processing*, y *Low Energy Consumption* en elementos *Participant*. La información especificada en esta tarea es utilizada en etapas de desarrollo posteriores para determinar las características de los recursos cloud a ser aprovisionados con el propósito de satisfacer los términos de SLA de los incrementos desplegados en éstos.

#### *4.3.1.4 Refactorización de la arquitectura del incremento*

En este paso, con el propósito de satisfacer términos SLA o los requisitos de recursos cloud especificados en el paso anterior, y obtener beneficio de los entornos cloud, el arquitecto determina la necesidad de refactorizar la arquitectura descrita en el *Modelo Extendido de la Arquitectura del Incremento*. El arquitecto realiza la tarea de refactorización a través de aplicar patrones de diseño arquitectónicos o a través de incluir elementos arquitectónicos que representan recursos cloud que ayudarán a satisfacer los requisitos (p. ej., base de datos, balanceador de carga). El arquitecto incluye elementos *Participant* que representan recursos cloud, para lo cual asigna un valor al atributo *hostResource* (ver elementos de modelado del *ADL para integración incremental de servicios cloud* en la Sección 4.1.2.1). Adicionalmente, el arquitecto incluye elementos *Service Contract* para describir el protocolo de interacción entre elementos *Participant* que representan recursos cloud con elementos *Participant* que consumirán el servicio ofrecido por recursos cloud, y elementos *Role Binding* para definir roles y establecer enlaces entre ellos. Finalmente, el arquitecto establece el impacto arquitectónico de los elementos incluidos.

### 4.3.2 Implementación del incremento

Los artefactos cloud (artefactos de software específicos para la plataforma de despliegue) implementan tanto los elementos arquitectónicos del *Modelo Extendido de la Arquitectura del Incremento*, así como decisiones de implementación, aprovisionamiento y despliegue que satisfacen la especificación de la integración. Sin embargo, existen retos que deben ser superados para tomar ventaja de los entornos cloud: i) Los artefactos cloud deben satisfacer los requisitos de la plataforma cloud (p. ej., lenguaje de programación); ii) la organización de los artefactos cloud debe facilitar el despliegue/re-despliegue de servicios en diferentes plataformas cloud y facilitar su interoperabilidad; y iii) a pesar de que la aplicación de principios SOA facilita la reconfiguración dinámica de aplicaciones basadas en servicios, estos principios no son totalmente entendidos durante la implementación. Las aplicaciones basadas en servicios son generalmente desarrolladas en una manera *ad hoc*, sin tomar en cuenta técnicas de ingeniería de software (Perepletchikov y otros, 2008). En este contexto, no es suficiente especificar la integración aplicando principios SOA, se requiere que estos principios también sean cumplidos durante la implementación.

Con el propósito de hacer frente a estos retos, y siguiendo el enfoque DSDM propuesta en este trabajo de tesis, en esta actividad el desarrollador ejecuta transformaciones M2M que generan, agrupan y organizan descripciones de los artefactos cloud requeridos para implementar *Modelo Extendido de la Arquitectura del Incremento*. Lo anterior es documentado en un modelo a un nivel de abstracción independiente del proveedor cloud. Luego ejecuta transformaciones M2T que generan implementaciones de los artefactos cloud descritos previamente y scripts que empaquetan las implementaciones en artefactos de despliegue de acuerdo a la descripción de la organización. A continuación se describen los artefactos y tareas de esta actividad (ver Figura 4-5).

Esta actividad toma como entrada los artefactos:

- *Modelo Extendido de la Arquitectura del Incremento.*

Los artefactos resultantes de esta actividad son:

- *Modelo de Artefactos Cloud del Incremento.* Describe de forma independiente de la plataforma cloud de despliegue los artefactos cloud y recursos cloud requeridos para implementar los elementos arquitectónicos del *Modelo Extendido de la Arquitectura del Incremento* (p. ej., *Service Contract*, *Participant* y *Role Binding*). Para favorecer la interoperabilidad entre servicios desplegados en entornos cloud se requiere de mecanismos de orquestación

(Parameswaran y Chaddha, 2009). Por lo tanto, a través de permitir describir la estructura en la que se organizan los artefactos cloud, este modelo promueve la aplicación de principios que facilitan la reconfiguración dinámica y la implementación de un servicio cloud por cada elemento *Participant*, así como un servicio cloud de orquestación que implemente el protocolo de interacción o lógica de integración definidos en cada elemento *Service Contract*.

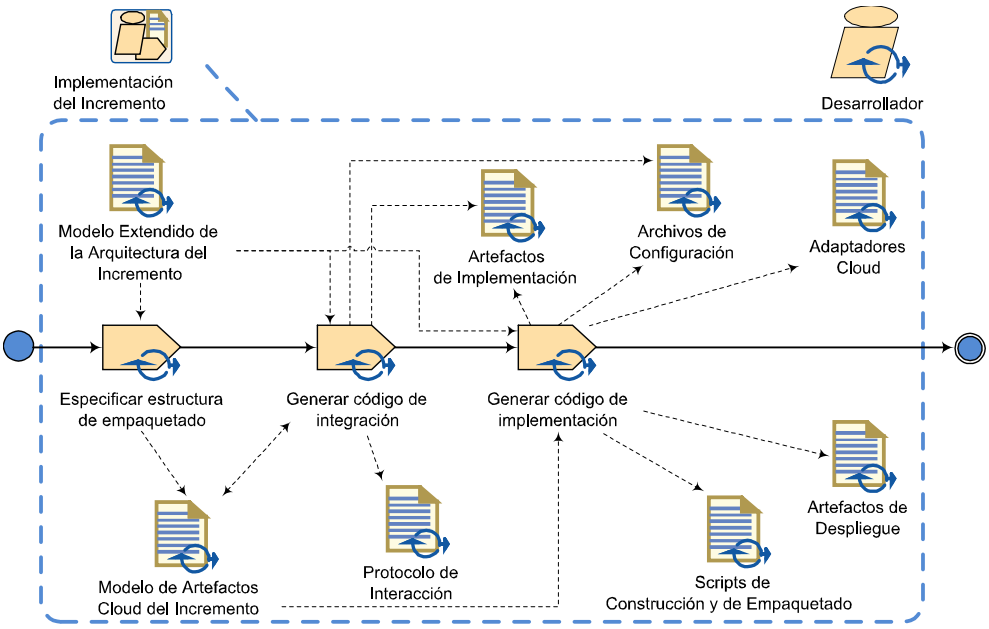



**Figura 4-5 Actividad Implementación del incremento**

- *Protocolo de Interacción.* Artefacto cloud específico para la plataforma cloud en la que serán desplegados los servicios de orquestación. Implementa el protocolo de interacción o lógica de integración de elementos *Service Contract* en el *Modelo Extendido de la Arquitectura del Incremento*. Este artefacto será desplegado como servicio cloud de orquestación.

- *Artefactos de Implementación.* Es un conjunto de artefactos que implementan la lógica de operación expuesta por los servicios incluidos en el incremento y descrita por elementos *Participant* en el *Modelo Extendido de la Arquitectura del Incremento*. Estos artefactos implementan definiciones interfaces, mensajes y tipos de datos. Son empaquetados y desplegados como parte de servicios cloud de orquestación o servicios cloud que exponen operaciones de los participantes incluidos en el incremento, dependiendo de cómo se haya definido su organización.

- *Archivos de Configuración.* Artefactos cloud específicos para la plataforma de despliegue. Contienen ajustes en configuraciones que cambian en tiempo de ejecución, correspondientes a los servicios incluidos en un incremento. Por ejemplo, información de puntos de acceso (*End Points*) a servicios relacionados. Estos artefactos son desplegados de manera independiente de paquetes que contienen los artefactos de implementación del servicio, evitando desplegar nuevamente paquetes de implementación cuando se producen cambios en ajustes, lo cual facilita la reconfiguración dinámica. Pueden existir tantos archivos de configuración como entornos de despliegue (p. ej., desarrollo, pruebas, producción).

- *Adaptadores Cloud.* Corrigen incompatibilidades entre interfaces de servicios (p. ej., nombres de métodos diferentes, orden de parámetros incorrecto), permitiendo su interoperabilidad. Son invocados desde artefactos cloud que implementan la lógica de operación expuesta por los participantes incluidos en el incremento, y forman parte del servicio cloud correspondiente.

- *Scripts de Construcción y Empaquetado.* Scripts específicos para el Entorno de Desarrollo Integrado (Integrated Development Environment – IDE) o marco de desarrollo utilizado en el que se construirán los servicios. Estos scripts ensamblan los componentes del incremento, incorporan las librerías necesarias para su ejecución, las compilan, vinculan y empaquetan.

- *Artefactos de Despliegue.* Representa las unidades de distribución que serán desplegadas en una plataforma cloud (p. ej., archivos con extensión cab, exe, war, zip, jar). Estos artefactos son generados como resultado de la ejecución de los *Scripts de Construcción y Empaquetado.* Se genera un artefacto de despliegue (p. ej., paquete) por cada elemento *Participant* y por cada elemento *Service Contract* (o un artefacto de despliegue por varios elementos *Participant* consolidados y por varios elementos *Service Contract* consolidados).

#### *4.3.2.1 Especificación de la estructura del empaquetado y despliegue*

En esta actividad, el desarrollador ejecuta transformaciones M2M que toman como entrada el *Modelo Extendido de la Arquitectura del Incremento* y generan el *Modelo de Artefactos Cloud del Incremento.* El *Modelo de Artefactos Cloud del Incremento* no solo define una estructura de empaquetado y despliegue que organiza los artefactos cloud en proyectos de tal manera que promueve el cumplimiento de principios SOA u otros principios que facilitan la reconfiguración dinámica durante

la implementación; sino también desacopla los artefactos cloud que implementan la orquestación entre servicios (descrita por el protocolo de interacción o lógica de integración de elementos *Service Contract*) de los artefactos cloud que implementan la lógica de operación expuesta por los servicios incluidos en el incremento (descrita por elementos *Participant*). La estructura de empaquetado y despliegue facilita el despliegue/re-despliegue de servicios en diferentes plataformas cloud. Adicionalmente, las transformaciones M2M definidas propagan el impacto arquitectónico especificado en el *Modelo Extendido de la Arquitectura del Incremento* hacia las descripciones de los artefactos cloud requeridos para la implementación; por lo que los artefactos cloud descritos también incluyen el atributo *architectural Impact* que indican las acciones que se deberán implementar en los diferentes artefactos en actividades posteriores (p. ej., crear/eliminar artefactos de implementación, desplegar/replegar servicios). El mapeo entre los elementos arquitectónicos del *Modelo Extendido de la Arquitectura del Incremento* con elementos del *Modelo de Artefactos Cloud del Incremento* se presenta en la Figura 4-6.

Las transformaciones M2M generan Proyectos de Interacción (*Interaction Project*) que incluyen descripciones de artefactos cloud que implementarán la orquestación entre participantes; es decir implementarán los elementos interiores de elementos *Service Contract* descritos en el *Modelo Extendido de la Arquitectura del Incremento*. Las descripciones de artefactos cloud incluidos en este tipo de proyectos son: Interfaces (*Interface*), que son implementadas por elementos *Participant* con el propósito de interactuar consumiendo o proveyendo servicios; tipos de mensajes (*MessageType*)que se intercambian entre los participantes durante la interacción; tipos de datos (*DataType*) que se utilizan en los parámetros de los mensajes transmitidos; y el protocolo de interacción (*Interaction Service*) entre elementos *Participant*. En las siguientes tareas de esta actividad, la implementación de los artefactos descritos en proyectos *Interaction Project* es empaquetada y desplegada como un servicio cloud de orquestación.

Por otro lado, las transformaciones M2M generan Proyectos de Implementación (*Implementation Project*) que incluyen descripciones de artefactos cloud que implementarán la lógica de operación expuesta por los elementos *Participant* incluidos en el incremento. Proyectos *Implementation Project* se corresponden con elementos *Participant* que cumplen un rol definido en elementos *Service Contract* en el *Modelo Extendido de la Arquitectura del Incremento*. Estos proyectos incluyen descripciones de los siguientes artefactos cloud: Implementación de las interfaces correspondientes al rol que cumple un elemento *Participant*, implementación denominada *Front End Service*. Implementación de las interfaces correspondientes al rol que cumple un elemento *Participant* que representan recursos cloud consumidos por otro elemento *Participant* del incremento, implementación denominada *Back End*

*Service*. Implementación de las operaciones incluidas en interfaces con el propósito de que corregir incompatibilidades entre interfaces de servicios, implementación denominada *Cloud Adaptors*. Adicionalmente, los proyectos *Implementation Project* que corresponden a elementos *Participant* que cumplen el rol de consumidor de servicio incluyen un artefacto cloud (*Client Object*) que implementa la lógica necesaria para iniciar la ejecución del servicio a través invocar al servicio cloud de orquestación correspondiente.

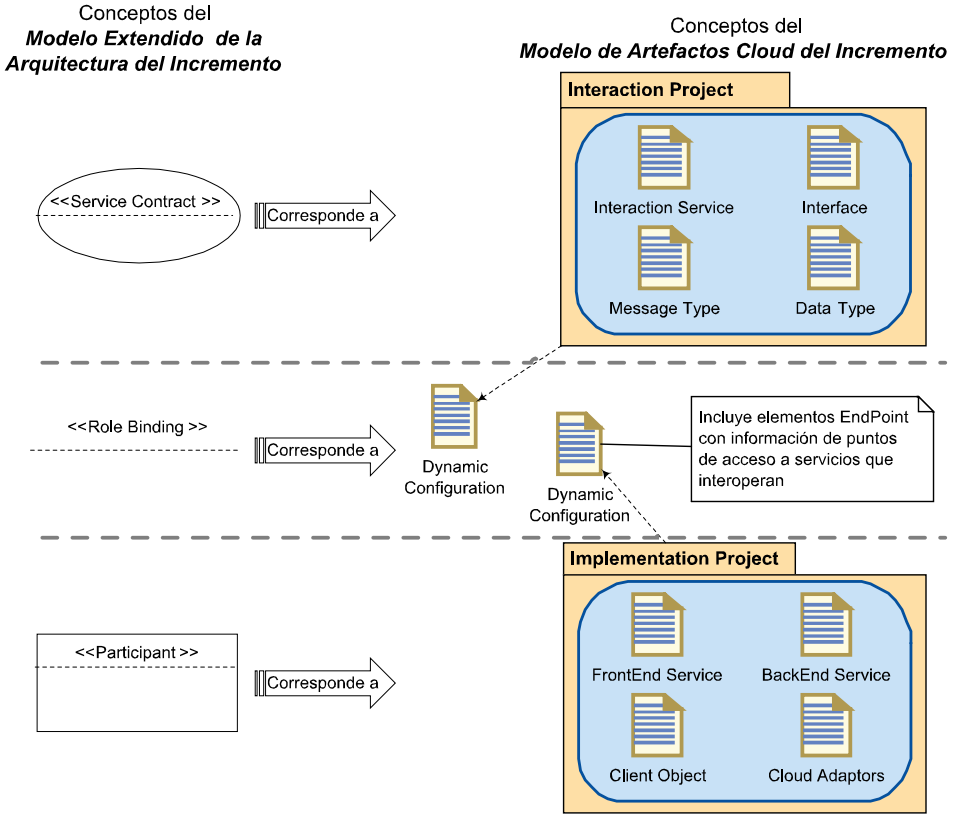



**Figura 4-6 Mapeo entre elementos del *Modelo Extendido de la Arquitectura del Incremento* y artefactos cloud del *Modelo de Artefactos Cloud del Incremento***

Los patrones de diseño cloud para la reconfiguración dinámica y las mejores prácticas en entrega continua (Continuous Delivery – CD) sugieren que ajustes en configuraciones que cambian en tiempo de ejecución sean almacenados fuera del paquete de despliegue que contiene los artefactos de implementación del servicio (Homer y otros, 2014) (Fehling y otros, 2014); permitiendo que estos ajustes sean actualizados sin que requiera desplegar nuevamente todo el paquete. Por lo tanto, con el propósito de facilitar la reconfiguración arquitectónica dinámica

a través de actualizar las dependencias entre servicios, el *Modelo de Artefactos Cloud del Incremento* incluye descripciones de artefactos *Dynamic Configuration*. Estos artefactos además de incluir ajustes (*Settings*) de servicios que cambiarán en tiempo de ejecución, son mapeados a elementos *Role Binding* del *Modelo Extendido de la Arquitectura del Incremento* con el propósito de obtener información acerca de los puntos de acceso (*End Points*) que los elementos *Participant* involucrados en la interacción utilizan para exponer sus funcionalidades o para invocar a elementos *Participant* dependientes.

La manera en la cual los servicios son desplegados tiene influencia en el cumplimiento de los términos de SLA u otros requisitos no funcionales (p. ej., agilidad en el despliegue, costo de ejecución) (Costa y otros, 2014). Por lo tanto, con el propósito facilitar el cumplimiento de términos SLA y brindar flexibilidad en la toma de decisiones de despliegue, las transformaciones M2M consolidan en proyectos *Deployment Project* los proyectos *Interaction Project* e *Implementation Project* correspondientes a elementos *Service Contract* o *Participant* con igual impacto arquitectónico y requisitos de uso de recursos cloud similares. Proyectos *Interaction Project* y proyectos *Implementation Project* no son consolidados en un mismo proyecto *Deployment Project*.

En la Figura 4-7(a) se presenta un ejemplo en el cual transformaciones M2M han consolidado dos proyectos *Interaction Project* en un único proyecto *Deployment Project*. Adicionalmente, el proyecto *Deployment Project* ha sido relacionado con un artefacto *Dynamic Configuration* en el que se incluirán los ajustes de los servicios de orquestación relacionados a los proyectos *Interaction Project* consolidados.

Posteriormente, los proyectos *Interaction Project* consolidados en el *Deployment Project* serán empaquetados en un mismo *Artefacto* de *Despliegue* y desplegados en una misma unidad de computo. Los servicios consolidados en un proyecto *Deployment Project* compartirán durante la ejecución los recursos ofrecidos por los proveedores cloud en los que son desplegados (p. ej., máquina virtual, entorno de ejecución, almacenamiento). La Figura 4-7(b) muestra un ejemplo en el cual un único proyecto *Implementation Project* es incluido en proyecto *Deployment Project*; por lo tanto, el servicio relacionado al proyecto de implementación hará uso exclusivo de los recursos ofrecidos por los proveedores cloud en los que es desplegado.

Finalmente, luego de ejecutar las transformaciones M2M el desarrollador analiza los requisitos especificados en el *Modelo Extendido de la Arquitectura del Incremento* y, en caso de considerarlo necesario, modificará la especificación de consolidación en proyectos de despliegue a través de asignar *Interaction Project* o *Implementation Project* a diferentes *Deployment Project*. La aproximación tecnológica aplicada

para implementar los conceptos definidos para el *Modelo de Artefactos Cloud del Incremento* se presenta en la Sección 5.3.

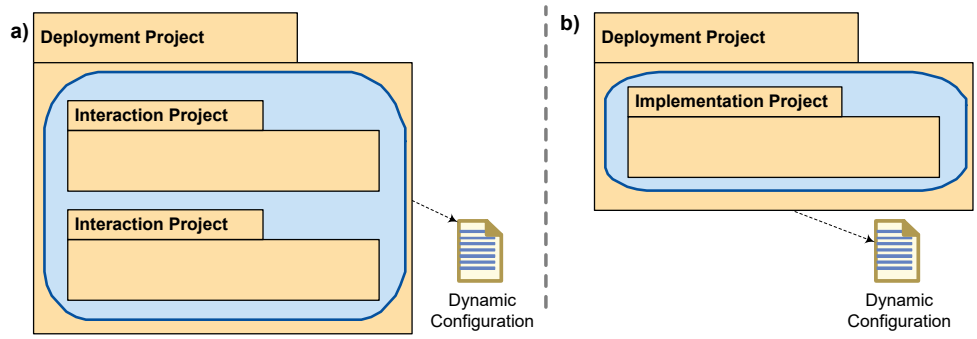



**Figura 4-7 Consolidación de artefactos cloud en proyectos de despliegue**

### *4.3.2.2 Generación del código de integración*

En esta actividad, el desarrollador toma decisiones de implementación que mejor cumplen los requisitos de los servicios incluidos en un incremento y completa el *Modelo de Artefactos Cloud del Incremento* generado previamente, especificando: i) la tecnología en la cual cada artefacto descrito en el modelo será implementado, por ejemplo lenguaje de programación o representación del artefacto (p. ej., clase, archivo XML, Archivo BPEL, WCF Workflow de Microsoft Azure©[7]); ii) ajustes de servicios que cambian en tiempo de ejecución, a través de actualizar o crear descripciones de artefactos tipo *Dynamic Configuration* o *Setting*; iii) información acerca de la implementación de servicios, por ejemplo estilo de servicio SOAP/REST); y iv) la ubicación de los artefactos cloud a ser generados.

Una vez que el desarrollador ha completado el *Modelo de Artefactos Cloud del Incremento*, ejecuta transformaciones M2T que toman como entrada tanto las descripciones de proyectos *Interaction Project* de este modelo como el diseño arquitectónico documentado en el *Modelo Extendido de la Arquitectura del Incremento* y generan la implementación de elementos arquitectónicos *Service Contract* de acuerdo a la tecnología especificada en el *Modelo de Artefactos Cloud del Incremento*. Adicionalmente, consideran el impacto arquitectónico propagado hacia el *Modelo de Artefactos Cloud del Incremento* a través del atributo *Architectural Impact* en las diferentes descripciones de artefactos cloud; si su valor es *Add* o *Modify* genera artefactos que implementan el artefacto cloud o lo modifican, si su valor es *Delete* genera scripts que eliminan los artefactos existentes o sus dependencias.

[7] https://azure.microsoft.com/

Las reglas de transformación tienen las siguientes consideraciones para los artefactos de salida generados en esta tarea:

- *Protocolo de Interacción.* Transformaciones M2T generan estos artefactos tomando información de descripciones de elementos *Interaction Service* del *Modelo de Artefactos Cloud del Incremento* (p. ej., información de *Architectural Impact* o tipo de representación del artefacto tal como un archivo BPEL o WCF Workflow de Microsoft Azure©) e implementando el diseño del protocolo de interacción descrito como elemento interno de elementos arquitectónicos *Service Contract* del *Modelo Extendido de la Arquitectura del Incremento.*

- *Artefactos de Implementación.* Transformaciones M2T generan estos artefactos tomando información de descripciones de elementos *Interface, Message Type* y *Data Type* del *Modelo de Artefactos Cloud del Incremento* e implementando el diseño de los elementos de internos de elementos arquitectónicos *Service Contract* del *Modelo Extendido de la Arquitectura del Incremento.* Estos artefactos, conjuntamente con el *Protocolo de Interacción* implementan la lógica de integración o interacción entre elementos *Participant* incluidos en el incremento.

- *Archivos* de *Configuración*. Transformaciones M2T generan estos artefactos tomando información de elementos *Dynamic Configuration* y *Setting* del *Modelo de Artefactos Cloud del Incremento* y elementos *Role Binding* del *Modelo Extendido de la Arquitectura del Incremento.* Por cada definición de artefacto *Dynamic Configuration* relacionada con una definición de elementos *Interaction Service* se genera la implementación de un *Archivo* de *Configuración* y por cada definición de elemento *Setting* relacionado se genera una entrada (p. ej., se declara una variable o parámetro) en el *Archivo* de *Configuración*. Artefactos *Interaction Services* gestionan la interacción a través de invocar operaciones de los participantes involucrados, en donde los puntos de acceso a los servicios ofrecidos por los participantes involucrados son documentados en elementos *Setting*; por lo tanto, el *Archivo* de *Configuración* incluye también entradas con información a esos puntos de acceso.

Dependiendo de la estructura del empaquetado y despliegue especificada previamente, en la siguiente tarea y actividad, el desarrollador organiza estos artefactos en proyectos *Deployment Project* que son empaquetados y desplegados como servicios de orquestación, referidos en este trabajo como servicios cloud de orquestación.

#### *4.3.2.3 Generación del código de implementación*

Una vez que el desarrollador ha generado la implementación de los servicios de orquestación ejecuta transformaciones M2T que toman como entrada tanto descripciones de proyectos *Implementation Project* del *Modelo de Artefactos Cloud del Incremento* como el diseño arquitectónico de elementos *Participant* documentado en el *Modelo Extendido de la Arquitectura del Incremento* y generan la implementación de elementos arquitectónicos *Participant* de acuerdo a la tecnología especificada en la descripción de los artefactos cloud. Las reglas de transformación incluidas en las transformaciones M2T consideran los valores de atributos *Architectural Impact*, tal como se explicó anteriormente.

Las salidas que se generan a través de transformaciones M2T (ver Figura 4-5) y las reglas de transformación tienen las siguientes consideraciones para los artefactos de salida generados en esta tarea:

- *Archivos* de *Configuración*. La interacción entre elementos *Participant* que consumen y proveen servicios inicia cuando el elemento *Participant* que cumple el rol de consumidor invoca al servicio cloud de orquestación. Por lo tanto, a diferencia del archivo de configuración relacionado a descripción de artefactos *Interaction Service*, los archivos de configuración relacionados a descripciones de elementos *Front End Service* incluyen entradas con información de los puntos de acceso a los servicios de orquestación invocadas para iniciar la interacción.

- *Artefactos de Implementación*. Transformaciones M2T generan estos artefactos tomando información de descripciones de elementos *Front End Service* o *Back End Service* del *Modelo de Artefactos Cloud del Incremento* e implementan el diseño de elementos arquitectónicos *Participant*. El nivel de detalle de la implementación depende del nivel de detalle del diseño de elementos *Participant*, pudiendo ser esqueletos de implementación de interfaces o implementaciones completas. Adicionalmente, en caso de elementos *Participant* que cumplen el rol de consumidores de servicio, Transformaciones M2T generan implementaciones de descripciones de artefactos *Client Object*. En la implementación de artefactos *Client Object* se incluyen instrucciones, que antes de invocar al servicio cloud de orquestación, leen del *Archivo de Configuración* correspondiente el punto de acceso a éste servicio.

- *Scripts de Construcción y de Empaquetado*. Transformaciones M2T generan estos artefactos tomando información de descripciones de proyectos *Deployment Project* e información de la estructura de empaquetado y desplie-

gue en base a sus relaciones con elementos *Interaction Project* e *Implementation Project*. Las reglas de transformación incluidas en las transformaciones M2T generan *Scripts de Construcción y de Empaquetado* para proyectos *Deployment Project* cuyo impacto arquitectónico es *Add* o *Modify;* sin considerar aquellos proyectos cuyo impacto arquitectónico es *Delete*.

- A*daptadores Cloud*. Transformaciones M2T generan estos artefactos tomando información de descripciones de elementos *Cloud Adaptor* e implementando las operaciones descritas como adaptadores en las interfaces de elementos *Participant*.

Una vez que el desarrollador ha ejecutado las transformaciones M2T anexa los artefactos *Protocolo de Interacción*, *Artefactos de Implementación*, *Archivos* de *Configuración* y A*daptadores Cloud* en el IDE utilizado para el desarrollo, completa los artefactos o esqueletos de artefactos incluidos, y ejecuta los *Scripts de Construcción y de Empaquetado* previamente generados; obteniéndose la última salida de esta actividad, *Artefactos de Despliegue*.

### 4.3.3 Despliegue y reconfiguración dinámica

La reconfiguración dinámica de la arquitectura se produce debido al despliegue de *Artefactos de Despliegue*, los cuales incluyen los servicios correspondientes al incremento de software a ser integrado. En donde, el despliegue crea instancias de los servicios incluidos en *Artefactos de Despliegue* en un entorno aprovisionado (reservado o preparado) para su ejecución, que en este trabajo de tesis se lo denomina *Entorno Cloud* (*Cloud Environment*). En el aprovisionamiento, paso inicial de la actividad de despliegue, el especialista en operaciones crea *Cloud Environments* a través de seleccionar, reservar, crear o configurar instancias de recursos cloud ofrecidos por proveedores cloud.

La automatización del aprovisionamiento y despliegue de aplicaciones cloud son factores clave para la reducción de costos en actividades de operación (Leymann, 2009). Sin embargo, los proveedores cloud ofrecen recursos cloud con diferentes funcionalidades o características, y proveen soluciones propietarias para aprovisionar recursos en sus nubes (p. ej., basados en scripts, servicios Web, APIs o herramientas propias). Esto dificulta la selección y comparación de recursos cloud, así como la automatización de tareas de aprovisionamiento, despliegue y reconfiguración dinámica.

Con el propósito de hacer frente a la heterogeneidad en la oferta de recursos cloud y mecanismos de aprovisionamiento o despliegue provistos por proveedores cloud, en esta actividad el especialista en operaciones: i) documenta sus

decisiones de aprovisionamiento y despliegue en un modelo cuyo nivel de abstracción le permite especificar los *Cloud Environment* en los cuales desplegará los servicios incluidos en *Artefactos de Despliegue* de acuerdo a ofertas específicas de un proveedor cloud; y ii) ejecuta transformaciones M2T que generan scripts de aprovisionamiento, despliegue y reconfiguración dinámica. A continuación se describen los artefactos y tareas de esta actividad (ver Figura 4-8).

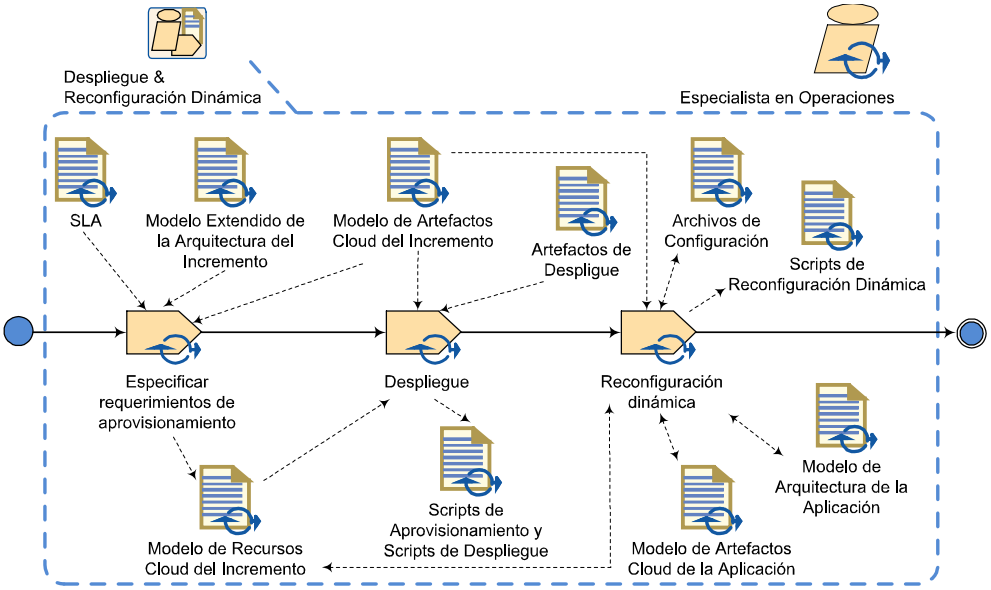



**Figura 4-8 Actividad Despliegue y reconfiguración dinámica**

Esta actividad toma como entrada los artefactos:

- *Modelo Extendido de la Arquitectura del Incremento*, *SLA*, *Modelo de Artefactos Cloud del Incremento*, *Artefactos de Despliegue*

Los artefactos resultantes de esta actividad son:

- *Modelo de Recursos Cloud del Incremento*. Describe los recursos cloud que tienen que ser aprovisionados para el despliegue de *Artefactos de Despliegue* y la ejecución de los servicios incluidos en este. Adicionalmente, describe la información de subscripción (p. ej., credenciales, parámetros de conexión) necesarias para gestionar el aprovisionamiento y despliegue en el (o los) proveedor cloud seleccionado para el despliegue.
- *Scripts de Aprovisionamiento y Scripts de Despliegue*. Incluyen instrucciones para aprovisionar los recursos cloud (aprovisionar el *Cloud Environment*), descritos en el *Modelo de Recursos Cloud del Incremento*, necesarios para la ejecución de los servicios incluidos en *Artefactos de Despliegue*. Incluyen

también instrucciones para el desplegar *Artefactos de Despliegue*. Estos scripts satisfacen los requisitos de los mecanismos de aprovisionamiento y despliegue específicos de los proveedores cloud seleccionados.

- *Scripts de Reconfiguración Dinámica*. Estos scripts actualizan las entradas de los *Archivos de Configuración* relacionados a servicios cloud de orquestación o a servicios ofrecidos por elementos *Participant* que cumplen un rol de consumidor de servicio. Las entradas actualizadas corresponden a los puntos de acceso (End Points) a los servicios con los que interactúan. Adicionalmente, este conjunto de artefactos incluye scripts para el despliegue de los *Archivos de Configuración* actualizados. Estos scripts son específicos de la tecnología de implementación y proveedor cloud de despliegue.

#### *4.3.3.1 Especificación de requisitos de aprovisionamiento*

*Cloud Environments* están conformados por una infraestructura virtual, respaldada por una infraestructura física, y por un conjunto de servicios a nivel de plataforma que soportan la ejecución de servicios cloud; todos con sus características, relaciones y requisitos; los primeros ofrecidos por proveedores cloud bajo un modelo de servicios IaaS y los segundos en un modelo de servicios PaaS.

La oferta de *Cloud Environments* por parte de proveedores cloud puede estar restringida a la selección de configuraciones o instancias predefinidas de *Cloud Environments* (p. ej., "m1.large" de Amazon EC2 o "Standard A2" de Microsoft Azure©) clasificadas de acuerdo a su poder computacional; o ser flexibles, permitiendo al especialista en operaciones personalizar las características de los recursos cloud que conformarán el *Cloud Environment*. En donde, generalmente, cloud públicas ofrecen instancias de *Cloud Environment* predefinidas, mientras cloud privadas permiten crear instancias personalizadas.

En esta actividad, una vez que los equipos de desarrollo han implementado los servicios de la nube y los han liberado en forma de *Artefactos de Despliegue*, el analista en operaciones prepara estos artefactos para su despliegue en el entorno de producción (o entorno de pruebas u otro), el mismo que será aprovisionado en uno o varios proveedores cloud. Para esto, el especialista en operaciones especifica los recursos en la nube necesarios para satisfacer los requisitos de los servicios incluidos el *Artefacto de Despliegue*; documentando estos requisitos en el *Modelo de Recursos Cloud del Incremento*. El especialista en operaciones escoge tanto los proveedores cloud como el modelo de servicios (p. ej., IaaS o PaaS) que adoptará por cada proveedor para desplegar los *Artefactos de Despliegue*. Luego, selecciona, de entre las ofertas de cada proveedor, ya sea instancias de *Cloud En-*

*vironment* predefinidas o las personaliza seleccionando los recursos cloud que satisfagan los requisitos. El especialista en operaciones toma decisiones basándose en: i) los servicios empaquetados en los *Artefactos de Despliegue*, documentados en el *Modelo de Artefactos Cloud del Incremento*; ii) los requisitos de recursos cloud correspondientes a los servicios empaquetados en cada *Artefacto de Despliegue*, documentados en el *Modelo Extendido de la Arquitectura del Incremento*; iii) términos SLA; iv) características de los recursos cloud ofertadas por cada proveedores cloud; v) costo de recursos cloud; y vi) otros requisitos de negocio. Adicionalmente, utiliza el *Modelo de Recursos Cloud del Incremento* para describir información de subscripción de los proveedores cloud en los que se desplegarán los servicios incluidos en los *Artefactos de Despliegue*.

#### *4.3.3.2 Despliegue*

Luego de especificar los requisitos de aprovisionamiento, el especialista en operaciones ejecuta transformaciones M2T que toman como entrada el *Modelo de Recursos Cloud del Incremento* y generan *Scripts de Aprovisionamiento* de *Cloud Environments* específicos para un proveedor cloud. El modelo de servicios (p. ej., IaaS o PaaS) elegido por el especialista determina el mayor o menor control que tendrá sobre el aprovisionamiento y despliegue, así como el nivel de abstracción de los scripts generados. Por lo tanto, existirán diferentes transformaciones M2T dependiendo del proveedor elegido para el despliegue y del modelo de servicios seleccionado de cada uno de ellos.

Las instrucciones de aprovisionamiento incluidas en los *Scripts de Aprovisionamiento* son acordes al impacto arquitectónico del proyecto *Deployment Project* en base al cual se generaron los *Artefactos de Despliegue* durante la actividad *Implementación del Incremento* (ver Sección 4.3.2). Es decir, las transformaciones M2T aprovisionan un *Cloud Environment* si es que el impacto arquitectónico del proyecto *Deployment Project* es *Add*, o lo re-configuran si es que es *Modify*. Si el impacto arquitectónico del proyecto *Deployment Project* es *Delete* significa que el *Cloud Environment* de los servicios empaquetados en el *Deployment Project*, y actualmente desplegados, debe ser desaprovisionado. Por lo tanto, las transformaciones M2T generan *Scripts de Aprovisionamiento* que liberan los recursos cloud actualmente reservados.

Una vez que el especialista ha generado *Scripts de Aprovisionamiento* (o desaprovisionamiento), ejecuta transformaciones M2T que generen *Scripts de Despliegue* (o repliegue) dependiendo del impacto arquitectónico, de manera similar a la generación de *Scripts de Aprovisionamiento*. Finalmente, el especialista en operaciones ejecuta los scripts generados previamente, desplegando los servicios incluidos en el *Modelo Extendido de la Arquitectura del Incremento* e integrándolos a la arquitectura

actual de la aplicación de acuerdo al impacto arquitectónico especificado durante la actividad Especificación de la integración del incremento (ver 4.3.1).

#### *4.3.3.3 Reconfiguración dinámica*

Una vez que el especialista en operaciones ha desplegado los servicios del incremento (incluidos los servicios cloud de orquestación) actualiza el *Modelo de Recursos Cloud del Incremento* con información de los puntos de acceso (*End Point*) utilizados por los servicios desplegados tanto para exponer sus funcionalidades como para invocar otros servicios. Luego ejecutan transformaciones M2T que generan *Scripts de Reconfiguración* que: i) actualizan las entradas de *Archivos de Configuración* que tienen información de *End Points* de acuerdo al impacto arquitectónico, modificándolas o eliminándolas; ii) actualizan las entradas de *Archivos de Configuración* correspondientes a los ajustes (*Setting*) de los servicios incluidos en *Artefactos de Despliegue* (p. ej., número de instancias) y que cambian en tiempo de ejecución; y iii) despliegan los *Archivos de Configuración* actualizados. Luego, el especialista en operaciones ejecuta los *Scripts de Reconfiguración*, desplegando los *Archivos de Configuración* actualizados sin afectar la ejecución de la aplicación.

El despliegue de *Archivos de Configuración* reconfigura dinámicamente la arquitectura de la aplicación a través de establecer o actualizar los enlaces entre los servicios desplegados debido a la integración del incremento con los servicios existentes en la aplicación actual.

Finalmente, el arquitecto ejecuta transformaciones M2M que actualizan los artefactos de diseño de la aplicación actual, incorporando elementos del incremento en estos artefactos de diseño. Una primera transformación toma como entrada el *Modelo Extendido de la Arquitectura del Incremento* y el *Modelo de Arquitectura de la Aplicación* actual y actualizan este último incluyendo los elementos arquitectónicos del primer modelo de acuerdo al impacto arquitectónico especificado para cada uno de ellos. Por otro lado, una última transformación toma como entrada el *Modelo de Artefactos Cloud del Incremento* y el *Modelo de Artefactos Cloud de la Aplicación* actual e integra los elementos del primer modelo en el segundo.

## 4.4 Conclusiones

En este capítulo se ha presentado la principal contribución metodológica de esta tesis: la definición del método DIARy, el cual brinda soporte a la reconfiguración dinámica de la arquitectura de aplicaciones cloud producida por la integración incremental de servicios cloud.

El método propuesto define un proceso que sigue un enfoque DSDM y que define un conjunto de actividades sistematizadas que facilitan la reconfiguración dinámica en entornos cloud. Adicionalmente propone un conjunto de modelos a diferentes niveles de abstracción como artefactos de entrada y salida de las actividades que los transforman.

En una primera actividad, el arquitecto utiliza el *ADL para la integración incremental de servicios cloud*, el cual le permite centrarse en sus actividades y especificar, en un alto nivel de abstracción, el impacto arquitectónico que tendrá la integración sobre la aplicación actual, la lógica de integración y los requisitos de recursos cloud. En la segunda actividad el desarrollador utiliza modelos que promueven, durante la implementación, la aplicación de principios de ingeniería de software que facilitan la reconfiguración dinámica. Luego ejecutan transformaciones de modelos que generan artefactos de software específicos para un proveedor cloud, que implementen la especificación. En la última actividad, el especialista en operaciones utiliza modelos que le permiten describir, utilizando conceptos independientes de un proveedor cloud específico, el entorno cloud en el que se desplegarán los servicios a ser integrados; satisfaciendo además los requisitos especificados en la primera actividad. Luego se ejecutan las transformaciones de modelos que generan los scripts, específicos para los proveedores cloud seleccionados, que automatizan la integración de servicios en la aplicación actual. De esta forma se da soporte al aprovisionamiento de entornos cloud, al despliegue de los servicios a ser integrados y finalmente a la reconfiguración dinámica de la arquitectura actual de la aplicación cloud.

En el siguiente capítulo se muestra una aproximación tecnológica para soportar las actividades del proceso de DIARy, desde la especificación de la integración del incremento, pasando por la implementación del incremento hasta el despliegue de los servicios incluidos en el incremento y la reconfiguración dinámica de la aplicación actual.

# Capítulo 5. Herramientas de soporte a DIARy

En este capítulo se presenta la aproximación tecnológica que da soporte a las actividades del método de reconfiguración dinámica e incremental de arquitecturas de aplicaciones cloud descrito en el capítulo anterior. Esta aproximación consiste en un prototipo completamente funcional de la infraestructura software desarrollada en el contexto de esta tesis doctoral; el cual es utilizado para ilustrar la aplicabilidad de las actividades del método mediante el ejemplo descrito en el Capítulo 6.

La sección 5.1 describe, en forma general, la aproximación tecnológica propuesta en el método DIARy, sus componentes y relaciones entre ellos.

La sección 5.2 describe la aproximación tecnológica de soporte a la actividad de *Especificación de la integración del incremento*.

La sección 5.3 describe la aproximación tecnológica de soporte a la actividad *Implementación del incremento*.

La sección 5.4 describe la aproximación tecnológica de soporte a la actividad *Despliegue y reconfiguración dinámica*.

La sección 5.5 describe la aproximación tecnológica de soporte a la generación de modelos que reflejan los aspectos de implementación y despliegue; así como la generación de código de integración, implementación, y scripts de despliegue y reconfiguración dinámica.

La sección 5.6 presenta las conclusiones del capítulo.

## 5.1 Vista general de la aproximación tecnológica

En esta sección presentamos una infraestructura software que ha sido concebida para facilitar la reconfiguración dinámica de la arquitectura de aplicaciones cloud producida por la integración de incrementos de software. Esta infraestructura software, soporta la propagación del impacto arquitectónico de la integración desde artefactos de diseño hasta artefactos de implementación.

La Figura 5-1 ilustra los componentes involucrados en la infraestructura software y la relaciones entre ellos. La infraestructura software provee:

- *DSLs* que brindan soporte a las actividades del método DIARy; facilitando la documentación las decisiones tomadas en las actividades, estos incluyen:
    - Metamodelos (ver Figura 5-1(4-7)) que definen la sintaxis abstracta, conceptos y relaciones entre conceptos de los DSLs.
    - Editores gráficos que definen la sintaxis concreta de los DSLs.
- Motor de transformación que incluye:
    - Motor generador de modelos: Transformaciones M2M que propagan el impacto arquitectónico, descrito durante la actividad *Especificación de la integración del incremento*, hacia modelos que describen las decisiones de implementación y de despliegue en entornos cloud.
    - Motor generador de código: Transformaciones M2T que generan los artefactos de implementación, y scripts de despliegue y reconfiguración dinámica propuestos como salidas de las actividades del método DIARy. Los scripts son generados conforme a la tecnología del proveedor cloud seleccionado para el despliegue y el modelo de servicio seleccionado (p. ej., IaaS, PaaS). Este componente incrementa su funcionalidad, y capacidad de generar artefactos específicos para una plataforma de despliegue particular, a través de la incorporación de *Plug-ins* de *transformación* que generan artefactos de implementación y scripts específicos para un proveedor cloud.

La infraestructura software ha sido implementada como un conjunto de plug-ins de Eclipse Modelling Framework (Eclipse 2017ª). Eclipse define Ecore, una implementación del estándar Essential MOF (EMOF) (Object Management Group Inc., 2013). Esta solución emplea los niveles M1 y M2 previstos en la

arquitectura MOF. Por lo tanto, los metamodelos de los DSLs son implementados como modelos Ecore.

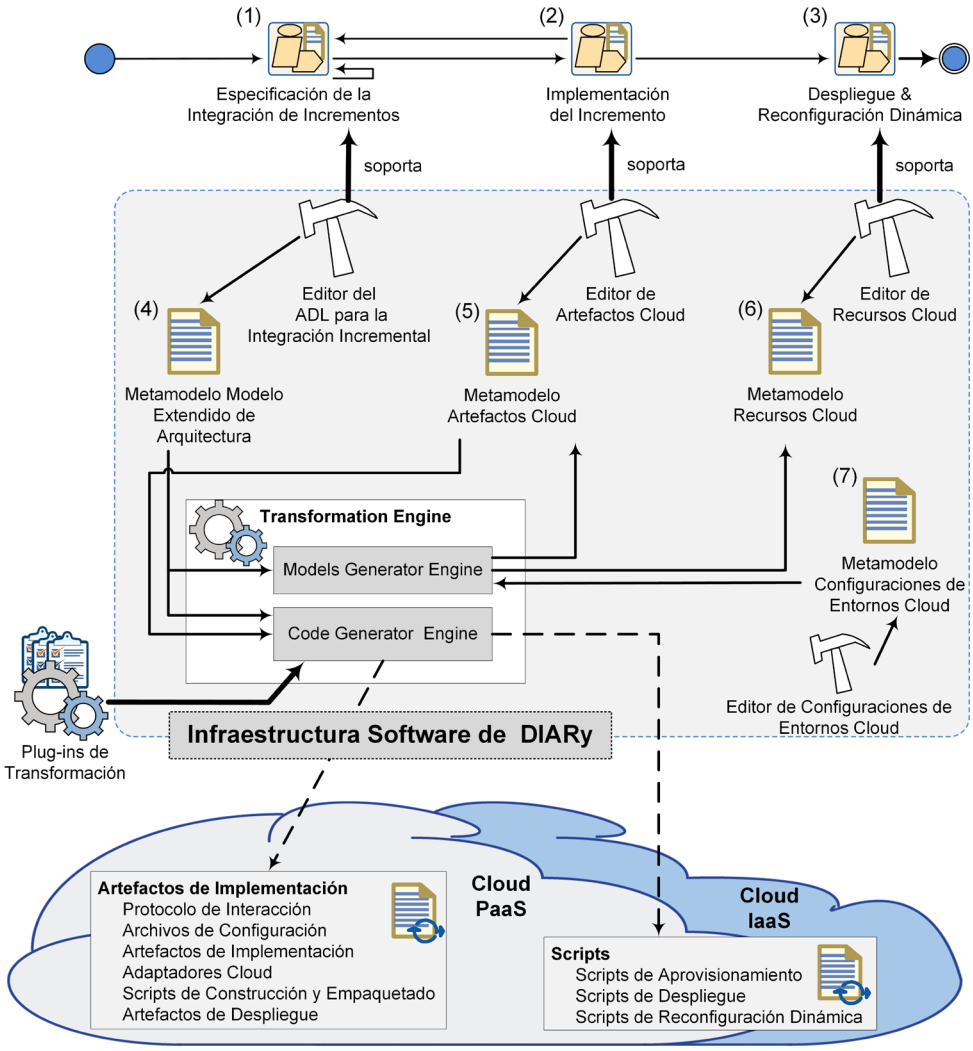



**Figura 5-1 Infraestructura software de DIARy**

Con respecto a la construcción de los editores gráficos de los DSLs, existen dos aproximaciones comunes para definir DSLs basados en UML. La primera aproximiación es un mecanismo ligero que consiste en definir Perfiles UML utilizando estereotipos y valores etiquetados. La segunda aproximación es utilizar MOF para extender el metamodelo de UML o para crear directamente un nuevo lenguaje de modelado sin ninguna dependencia en UML. La primera aproxima-

ción es soportada por herramientas UML que permiten la definición de estereotipos y valores etiquetados personalizados, permitiendo incluso cambiar la presentación de las notaciones utilizadas. Sin embargo, los Perfiles UML definen la sintaxis abstracta únicamente de manera implícita y algunos aspectos de la sintaxis no pueden ser convenientemente definidos (Brucker y Doser, 2007).

Inicialmente, utilizamos una tercera aproximación que combina las dos primeras a través de: i) crear un metamodelo (Ecore) que extiende el metamodelo de UML y define la sintaxis abstracta; y ii) crear un Perfil UML en el ambiente de modelado de Papyrus que implementa el perfil de la Figura 4-1 y define sintaxis concreta. Sin embargo, esta tercera aproximación presenta inconvenientes. En particular, no provee un mapeo bien definido entre los modelos UML con los que trabaja el arquitecto y las instancias del metamodelo del DSL que define el significado de los modelos, dificultando la obtención automática de instancias del metamodelo a partir de los modelos UML. Además, el uso de Perfiles UML no brinda la simplicidad requerida para la creación de modelos ya que primero se requiere crear elementos de modelado de UML (p. ej., clases, colaboraciones) para luego aplicar estereotipos y valores etiquetados en cada elemento, ralentizando la actividad de especificación. Por lo tanto, para la implementación de los DSLs de la infraestructura software utilizamos la segunda aproximación, diseñamos un metamodelo que define la sintaxis abstracta y utilizamos la herramienta Obeo Designer (Juliot y Benois, 2010), un plug-in de Eclipse Basado en GMF (Eclipse, 2011), para crear editores gráficos que definen la sintaxis concreta.

Con respecto a la validación de la consistencia de los modelos se utilizó OCL; para su implementación se analizaron dos alternativas. La primera de ellas consistía en llevar a cabo llamadas a los validadores de OCLTools (Eclipse, 2017b) en tiempo de ejecución, pasando por parámetro el código de las formulas OCL con las que validar las restricciones. La segunda alternativa consistía en definir las restricciones como invariantes de las propias relaciones entre vistas. Implementamos la segunda alternativa ya que aporta flexibilidad y mantenibilidad, permitiendonos desacoplar la definición de las restricciones, de manera que puedan ser modificadas directamente sobre el metamodelo, sin necesidad de modificar el código fuente de los editores gráficos.

## 5.2 Aproximación tecnológica de soporte a la actividad de *Especificación de la integración del incremento*

La infraestructura software propuesta en este trabajo de tesis brinda soporte a la actividad de *Especificación de la integración del incremento* (ver Figura 5-1(1)) mediante el DSL para el *ADL para la Integración Incremental* definido en la sección 4.1.2. El

metamodelo del DSL (el *Metamodelo Modelo Extendido de Arquitectura* (ver Figura 5-1(4))) emplea los conceptos identificados la sección 4.1.2.1.

### 5.2.1 *Metamodelo Modelo Extendido de Arquitectura*

Es recurrente que los diseñadores utilicen únicamente un pequeño sub conjunto de diagramas UML (France y otros, 2006) por lo que, con el propósito de facilitar la creación de modelos sin distraer la atención de los arquitectos con elementos de modelado que no se requieren durante la actividad de especificación, creamos un metamodelo que contiene únicamente los elementos UML que requieren ser extendidos para soportar la creación del *ADL para la integración incremental*. Para crear este metamodelo, primero copiamos la porción del metamodelo de UML requerida (elementos sin color de fondo en la Figura 5-2), luego los extendimos con los elementos de SoaML y finalmente los extendimos de acuerdo al diagrama de perfiles UML de la Figura 4-1.

La Figura 5-2 muestra el *Metamodelo Modelo Extendido de Arquitectura* el cual permite describir tanto la arquitectura de la aplicación, como la arquitectura del incremento de software a través de sus distintas capas: capa de servicios de aplicación, la capa de servicios de negocio y la capa de orquestación de servicios. Con respecto a los elementos que fueron tomados del metamodelo de UML, la Figura 5-2 muestra, por razones de simplicidad, únicamente aquellos que permiten describir la orquestación o interacción entre servicios.

### 5.2.2 Editor del ADL para la Integración Incremental

El editor del DSL desarrollado se implemento siguiendo un diseño basado en múltiples pestañas. Cada una de las pestañas, tal como se muestra en la Figura 5-3 sección (a), contiene un editor dedicado a describir un tipo de elemento arquitectónico: Arquitectura Extendida, Contratos de Servicio, Participantes, Enlaces entre Roles, Interacciones, Interfaces, etc. El editor nos permite importar el *Modelo de Arquitectura del Incremento* (entrada del método DIARy) para la cual se requiere su descripción en SoaML en formato XMI. Adicionalmente el editor permite crear la arquitectura del incremento desde cero.

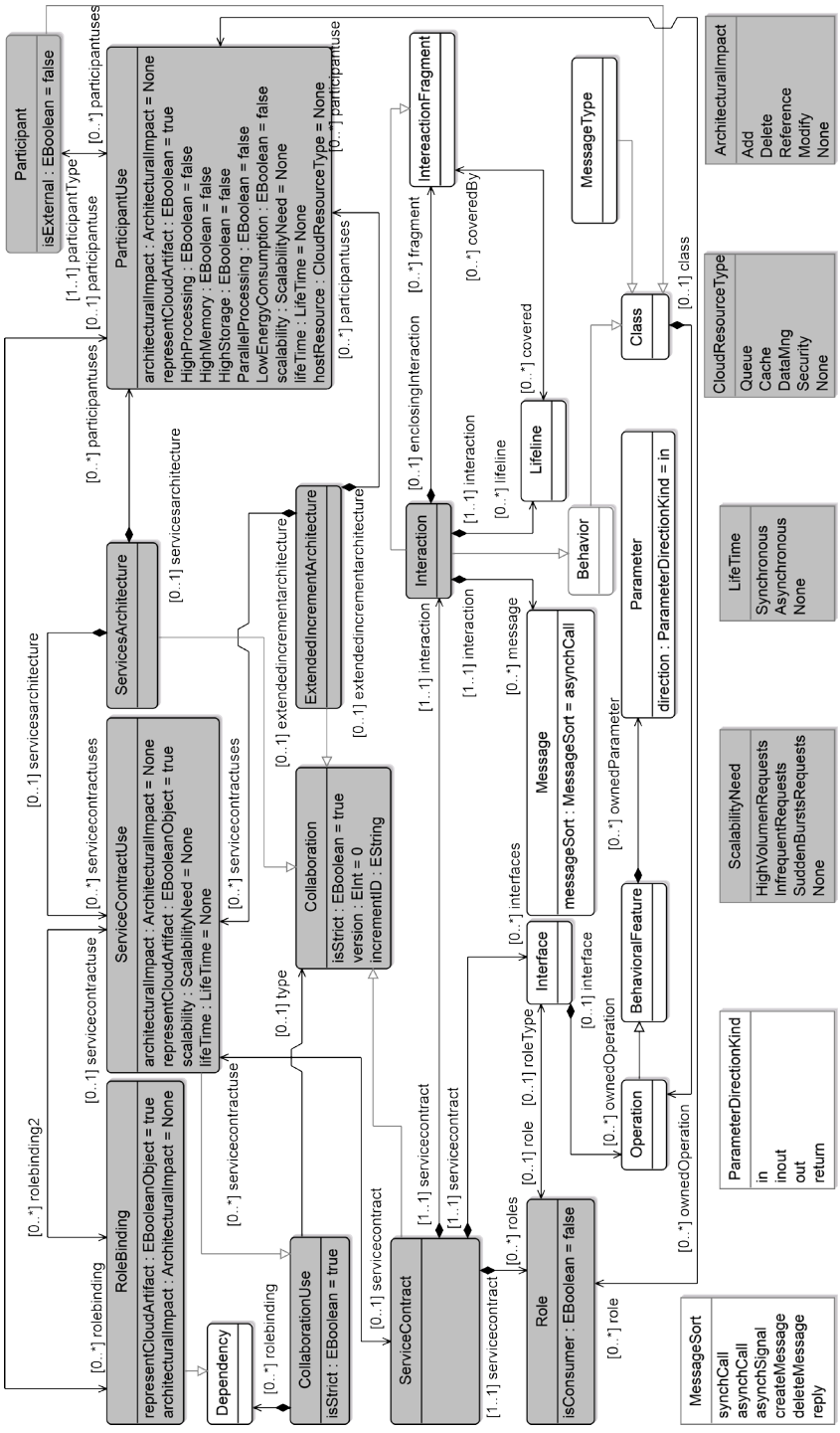



**Figura 5-2 Metamodelo *Modelo Extendido de la Arquitectura del Incremento***

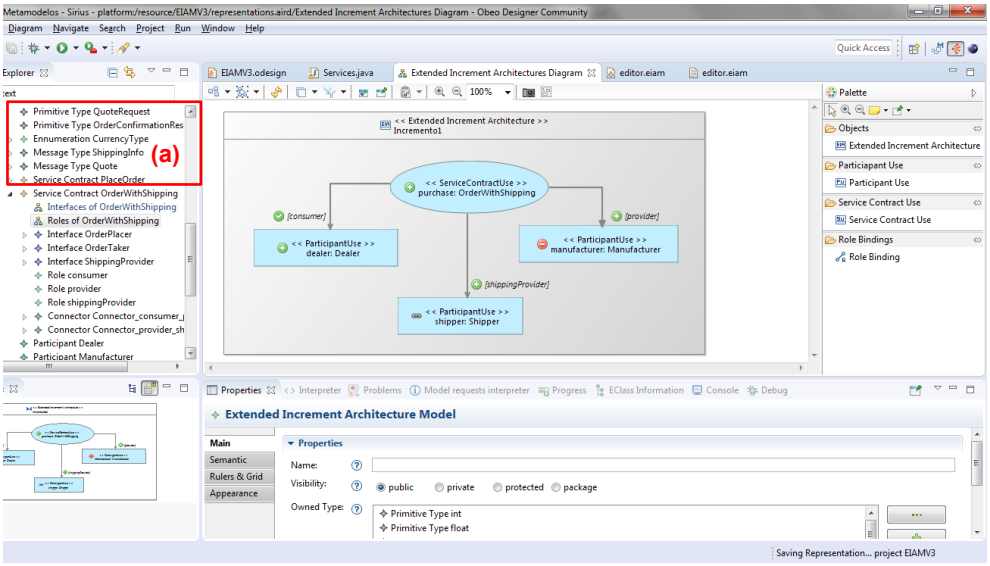



**Figura 5-3 Editor del ADL de integración incremental**

El editor permite al arquitecto especificar la integración del incremento a través de extender el *Modelo de la Arquitectura del Incremento* con el impacto arquitectónico, los requisitos de recursos cloud y refactorizar la arquitectura. Como se puede apreciar en la Figura 5-3 los elementos arquitectónicos son etiquetados con diferentes iconos dependiendo el impacto arquitectónico especificado. Esto ayuda al arquitecto en su tarea de especificación, evitando que deba navegar entre los atributos de los elementos arquitectónicos para conocer el impacto arquitectónico que tendrá la integración del elemento en la arquitectura actual de la aplicación. Por ejemplo, los elementos arquitectónicos con el icono "+" tienen impacto arquitectónico *Add*, con el icono "-" tienen impacto arquitectónico *Delete*, con el icono "√" tienen impacto arquitectónico *Modify*, y con el icono "∞" tienen impacto arquitectónico *Reference*. Con respecto a la especificación de la lógica de integración y la refactorización, éstas son soportadas por el editor a través de la creación, modificación o eliminación de elementos arquitectónicos en las diferentes vistas.

## 5.3 Aproximación tecnológica de soporte para la actividad *Implementación del incremento*

La tarea *Especificación de la estructura de empaquetado y despliegue* en la actividad *Implementación del incremento* (ver Figura 5-1(2)), es soportada por el componente de la infraestructura software *DSL de Artefactos Cloud*, el cual está compuesto por el

*Metamodelo Artefactos Cloud* (ver Figura 5-1(5)) y el *Editor de Artefactos Cloud*, ambos implementados en base a los conceptos identificados la sección 4.3.2.1. El *Modelo de Artefactos Cloud del Incremento* es construido conforme al *Metamodelo Artefactos Cloud*, y editado con el *Editor de Artefactos Cloud*.

Por otro lado, las tareas de *Generación del código de integración* y *Generación del código de implementación* son soportadas por transformaciones M2T que son parte del componente de la infraestructura software *Motor de Transformación* y son explicadas en la sección 5.5. A continuación, se explican las partes que constituyen el *DSL de artefactos cloud.*

### 5.3.1 *Metamodelo Artefactos Cloud*

El *Metamodelo Artefactos Cloud* (Figura 5-4) facilita la descripción y organización de los artefactos cloud que implementan los elementos arquitectónicos del *Modelo Extendido de la Arquitectura del Incremento*. La metaclase *DIARyArchitecturalElement* permite establecer la correspondencia entre elementos arquitectónicos del *Modelo Extendido de la Arquitectura del Incremento* con los artefactos cloud que los implementan, descritos en el *Modelo Artefactos Cloud del Incremento*. Como se puede observarse en la Figura 5-4, el *Metamodelo de Artefactos Cloud* agrupa los artefactos de software en tres tipos de proyectos, los cuales pueden ser empaquetados, construidos y desplegados independientemente en diferentes proveedores cloud de acuerdo a decisiones tomadas durante el proceso de desarrollo:

- *Interaction Project*: Incluye los artefactos de software necesarios para implementar los protocolos de interacción (orquestación), descritos como *ServiceContract* en el *Metamodelo Modelo Extendido de Arquitectura*. La implementación y despliegue de este proyecto permitirá ofrecer la orquestación de servicios como otro servicio. Entre estos artefactos están las interfaces y diagrama de secuencia (metaclase *Interaction Service*) que forman parte de un *ServiceContract*.
- *Implementation Project*: Incluye los artefactos de software necesarios para implementar el diseño de los servicios descritos como *Participant* en el *Metamodelo Modelo Extendido de Arquitectura*.
- *Deployment Project*: Incluye proyectos de implementación o interacción que serán desplegados en un mismo nodo (p. ej., máquina virtual) de un entorno cloud. Todos los servicios incluidos en los proyectos de implementación o interacción compartirán los recursos del nodo.

Algunos de los elementos que caben destacar de este metamodelo son:

- *architecturalImpact*: Atributo que tiene la misma funcionalidad que en el metamodelo *Metamodelo Modelo Extendido de Arquitectura*, indica la acción a realizar con el artefacto de despliegue. Valor propagado desde el atributo impacto arquitectónico de elementos arquitectónicos.
- *sourceCodeTechnology*: Atributo de artefactos *Deployment Project* que indica la tecnología en la cual se implementarán los artefactos incluidos en los proyectos de implementación o interacción asignados a un proyecto de despliegue.
- *hostResource*: Atributo de artefactos *BackEndService* que describe el tipo de recurso cloud al que permitirá acceder la implementación del artefacto.
- *EndPoint*: Metaclase que representa entradas con la localización o punto de acceso en la que se puede encontrar un servicio y/o participante. Es necesario conocerlo con tal de poder interactuar con el elemento en cuestión.
- *Dynamic Configuration*: Metaclase que representa artefactos que almacenan información de ajustes de servicios que cambiarán en tiempo de ejecución. Es donde los ajustes son definidos dependiendo del entorno (atributo *environment*) en que se desplegarán los artefactos de implementación; por ejemplo, entorno de ejecución, pruebas o producción.

### *5.3.2* Editor de Artefactos Cloud

El editor provee a los desarrolladores con una interface intuitiva que les permite describir los artefactos cloud que implementan los diferentes elementos arquitectónicos, y consolidar proyectos de interacción e implementación en proyectos de despliegue que en actividades posteriores serán empaquetados y desplegados conjuntamente en un mismo nodo. En la Figura 5-5 sección (a) se muestra la descripción de los artefactos cloud necesarios para implementar un proyecto de interacción, incluyendo artefactos *Dynamic Configuration* y *Settings* que cambiarán en tiempo de ejecución, como por ejemplo el punto de acceso (EndPoints o URIs) de los servicios ofrecidos por los participantes involucrados en la interacción.

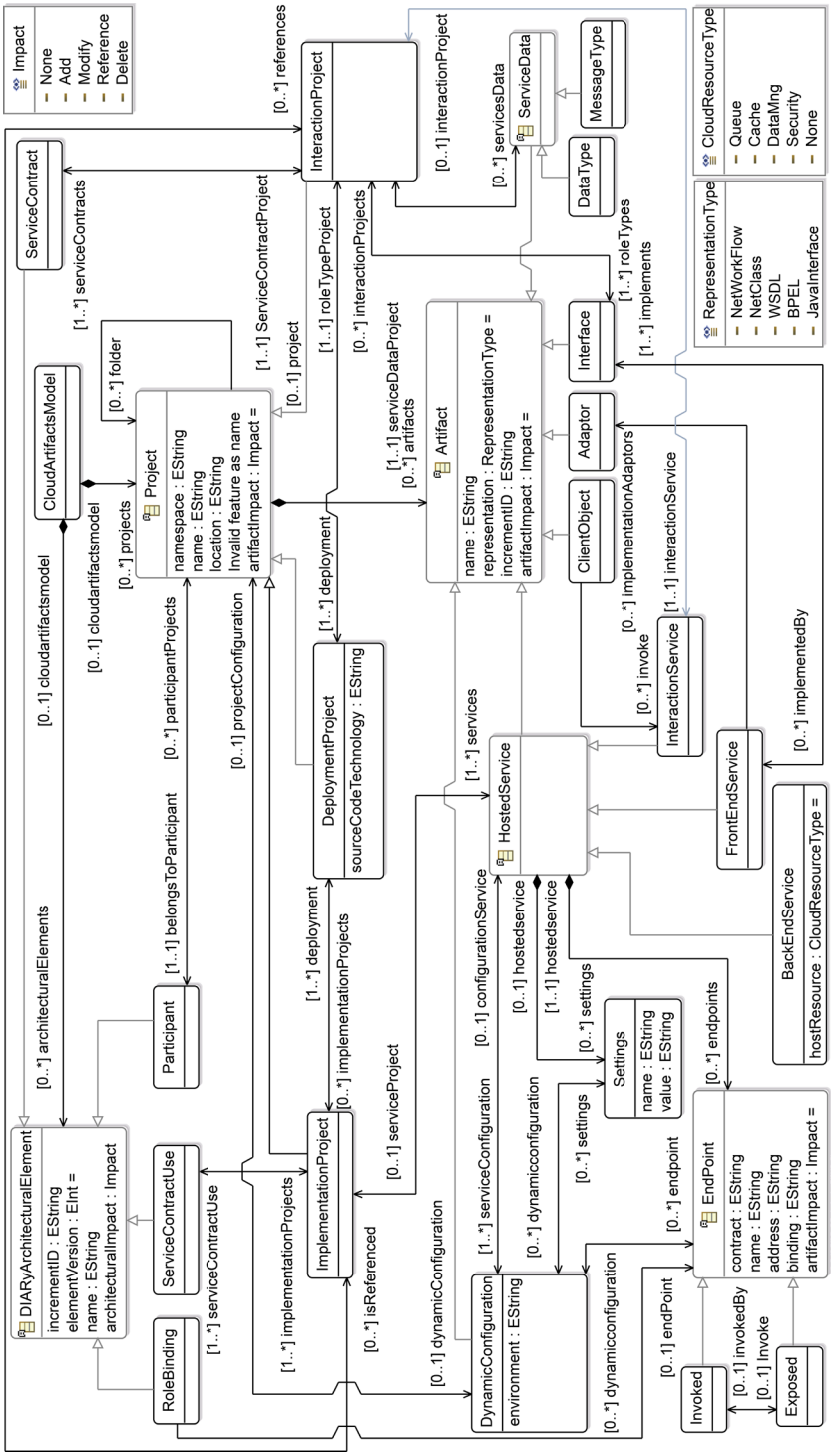



**Figura 5-4 *Metamodelo Artefactos Cloud***

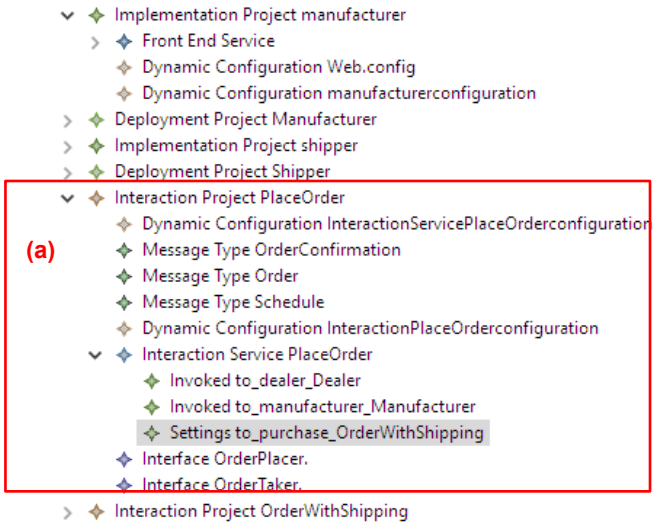



**Figura 5-5 Editor de Artefactos Cloud – Descripción de artefactos cloud**

La Figura 5-6 ilustra cómo consolidar diferentes proyectos de interacción (sección b) en un único proyecto de despliegue (sección a).

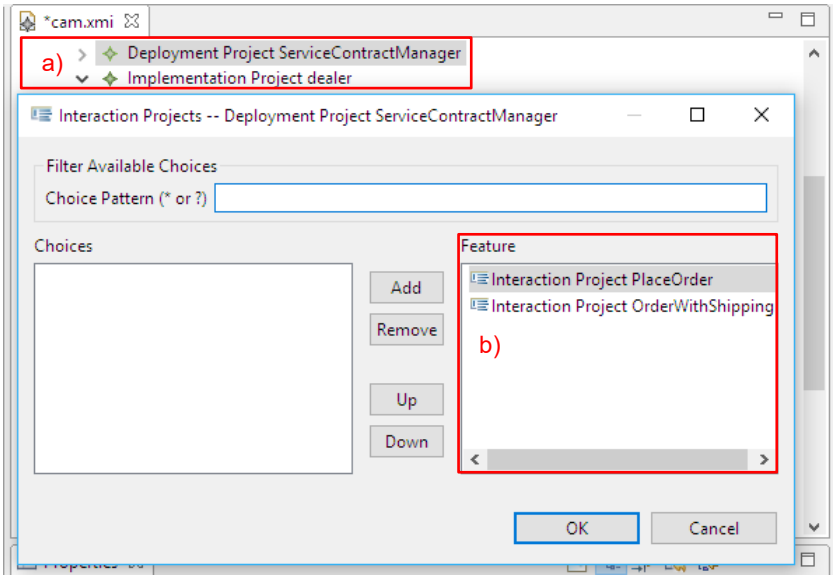



**Figura 5-6 Editor de Artefactos Cloud – Especificar consolidación de proyectos**

Adicionalmente el desarrollador de servicios cloud utiliza el *Editor de Artefactos Cloud* para, a través de dar valor a atributos, especificar la tecnología de implementación o representación de los diferentes artefactos cloud. Por ejemplo, la

Figura 5-7 muestra cómo especificar la representación que tendrá la implementación del protocolo de interacción.

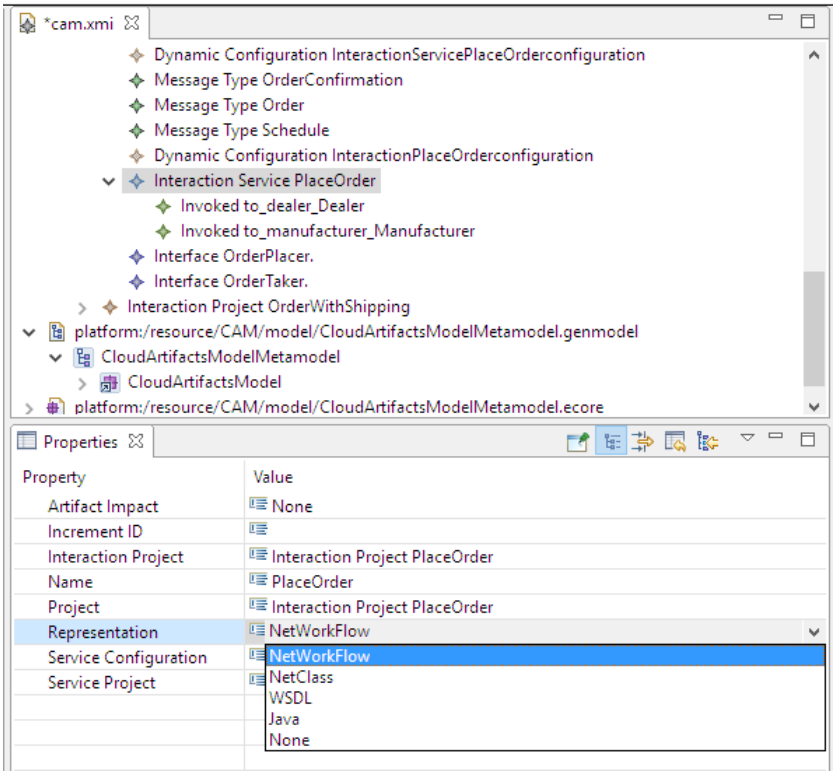



**Figura 5-7 *Editor de Artefactos Cloud* – Especificar representación de artefacto**

## 5.4 Aproximación tecnológica de soporte para la actividad *Despliegue y reconfiguración dinámica*

La tarea *Especificación de requisitos de aprovisionamiento* en la actividad *Despliegue y reconfiguración dinámica* (ver Figura 5-1(3)) es soportada por el componente de la infraestructura *Metamodelo Recursos Cloud* (ver Figura 5-1(6)) conforme al cual se crean *Modelos de Recursos Cloud del Incremento*.

Adicionalmente, la infraestructura software permite crear configuraciones o instancias de entornos cloud predefinidas. Estas instancias predefinidas son utilizadas como plantillas para crear *Modelos de Recursos Cloud del Incremento*. Es decir, los especialistas en operaciones pueden crear *Modelos de Recursos Cloud del Incremento* especificando uno a uno sus componentes, o a partir de una instancia de entorno cloud predefinida, la cual es copiada ejecutando transformaciones M2M (Sección

5.5.1.2). Las instancias predefinidas son creadas con el *DSL de Configuraciones de Entornos Cloud*, el cual está compuesto por el *Metamodelo Configuraciones de Entornos Cloud* (ver Figura 5-1(7)) y el *Editor de Configuraciones de Entornos Cloud*.

Por otro lado, las tareas de *Generación del código de integración* y *Generación del código de implementación* son soportadas por transformaciones M2T que son parte del componente de la infraestructura software *Motor de Transformación* y son descritas en la sección 5.5.

### 5.4.1 *Metamodelo Recursos Cloud*

Con el propósito de hacer frente al problema a la heteregeonidad en la descripción de recursos cloud, y facilitar la portabilidad, este metamodelo permite especificar de manera independiente de la tecnología y oferta de proveedores los recursos a ser aprovisionados con el propósito de desplegar los servicios. Este metamodelo ha sido definido en base a la comparación semántica y la unificación de los conceptos utilizados para referirse a recursos cloud por diferentes fuentes; las fuentes analizadas fueron: i) esfuerzos de estandarización tales como: Topology and Orchestration Specification for Cloud Applications (TOSCA) (Binz y otros, 2014), Open Cloud Computing Interface (OCCI) (Metsch y otros, 2010) y Distributed Management Task Force (DMTF) (Distributed Management Task Force Inc., 2010, 2009); ii) ofertas de proveedores cloud ya sea de modelos de servcio IaaS o PaaS tales como Microsoft Azure© y Google Cloud Platform y iii) soluciones o frameworks que facilitan la automatización del aprovisionamiento o despliegue de entornos cloud tales como Chef y Cloudify.

Los recursos cloud que forman parte de *Cloud Environments* descritos en el *Modelo de Recursos Cloud del Incremento* puden ser aprovisionados en diferentes proveedores cloud; así mismo, proveedores cloud pueden ser referenciados en diferentes *Modelo de Recursos Cloud del Incremento*. Por lo tanto, con el objetivo de facilitar la reutilización de la definición de proveedores cloud, la cual incluye información de subscripción necesaria para el acceso a la plataforma cloud y la generación de scripts de aprovisionamiento y despliegue, el *Modelo de Recursos Cloud del Incremento* describe la información de proveedores en *Modelos de Proveedores Cloud* relacionados. Estos modelos son construidos conforme al *Metamodelo de Proveedores Cloud* que se muestra en la Figura 5-8; en donde, la relación con *Modelos de Recursos Cloud del Incremento* construidos conforme al *Metamodelo de Recursos Cloud* (ver Figura 5-9) se establece a través de la relación *subscription* de tipo *Subscription* definida en la metaclase *Cloud Environment* de este último metamodelo. La metaclase *Parameter* de la Figura 5-8 permite describir los parámetros necesarios para tener acceso a la subscripción de un proveedor específico; en

donde la cantidad, nombre y naturaleza del parámetro son específicos de cada proveedor cloud, estos pueden ser: identificador de la subscripción, clave, certificado digital u otros. La metaclase *PlatformImage* permite documentar imágenes de recursos de plataforma preinstalados que podrán ser utilizadas para agilizar el proceso de aprovisionamiento.

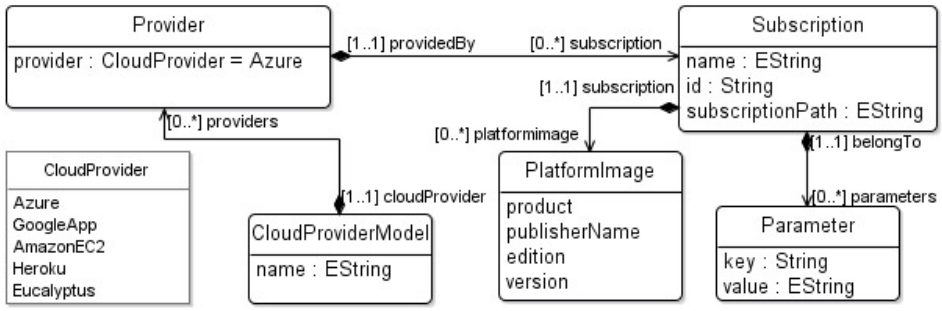



**Figura 5-8** ***Metamodelo de Proveedores Cloud***

El *Metamodelo Recursos Cloud* de la Figura 5-9 muestra que los constructures de este metamodelo permiten describir como un paquete a ser desplegado (*DeploymentArtifact*), cuya ubicación se determina por el atributo *packagesFile*, puede ser hospedado en varios entornos (*CloudEnvironment*) provistos por diferentes proveedores cloud especificados a traves de la reción *subscription* (no visible en la Figura 5-9). Cada *Deployment Artifact*, pude incluir uno o mas servicios (*Service*) que harán uso y compartirán los recursos cloud (*Resource*) aprovisionados en un *CloudEnvironment*. Los recursos pueden ser, ya sea de infraestructura (*Infraestructure*) o de plataforma (*Platform*). Adicionalmente se pueden especificar las relaciones entre los diferentes elementos *Resource* a provisionar; por ejemplo, un sistema operativo (*PaaS_SO)* se hospeda en una unidad de computo (*IaaS_Computing*), o un recurso runtime *PaaS_Runtime* (p. ej., java, php) se instala en un *PaaS_SO*, o un gestor de base de datos (*PaaS_DBManager* ) se instala en un *PaaS_S0*.

Algunas de los elementos que caben destacar de este metamodelo son:

- *architecturalImpact*: Atributo de la metaclase *Deployment Artifact* que tiene la misma funcionalidad que en el metamodelo *Modelo de Artefactos Cloud*, indica la acción a realizar con el artefacto de despliegue (p.ej., desplegar, replegar, etc.). Valor propagado desde el atributo impacto arquitectónico de proyectos de despliegue en *Modelos de Artefactos Cloud*.

- *environment*: Atributo de la metaclase *Cloud Environment* que indica el tipo de entorno en el cual se desplegará el *Deployment Artifact*, por ejemplo: entorno de desarrollo, pruebas, producción, etc.

- *InvokedEndPoint*: Metaclase que representa entradas con información relacionad a los puntos de acceso a los servicios invocados por cada servicio incluido en un *Deployment Artifact*. Esta información será empleada por el componente *Transformation Engine* durante la generación de scripts de reconfiguración dinámica para modificar las entradas, que contienen información de dependencias entre servicios, de los archivos de configuración de los servicios desplegados.

- *ExposedEndPoint:* Metaclase que representa puntos de acceso a los servicios incluidos en un *Deployment Artifact*. El punto de acceso que ofrece cada servicio estará constituido por el punto de acceso al *Cloud Environment* en el cual es desplegado, complementado con el contrato (atributo *contract* de la metaclase *Service*) ofrecido por el servicio.

- *Setting*: Metaclase que representa ajustes a servicios (*Service*) o recursos cloud (*Resource*) que cambian en tiempo de ejecución. Esta información será empleada por el componente *Transformation Engine* durante la generación de scripts de reconfiguración dinámica para modificar las entradas de los archivos de configuración de los recursos cloud aprovisionados o servicios desplegados.

- *Resouce Parameter*: Metaclase que representa parámetros necesarios para el aprovisionamiento o configuración de recursos cloud en un proveedor específico; en donde, dependiendo de cada proveedor cloud, diferentes parámetros son requeridos para los recursos cloud. Por ejemplo, versión del sistema operativo, distribución del sistema operativo, etc.

- *PlatformImage*: Representan imágenes predefinidas de recursos cloud. Utilizadas durante el aprovisionamiento para acelerar el proceso. Por ejemplo: la imagen del sistema operativo a hospedar en el recurso IaaS_Computing.

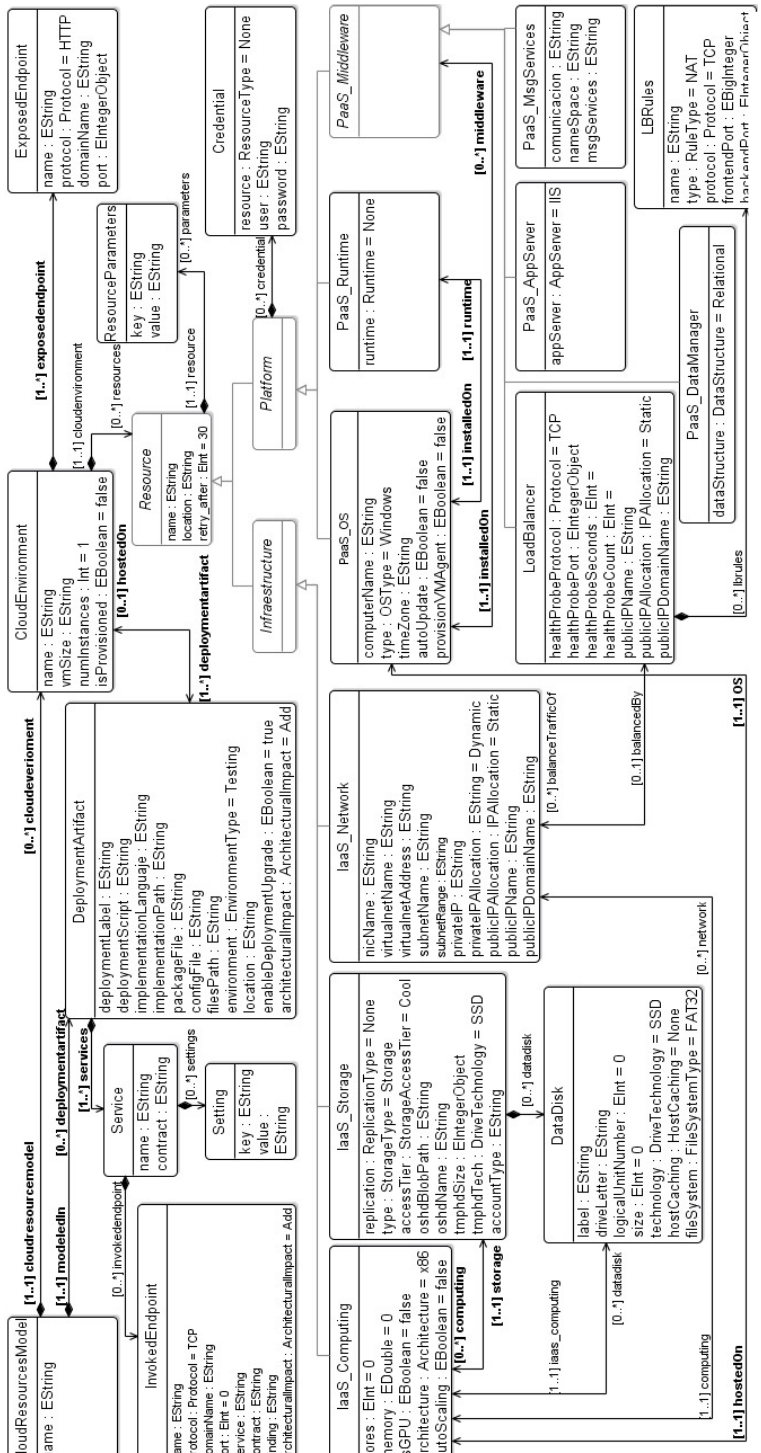



**Figura 5-9 *Metamodelo de Recursos Cloud***

### 5.4.2 *Metamodelo Configuraciones de Entornos Cloud*

Uno de los principales problemas de la construcción de aplicaciones a ser desplegadas en entornos cloud es la variabilidad, no solo en las características de los recursos cloud de infraestructura y plataforma ofertados por proveedores cloud, sino en la manera en la que estos recursos cloud son combinados por los proveedores y ofrecidos como instancias de entornos cloud predefinidas.

El M*odelo de Configuraciones de Entornos Cloud* permite expresar la variabilidad en la oferta de recursos cloud. Este modelo es construido conforme al M*etamodelo Configuraciones de Entornos* Cloud, el cual se basa en modelos de características; en donde, las características son tipos de recursos, y atributos de las características son particularidades de los recursos. Adicionalmente, las diferentes maneras de combinar los recursos cloud para ofrecerlos como instancias predefinidas es afrontada a través de crear configuraciones del M*odelo de Configuraciones de Entornos Cloud*. Una de las principales notaciones es el modelo de características propuesto por Kang y otros (1900). Sin embargo con el objetivo de permitir un modelo de características con cardinalidades se ha utilizado el metamodelo para expresar la variabilidad externa propuesto por Gómez (2012). No obstante, este modelo no incluye constructores que soporten la especificación de configuraciones; por lo tanto, lo extendemos incluyendo las metaclases y relaciones requeridas, dibujadas con fondo gris en la Figura 5-10.

La Figura 5-10 muestra un extracto del metamodelo definido para soportar el modelo de características con cardinalidades. La clase contenedora principal es el modelo de características (*FeatureModel*). Este modelo de características está compuesto de características (*Features*) que se pueden estructurar en grupos (*Groups*). A diferencia de otras notaciones, como la propuesta por Kang y otros (1990), en esta notación se diferencia entre la notación del grupo y la notación de las características hijas. La cardinalidad de grupo restringe cuantas características se pueden instanciar sin importar el número de instancias de la característica. Con respecto las configuraciones (*Configuration*), cada configuración incluye únicamente las características seleccionadas (*SelectedFeature*) y los atributos a los cuales el usuario asigno un valor durante la especificación de la configuración (*AttributeValue*).

Además, pueden establecerse relaciones entre características (*Relationships*). Se distinguen los siguientes tipos de relaciones:

- *Implicación (A -> B):* Si una instancia de una característica A existen al menos debe de existir una instancia de la característica B.

- *Bi*-condicional (A <-> B): Si una instancia de una característica A existen al menos debe de existir una instancia de la característica B y viceversa.

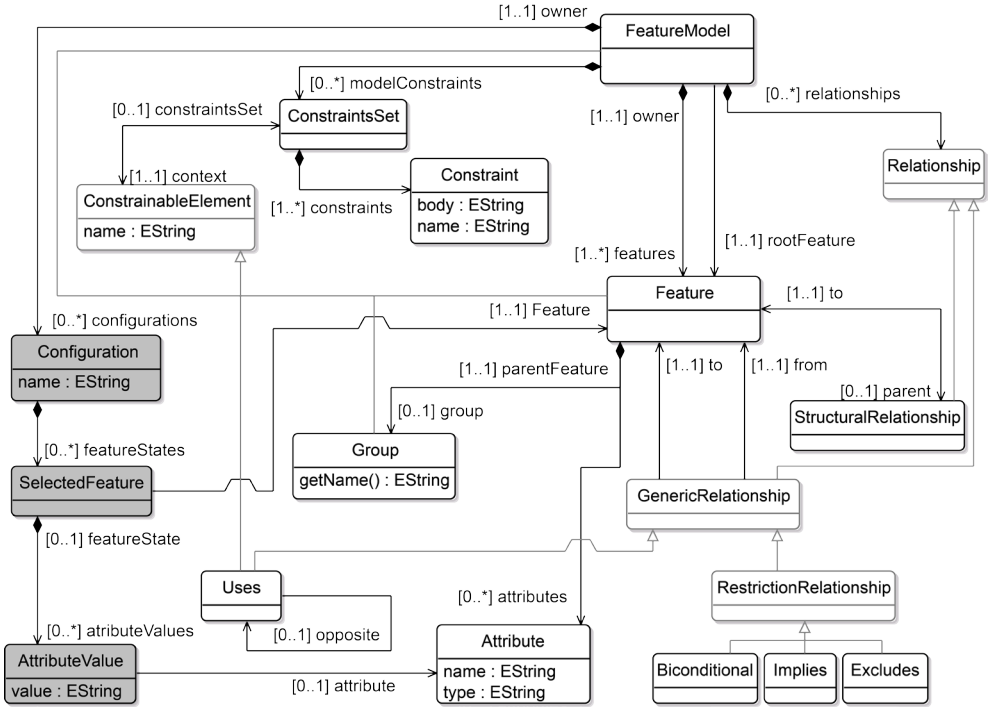



**Figura 5-10 Metamodelo *Configuraciones de Entornos Cloud***

- Exclusión (A >-< B): Si una instancia de una característica A existe, no puede existir ninguna instancia de la característica B y viceversa.
- Uso (A-->B): Esta relación se define a nivel de configuración e indica que una instancia específica de una característica A se relaciona con una o más instancias específicas de B definidas por su límite superior (n).
- Por último, pueden definirse restricciones (Constraints) sobre el modelo. Estas restricciones se definen mediante el lenguaje Feature Modeling Constraint Language (FMCL) (Gómez, 2012), con una sintaxis y semántica similar a OCL.
- Por otra parte, las relaciones de obligatoriedad se indican con un punto negro en el extremo de la relación.
- Otro tipo de relación entre características es la de implicación, que indica que la primera solo podrá estar seleccionada en una configuración si la característica desde la que parte la implicación es seleccionada.

- Otro tipo de elementos que se utilizan en el modelo de características son los grupos. Los grupos de alternativas, indicados con un triángulo blanco en la notación, permiten seleccionar una y sólo una característica de las características hijas.
- Otro tipo de grupos son los grupos de selección, indicado en la notación gráfica con un triángulo negro; en donde, se debe de seleccionar como mínimo una de las características hijas y como máximo el valor especificado por la cardinalidad.

### 5.4.3 Editor de Configuraciones de Entornos Cloud

El editor provee a los especialistas en operaciones con una interface intuitiva que les permite describir configuraciones predefinidas de entornos cloud, las cuales son descritas en M*odelos de Configuraciones de Entornos Cloud.*

La Figura 5-11 muestra la interface principal del *Editor de Configuraciones de Entornos Cloud*, misma que, al igual que el *Metamodelo Configuraciones de Entornos Cloud* ha sido inspirada en el trabajo de Gómez y Ramos (2010), por lo tanto se ven similares. Sin embargo, el *Editor de Configuraciones de Entornos Cloud* propuesto en este trabajo de tesis permite definir configuraciones sobre el modelo de características especificado.

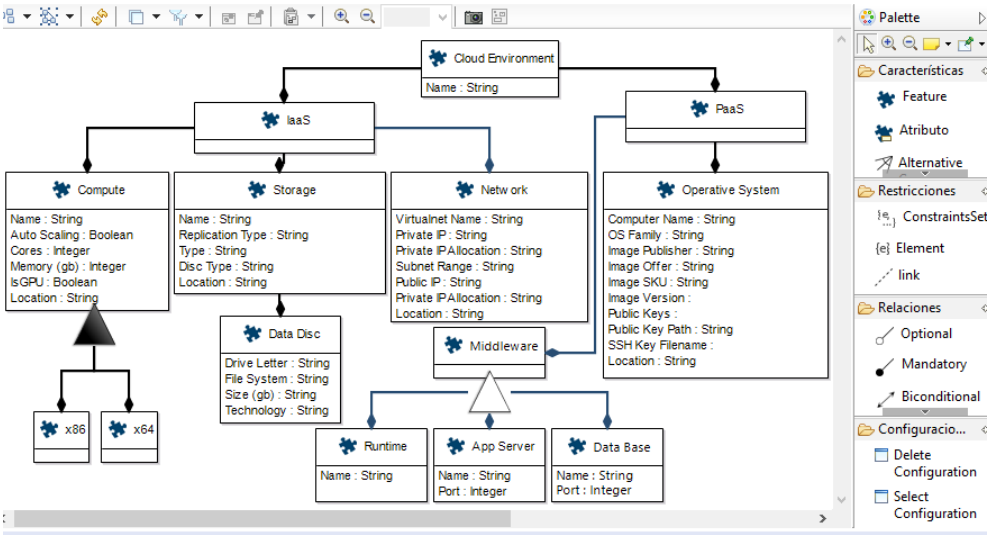



**Figura 5-11 Editor de Configuraciones de Entornos Cloud**

El especialista en operaciones utiliza este editor para crear modelos de características que describen los posibles recursos cloud ofrecidos por los proveedores

cloud; en donde, los recursos cloud son características del modelo, las particularidades de los recursos cloud son atributos de las características, y las relaciones entre recursos cloud son relaciones entre características. Una vez que el especialista en operaciones ha definido el modelo de características, procede a definir configuraciones predefinidas de entornos cloud. Para definir las configuraciones, el especialista en operaciones crea una vista del modelo de características, selecciona las características (recursos cloud) que desea incluir en la configuración, y así valor a los atributos según las características de la configuración predefinida.

### 5.4.4 Editor de Recursos Cloud

El *Editor de Recursos Cloud* (ver Figura 5-12) provee a los especialistas en operaciones con una interface intuitiva que les permite describir los entornos cloud que se requieren aprovisionar para desplegar un artefacto de despliegue; soportando la tarea *Especificación de requisitos de aprovisionamiento* en la actividad *Despliegue y reconfiguración dinámica* (ver Figura 5-1(3)).

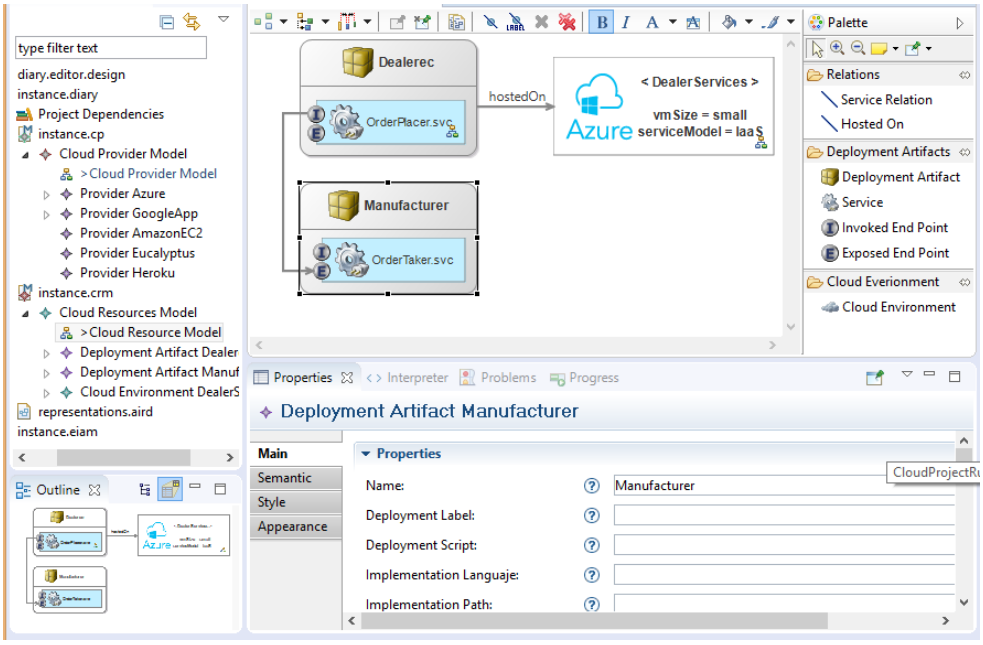



**Figura 5-12 Editor de Recursos Cloud**

El especialista podrá seleccionar el proveedor cloud en el cual se creará el entorno cloud, el modelo de servicio (p. ej., IaaS o PaaS) a emplear del proveedor, y los recursos cloud (y sus características) que se requieren aprovisionar en el proveedor cloud a fin de satisfacer los requisitos de los servicios incluidos en un

artefacto de despliegue. El *Editor de Recursos Cloud* permite editar *Modelos de Recursos Cloud*; en donde, por cada artefacto de despliegue (p. ej., Manufacturer) el especialista en operaciones describe los servicios cloud incluidos en éste (p. ej., Order Taker.svc), los puntos de acceso a los servicios (ícono con letra "E") y los puntos de acceso a los servicios invocados (ícono con letra "I") por un servicio. Adicionalmente por cada artefacto de despliegue permite especificar el (o los) entornos cloud en los que se desplegará (hospedará) el servicio.

## 5.5 Motor de transformación

Este componente de la infraestructura software de DIARy implementa transformaciones M2M y M2T que propagan el impacto arquitectónico descrito en el *Modelo Extendido de la Arquitectura del Incremento* hacia los modelos y código que soportan las actividades *Implementación del incremento* y *Despliegue y reconfiguración dinámica* (ver Figura 5-1(2 y 3)). Las transformaciones son implementadas como plug-ins en los sub-componentes del Motor de Transformación (*Transformation Engine*), el Motor de Generación de Modelos (*Models Generator Engine*) y el Motor de Generación de Código (*Code Generator Engine*) respectivamente.

Los plug-ins de los sub-componentes del Motor de Transformación son invocados a través de seleccionar acciones de un menú contextual incluido en el entorno de desarrollo de Eclipse, ver Figura 5-13.

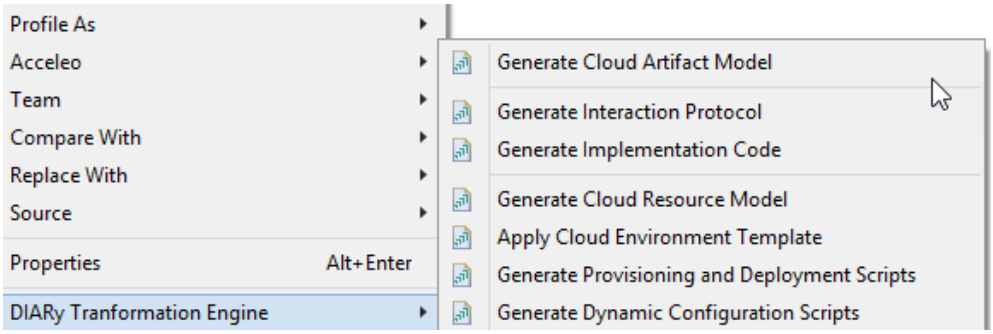



**Figura 5-13 Menú contextual para la invocación de transformaciones**

### 5.5.1 Motor de generación de modelos

Este motor de transformación está compuesto por plug-ins de transformaciones M2M implementados en Atlas Transformation Language (ATL). Las transformaciones generan parcialmente modelos con descripciones de los artefactos cloud y recursos cloud necesarios para implementar y desplegar los servicios que conforman un incremento. Posteriormente, los desarrolladores completan estos

modelos con información referente a las tecnologías en las cuales se implementarán y aprovisionarán los artefactos y recursos cloud respectivamente (p. ej., lenguaje de programación, API de interacción con recursos cloud). Las trasformaciones M2M incluidas en la infraestructura software son:

#### 5.5.1.1 *Generación de Modelos de Artefactos Cloud*

Este plug-in se invoca mediante la acción *Generate Cloud Artifacts Model* del menú contextual de la Figura 5-13. Reglas de transformaciones M2M toman como entrada el *Modelo Extendido de la Arquitectura del Incremento* y generan parcialmente el *Modelo de Artefactos Cloud de Incremento*. Este modelo describe los artefactos de software necesarios para implementar la arquitectura del incremento, implementar la integración (orquestación entre los servicios del incremento con los servicios de la aplicación actual), y reconfigurar dinámicamente la arquitectura actual de la aplicación. En este apartado se definen las correspondencias y reglas de transformación que toman como entrada modelos creados conforme al *Metamodelo Modelo Extendido de Arquitectura* (ver Figura 5-2) y generan modelos conforme al *Metamodelo Artefactos Cloud* (ver Figura 5-4). Las principales correspondencias empleadas en las transformaciones, y de acuerdo a Figura 4-6, son las siguientes:

Por cada elemento *Participant* origen generamos un *DeploymentProject* con sus configuraciones (*DynamicConfiguration*), ver líneas 3, 5, 9, 15 en Listado 5-1.

**Listado 5-1 Extracto de la regla de transformación de *Participant***

```
1  rule Participant {
2    From
3      ParticipantInput : EIAM!Participant
4    to
5      Participant : CAM!Participant (
6        name <- ParticipantInput.name,
7        cloudartifactsmodel <- ParticipantInput.extendedincrementarchitecturemodel
8      ),
9      DeploymentProject : CAM!DeploymentProject (
10       name <- ParticipantInput.name,
11       cloudartifactsmodel <- ParticipantInput.extendedincrementarchitecturemodel,
12       belongsToParticipant <- Participant,
13       artifacts <- Set{ServiceConfiguration}
14     ),
15     ServiceConfiguration : CAM!DynamicConfiguration (
16       name <- 'ServiceConfiguration.cloud.cscfg',
17       projectConfiguration <- DeploymentProject,
18       project <- DeploymentProject
19     )
20 }
```

Por cada elemento *ParticipantUse* (instancia de un Participant dependiendo del contexto) generamos un *ImplementationProject* (una implementación del participante en un contexto concreto) el cual está enlazado con el *DeploymentProject* del Participant al que pertenece, ver líneas 3, 6, 13 en Listado 5-2. Así mismo también se generan la configuración específica para el *ImplementationProject* y los *EndPoint* (a partir de los *RoleBinding* relacionados con el elemento *ParticipantUse*) necesarios para poder interaccionar con este. En caso que la interface que implementa el participante tenga el atributo *isConsumer* = true, también se generará un *ClientObject* que describe el artefacto que implementa el código para invocar al servicio de orquestación, e iniciar la interacción.

**Listado 5-2 Extracto de la regla de transformación de ParticipantUse**

```
1  rule ParticipantUse {
2    From
3      ParticipantUseInput : EIAM!ParticipantUse
4        (ParticipantUseInput.scalability <> #HighVolumenRequests)
5    to
6      ImplementationProject : CAM!ImplementationProject (
7        name <- ParticipantUseInput.name,
8        architecturalImpact <- ParticipantUseInput.architecturalImpact,
9        cloudartifactsmodel <- ParticipantUseInput.extendedincrementarchitecture.
10         extendedincrementarchitecturemodel,
11       belongsToParticipant <-
12         thisModule.resolveTemp(ParticipantUseInput.participantType, 'Participant'),
13       deployment <-
14         thisModule.resolveTemp(ParticipantUseInput.participantType,
15       artifacts <- Set{ImplementationProjectConfiguration, FrontEndService,
16       services <- FrontEndService
17     ),
18     ImplementationProjectConfiguration : CAM!DynamicConfiguration (
19       name <- 'ServiceConfiguration.cloud.cscfg',
20       projectConfiguration <- ImplementationProject
21     ),
22     FrontEndService : CAM!FrontEndService (
23       project <- ImplementationProject
24     ),
25     FrontEndConfiguration : CAM!DynamicConfiguration (
26       name <- 'Web.config',
27       configurationService <- FrontEndService
28     )
29 }
```

Con respecto al soporte a la consolidación de los servicios que tienen similitudes en su perfil de requisitos de aprovisionamiento (sección 4.3.3.1), la línea 4 del

Listado 5-2 permite filtrar *Participants* con requisitos de escalabilidad que no requieren altos volúmenes de petición de servicio (*.scalability <> #HighVolumenRequests*) y consolidar, en un proyecto de despliegue común (Líneas 13 y 14), los artefactos cloud correspondientes a los servicios (p. ej., *FrontEndServices*) ofrecidos por esos participantes. En el caso de que un participante tenga requisitos de escalabilidad que requieran altos volúmenes de petición de servicio se genera un *DeploymentProject* exclusivo para el participante. Adicionalmente, la línea 8 del Listado 5-2 muestra como el impacto arquitectónico descrito en el *Modelo Extendidos de la Arquitectura del Incremento* es propagado hacia el proyecto correspondiente en el *Modelos de Artefactos Cloud del Incremento*.

Por cada *ServiceContract* en el modelo origen se genera un *ServiceContract* (SC) en el modelo destino, un *InteractionProject*, y el artefacto de configuración correspondiente al SC (*DynamicConfiguration*). Al interior del *InteractionProject* se genera un *InteractionService* (a partir del diagrama de secuencia del SC indicando así el comportamiento de este), sus *EndPoints* a partir de los *RoleBindings* que posea, y la configuración del *InteractionService*. Además, en el *InteractionProject* se generan también los *MessageType* que utiliza el SC para comunicarse a partir de los *MessageType* originales. Ver líneas 3, 5, 9, 16 y 20 en Listado 5-3.

**Listado 5-3 Extracto de la regla de transformación de ServiceContract**

```
1  rule ServiceContract {
2    from
3      ServiceContractInput : EIAM!ServiceContract
4    to
5      ServiceContract : CAM!ServiceContract (
6        name <- ServiceContractInput.name,
7        cloudartifactsmodel <- ServiceContractInput.extendedincrementarchitecturemodel
8      ),
9      InteractionProject : CAM!InteractionProject (
10       name <- ServiceContractInput.name,
11       cloudartifactsmodel <- ServiceContractInput.extendedincrementarchitecturemodel,
12       artifacts <-
13         Set{InteractionProjectConfiguration, InteractionService,
14           InteractionServiceConfiguration}
15     ),
16     InteractionProjectConfiguration : CAM!DynamicConfiguration (
17       name <- 'ServiceConfiguration.cloud.cscfg',
18       projectConfiguration <- InteractionProject
19     ),
20     InteractionService : CAM!InteractionService (
21       name <- ServiceContract.name,
22       interactionProject <- InteractionProject
23     )
28 }
```

Por cada *ServiceContractUse* (instancias de SC, permitiendo reutilizar el SC sin tener que duplicarlo) se genera un *ServiceContractUse* que está relacionado con el SC al que apunta y se crea la relación entre el *ServiceContractUse* y los proyectos de implementación que implementan las interfaces correspondientes a los roles definidos en el SC relacionado. Ver líneas 3, 5, 10 y 11 en el Listado 5-4.

**Listado 5-4 Extracto de la regla de transformación de ServiceContractUse**

```
1  rule ServiceContractUse {
2    from
3      ServiceContractUseInput : EIAM!ServiceContractUse
4    to
5      ServiceContractUse : CAM!ServiceContractUse (
6        name <- ServiceContractUseInput.name,
7        cloudartifactsmodel <-
8          ServiceContractUseInput.extendedincrementarchitecture
9            .extendedincrementarchitecturemodel,
10       implementationProjects <-
11         ServiceContractUseInput.rolebinding->
12           iterate(Rolebinding; participantUses: Set(soaml!ParticipantUse) = Set {} |
13         Rolebinding.supplier->
14           iterate(participantUse; res2: Set(soaml!ParticipantUse) = Set{} |
15         participantUses->including(participantUse)
16         ))->collect(participantUse |
17         thisModule.resolveTemp(participantUse, 'ImplementationProject'))
18     )
19 }
```

Por cada *Interface* generamos una *Interface* dentro de cada *SC* que la define, indicando el *ImplementationProject* que lo implementa. Ver líneas 3 y 5 en Listado 5-5.

**Listado 5-5 Extracto de la regla de transformación de Interface**

```
1  rule Interface {
2    From
3      InterfaceInput : EIAM!Interface
4    to
5      Interface : CAM!Interface (
6        name <- InterfaceInput.name,
7        project <-
8          thisModule.resolveTemp(InterfaceInput.servicecontract, 'InteractionProject')
9      )
10 }
```

La Tabla 5-1 muestra las principales correspondencias que hemos nombrado anteriormente.

**Tabla 5-1 Correspondencia entre el Modelo Extendido de la Arquitectura del Incremento y el Modelo de Artefactos Cloud del Incremento**

| Origen | Destino |
|---|---|
| Participant | Participant<br>DeploymentProject<br>DynamicConfiguration |
| ParticipantUse | ImplementationProject<br>FrontEndService<br>DynamicConfiguration<br>ClientObject (En caso de participantes que tienen un rol de consumidor del servicio) |
| ServiceContract | ServiceContract<br>Deployment Project<br>InteractionProject<br>DynamicConfiguration<br>InteractionService |
| Interface | Interface |
| Message Type | Message Type |
| Data Type | Data Type |
| RoleBinding | Exposed EndPoint<br>Invoked EndPoint |
| ServiceContractUse | ServiceContractUse |

### *5.5.1.2* Generación *de Modelos de Recursos Cloud del Incremento*

Este plug-in es invocado mediante la acción *Generate Cloud Resources Model* del menú contextual de la Figura 5-13. Reglas de transformacion M2M toman como entrada el *Modelo de Artefactos Cloud de Incremento* y generan parcialmente el *Modelo de Recursos Cloud del Incremento*. Este modelo describe tanto los recursos cloud que se requieren aprovisionar para desplegar los servicios que conforman el incremento; así como información que facilita la automatización del despliegue y reconfiguración dinámica.

Las reglas de transformación generan elementos *Deployment Artifact* en el modelo destino por cada elemento *Deployment Project* en el modelo origen; por cada proyecto de despliegue existirá un artefacto de despliegue a ser desplegado. El impacto arquitectónico (*architecturalImpact*) del *Deployment Project* origen es propagado hacia el *Deployment Artifact* destino. Adicionalmente por cada *Hosted Service* en el modelo origen se genera un elemento *Service* en el modelo destino. Finalmente, por cada elemento *Setting* y elementos *EndPoint* relacionados a un *HostedService* en el modelo origen, se generan elementos *Setting* e *InvokedEndPoint* relacionados a elementos *Service* en el modelo destino.

#### 5.5.1.3 Aplicar plantilla de entorno cloud

Este plug-in es invocado mediante la acción *Apply Cloud Environment Template* del menú contextual de la Figura 5-13. En la tarea *Especificación de requisitos de aprovisionamiento* de la actividad *Despliegue y reconfiguración dinámica* (ver Figura 5-1(3)), el especialista en operaciones tiene la posibilidad de seleccionar una configuración de entorno cloud predefinida (o plantilla de entorno cloud) en lugar de especificar uno a uno los recursos cloud que conforman el *Cloud Environment* a ser aprovisionado.

Las configuraciones de entornos predifinidas se describen en *Modelos de Configuración de Entornos Cloud*; por lo tanto, las reglas de transformación de este plug-in refactorizan *Modelos de Recursos Cloud del Incremento* a partir de *Modelos de Configuración de Entornos Cloud*. Estas reglas de transformación crean elementos *Resource* en *Modelos de Recursos Cloud del Incremento* de acuerdo a elementos *Feature* descritos en *Modelos de Configuración de Entornos Cloud*; en donde, elementos *Feature* describen recursos cloud de un entorno cloud preefinido. Adicionalmente elementos *Atribute* relacionados a elementos *Feature* en el modelo origen son transformados en elementos *ResourceParameter* de elementos *Resource* en el modelo destino.

### 5.5.2 Motor de generación de código

Este sub-componente está constituido por un conjunto de plug-ins de transformación M2T que generan código de implementación, código de integración, scripts de aprovisionamiento y despliegue, y scripts de reconfiguración dinámica de acuerdo a la tecnología descrita en el *Modelo de Artefactos Cloud del Incremento* y el *Modelo de Recursos Cloud del Incremento*.

Los plug-ins son creados e incorporados cada vez que se requiere generar código conforme a una nueva tecnología. Es decir, los desarrolladores diseñan y construyen plug-ins de transformación la primera vez que requieren generar código conforme a una tecnología específica, los incorporan en este sub-componente, y los re-utilizan cada vez que los requieren. En este trabajo de tesis se ha utilizado la extensión de Eclipse Acceleo para crear las trasformaciones M2T incorporadas en este sub-componente. Los tipos de plug-in incorporados son:

#### 5.5.2.1 Generador del protocolo de interacción

Este tipo de plug-in genera el código que implementa el protocolo de interacción o lógica de integración de los servicios ofrecidos por participantes del incremento con los servicios ofrecidos por participantes en la aplicación actual. El código generado implementa el diseño de elementos *Service Contract* descritos durante la actividad *Especificación de la integración del incremento* (ver Figura 5-1(1)). Los

plug-in de este tipo son invocados mediante la acción *Generate Interaction Protocol* del menú contextual de la Figura 5-13, toman como entrada *Modelos Extendidos de la Arquitectura del Incremento* y *Modelos de Artefactos Cloud del Incremento,* y generan los artefactos cloud que implementan el protocolo de interacción.

Una vez que el desarrollador ha invocado el *Generador del protocolo de interacción*, selecciona el *Modelo de Artefactos Cloud del Incremento* a partir del cual generará los artefactos cloud que implementan el protocolo de interacción (ver campo *Model* en Figura 5-14). El modelo seleccionado incluye una referencia al *Modelo Extendido de la Arquitectura del Incremento* correspondiente. Adicionalmente, el desarrollador selecciona la ubicación en la que se almacenarán los artefactos cloud generados (campo *Output directory* en Figura 5-14).

Los plug-in de transformación de este tipo utilizan el *Modelo de Artefactos Cloud del Incremento* para obtener la descripción de los artefactos cloud a ser generados; tomando información referente a la representación de los diferentes artefactos a través del atributo *representation* de elementos *Artifact*. Adicionalmente utilizan el atributo *location* de los proyectos de interacción (elementos *Interaction Project*) para determinar la ubicación en la que guardarán los artefactos generados; esta ubicación corresponde a la especificación de la estructura de empaquetado y despliegue descrita en la sección 4.3.2.1. La ubicación especificada en el modelo complementa la ubicación seleccionada por el desarrollador al invocar el plug-in. Por otro lado, los plug-in de transformación de este tipo utilizan el *Modelo Extendido de la Arquitectura del Incremento* para obtener información de diseño de los artefactos cloud descritos en el *Modelo de Artefactos Cloud del Incremento*.

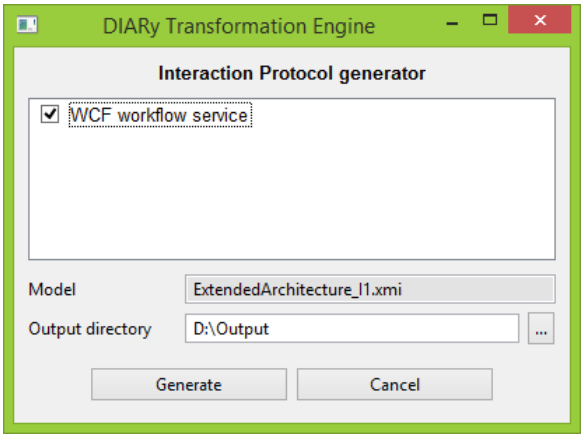



**Figura 5-14 Plug-in Generar protocolo de interacción**

Este trabajo de tesis incluye un plug-in de este tipo, el cual genera artefactos cloud de interacción específicos para el IDE Microsoft Visual Studio 2013. Los

artefactos cloud son creados conforme los requisitos de servicios WCF (Windows Communication Foundation) Workflows y serán desplegados como servicios de orquestación en Microsoft Azure©. En este trabajo se utiliza Microsoft Azure© para el despliegue ya que es el más adecuado para aplicaciones basadas en SOA (Muppalla y otros, 2013). La Tabla 5-2 muestra las correspondencias entre los elementos *Service Contract* del *Modelo de Artefactos Cloud del Incremento* y los artefactos cloud que deberán ser generados para la implementación en Microsoft Azure©.

**Tabla 5-2 Correspondencia entre elementos *Service Contract* del Modelo de Artefactos Cloud y artefactos cloud conforme servicios WCF Workflow**

| Origen | Destino |
|---|---|
| Deployment Project | *Azure Cloud Service Project |
| InteractionProject | *WCF Service Web Role Project |
| DynamicConfiguration | ServiceConfiguration.Cloud.cscfg |
| | ServiceDefinition.csdef |
| InteractionService | WCF Workflow Service |
| Interface | Interface decorada con [ServiceContract] |
| Message Type | Clase decorada con [Message Contract] y atributos decorados con [Message BodyMember] |
| Data Type | Clase decorada con [DataContract] o Enumeración |
| Exposed EndPoint | Entradas en archivos Web.config que indican el punto de acceso al servicio. Este punto de acceso será referenciado como Invoked End Point desde el servicio correspondiente al participante con rol de consumidor de servicio. |
| Invoked EndPoint | Entradas en archivos ServiceDefinition.csdef y ServiceConfiguration.Cloud.cscfg que indican los puntos de acceso a los servicios a ser orquestados. Una entrada por cada servicio a ser orquestado. |

* No respresentan artefactos cloud sino proyectos que el desarrollador deberá crear en el IDE Microsoft Visual Studio 2016. En estos proyectos se incluirán los artefactos clou generados.

Con respecto a la propagación del impacto arquitectónico el plug-in *Generador del protocolo de interacción* soporta la generación de artefactos cloud cuando el impacto arquitectónico del Service Contract es *Add*. El desarrollador tendrá que gestionar la modificación o la eliminación de los artefactos cloud en caso de un valor de impacto arquitectónico diferente. Esto debido a que este plug-in no implementa mecanismos de consistencia incremental; es decir, si es que el desarrollador realiza cambios en los artefactos generados, esos cambios no se mantendrán si es que la transformación es ejecutada nuevamente.

#### *5.5.2.2 Generador de código de implementación*

Este tipo de plug-in genera el código que implementa la lógica o esqueletos de

la lógica de negocio correspondiente al diseño de elementos *Participant* descritos durante la actividad *Especificación de la integración del incremento* (ver Figura 5-1(1)). Los desarrolladores invocan este tipo de plug-in de transformación mediante la acción *Generate Implementation Code* del menú contextual de la Figura 5-13. Una vez que el desarrollador ha invocado el *Generador del protocolo de interacción*, selecciona el *Modelo de Artefactos Cloud del Incremento* a partir del cual generará los artefactos cloud que implementan el protocolo de interacción (ver campo *Model* en la Figura 5-15). El modelo seleccionado incluye una referencia al *Modelo Extendido de la Arquitectura del Incremento* correspondiente. Adicionalmente, el desarrollador selecciona la ubicación en la que se almacenarán los artefactos cloud generados (campo *Output directory* en Figura 5-15).

Las transformaciones de este tipo toman como entrada *Modelos Extendidos de la Arquitectura del Incremento* y *Modelos de Artefactos Cloud del Incremento* y generan artefactos cloud conforme a una tecnología de implementación específica. Al igual que el generador del protocolo de interacción, las transformaciones de este tipo utilizan el *Modelo de Artefactos Cloud del Incremento* para para obtener la información referente a la representación de los diferentes artefactos, e información referente a la especificación de la estructura de empaquetado y despliegue. Por otro lado, utilizan el *Modelo Extendido de la Arquitectura del Incremento* para obtener información de diseño de los artefactos cloud descritos en el *Modelo de Artefactos Cloud del Incremento* y generan los artefactos cloud en base a la información de diseño.

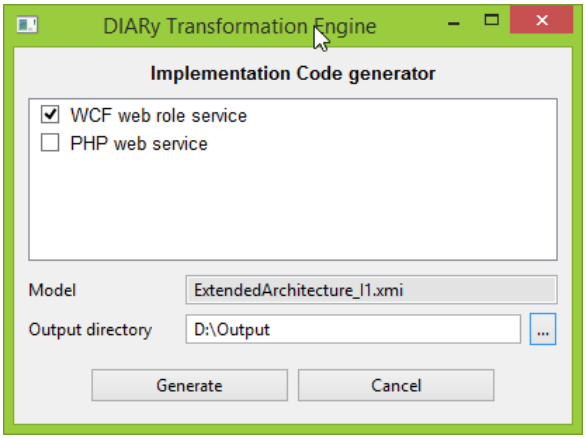



**Figura 5-15 Plug-in Generar código de implementación**

Este trabajo de tesis incluye dos plug-ins de este tipo que generan artefactos cloud de interacción específicos para servicios Web PHP y para el IDE Microsoft Visual Studio 2013. Por ejemplo, en el segundo plug-in, los artefactos cloud

son creados conforme los requisitos de servicios WCF Web Roles (C#). La Tabla 5-3 muestra las correspondencias entre los elementos *Participant* del *Metamodelo Artefactos Cloud* y los artefactos cloud que deberán ser generados para la implementación de servicios WCF Web Roles.

**Tabla 5-3 Correspondencia entre elementos *Participant Use* del Modelo de Artefactos Cloud y artefactos cloud conforme servicios WCF WebRole**

| Origen | Destino |
|---|---|
| Deployment Project | *Azure Cloud Service Project. nombre= nombre elmento *Participant* |
| ImplementationProject | *WCF Service Web Role Project. nombre= nombre elmento *Participant* + "Services", p. ej., DealerServices |
| DynamicConfiguration | ServiceConfiguration.Cloud.cscfg |
| | ServiceDefinition.csdef |
| FrontEndService | WCF Web Role. Incluye clases que implementan las interfaces definidas en elementos *Service Contract* relacionados; las interfaces se determinan en base al rol que cumple el participante. |
| BackEndService | WCF Web Role/ WCF Worker Role. En caso de elementos *Participant* que representan recursos cloud. |
| ClientObject | Clase que incluye código para invocar al servicio de orquestación. Solo en caso de elementos *Participant* que cumplen el rol de consumidor de servicio. |
| Adaptor | Clases que resuleven incompatibilidades entre interfaces |
| Exposed EndPoint | Entradas en archivos Web.config que indican el punto de acceso al servicio. Este punto de acceso será referenciado como Invoked End Point en el achivo de configuración del servicio de orquestación. |
| Invoked EndPoint | Entradas en archivos ServiceDefinition.csdef y ServiceConfiguration.Cloud.cscfg que indican los puntos de acceso al servicio de orquestación. Estas entradas se generan únicamente cuando se trata de elementos *Participant* que cumplen un rol de consumidor de servicio. |

* No respresentan artefactos cloud sino proyectos que el desarrollador deberá crear en el IDE Microsoft Visual Studio 2013. En estos proyectos se incluirán los artefactos clou generados.

El Listado 5-1 muestra el código Acceleo correspondiente a la regla de transformación que genera código de la implementación de la lógica de negocio (*FrontEndService*) de participantes conforme a los requisitos de servicios WCF Web Role. La regla de transformación genera código de implementación únicamente para participantes que cumplen el rol de consumidor de servicio (línea 4). La transformación determina la interface que debe ser implementada (línea 6), y procede a crear el archivo (línea 7) que incluirá esqueletos de la implementación de las operaciones de la interface. La transformación genera código que incluye referencias a librerías necesarias para la construcción de una aplicación conforme

los requisitos de servicios WCF Web Role, así como una referencia (línea 11) al proyecto correspondientes al servicio de orquestación; esta referencia permite hacer uso de los proxis necesarios para la implementación de las interfaces. La línea 13 genera código para crear una clase cuyo nombre es similar al nombre de la interface que implementa. A partir de la línea catorce se genera el código que implementa esqueletos de las operaciones.

**Listado 5-6 Extracto de la regla de transformación para la generación de la lógica de implementación de elementos *Participant***

```
1  [for(RB:RoleBinding | SCU.rolebinding2->
2    select(e:RoleBinding | e.architecturalImpact = ArchitecturalImpact::Add))]
3    [for(R:Role | RB.client->select(oclIsTypeOf(Role)))]
4      [if (R.isConsumer)]
5        [if (R.roleType->notEmpty())]
6          [let inter:Interface = R.roleType]
7            [file (inter.name.concat('.svc.sc'), false)]
8              using Microsoft.WindowsAzure.ServiceRuntime;
9              using System.ServiceModel.Web;
10             …
11             using [R.servicecontract.name.toUpperFirst()/]ServiceContract;
12             …
13           public class [inter.name.toUpperFirst()/] : I[inter.name.toUpperFirst()/] {
14             [for(Op:Operation | inter.ownedOperation) separator ('\n')]
15               public
16               [if(Op.ownedParameter->select(e:Parameter | e.direction =
17               ParameterDirectionKind::return)->isEmpty())]void[else][Op.ownedParameter->
18               select(e:Parameter | e.direction = ParameterDirectionKind ::return).type.name/][/if]
19               [Op.name/]
20              ([if(Op.ownedParameter->select(e:Parameter | e.direction =
21               ParameterDirectionKind::_in)->
22               notEmpty())][for(Param:Parameter | Op.ownedParameter->
23               select(e:Parameter | e.direction = ParameterDirectionKind::_in)) separator(', ')]
24               [if(Param.type->notEmpty())][Param.type.name/]
25               var[Param.type.name/][/if][/for][/if]) {
26               …
27               //PUT HERE YOUR CODE
28               …
29               [if(Op.ownedParameter->select(e:Parameter | e.direction =
30               ParameterDirectionKind::return)->notEmpty())]return new[Op.ownedParameter->
31               select(e:Parameter | e.direction = ParameterDirectionKind::return).type.name/]();[/if]
32             }
33             [/for]
```

Con respecto al código de implementación de elementos *Participant* que cumplen el rol de consumidor de servicio, este código incluye una operación que invoca

al servicio de orquestación; el Listado 5-7 muestra un extracto de la transformación Acceleo que genera dicha operación, la cual corresponde al elemento *ClientObject* del *Modelo de Artefactos Cloud del Incremento*. Las líneas 1 y 2 del listado generan el código de implementación necesario para recuperar del archivo de configuración, el cual es actualizado dinámicamente, el punto de acceso (*Invoked End Point*) al servicio de orquestación; permitiendo que el participante que consume un servicio inicie la interacción mediante la invocación al servicio de orquestación. La línea 3 generará el código de implementación que invoca el servicio de orquestación correspondiente. La línea 4 genera el código necesario para crear una instancia del objeto que será pasado como parámetro; en donde, de la línea 5 en adelante se inicializa los atributos del objeto.

**Listado 5-7 Extracto de la regla de transformación para la generación de la lógica de implementación de elementos *Participant* que consumen servicios**

```
1  client.Endpoint.Address = new EndpointAddress(
2    RoleEnvironment.GetConfigurationSettingValue("to_[SCU.name/]_EndPoint"));
3    resultado = client.place[Op.name.toUpperFirst()/](
4      new place[Op.name.toUpperFirst()/]WorkFlow.place[Op.name.toUpperFirst()/](){
5        [for(Param:Parameter | Op.ownedParameter->
6        select(e:Parameter | e.direction = ParameterDirectionKind::_in))]
7          [for(MT:MessageType | Param.type->select(oclIsTypeOf(MessageType)))]
8            [for(Pro:Property | MT.ownedAttribute) separator (',\n\t\t\t\t\t')]par_[Pro.name/] =
9            p_[Pro.name/]
10       [/for][/for][/for]
11     });
```

Con respecto a la propagación del impacto arquitectónico, este plug-in soporta la generación de artefactos cloud cuando el impacto arquitectónico del elemento *Participant Use* es *Add*. El desarrollador tendrá que gestionar la modificación o la eliminación de los artefactos cloud en caso de un valor de impacto arquitectónico diferente, ya que este plug-in no implementa mecanismos de consistencia incremental.

### *5.5.2.3 Generador de scripts de aprovisionamiento y despliegue*

Este tipo de plug-in genera el código que implementa scripts de aprovisionamiento y despliegue específicos para un proveedor cloud y modelo de servicio seleccionado del proveedor cloud (p. ej., PaaS o IaaS). Los especialistas en operaciones invocan este tipo de plug-in de transformación mediante la acción *Generate Provisioning and Deployment Scripts* del menú contextual de la Figura 5-13. Una vez que el especialista en operaciones ha invocado el *Generador de scripts de aprovisionamiento y despliegue*, selecciona el *Modelo de Recursos Cloud del Incremento* a partir

del cual generará los scripts (ver campo *Model* en la Figura 5-16). El modelo seleccionado incluye una referencia al *Modelo de Artefactos Cloud del Incremento* correspondiente, el cual describe información de *Deployment Projects* relacionados a *Deploymen Artifacts* a desplegar. Adicionalmente, el desarrollador selecciona la ubicación en la que se almacenarán los scripts generados (campo *Output directory* en Figura 5-16).

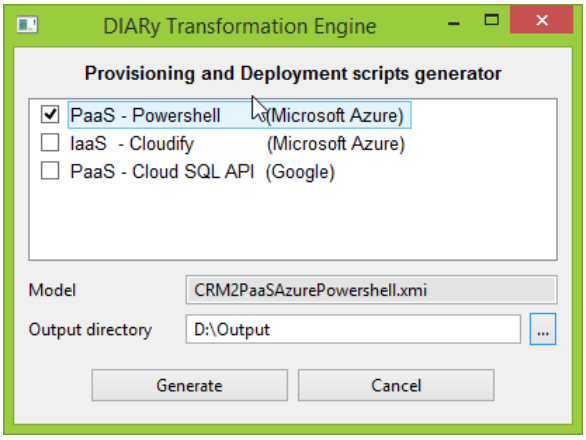



**Figura 5-16 Plug-in Generar scripts de aprovisionamiento y despliegue**

Las reglas de transformación de plug-ins de este tipo utilizan el *Modelo de Recursos Cloud del Incremento* para obtener información del entorno cloud (elemento *Cloud Environment*) que debe ser aprovisionado para el despliegue de un *Deployment Artifact*, información de la subscripción (*Subscription*) al proveedor cloud seleccionado para aprovisionar el *Cloud Environment,* el modelo de servicios a utilizar en la subscripción (atributo *serviceModel* de elementos *Cloud Environment*), y los recursos cloud (*Resources*) que requieren ser aprovisionados para el despliegue. En el caso de que el modelo de servicio seleccionado de un proveedor cloud sea PaaS, las reglas de transformación generan scripts de aprovisionamiento que crean instancias predefinidas de entornos cloud ofrecidas por los proveedores cloud; por lo tanto, no utilizan la descripción de elementos *Resource,* utilizan únicamente el nombre de a instancia predefinida descrita por el atributo *vmSize* de elementos *Cloud Environment*. Los ajustes particulares que permiten acceder y hacer uso de la subscripción, así como los ajustes o características particulares de los recursos cloud ofrecidos por el proveedor, especificados durante la tarea *Especificación de requisitos de aprovisionamiento* de la actividad *Despliegue y reconfiguración dinámica* (ver Figura 5-1(3)), son obtenidos de los elementos *Parameter* y *Resource Parameters* relacionados a elementos *Subscription* y *Resource* respectivamente.

Con respecto a la propagación del impacto arquitectónico, las reglas de transformación hacen uso del atributo *architecturalImpact* del elemento *Deployment Artifact*. Si este atributo tiene el valor *Add*, las reglas de transformación generan scripts de aprovisionamiento y despliegue, si el valor del atributo es *Modify*, las reglas de transformación generan scripts de aprovisionamiento y despliegue que utilizan los entornos cloud previamente aprovisionados.

Por otro lado, en caso de servicios cloud que deben ser replegados, éstos no son documentados en el *Modelo de Recursos Cloud del Incremento*; por lo tanto, las reglas de trasformación emplean elementos *Deployment Projects* cuyo atributo *architecturalImpact* es *Delete* en el *Modelo de Artefactos Cloud del Incremento* relacionado, y generan scripts que liberan los recursos cloud y repliegan el servicio ofrecido por el participante correspondiente.

Este trabajo de tesis incluye tres plug-in de este tipo (ver Figura 5-16), el primero genera scripts de aprovisionamiento y despliegue que hacen uso del modelo de servicios PaaS del proveedor Microsoft Azure©. Este plug-in genera scripts de despliegue y aprovisionamiento a ser ejecutados en Windows PowerShell[8], una interfaz de consola con posibilidad de escritura y unión de comandos por medio de instrucciones. El segundo plug-in de transformación genera scripts de aprovisionamiento y despliegue que hacen uso del modelo de servicios IaaS del proveedor Microsoft Azure©. Para este segundo plug-in se generaron scripts conforme al DSL de cloudify[9] el cual es escrito en YAML[10], requiriéndose el cliente Cloudify 3.4.0 Cli para la ejecución de los scripts. Se escogió utilizar Cloudify en lugar de generar scripts conforme a los requisitos específicos de Microsoft Azure© debido a que el DSL de Cloudify abstrae las instrucciones de aprovisionamiento y despliegue de los requisitos específicos de un proveedor cloud, lo cual facilitará la incorporación de plug-ins específicos para otros proveedores cloud. Finalmente, el último plug-in de transformación genera scripts de aprovisionamiento del servicio Cloud SQL API de Google Cloud Platform.

El Listado 5-8 muestra el pseudocódigo de la definición de transformación escrita en Acceleo para la generación de scripts de aprovisionamiento.

[8] https://msdn.microsoft.com/en-us/powershell

[9] http://getcloudify.org/

[10] http://yaml.org/

**Listado 5-8 Pseudocódigo de la transformación para la generación del script de aprovisionamiento en un proveedor cloud con modelo de servicio IaaS**

```
1  Recorrer CloudEnverioments
2    Crear carpeta CloudEverioments.name: Contiene los archivos de despliegue
3    Recorrer CloudEnverioments.Subscription.Parameters
4      Crear archivo definition.yaml: Contiene declaración de imports e inputs del blueprint
5      Crear archivo inputs.yaml: Contiene los valores de los imputs de definition.yaml
6    Fin recorrer
7    Recorrer CloudEnverioments.Resources
8      Crear archivo bluprint.yaml: Contiene los comandos de aprovisionamiento de recursos
9        nombre de recurso =   [nombre_recurso] + _[CloudEnverioments.name]"
10       Agregar creación recurso "resource_group"
11       Recorrer CloudEnverioments.Resources.IaaS
12         Recorrer CloudEnverioments.Resources.IaaS_Network
13           Agregar creación recurso "virtual_network"
14           Agregar creación recurso "network_security_group"
15           Agregar creación recurso "subnet"
16           Agregar creación recurso "ip_configuration"
17           Agregar creación recurso "public_ip_address"
18           Agregar creación recurso "nic"
19         Fin recorrer
20         Recorrer CloudEnverioments.Resources.IaaS_Storage_Account
21           Agregar creación recurso "storage_account"
22         Fin recorrer
23         Recorrer CloudEnverioments.Resources.IaaS_Computing
24           Agregar creación recurso "virtual_machine"
25         Fin recorrer
26       Fin recorrer
27       Recorrer CloudEnverioments.Resources.PaaS
28         Recorrer CloudEnverioments.Resources.PaaS_AppServer
29           Agregar creación recurso "app_server"
30         Fin recorrer
31         Recorrer CloudEnverioments.Resources.PaaS_Runtime
32           Agregar creación recurso "runtime"
33         Fin recorrer
34       Fin recorrer
35     Fin crear archivo
36   Fin recorrer
37 Fin recorrer
```

El Listado 5-9 muestra un extracto de la regla de transformación escrita en Acceleo que genera el script de despliegue de un *Deployment Artifact* en el proveedor cloud Microsoft Azure©. El script de despliegue es generado conforme al DSL de Cloudify; en donden en las líneas 10 -13 se muestra la generación de instrucciones que obtienen el *Deployment Artifact*, lo descomprime, lo copia en el

directorio de publicación de apache, y finalmente una vez desplegado, elimina el artefacto de despliegue comprimido.

**Listado 5-9 Extracto de la regla de transformación para la generación del script de despliegue en un proveedor cloud con modelo de servicio IaaS**

```
service_[anDeploymentArtifact.deploymentLabel/]:
  type: cloudify.nodes.WebServer
  interfaces:
    cloudify.interfaces.lifecycle:
      create:
        implementation: fabric.fabric_plugin.tasks.run_commands
      inputs:
      commands:
        - wget [anDeploymentArtifact.packageFile/] -P /var/www/html/
        - unzip /var/www/html/
          [anDeploymentArtifact.deploymentScript/].zip -d /var/www/html/
        - rm /var/www/html/[anDeploymentArtifact.deploymentScript/].zip
      use_sudo: true
      fabric_env:
        user: '[getIaaS_Computing(hostedOn).OS.computerName/]'
      key_filename:
        '[getPlatformParameter('ssh_key_filename',getIaaS_Computing(
          anDeploymentArtifact.hostedOn).OS)/]'
      host_string:
        { get_attribute: ['['/]virtual_machine_[
          anDeploymentArtifact.hostedOn.name/], public_ip[']'/] }
      always_use_pty: true
    relationships:
        - type: cloudify.relationships.contained_in
      target: virtual_machine_[anDeploymentArtifact.hostedOn.name/]
```

### *5.5.2.4 Generador de scripts de reconfiguración dinámica*

La reconfiguración dinámica de la arquitectura consiste en integrar los servicios de la arquitectura del incremento en la arquitectura de la aplicación y remplazar las dependencias entre servicios. Las dependencias entre servicios se almacenan en los artefactos de configuración de servicios; por lo tanto, este tipo de plug-in genera el código que implementa scripts que actualizan artefactos de configuración. Los especialistas en operaciones invocan este tipo de plug-in de transformación mediante la acción *Generate Dynamic Reconfiguration Scripts* del menú contextual de la Figura 5-13, seleccionan el tipo de script a generar y la carpeta en la que se almacenará el script generado (ver Figura 5-17).

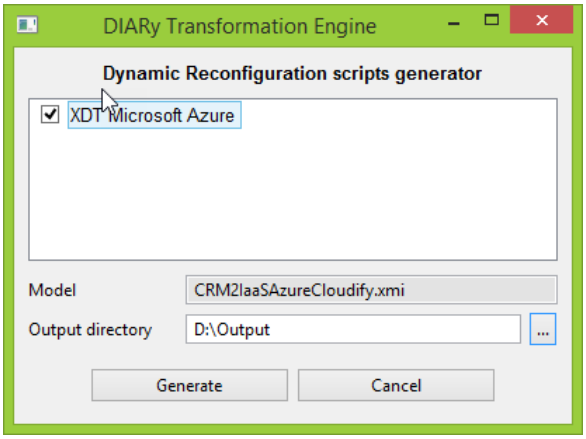



**Figura 5-17 Plug-in Generar scripts de reconfiguración dinámica**

Los plug-in de transformación de este tipo utilizan el *Modelo de Recursos Cloud del Incremento* para para obtener información de dependencias entre servicios (*InvokedEndpoint*) correspondientes a los servicios (*Service*) incluidos en un *Deployment Artifact*, y generan archivos XDT utilizados para actualizar los artefactos de configuración. Las reglas de transformación hacen uso del atributo *architecturalImpact* del elemento *InvokedEndPoint* y de acuerdo a su valor actualiza entradas artefactos de configuración. Adicionalmente, las reglas de transformación de este tipo de plug-in utilizan la información de elementos *Setting* relacionados a elementos *Service* para actualizar ajustes de servicios, almacenados en artefactos de configuración, que cambian en tiempo de ejecución.

Este trabajo de tesis incluye un plug-in de este tipo, el cual actualiza entradas en artefactos de configuración ServiceConfiguration.cscfg que son utilizados por servicios WCF Service Web Role desplegados en Microsoft Azure© para facilitar la reconfiguración dinámca, permitiendo actualizar ajustes a servicios sin requerir el redespliegue el paquete de software correspondiente. El Listado 5-10 muestra un extracto de la regla de transformación definida en este plug-in; en donde, por cada *Deployment Artifact* descrito en el *Modelo de Recursos Cloud de Incremento* genera código que crea un archivo conforme al esquema de archivos XDT (líneas 3-6).

Con respecto a la propagación del impacto arquitectónico descrito durante la especificación de la integración, reglas de transformación evaluan el valor del atributo *architecturalImpact*, generando código que actualiza las entradas con información (línea 20) de los puntos de acceso (línea 19) correspondientes a los servicios externos invocados por el elemento *Service* que se esta analizando (línea 14). Las reglas de tranformación no considera los valores *Reference* y *Delete* del atributo *architecturalImpact* (líneas 16 y 17), ya que el primer valor indica que el

elemento Service correspondiente no sera afectado por la integración; mientras que el segundo valor indica que el elemento *Service* será eliminado, por lo tanto los enlaces tambíen serán eliminados.

**Listado 5-10 Extracto de la regla de transformación para la generación del script de reconfiguración dinámica conforme al esquena de archivos XDT**

```
1  [template public generateElement(anCloudResourcesModel : CloudResourcesModel )]
2  [comment @main/]
3  [for (anDeploymentArtifact : DeploymentArtifact|
4     anCloudResourcesModel.deploymentartifact)]
5   [file (anDeploymentArtifact.deploymentScript.concat('\\')+
6     anDeploymentArtifact.deploymentLabel.concat('NewConfig.XDT'), false)]
7     <?xml version="1.0" encoding="utf-8"?>
8     <ServiceConfiguration
9       [for (anCloudEnvironment : CloudEnvironment|
10        anDeploymentArtifact.cloudenvironment)]
11        [if (anCloudEnvironment.belongTo = anSubscription)]
12          <Role name="[anCloudEnvironment.name/]">
13          <ConfigurationSettings xdt:Transform="InsertIfMissing">
14          [for (anService : Service| anCloudEnvironment.services)]
15            [for (anInvokedEndpoint : InvokedEndpoint| anService.invokedendpoint)]
16              [if (anInvokedEndpoint.architecturalImpact <> ArchitecturalImpact::None) and
17                (anInvokedEndpoint.architecturalImpact <> ArchitecturalImpact::Delete) and
18                (anInvokedEndpoint.architecturalImpact <> ArchitecturalImpact::Reference)]
19                <Setting name="[anInvokedEndpoint.name.concat('_EndPoint')/]"
20                value="[anInvokedEndpoint.protocol.toString().concat('://')
21                  .concat(anInvokedEndpoint.domainName
22                  .concat(':').concat(anInvokedEndpoint.port.toString()).concat('/')
23                  .concat(anInvokedEndpoint.service))/]"
24                  [parseXDT(anInvokedEndpoint.architecturalImpact)/]/>
25              [/if]
26            [/for]
27          [/for]
28          </ConfigurationSettings>
29          </Role>
30        [/if]
31      [/for]
32    </ServiceConfiguration>
33 [/file]
```

Los archivos XDT incluyen instrucciones que indican la acción (p. ej., añadir, eleminar, insertar si no existe, etc) a realizar con las entradas de achivos Service-Configuration.cscfg, por lo que en la línea 24 se invoca a una función que devuelve la instrucción conforme al esquema XDT que corresponde al valor del atributo *architecturalImpact*.

## 5.6 Conclusiones

En este capítulo se ha presentado la aproximación tecnológica de soporte al método DIARy, la cual cubre las actividades de *Especificación de la integración del incremento*, *Implementación del incremento*, y *Despliegue y reconfiguración dinámica* (ver Figura 5-1(1-3)).

La aproximación tecnológica, una infraestructura software, que hemos implementada como un conjunto de plug-ins para ser integrados en el ambiente de desarrollo común Eclipse EMF. Esto no solo evita que los desarrolladores requieran utilizar otros entornos, sino que facilita la ejecución de las actividades del método DIARy, disminuye la ambigüedad y la introducción de errores. La infraestructura software incluye: i) Un conjunto de DSLs, conformados tanto por metamodelos (ver Figura 5-1(4-7)) que describe su sintaxis abstracta y promueven la aplicación de patrones de diseño o principios que favorecen la reconfiguración dinámica; así como por editores gráficos que describen su sintaxis concreta; ii) Un conjunto de transformaciones que permiten automatizar tanto: la generación de modelos a diferentes niveles de abstracción que reflejan diferentes aspectos del incremento de software a ser integrado; así como la generación de artefactos específicos para un proveedor cloud, que facilitan la reconfiguración dinámica de la arquitectura actual de la aplicación.

La infraestructura software integra diferentes herramientas y mecanismos del desarrollo dirigido por modelos para facilitar el razonamiento sistemático del impacto arquitectónico de la integración y su propagación hacia otros niveles de abstracción. Esta infraestructura, que se encuentra en estado de prototipo, nació con la intención de mejorar la productividad en la creación y edición de los modelos propuestos en el método DIARy, y como resultado de la validación empírica. En donde, durante la validación empírica, en lugar de utilizar editores gráficos creados específicamente para los DSLs propuestos, se emplearon Perfiles UML, identificándose mejoras que deberían ser implementadas y que afectan sobre todo a la usabilidad de las herramientas provistas.

# Capítulo 6. Aplicación del método DIARy

Este capítulo presenta un ejemplo que ilustra la aplicabilidad del método DIARy como soporte a la reconfiguración dinámica e incremental de aplicaciones cloud. Para ello, se planteó un escenario de desarrollo incremental e integración en cuya solución se aplicaron las diferentes actividades del método.

Las siguientes secciones describen cómo el método DIARy y su infraestructura software han sido aplicadas en este ejemplo; tanto en la especificación de las decisiones tomadas en las diferentes actividades, así como en la generación de artefactos cloud que facilitan la implementación, aprovisionamiento, despliegue y reconfiguración dinámica de la arquitectura actual de la aplicación cloud.

La sección 6.1 describe el escenario que será utilizado como ejemplo para ilustrar la aplicabilidad del método DIARy.

La sección 6.2 describe cómo los arquitectos ejecutaron la actividad *Especificación de la Integración del Incremento* a fin de especificar como los elementos de la arquitectura del incremento colaborarán para reconfigurar la arquitectura actual de la aplicación. Especificando, además, la demanda esperada y naturaleza del trabajo de los servicios incluidos en el incremento, información que soportará la toma de decisiones de implementación, aprovisionamiento y reconfiguración en actividades posteriores.

La sección 6.3 describe cómo los desarrolladores ejecutaron la actividad *Implementación del incremento* a fin de generar la implementación del servicio de orquestación y la lógica de los participantes (o servicios) involucrados en la integración, implementando soluciones que facilitan la integración y reconfiguración dinámica.

La sección 6.4 describe cómo los especialistas en operaciones ejecutan la actividad *Despliegue y reconfiguración dinámica* a fin de especificar y aprovisionar los entornos cloud que soportarán la demanda y naturaleza de trabajo de los servicios incluidos en un incremento de software; así como la reconfiguración dinámica e interacción entre servicios cloud desplegados en diferentes entornos cloud.

La sección 6.5 presenta las conclusiones del capítulo.

## 6.1 Ejemplo de aplicación del método DIARy

Para ilustrar el uso de nuestro enfoque, en esta sección introducimos un escenario de desarrollo incremental e integación en el cual una compañía de manufactura desea mejorar el soporte tecnológico ofrecido a sus socios. Con este propósito, ha iniciado el desarrollo de una aplicación que incorporará funcionalidades de manera incremental. La versión inicial de la aplicación provee servicios cloud a sus distribuidores para que gestionen sus órdenes, facilitando la interacción directa entre el sistema IT de los distribuidores y el de la compañía. Ahora, la compañía requiere incorporar un proceso de entregas como parte de su gestión de pedidos; por lo tanto, en el Incremento-1 proveerá de servicios cloud a sus socios de empresas de reparto, esto les permitirá gestionar órdenes de entrega emitidos por la compañía al transportista. Este escenario ha sido adaptado y extiende el propuesto por Berre (2008). La Figura 6-1, muestra un extracto del *Modelo de Arquitectura de la Aplicación* actual el cual evolucionará luego de la integración del incremento; este modelo es descrito con el *Editor del ADL para la Integración Incremental* ().

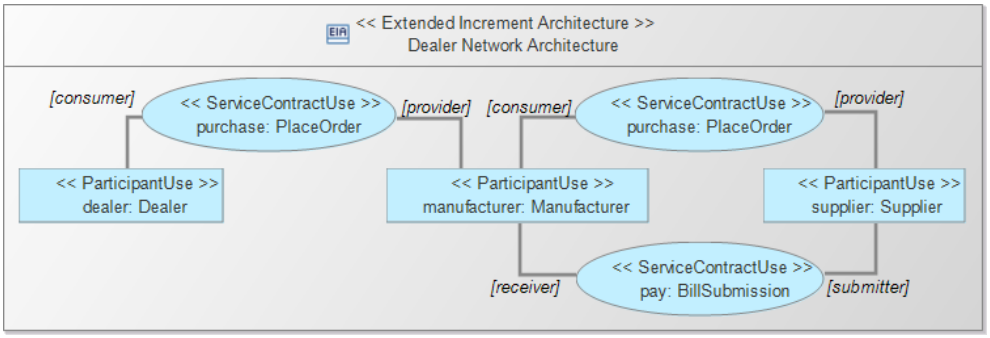



**Figura 6-1 *Modelo de Arquitectura de la Aplicación* actual**

El método DIARy toma como entrada: i) el *Modelo de Arquitectura de la Aplicación* actual (ver Figura 6-1), el cual es utilizado por los arquitectos para identificar los elementos arquitectónicos de la arquitectura actual de la aplicación que, luego de la integración, cambiarán o interactuarán con elementos arquitectónicos del incremento; ii) el *Modelo de Arquitectura del Incremento* (ver Figura 6-2), el cual ha sido editado con el *Editor del ADL para la Integración Incremental*; y iii) términos del SLA. Los modelos forman parte de la descripción del escenario de ejemplo.

La Figura 6-2 muestra una vista de alto nivel del *Modelo de la Arquitectura del Incremento* del Incremento-1, que incluye los servicios a ser integrados y las relaciones entre ellos. Este modelo es utilizado por arquitectos de software para especificar la integración del incremento. Los participantes (otros servicios o aplicaciones)

que se espera estén involucrados en la interacción son especificados por elementos *ParticipantUse*, mientras que los contratos de servicio son especificados por elementos *ServiceContractUse*; esto se debe a que las definiciones de *Participant* y *Service Contract* pueden ser reutilizados en diferentes interacciones.

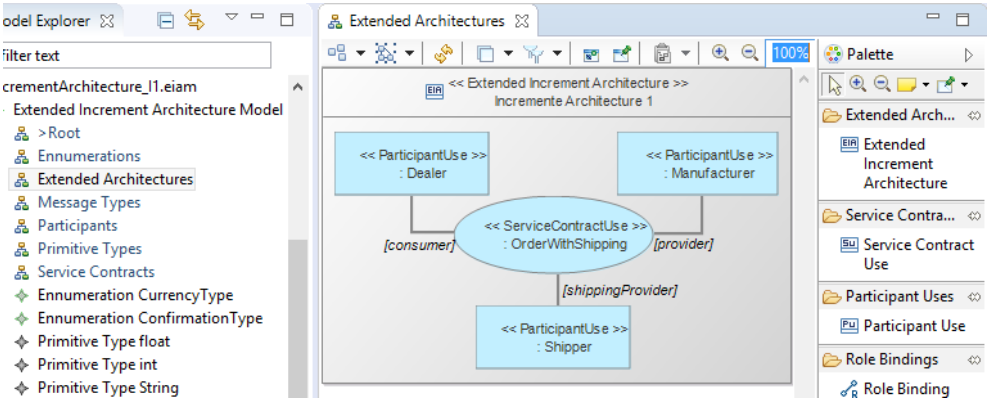



**Figura 6-2 *Modelo de Arquitectura del Incremento* – Vista Arquitectura de Servicios**

La integración del Incremento-1 reemplazará el contrato de servicio *Place Order* de la arquitectura actual (Figura 6-1) con el contrato de servicio *Order With Shipping* (Figura 6-2) de la arquitectura del incremento. Además, incorporará el participante *Shipper* y actualizará la implementación de la lógica del participante *Manufacturer* a fin de implementar las interfaces que le permitan cumplir su rol en el contrato de servicios *Order With Shipping*. Con respecto a la implementación del participante *Dealer*, ésta no se verá afectada por la integración.

A pesar de que para ejecutar la actividad *Especificación de la integración del incremento* los arquitectos hacen uso de una vista de alto nivel, la vista *Arquitectura de Servicios* que se muestra en la Figura 6-2, el modelo de la arquitectura del incremento incluye el modelado de sus elementos internos, cuyo extracto es presentado en la Figura 6-3 a la Figura 6-5. La Figura 6-3 muestra el modelado del elemento *Service Contract Order With Shipping*, el cual describe los roles (p. ej., consumer, provider, shipping provider) que deben cumplir los participantes involucrados en un servicio con el fin de interactuar; las interfaces que modelan explícitamente las operaciones que los participantes deben implementar para cumplir un rol y ofrecer la funcionalidad del servicio (p. ej., Order Placer, Order Taker, Shipping Provider); y el protocolo de interacción que especifica la interacción entre participantes sin definir sus procesos internos.

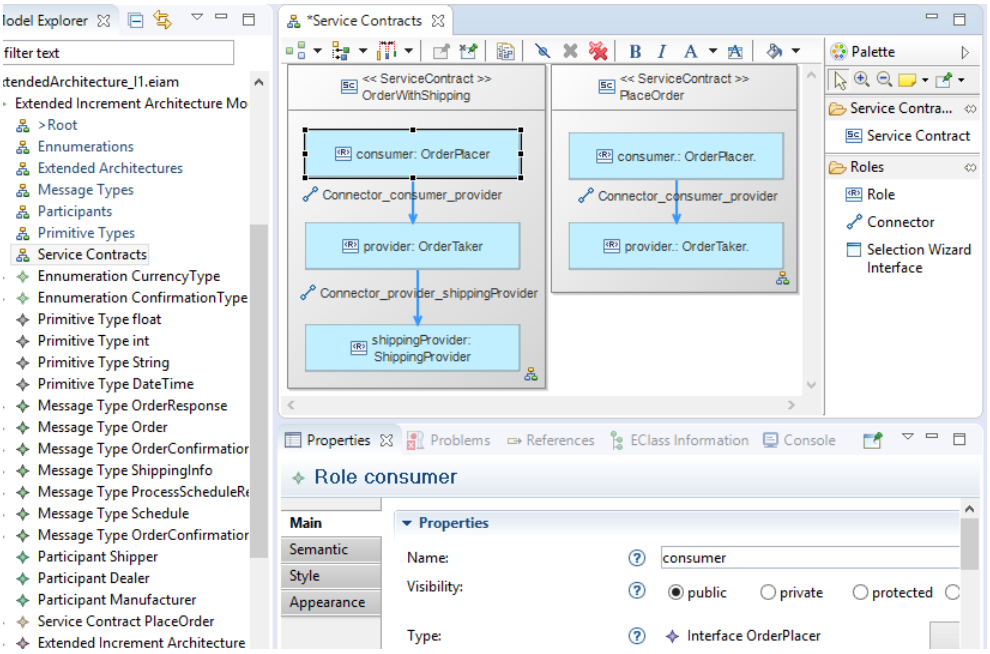



**Figura 6-3 *Modelo de Arquitectura del Incremento* – Vista Service Contract**

La Figura 6-4 muestra la descripción de las interfaces que definen los roles que deben cumplir los participantes del contrato de servicio *Order With Shipping*.

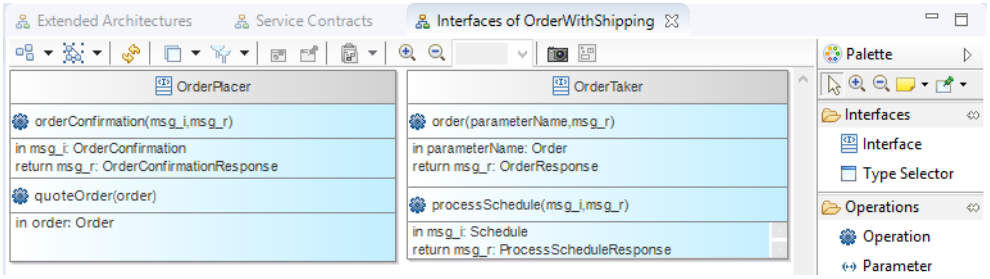



**Figura 6-4 *Modelo de Arquitectura del Incremento* – Vista Interfaces**

La Figura 6-5 muestra el protocolo de interacción que describe como los proveedores que cumplen un rol en la arquitectura del incremento interactuarán. Esta vista se muestra en modo de árbol con el objetivo de presentar los constructores que se han tomado de UML para la descripción de la interacción.

## 6.2 Especificación de la integración del incremento

Esta actividad se inicia tomando como entrada el *Modelo de Arquitectura del Incremento* el cual ha sido previamente diseñado con el *Editor del ADL para la Integración Incremental*.

Los arquitectos complementan el *Modelo de Arquitectura del Incremento* con información que describe la integración, produciendo el *Modelo Extendido de la Arquitectura del Incremento* (ver Figura 6-2 ). El *Modelo Extendido de la Arquitectura del Incremento* describe cómo los elementos arquitectónicos descritos en el *Modelo de Arquitectura del Incremento* colaborarán para cambiar la arquitectura actual de la aplicación cloud. Adicionalmente, describe la demanda y naturaleza de trabajo esperadas para los servicios incluidos en el incremento. La información de demanda y naturaleza esperada será utilizada en actividades posteriores para tomar decisiones de implementación y aprovisionamiento en entornos cloud. Los arquitectos siguen los pasos definidos para esta actividad (ver 4.3.1) y aplican la *Guía para la Especificación de la Integración de Incrementos* (ver Apéndice A.1).

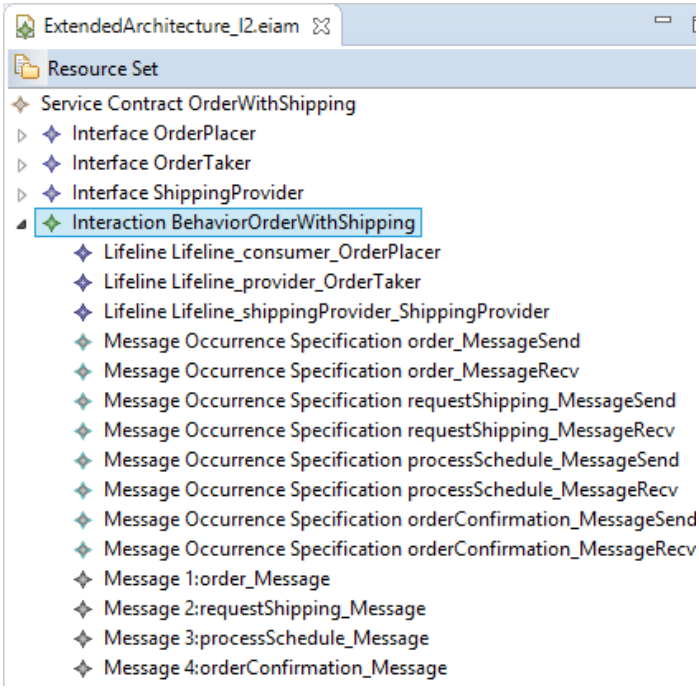



**Figura 6-5 *Modelo de Arquitectura del Incremento* – Vista Protocolo Interacción**

En el primer paso de esta actividad los arquitectos especifican la lógica de integración. Si es que el protocolo de interacción entre los servicios a incluir en el

incremento no ha sido descrito en *Modelo de Arquitectura del Incremento*, o no está completo, lo describen o modifican; especificando además el rol que inicia con la interacción mediante la asignación del valor true al atributo *isConsumer* del rol correspondiente. Luego, los arquitectos identifican los elementos arquitectónicos del incremento que existen en la arquitectura actual. Por ejemplo, los *ParticipantUse* de tipo *Dealer* y *Manufacturer* del *Modelo Extendido de la Arquitectura del Incremento* ya existen en el *Modelo de Arquitectura de la Aplicación* actual (el tipo y rol que cumplen los participantes son iguales en ambas arquitecturas); por lo tanto, asignan los nombres de instancia de los elementos de la arquitectura actual a los elementos de la arquitectura del incremento. El *Service Contract* Place Order será eliminado de la arquitectura actual debido a la integración; sin embargo, éste elemento no existe en *Modelo Extendido de la Arquitectura del Incremento* por lo que lo incluyen a fin de poder especificar su eliminación.

En el siguiente paso de esta actividad, el arquitecto corrige posibles inconsistencias entre las interfaces de la arquitectura del incremento y las interfaces de la arquitectura actual de la aplicación; incluyendo para ello elementos arquitectónicos (p. ej., modelando clases o métodos) que corrigen las inconsistencias. En este ejemplo no existieron inconsistencias entre las interfaces por lo tanto los arquitectos procedieron a especificar el impacto arquitectónico de la integración. Para ello asignaron un valor al atributo *architectural Impact* de cada uno de los elementos internos del *Modelo Extendido de la Arquitectura del Incremento*. Por ejemplo, el *ServiceContractUse* Order With Shipping y sus *RoleBinding* relacionados fueron etiquetados con *architectural Impact* = Add. El *ServiceContractUse* purchase:PlaceOrder y sus *RoleBinding* relacionados que fueron incluidos en el paso anterior por no ser parte de la arquitectura del incremento, pero serán eliminados de la arquitectura actual durante la integración, fueron etiquetados con *architectural Impact* = Delete. Adicionalmente, la implementación actual de *Dealer* no sufrirá cambios producto de la integración, por lo tanto, fue etiquetado con *architectural Impact* = Reference; la implementación actual de *Manufacturer* cambiará a fin de implementar la interface que le corresponde en el contrato de servicios *Order With Shipiping*, por lo tanto, fue etiquetado con *architectural Impact* = Modify. La Figura 6-6 muestra la especificación de la integración para este ejemplo. Para facilitar la lectura del modelo el *Editor del ADL para la Integración Incremental* muestra el impacto arquitectónico especificado para cada elemento arquitectónico mediante iconos; el significado de los iconos es: "+" = *Add*, "-" = *Delete*, "√" = *Modify*, y "∞" = *Reference*.

En el siguiente paso de esta actividad, los arquitectos analizan la demanda y naturaleza del trabajo esperada de los elementos *Participant* y *Service Contract*. De acuerdo a los requisitos del usuario y términos SLA los arquitectos de software han previsto que el proveedor del contrato de servicio *Order With Shipping* tendrá

periodos de alta demanda y debe ser atendido sin retrasos. Por lo tanto, el *ParticipantUse* Manufacturer, quien cumple el rol de proveedor del servicio, es etiquetado con valores *scalability* = HighVolumenRequests y *lifetime* = Synchronous. Adicionalmente, el *ServiceContractUse* purchase: Order With Shipping gestionará la interacción entre los participantes involucrados en este contrato de servicio; por lo tanto, es etiquetado con valores iguales (Figura 6-6).

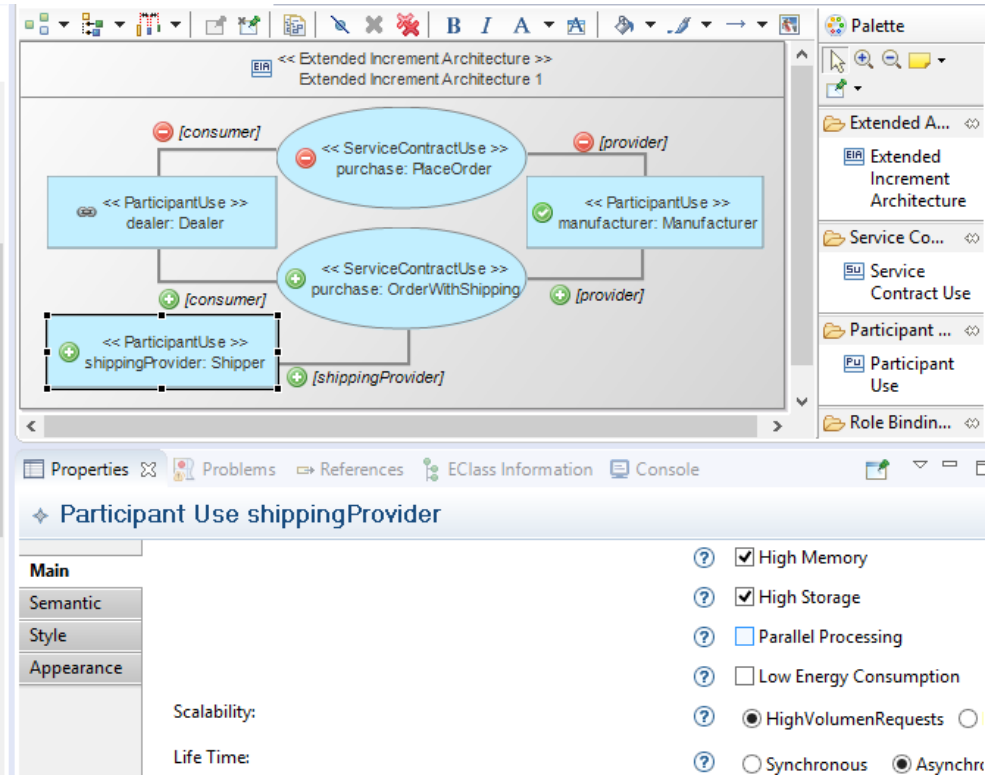



**Figura 6-6 *Modelo Extendido de la Arquitectura del Incremento* (Incremento-1)**

Finalmente, en el paso *Refactorización de la arquitectura del incremento*, los arquitectos identifican la oportunidad de sacar ventaja de los servicios de gestores de bases de datos ofrecidos por los proveedores cloud. Por lo tanto, incluyen en el *Modelo Extendido de la Arquitectura del Incremento* los elementos *ParticipantUse* (dbShipper: shipmentData) y *ServiceContractUse* (dataMng: DataManager) que representan respectivamente un servicio cloud de gestión de datos ofrecido por una plataforma cloud y el servicio de orquestación requerido para la interacción. Los arquitectos especifican que el nuevo elemento *ParticipantUse* será provisto por una plataforma cloud, para lo cual asignan el valor DataMng al atributo *hostResouce*. Adi-

cionalmente, incluyen los elementos *RoleBinding* y etiquetan los nuevos elementos arquitectónicos incluidos con *architectural Impact* = Add, ver Figura 6-7.

## 6.3 Implementación del incremento

En esta actividad los desarrolladores toman decisiones no solo de la tecnología de implementación sino acerca de la estructura de empaquetado y despliegue de los artefactos que implementan (artefactos cloud de implementación) el diseño arquitectónico de la especificación del incremento.

La estructura de empaquetado y despliegue consolida los artefactos de implementación en proyectos, promoviendo durante la implementación el cumplimiento de principios SOA y otros principios que facilitan la reconfiguración dinámica. Las decisiones tomadas en esta actividad son documentadas en el *Modelo de Artefactos Cloud del Incremento*. Los desarrolladores ejecutan esta actividad de acuerdo a lo descrito en la sección 4.3.2.

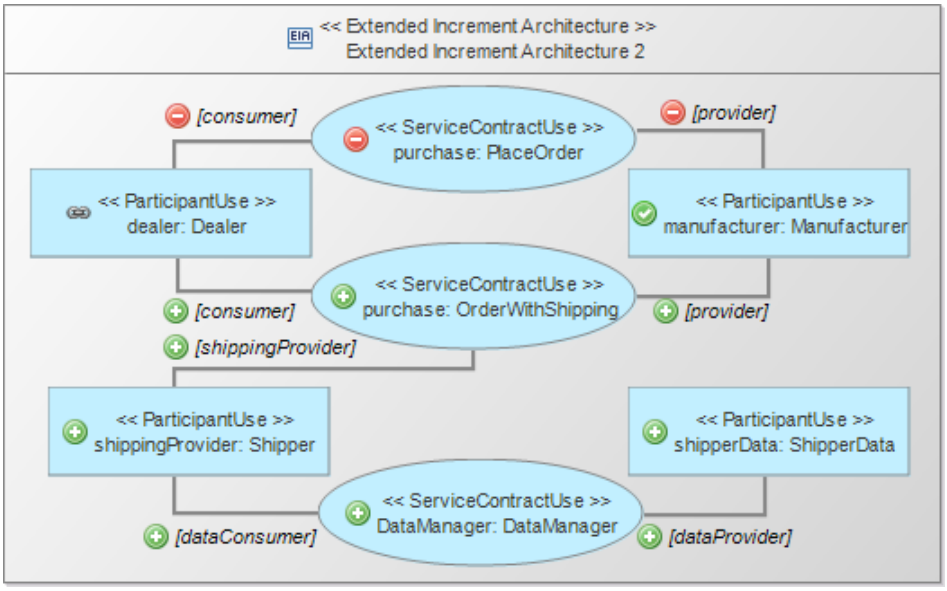



**Figura 6-7 *Modelo Extendido de la Arquitectura del Incremento* (Incremento-1) – Luego de Refactorización**

En el primer paso de esta actividad los desarrolladores utilizaron el componente *Generación de Modelos de Artefactos Cloud* (ver 5.5.1.1 ) el cual, a partir del *Modelo Extendido de la Arquitectura del Incremento* (Figura 6-7), generó parcialmente el *Modelo de Artefactos Cloud del Incremento* (Figura 6-8). La ejecución de las reglas de

transformación generó y organizó las descripciones de artefactos cloud de implementación de tal manera que desacoplaron los artefactos cloud que implementan la orquestación entre servicios (o lógica de integración descrita por elementos *Service Contract*) de los artefactos cloud que implementan la lógica de operación expuesta por los servicios incluidos en el incremento (descrita por elementos Participant). Adicionalmente, la ejecución de reglas de transformación consolidó (o agrupó) los artefactos de implementación en proyectos de despliegue de acuerdo a: la naturaleza de trabajo, demanda esperada, tipo de elementos arquitectónico que implementan (p. ej., *Participant* o *Service Contract*), y el impacto arquitectónico. Luego propagaron el impacto arquitectónico desde los elementos arquitectónicos hasta los proyectos de despliegue correspondientes. Por ejemplo, en el escenario de ejemplo, los elementos arquitectónicos de tipo *ParticipantUse* y de tipo *ServiceContractUse* tienen requisitos *scalability* = HighVolumenRequests; por lo tanto, a fin de satisfacer esos requisitos, las descripciones de artefactos cloud fueron organizados en proyectos de despliegue exclusivos.

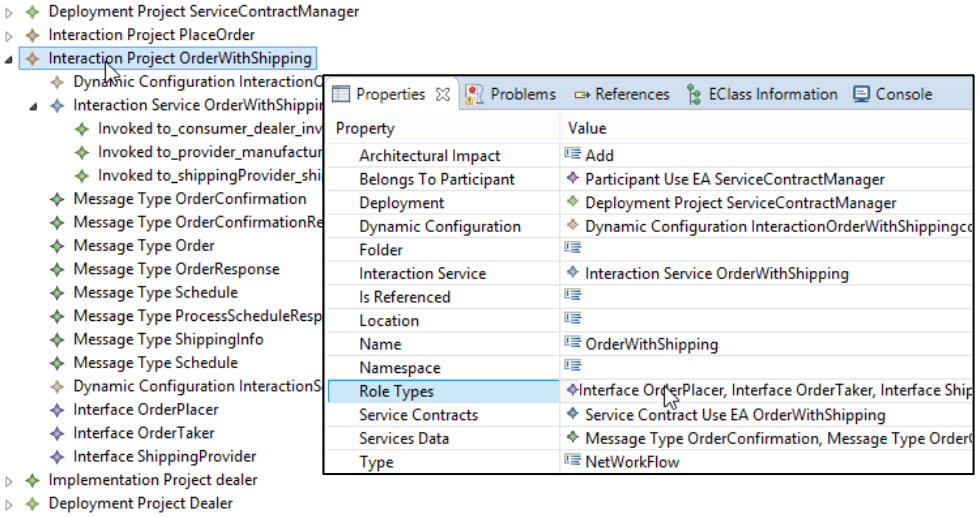



**Figura 6-8 *Modelo de Artefactos Cloud* (Incremento-1)**

Finalmente, para satisfacer patrones de diseño cloud para la reconfiguración dinámica, se generaron entradas en archivos de configuración de los ajustes a servicios que cambiaran en tiempo de ejecución. Esto permitirá que los ajustes sean actualizados en tiempo de ejecución sin que requiera desplegar nuevamente todo el paquete. Por ejemplo, los puntos de acceso a los servicios invocados por un servicio consumidor (Invoked End Points) fueron definidos como ajustes del servicio consumidor y generados como entradas en el archivo de configuración correspondiente.

En un segundo paso de esta actividad, los desarrolladores utilizaron el *Editor de Artefactos Cloud* para modificar, en caso de requerirlo, la consolidación artefactos (o estructura de empaquetado) generada por el motor de generación de modelos y descrita en el *Modelo de Artefactos Cloud del Incremento*. Luego los desarrolladores toman decisiones de implementación y las documentan en el *Modelo de Artefactos Cloud del Incremento*; decisiones tales como: la ubicación en la que se generarán los artefactos que implementan los elementos arquitectónicos, el lenguaje de implementación (atributo *technology* en *Deployment Project*), la implementación de los servicios (atributo *serviceImplementation* en *Deployment Project*), los ajustes de servicios que cambiarán en tiempo de ejecución (elementos *Settings* relacionados a elementos *Dynamic Configuration*. Por ejemplo, los desarrolladores especificaron que el los servicios incluidos en el proyecto de despliegue correspondiente al *Participant* Manufacturer tendrán una implementación basada en SOAP (*serviceImplementation* = SOAP) en un lenguaje de programación conforme los requisitos de servicios WCF Web Role (*technology* = WCF Web Role); los servicios incluidos en el proyecto de despliegue correspondiente al *Participant* Shipper tendrán una implementación basada en REST (*serviceImplementation* = REST) en un lenguaje de programación conforme a los requisitos de PHP (*technology* = PHP); mientras que los servicios incluidos en el proyecto de despliegue correspondiente al *Service Contract* Order With Shipping tendrán una implementación basada en SOAP (*serviceImplementation* = SOAP) con un lenguaje conforme a los requisitos de servicios WCF Workflow (*technology* = WCF Workflow) (ver Figura 6-8).

Una vez que los desarrolladores han especificado sus decisiones de implementación, utilizan el componente de la infraestructura software *Generador del Protocolo de Interacción* (5.5.2.1 ) para generar la implementación del código de integración (servicio de orquestación). La Figura 6-9 muestra el proyecto de tipo Cloud Service Microsoft Azure© de Microsoft Visual Studio 2013 generado, el cual implementa el diseño del elemento arquitectónico *Service Contract* Order With Shipping modelado durante la especificación de la integración.

El impacto arquitectónico del proyecto de despliegue al que se asignaron los artefactos que implementan el *Service Contract* fue Add, por lo tanto, El *Generador del Protocolo de Interacción* generó un proyecto nuevo. El proyecto fue generado íntegramente por el generador e incluye: i) la implementación del protocolo de interacción (servicio de orquestación) como un servicio WCF Workflow (archivo *.xamlx); ii) la implementación de las interfaces, mensajes y tipos de datos como clases (archivos *.cs); y iii) archivos de configuración con entradas que corresponden a los puntos de acceso a los servicios cuya interacción será orquestada (archivos ServiceDefinition.csdef y ServiceConfiguration.Cloud.cscfg). La Figura 6-10 muestra, en modo gráfico, el servicio de orquestación (Order-

WithShipping.xamlx) generado; en esta figura se puede apreciar como los servicios orquestados son invocados utilizando diferentes mecanismos de acuerdo a su implementación. Por ejemplo, el servicio correspondiente al participante *Manufacturer* es implementado como un servicio SOAP mientras que el servicio ofrecido por el participante *Shipper* es implementado como un servicio REST.

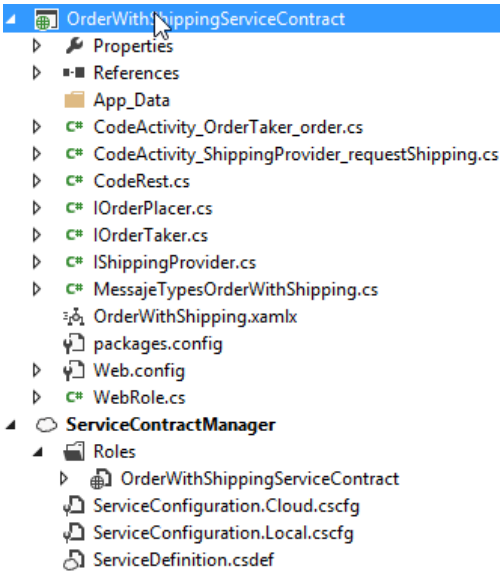



**Figura 6-9 Implementación del Servicio de Orquestación (Incremento-1) - Proyecto Cloud Service Microsoft Azure©**

El Listado 6-1 muestra uno de los archivos de configuración (ServiceConfiguration.Cloud.cscfg) que soporta la reconfiguración dinámica de servicios WCF Workflow y que ha sido generado por el *Generador del protocolo de interacción*.

**Listado 6-1 Archivo de configuración dinámica del servicio de orquestación**

```
3  ************************
4  *                                                              *
5  *  Source code partially generated by the DIARy transformation engine  *
6  *                                                              *
7  ************************
8  -->
9  <ServiceConfiguration serviceName="ServiceContractManager"
10     xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration"
11     osFamily="4" osVersion="*" schemaVersion="2015-04.2.6">
12   <Role name="OrderWithShippingServiceContract">
13     <Instances count="1" />
14     <ConfigurationSettings>
15       <Setting name="to_consumer_dealer_endpoint" value="http://..." />
16       <Setting name="to_provider_manufacturer_endpoint" value="http://..." />
17       <Setting name="to_shippingProvider_shipper_endpoint" value="http://..." />
18     </ConfigurationSettings>
19   </Role>
20 </ServiceConfiguration
```

El Listado 6-1 incluye entradas con información de los puntos de acceso a los servicios cuya interacción será gestionada por el servicio de orquestación; es decir, información de los *Invoked End Points* del servicio de orquestación (ver líneas 15-17).

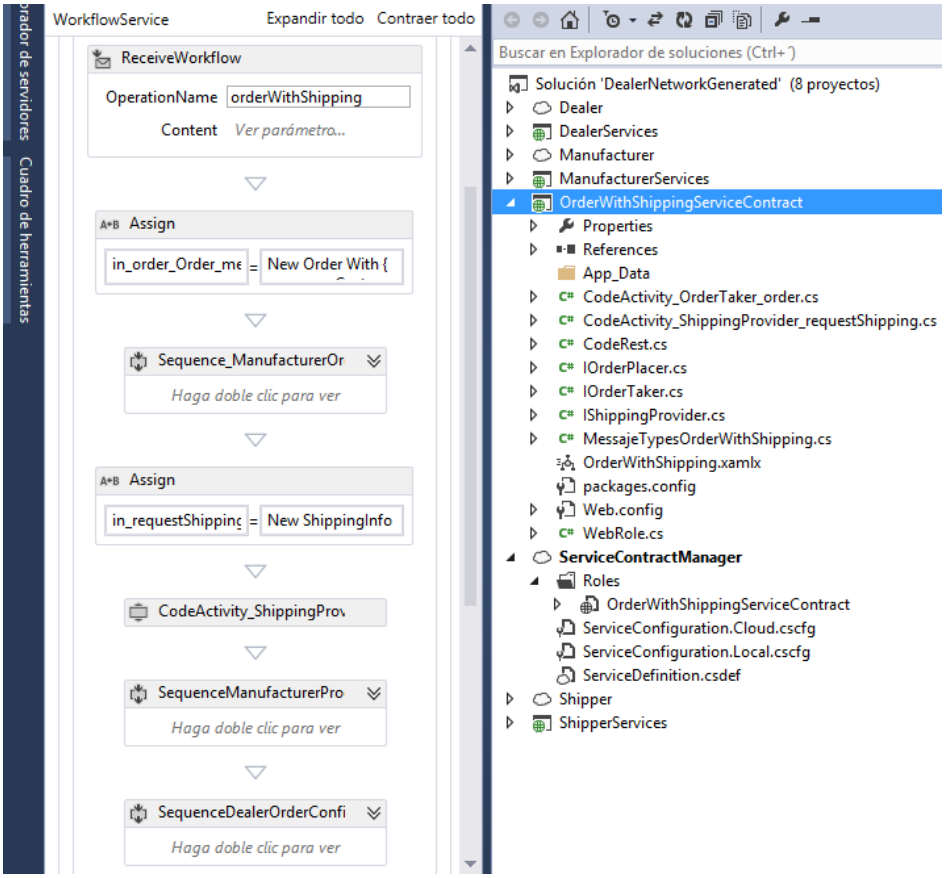



**Figura 6-10 Implementación del Servicio de Orquestación (Incremento-1) - WCF Workflow**

Una vez que los desarrolladores han generado el protocolo de interacción utilizan el componente de la infraestructura software *Generador de código de implementación* para generar la implementación de la lógica de los participantes involucrados en el contrato de servicios *Order With Shipping*; esto de acuerdo a la tecnología, tipo de implementación e impacto arquitectónico especificados para los proyectos de despliegue correspondientes a cada participante y contrato de servicios. Esta implementación permitirá a los servicios participantes integrarse e interactuar con los servicios de la aplicación actual mediante la implementación de las interfaces que le corresponden de acuerdo al rol que cumplen en el contrato de

servicios. De acuerdo a la especificación la tecnología de implementación del proyecto de despliegue en el que se empaquetarán los artefactos de implementación del participante Shipper es *technology* = PHP, la implementación *serviceImplementation* = REST y *architectural Impact* = Add. El Listado 6-2 muestra el esqueleto de la implementación generado que corresponde a lógica del nuevo participante, *Shipper*; las líneas 9, 11 y 15 definen los métodos estándar de la implementación de un servicio REST en PHP.

**Listado 6-2 Esqueleto de la lógica del participante Shipper (Incremento-1) - PHP**

```
1  <?php
2  /**************************************************************************
3   *                                                                        *
4   *  Source code partially generated by the DIARy transformation engine    *
5   *                                                                        *
6   **************************************************************************/
7   ...
8   ...
9   $app->add(new \Slim\Middleware\ContentTypes());
10
11  $app->get('/', function() {
12      echo 'Welcome to DIARy web service';
13  });
14
15  $app->post("/requestShipping", function () use($app) {
16      $app->response()->header("Content-Type", "application/json");
17
18      //Read and Asign input parametes
19      $in_in_requestShipping_ShippingInfo_msg = new ShippingInfo();
20      $str_in_requestShipping_ShippingInfo_msg =
21          $app->request-> post('in_requestShipping_ShippingInfo_msg');
22      $json_in_requestShipping_ShippingInfo_msg =
23          json_decode($str_in_requestShipping_ShippingInfo_msg);
24      $in_in_requestShipping_ShippingInfo_msg =
25          $json_in_requestShipping_ShippingInfo_msg->in_requestShipping_ShippingInfo_msg;
26
27      //PUT HERE YOUR CODE
28
29      //Return results
30      $return_requestShipping_Schedule_response = new Schedule();
31      echo json_encode($return_requestShipping_Schedule_response);
32  });
33  $app->run();
```

Una vez que los esqueletos de la lógica correspondiente a los participantes descritos en el incremento han sido generados, los desarrolladores los incorporaron en el IDE de desarrollo seleccionado para cada tecnología de implementación, los completan, y compilan o empaquetan; generando los artefactos de despliegue resultantes de esta actividad.

## 6.4 Despliegue y reconfiguración dinámica

En esta actividad la reconfiguración dinámica se produce tanto al desplegar, re-

desplegar o replegar artefactos de despliegue que incluyen los servicios del incremento a ser integrado; así como, actualizar las dependencias entre servicios especificadas en archivos de configuración; ambos de acuerdo su impacto arquitectónico descrito durante la especificación de la integración. Los especialistas en operaciones ejecutan esta actividad de acuerdo a lo descrito en la sección 4.3.3 y utilizan componentes de la infraestructura software como soporte en los diferentes pasos de ésta actividad.

Al iniciar esta actividad el especialista en operaciones especifica los recursos cloud necesarios para satisfacer los requisitos de los servicios incluidos en los artefactos de despliegue correspondientes al Incremento-1, documentando estos recursos en el *Modelo de Recursos Cloud del Incremento*. Para esto, en primer lugar utilizaron el componente de la infraestructura software *Generación de Modelos de Recursos Cloud* (5.5.1.2), el cual generó parcialmente el *Modelo de Recursos Cloud del Incremento a* partir del *Modelo de Artefactos Cloud del Incremento*. El modelo generado incluye las descripciones de los artefactos de despliegue a desplegar y las descripciones de los servicios incluidos en cada artefacto de despliegue. En el siguiente paso el especialista en operaciones utilizó el *Editor de Recursos Cloud* (5.4.4) para, por cada artefacto de despliegue, especificar tanto los proveedores cloud como el modelo de servicios (p. ej., IaaS o PaaS) que adoptará para aprovisionar el entorno cloud en el cual se desplegará el artefacto de despliegue. El especialista en operaciones especificó también los recursos cloud que conformarán el entorno cloud; para ello, seleccionó de entre las ofertas de cada proveedor, ya sea instancias de entornos cloud predefinidas o especificó entornos personalizados mediante seleccionar los recursos cloud necesarios para satisfacer la demanda y naturaleza de trabajo especificados durante la especificación del incremento. Por ejemplo, la Figura 6-11 muestra un extracto del *Modelo de Recursos Cloud del Incremento*; en dónde el servicio de orquestación *Order With Shipping* será desplegado en el proveedor Microsoft Azure© con un modelo de servicio PaaS utilizando la instancia predefinida Small; el servicio cloud correspondiente al participante Shipper será desplegado en el en el proveedor Microsoft Azure© con un modelo de servicios IaaS cuya unidad de computo hospedará un sistema operativo Linux-Centos, con servidor de Aplicaciones, runtime php, etc.

La descripción de las características del entorno cloud en el que se desplegará el artefacto de despliegue *Shipper* fue copiada de una instancia predefinida o plantilla de entorno cloud, la misma que corresponden a una configuración del *Modelo de Configuraciones de Entornos Cloud* de la Figura 5-11. La Figura 6-12 muestra la instancia predefinida, la misma que fue copiada haciendo uso del componente de la infraestructura software *Aplicar plantilla de entorno cloud* (5.5.1.2).

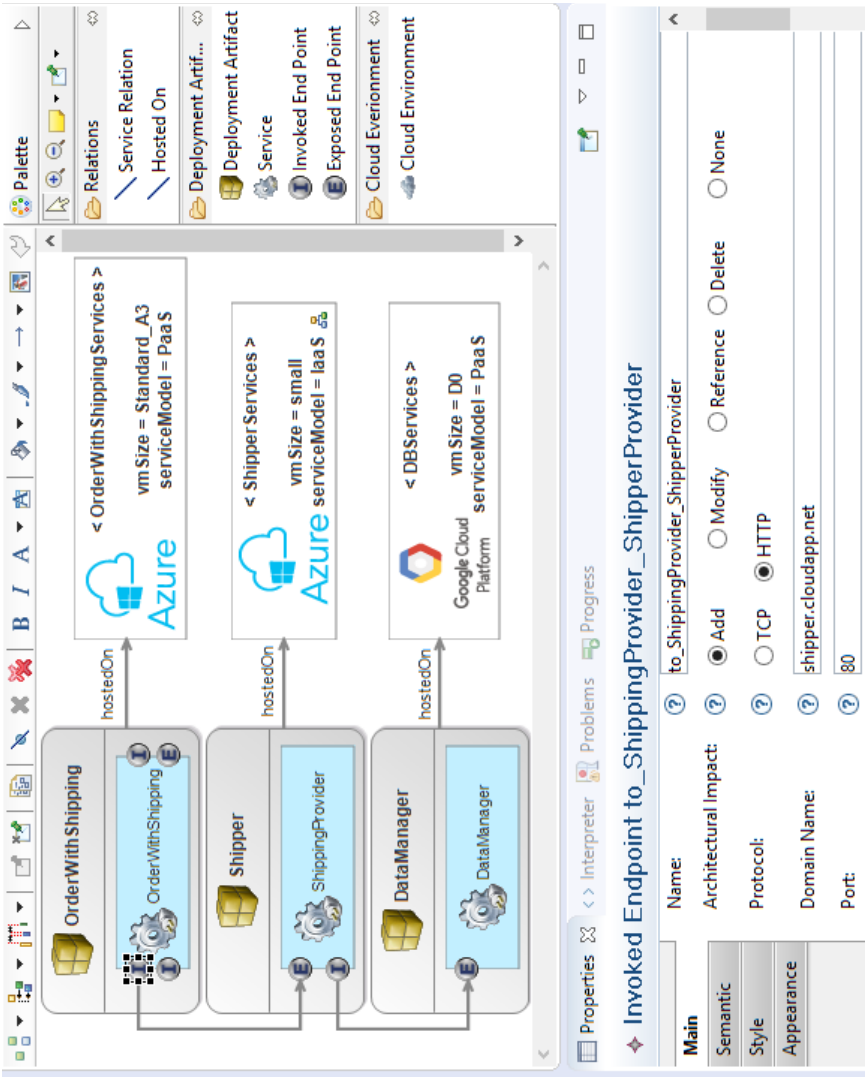



**Figura 6-11 *Modelo de Recursos cloud del Incremento* (Incremento-1) – Vista Topología de Alto Nivel**

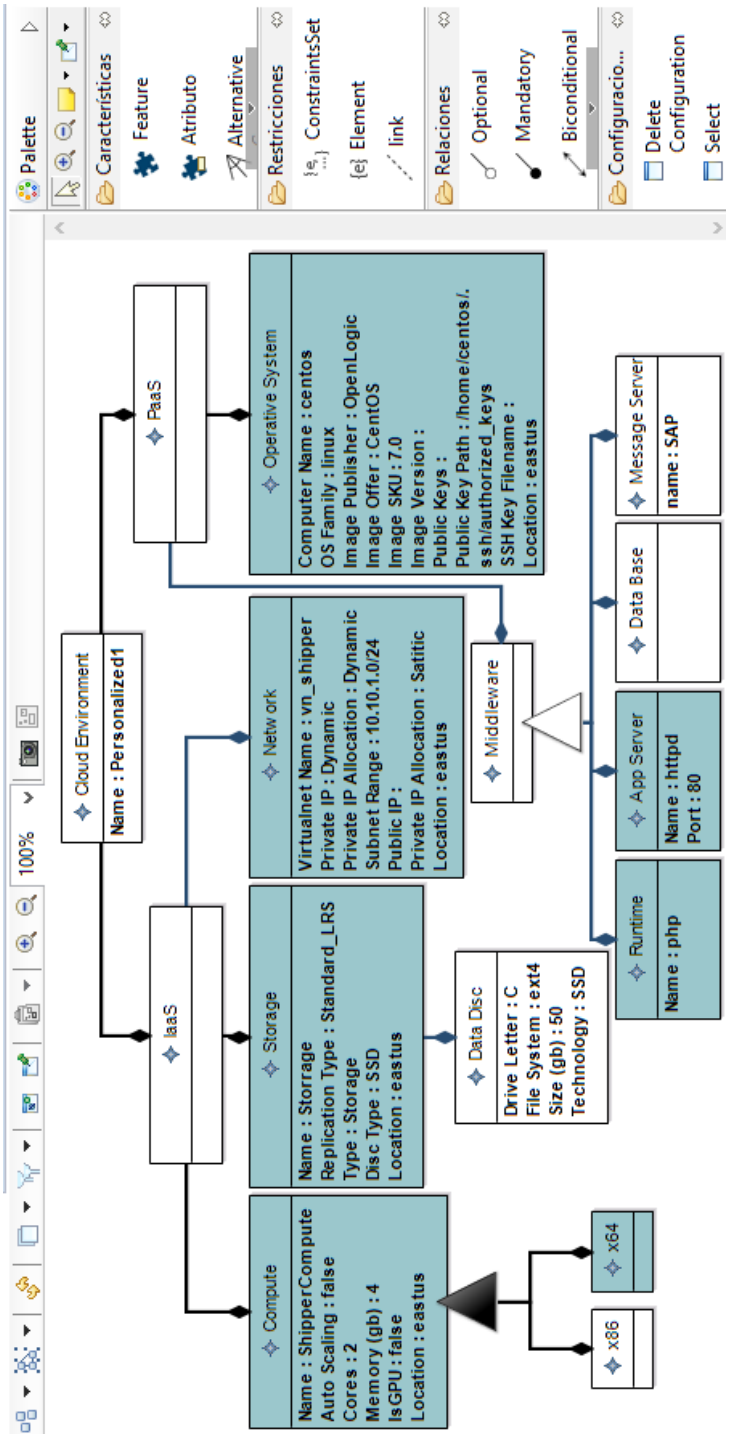



**Figura 6-12 Instancia de entorno cloud predefinida – Vista configuración del Editor de Configuraciones de Entornos Cloud**

La Figura 6-13 muestra la descripción del entorno de despliegue de *Shipper* luego de haber ejecutado la transformación *Aplicar plantilla de entorno cloud*, la cual copió la instancia predefinida descrita en la Figura 6-12.

Una vez que los entornos cloud en los que se desplegarán los artefactos de despliegue correspondientes al Incremento-1 han sido especificados, los especialistas en operaciones utilizan el componente de la arquitectura software *Generador de scripts de aprovisionamiento y despliegue* (5.5.2.3), obteniendo los scripts de aprovisionamiento y scripts de despliegue específicos de un proveedor cloud y modelo de servicios seleccionado. Por ejemplo, el Listado 6-3 muestra el script de aprovisionamiento (archivo blueprint.yaml) del artefacto de despliegue de *Shipper*, el cual ha sido generado conforme a los requisitos de Cloudify, facilitando la automatización del aprovisionamiento en el proveedor Microsoft Azure© con el modelo de servicio IaaS.

**Listado 6-3 Script de aprovisionamiento artefacto de despliegue Shipper**

```
tosca_definitions_version: cloudify_dsl_1_3

inputs:
  virtual machine public key data:
  virtual machine public key path:
  virtual machine public keys:

node_templates:

  # ****** Creating resourcegroup ******
  resource group cfyproject:
  # ****** Creating network ******
  virtual network cfyproject:
  network security group cfyproject:
  public subnet cfyproject:
  public ip address cfyproject:
  public ip configuration cfyproject:
  public nic cfyproject:
  # ****** Creating storage account ******
  storage account cfyproject:
  # ****** Creating virtual machine ******
  virtual_machine_cfyproject:
    type: cloudify.azure.nodes.compute.VirtualMachine
    properties:
      name: 'vmcfyproject'
      location: 'eastus'
      use_external_resource: false
      azure_config: { get_input: azure_configuration }
      os_family: 'linux'
      resource config:
      agent config:
    relationships:
    - type: cloudify.azure.relationships.contained in resource group
    - type: cloudify.azure.relationships.connected to storage account
    - type: cloudify.azure.relationships.connected to nic

  #Creating app servers
  app server httpd:
  #Creating runtimes
  runtime php:
```

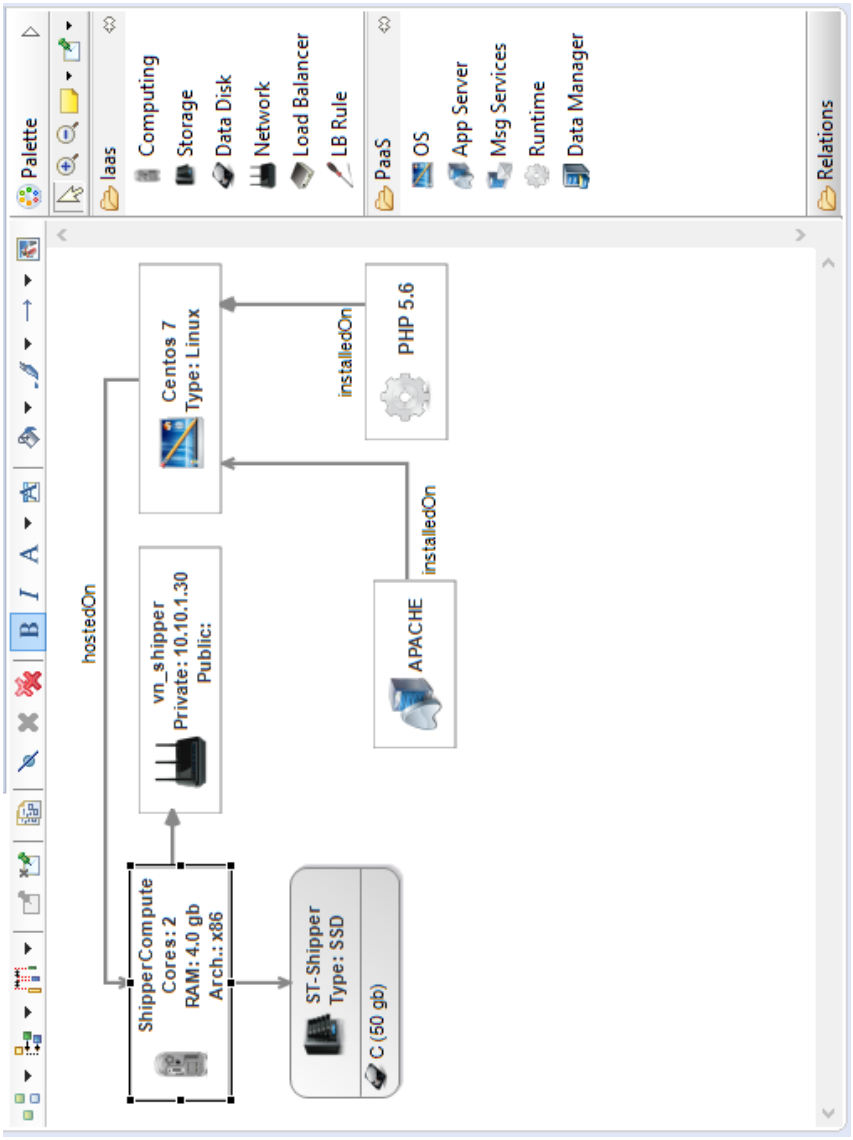



**Figura 6-13** ***Modelo de Recursos cloud del Incremento*** **(Incremento-1) – Vista Descripción de Entorno Cloud**

Las líneas cuyo texto están en color verde en el Listado 6-3 indican el inicio del bloque de instrucciones de aprovisionamiento de los diferentes recursos de infraestructura (línea 32, red; línea 159, máquina virtual con sistema operativo Linux; línea 195, servidor de aplicaciones Apache; línea 249, runtime PHP; etc). Por otro lado, el Listado 6-4 muestra el script de despliegue (archivo blueprint.yaml) del artefacto de despliegue de *Shipper*; el cual ha sido generado conforme a los requisitos de Cloudify para el despliegue en Microsoft Azure©. Las líneas 20 - 22 muestran las instrucciones que obtienen el artefacto de despliegue de *Shipper*, lo descomprime, lo copia en el directorio de publicación de Apache, y finalmente una vez desplegado, elimina el artefacto de despliegue comprimido.

**Listado 6-4 Script de despliegue del artefacto de despliegue Shipper**

```
1  #*************************************************************
2  #                                                            *
3  #  Source code generated by the DIARy transformation engine  *
4  #                                                            *
5  #*************************************************************
6
7  tosca_definitions_version: cloudify_dsl_1_3
8
9  node_templates:
10
11   #Creacion del nodo para ejecutar el script de instalacion del servicio
12   service_Shipperec:
13     type: cloudify.nodes.WebServer
14     interfaces:
15       cloudify.interfaces.lifecycle:
16         create:
17           implementation: fabric.fabric_plugin.tasks.run_commands
18           inputs:
19             commands:
20             - wget https://.../s/69j4eh.../shipper.zip -P /var/www/html/
21             - unzip /var/www/html/shipper.zip -d /var/www/html/
22             - rm /var/www/html/shipper.zip
23             use_sudo: true
24             fabric_env:
25               user: 'centos'
26               key_filename: '~\\.ssh\\codifyPrivateKey_rsa'
27               host_string: {
28                 get_attribute: [virtual_machine_cfyproject, public_ip] }
29               always_use_pty: true
30     relationships:
31       - type: cloudify.relationships.contained_in
32         target: virtual_machine_cfyproject
33
34 outputs:
35   service_Shipperec:
36     description: End Point Shipperec
37     value:
38       service_Shipperec_endpoint: { concat: [{ get_attribute:
39       [virtual_machine_cfyproject, public_ip] }, '/shipper' ]}
```

Luego de haber ejecutado los scripts de aprovisionamiento y despliegue, lo cual

aprovisionó los entornos cloud en los proveedores cloud especificados para cada servicio y desplegó los servicios incluidos en el incremento, los especialistas en operaciones utilizaron el *Editor de recursos cloud* para actualizar información de los *Invoked End Point* del servicio del participante *Dealer* y el servicio de orquestación *Order With Shipping*. El *Invoked End Point* (to_sc_purchase) del servicio *Dealer*, quien cumple el rol de consumidor de servicio e inicia la interacción mediante invocar al servicio de orquestación, fue actualizado con el valor del *Exposed End Point* del nuevo servicio de orquestación (*Order With Shipping*), mientras que los *Invoked End Point* del servicio de orquestación *Order With Shipping* fueron actualizados con los valores de los *Exposed End Points* de los servicios que participan en la interacción (Dealer, Shipper y Manufacturer).

Una vez que el *Modelo de Recursos Cloud del Incremento* ha sido actualizado, los especialistas en operaciones actualizaron los archivos de configuración de los servicios con la nueva información de puntos de acceso. Para lo cual, utilizaron el componente de la infraestructura software *Generador de scripts de reconfiguración* (5.5.2.4), el cual generó los scripts de reconfiguración: i) Un script de reconfiguración correspondiente al participante *Dealer* que actualiza su archivo de configuración con la dirección del punto de acceso al servicio de orquestación (*Invoked End Point* to_sc_purchase). ii) Un script de reconfiguración correspondiente al servicio de orquestación *Order With Shipping* que actualiza la información de sus puntos de acceso a los servicios involucrados en la interacción (Dealer, Manufacturer, Shipper). El servicio *Dealer* y el servicio de orquestación *Oder With Shipping* son implementados como servicios WCF por lo tanto los scripts de reconfiguración fueron generados como archivos XDT, archivos utilizados para modificar los archivos de configuración (ServiceConfiguration.Cloud.cscfg) de servicios WCF en tiempo de ejecución.

El Listado 6-5 muestra el script de reconfiguración correspondiente al participante *Dealer*; en donde, la línea 8 indica la acción a realizar (Replace) sobre la entrada de la línea 6 (to_sc_purchase) del archivo de configuración. La acción *Replace* tiene concordancia con el impacto arquitectónico especificado para el enlace entre el contrato de servicio *Order With Shipping* y el participante *Dealer* durante la especificación de la integración. La línea 7 muestra el nuevo valor de la entrada to_sc_purchase el cual corresponde al punto de acceso del servicio de orquestación, reemplazando al punto de acceso del servicio de orquestación *Place Order* que será eliminado (ver Figura 6-7).

Una vez que los scripts de reconfiguración fueron generados, los especialistas en operaciones los ejecutaron, actualizando las entradas en los archivos de configuración del participante *Dealer* y del servicio de orquestación *Order With Shipping*. Luego desplegaron los archivos de configuración actualizados produciéndose la

reconfiguración dinámica al actualizar los enlaces entre servicios en tiempo de ejecución.

**Listado 6-5 Script de reconfiguración dinámica - XDT**

```
<ServiceConfiguration serviceName="Dealer"
xmlns:xdt="http://schemas.microsoft.com/
XML-Document-Transform">
<Role name="DealerServices">
    <ConfigurationSettings>
    <Setting name="to_sc_purchase"
    value="http://../OrderWithShipping.xamlx"
    xdt:Transform="Replace"
    xdt:Locator="Match(name)" />
    </ConfigurationSettings>
    </Role>
</ServiceConfiguration>
```

Finalmente, los especialistas en operaciones ejecutaron transformaciones M2M que actualizaron los modelos de la aplicación actual. Esto es, actualizaron el *Modelo de Arquitectura de la Aplicación* actual y el *Modelo de Artefactos Cloud de la Aplicación*. La actualización se realiza conforme al impacto arquitectónico especificado para cada elemento arquitectónico del *Modelo Extendido de la Arquitectura del Incremento*. Por ejemplo, los elementos arquitectónicos con impacto arquitectónico Add serán incluidos en el *Modelo de Arquitectura de la Aplicación* actual y las descripciones de los artefactos que implementarán esos elementos arquitectónicos serán incluidas en el *Modelo de Artefactos Cloud de la Aplicación*. Por otro lado, los elementos arquitectónicos con impacto arquitectónico Delete serán eliminados del *Modelo de Arquitectura de la Aplicación* actual y las descripciones de los artefactos que implementarán esos elementos arquitectónicos serán eliminados en el *Modelo de Artefactos Cloud de la Aplicación*. La Figura 6-14 muestra el *Modelo de Arquitectura de la Aplicación* luego de la integración del Incremento-1

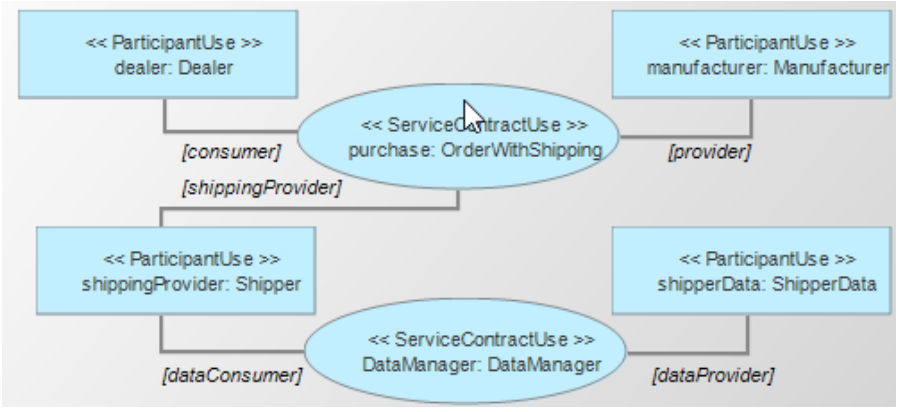



**Figura 6-14 Modelo de Arquitectura de la Aplicación - Luego de la Integración**

## 6.5 Conclusiones

En este capítulo se ha desarrollado un escenario de desarrollo incremental e integración que fue utilizado como ejemplo para ilustrar la aplicabilidad del método DIARy como soporte a la reconfiguración dinámica de arquitecturas de aplicaciones cloud producida por la integración de nuevos incrementos.

En primer lugar, se describió el escenario de ejemplo, el cual contiene la especificación de los requisitos funcionales de un incremento de software a ser integrado. El escenario de ejemplo incluyó también el modelo de la arquitectura actual de la aplicación que evolucionará luego de la integración, y el modelo de la arquitectura del incremento a ser integrado. Este capítulo mostró como el método DIARy, su proceso e infraestructura software, soportaron la reconfiguración dinámica durante las actividades propuestas por el método. Se mostró como la actividad *Especificación de la Integración del Incremento* y el *Editor del ADL para la Integración Incremental* facilitaron al arquitecto el razonamiento sistemático acerca de: el impacto de la integración, la demanda esperada de los servicios del incremento, y la naturaleza de trabajo de los servicios incluidos en el incremento.

La actividad *Implementación del incremento*, y el *Editor de Artefactos Cloud*, conjuntamente con el *Modelo de Artefactos Cloud del Incremento* y el *Motor de Transformaciones* ayudaron a los desarrolladores a aplicar principios y patrones arquitectónicos que facilitan la reconfiguración dinámica. En esta segunda actividad los desarrolladores tomaron decisiones tanto acerca de la estructura de empaquetado de los servicios, así como decisiones acerca de la tecnología de implementación. Luego obtuvieron automáticamente la implementación del servicio de orquestación que gestionará la integración e interacción de los servicios del incremento con los servicios de la aplicación actual. Adicionalmente obtuvieron esqueletos de la implementación de las interfaces que los participantes (otros servicios o aplicaciones) deben implementar para cumplir su rol en la interacción. Estos esqueletos integraron la lógica de negocio del participante, formando parte del servicio que ofrecen.

Luego, en la actividad *Despliegue y Reconfiguración Dinámica*, el *Editor de Recursos Cloud* conjuntamente con el *Modelo de Recursos Cloud del Incremento,* el *Modelo de Configuraciones de Entornos Cloud* y el *Motor de Transformaciones* soportaron a los especialistas en operaciones en la toma de decisiones de aprovisionamiento. Estas decisiones consisten de la especificación de los recursos cloud y características de los recursos cloud que soportarán la naturaleza de trabajo de los servicios incluidos en el incremento y satisfarán términos SLA, las cuales fueron documentadas en el Modelo *de Recursos Cloud del Incremento.* Luego los especialistas en operaciones utilizaron el *Motor de Transformaciones* para obtener automáticamente

scripts de aprovisionamiento, despliegue y reconfiguración dinámica de acuerdo al proveedor cloud seleccionado, modelo de servicio (p. ej., IaaS o PaaS), características de recursos cloud e impacto arquitectónico descrito durante la especificación del incremento.

La reconfiguración dinámica se produce debido al despliegue de los servicios incluidos en un incremento de software, el repliegue de los servicios de la aplicación actual que serán sustituidos o eliminados por la integración del incremento, y la actualización de enlaces entre servicios. En este contexto, para lograr esta reconfiguración dinámica los especialistas en operaciones ejecutaron los scripts de aprovisionamiento y despliegue, para luego ejecutar los scripts de reconfiguración dinámica que actualizaron los enlaces entre servicios sin interrumpir la ejecución de la aplicación cloud. Finalmente, transformaciones de modelo-a-modelo permitieron actualizar la arquitectura de la aplicación actual con los elementos de la arquitectura del incremento.

# Capítulo 7. Validación empírica del método DIARy

En este capítulo se presenta la validación empírica realizada sobre el método DIARy mediante un cuasi-experimento en el que se analiza la efectividad, eficiencia, facilidad de uso percibida, utilidad percibida e intención de uso. La estructura del capítulo es la siguiente:

La sección 7.1 presenta la introducción a esta validación empírica.

La sección 7.2 presenta la planificación del experimento.

La sección 7.3 presenta la ejecución del experimento.

La sección 7.4 se presentan los resultados del experimento.

La sección 7.5 se realiza un análisis pormenorizado de las amenazas a la validez.

La sección 7.6 presenta las conclusiones de este capítulo.

## 7.1 Introducción

Esta sección presenta un cuasi-experimento que tiene el objetivo de estudiar el método DIARy como soporte para la reconfiguración dinámica de arquitecturas de aplicaciones cloud producida por la integración de incrementos en una aplicación cloud. Este estudio propone también un marco de trabajo bien definido que puede ser reutilizado por otros investigadores para la validación empírica de sus procesos de reconfiguración dinámica. Un cuasi-experimento es un estudio empírico dirigido a probar hipótesis causales descriptivas a cerca de causas manipulables en el cual los sujetos no son asignados aleatoriamente (Shadish, Cook, y Campbell 2002). Una de las razones para seleccionar un cuasi-experimento para la validación del método DIARy es la ausencia de métodos ampliamente aceptados e integrados que soporten la reconfiguración dinámica de arquitecturas de servicios cloud producida por la integración de servicios (reconfiguración debido a cambios adaptativos). Este hecho incrementa la dificultad de realizar experimentos controlados que comparen el método DIARy con métodos similares.

El cuasi-experimento fue diseñado siguiendo las guías propuestas por Wohlin y otros (2012); por lo tanto, los pasos realizados han sido: i) Planificación del experimento, ii) Preparación y ejecución del experimento, iii) Análisis de datos, y iv) Amenazas a la validez.

## 7.2 Planificación del experimento

### 7.2.1 Objetivo del experimento

En primer lugar, se define el objetivo del cuasi-experimento, el cual según el paradigma Goal-Question-Metric (GQM) (Basili y otros 1994), es: *analizar* el método DIARy *con el propósito de* evaluar *respecto a* su facilidad de uso, utilidad percibida e intensión de uso *desde el punto de vista* de un conjunto de arquitectos de software noveles.

Las preguntas de investigación abordadas por el experimento fueron:

- RQ1: ¿Es el método DIARy percibido tanto fácil de usar como útil en la reconfiguración arquitectónica de aplicaciones cloud?
- RQ2: ¿Existe la intención de usar el método DIARy en el futuro?

### 7.2.2 Definición del contexto

El contexto del experimento está determinado por: i) la aplicación en la cual los servicios cloud serán incrementalmente integrados, y cuya arquitectura será reconfigurada dinámicamente; ii) las actividades del método DIARy que serán evaluadas; y iii) la selección de sujetos. Estos elementos son descritos en las siguientes secciones.

#### *7.2.2.1 Aplicación cloud*

El experimento hace frente a problemas reales (p. ej., gestionar el impacto arquitectónico de la integración de servicios) presentes en el desarrollo incremental de aplicaciones cloud. Proponemos un escenario de integración de complejidad media, en donde la funcionalidad de un Sistema de Reservas, que permite a agencias de viajes gestionar las reservas de sus clientes, es complementado incrementalmente a través de integrar nuevos servicios cloud, reconfigurando la arquitectura actual de la aplicación. La Figura B-8-1 del Apéndice B.2.1 muestra el *Modelo de Arquitectura de la Aplicación* correspondiente a la versión inicial del Sistema de Reservas utilizado en el experimento.

El Sistema de Reservas está compuesto por 5 contratos de servicio (*Service Contract*), 4 participantes (*Participant Use*) que trabajan proveyendo y consumiendo servicios, y 10 dependencias (*Role Binding*) que definen el rol que cada participante cumple en la interacción. A pesar de que el *Modelo de Arquitectura de la Aplicación* incluye el modelado de sus partes interiores (p. ej., interfaces que definen el tipo de un rol, protocolo de interacción, tipos de mensajes transmitidos entre los participantes), los sujetos del experimento requieren únicamente el protocolo de interacción del servicio que será afectado por la integración (servicio Reserva) para completar las tareas del experimento; por lo tanto, es el único incluido en los materiales del experimento, ver Figura B-8-2 del Apéndice B.2.1.

De acuerdo al escenario de integración propuesto en el experimento, la integración del incremento *Increment – 1* en la aplicación cloud actual cambiará su arquitectura: incorporando un nuevo participante, modificando la lógica de negocio de un participante existente, y reemplazando el protocolo de interacción de un contrato de servicio en el cual el nuevo participante está involucrado.

Para la creación de los modelos utilizados en el experimento se creó un perfil UML de acuerdo a la descripción presentada en la Figura 4-1. La Figura B-8-3 del Apéndice B.2.1 muestra el *Modelo de Arquitectura del Incremento* correspondiente al incremento *Increment-1*, el cual ha sido modelado haciendo uso del perfil antes mencionado. La arquitectura del incremento está conformada por 1 elemento *Service Contract*, 3 elementos *Participant Use*, y 3 elementos *Role Binding*. El *Modelo*

*de Arquitectura del Incremento* de la Figura B-8-3 es utilizado como hoja de respuestas en el cuasi-experimento. En donde, los sujetos tienen que razonar acerca del impacto arquitectónico de la integración de cada uno de los elementos arquitectónicos del incremento de software *Increment-1*, para luego llenar los espacios provistos para las respuestas con las decisiones de diseño que han tomado. Adicionalmente, los sujetos proveen respuestas para el nombre de la instancia del elemento arquitectónico y requisitos de aprovisionamiento. En cuanto a los requisitos de aprovisionamiento, por razones de simplicidad, se utilizaron los valores *delayLevel* y *elasticityLevel* para su especificación.

Al igual que el *Modelo de Arquitectura Actual*, la hoja de respuestas del material experimental incluye el modelado del protocolo de interacción, en este caso, correspondiente al contrato de servicio incluido en la arquitectura del incremento, ver la Figura B-8-4 del Apéndice B.2.1.

Con el propósito de soportar el escenario de integración propuesto y hacer posible verificar la reconfiguración arquitectónica producida por la integración proveemos una versión ejecutable del Sistema de Reservas cuyo comportamiento cambiará luego de la integración del incremento *Increment-1*. Esta versión inicial fue desarrollada siguiendo parcialmente el método DIARy, sin tener en cuenta las actividades en las cuales se requiere una versión actual de la arquitectura de la aplicación (p. ej., verificar la compatibilidad entre arquitecturas). Adicionalmente, durante la actividad *Especificación de la integración del incremento*, el impacto arquitectónico de los elementos arquitectónicos de la arquitectura del incremento fue especificado con el valor *Add*. Los servicios cloud que conforman el Sistema de Reservas fueron desplegados en el proveedor cloud Microsoft Azure© haciendo uso de un modelo de servicios PaaS; por lo tanto, los artefactos cloud fueron generados de acuerdo al mapeo entre elementos arquitectónicos y artefactos cloud descrito en las tablas Tabla 5-2 y Tabla 5-3. El Sistema de Reservas fue inspirado y adaptado del caso de negocio "Voyage travel agency" del tutorial "Service Modeling with SoaML" presentado en ECMFA'10.

#### *7.2.2.2 Actividades del método DIARy a ser evaluadas*

A pesar de que el método DIARy provee soporte a la reconfiguración dinámica de arquitecturas durante la especificación, implementación, y despliegue de servicios cloud; la reconfiguración arquitectónica toma lugar sustituyendo los enlaces entre servicios y gestionando las instancias de servicios mientras la aplicación cloud está en ejecución. Por lo tanto, incluimos como parte de la validación únicamente los pasos de las actividades del método DIARy necesarios para obtener los scripts de reconfiguración dinámica. Es decir, los sujetos del cuasi-experimento trabajan con el escenario de integración descrito anteriormente realizando

tareas que soportan tanto la *Especificación de la integración del incremento* y la *Generación de scripts de reconfiguración* necesarios para reconfigurar en tiempo de ejecución la arquitectura del Sistema de Reservas.

#### *7.2.2.3 Selección de sujetos*

A pesar de que evaluadores expertos son capaces de detectar más problemas que los evaluadores noveles (Hertzum y Jacobsen, 2001), nos enfocamos en el último grupo de perfil de evaluadores debido a que la intención es proveer un método de reconfiguración adecuado para ayudar a arquitectos de software sin mucha experiencia mientras realizan actividades de reconfiguración dinámica.

El experimento fue ejecutado en un contexto en línea en un entorno académico con sujetos seleccionados de dos grupos de investigación de la Universitat Politècnica de València, quienes han tomado asignaturas de modelado o desarrollo de servicios Web, y cuya participación fue voluntaria.

La Tabla 7-1 muestra un extracto del perfil de los participantes, en donde los sujetos en PERF1, PERF 2 y PERF 3 poseen un grado en Ciencias de la Computación. A pesar de que los sujetos fueron estudiantes cuando se ejecutó el cuasi-experimento, bajo ciertas condiciones, no existe gran diferencia entre estudiantes de últimos años y profesionales junior (Basili, Shull, y Lanubile 1999), (Höst, Regnell, y Wohlin 2000), y podrían ser considerados como la próxima generación de profesionales (Kitchenham y otros 2002). Por lo tanto, creemos que su habilidad para entender modelos arquitectónicos, principios de desarrollo de servicios, y evaluar el proceso de reconfiguración dinámica puede ser comparable a aquellos de profesionales junior.

**Tabla 7-1 Perfil de sujetos**

| Perfil | Descripción |
|---|---|
| PERF1 | Estudiantes de PhD, todos ellos trabajando en tópicos de investigación relacionados a varias áreas de la ingeniería del software |
| PERF 2 | Estudiantes de PhD, todos ellos trabajando en tópicos de investigación relacionados a varias áreas de las ciencias de la computación |
| PERF 3 | Estudiantes de maestría, todos ellos trabajando en tópicos de investigación relacionados a varias áreas de la ingeniería del software |
| PERF 4 | Estudiantes de pregrado, , todos ellos trabajando en tópicos de investigación relacionados a varias áreas de la ingeniería del software |

### 7.2.3 Selección de variables

#### *7.2.3.1 Variables independientes*

Debido a que en la actualidad no existe un proceso de reconfiguración dinámica de arquitecturas producida por la integración de servicios, no es posible utilizar un método de control para evaluar el método DIARy. Por lo tanto, la variable independiente tiene un único valor en una escala nominal, la cual es el método DIARy.

#### *7.2.3.2 Variables dependientes*

Con respecto a las variables dependientes, variables subjetivas como la aceptación del usuario se han convertido en un tema importante en el contexto de los sistemas de información (Markus y otros, 1994), (Gaynor, 1996). Por lo tanto, basamos la definición de las variables subjetivas de interés en los constructores de un modelo teórico para el análisis de la aceptación y adopción de tecnologías de información emergentes: el *Modelo de Aceptación Tecnológica* (Technology Acceptance Model – TAM) propuesto por Davis (1989). TAM ha recibido soporte empírico extensivo a través de validaciones y replicas (Venkatesh, 2000), y comparado con otros modelos tiene ventajas en la especificidad de tecnologías de la información, bases teóricas fuertes y soporte empírico (Hu y otros, 1999); sus principales constructores son las siguientes variables:

- *Facilidad de Uso Percibida* (FUP): se refiere al grado en el que los participantes creen que el aprendizaje y el uso de un método en particular podrá llevarse a cabo con un esfuerzo mínimo.
- *Utilidad Percibida (UP):* se refiere al grado en el que los participantes consideran que el uso de un método especifico aumentará su rendimiento dentro de un contexto organizacional.

*Intención de Uso (IU)*: se refiere a la medida en que un determinado participante tiene la intención de utilizar, en el futuro, un método particular. Esta última variable representa un juicio perceptual de la eficacia del método, es decir, si realmente es rentable. Esta variable es comúnmente empleada para predecir la probabilidad de aceptación de un determinado método en la práctica.

Con respecto a los instrumentos empleados para medir estas variables, apoyamos en un instrumento existente basado en TAM, el Método para la Evaluación de Modelos (Method Evaluation Model – MEM) (Moody, 2001), y lo adaptamos (básicamente reformulando declaraciones) para el uso en el contexto de un proceso de reconfiguración dinámica de arquitecturas. La Figura 7-1 muestra como

MEM ha sido adaptado en este trabajo de tesis de acuerdo a la Tabla 7-2, a través de definir un conjunto de preguntas por cada variable subjetiva.

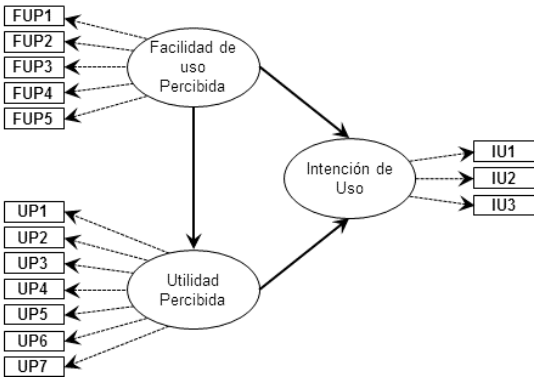



**Figura 7-1 Adaptación de MEM**

Estas tres variables subjetivas se midieron mediante un cuestionario Likert con un conjunto de 15 preguntas cerradas definidas en base a los elementos de la Tabla 7-2: 5 preguntas fueron correspondientes a la variable FUP, 7 a la variable UP y 3 a la variable IU. Las preguntas se formularon con el formato de preguntas con respuestas opuestas, y las respuestas fueron codificadas empleando una escala Likert (Likert, 1932) de 5 puntos. Cada pregunta contiene dos estados opuestos que representan los valores máximo y mínimo (5 y 1), considerándose el valor 3 como la percepción neutra. El participante selecciona el valor en la escala que más se adecua a su percepción u opinión. El valor agregado de cada una de las variables se calcula como la media aritmética de las respuestas a las preguntas asociadas a cada una de las variables subjetivas. El orden de las preguntas fue alterado de forma aleatoria con el fin de evitar el sesgo de respuestas sistemáticas; además, se reformularon algunas preguntas para convertirlas en enunciados negativos ubicados ya sea en la parte izquierda o derecha con el fin de evitar así las respuestas monótonas (Hu y Chan 1999).

El cuestionario de evaluación de las variables subjetivas (ver Apéndice B.3) también incluyó tres preguntas abiertas a fin de obtener retroalimentación de los sujetos del experimento, estas fueron: sugerencias pare mejorar el método DIARy, ii) razones por las cuales adoptarían el método DIARy en el futuro, y iii) comentarios acerca del método DIARy.

Estamos además interesados en evaluar cuán entendible es el método DIARy: para ello, estudios empíricos sugieren que las tareas de solución de problemas pueden ser utilizadas como instrumento para medir la entendibilidad (p. ej., (Gemino y Wand, 2005), (Bodart y otros, 2001)). Para responder preguntas de

solución de problemas el sujeto requiere razonar a cerca del dominio. Por lo tanto, se utilizaron dos variables basadas en el rendimiento:

- *Efectividad*: número de cambios arquitectónicos correctamente especificados (entendiendo por correctamente especificados aquellos cambios que son acordes con los requisitos descritos en el boletín del experimento).

**Tabla 7-2 Elementos de la encuesta para medir las variables subjetivas**

| Elemento | Descripción |
|---|---|
| FUP1 | El método para la reconfiguración dinámica e incremental de arquitecturas de servicios cloud me ha parecido simple y fácil de seguir. |
| FUP2 | De manera general, pienso que el método de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es fácil de entender. |
| FUP3 | Los pasos a seguir para especificar los cambios arquitectónicos que producirá la integración de nuevos servicios y reconfigurar la arquitectura de servicios cloud actual son claros y fáciles de entender. |
| FUP4 | El método de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es fácil de aprender. |
| FUP5 | Fue fácil para mí entender cómo integrar las arquitecturas cloud utilizando este método. |
| UP1 | Creo que este método reduciría el tiempo y el esfuerzo requerido para integrar y reconfigurar dinámicamente arquitecturas de servicios cloud |
| UP2 | De manera general, considero que el método de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es útil. |
| UP3 | Creo que los estereotipos propuestos en este método para marcar los distintos elementos arquitectónicos son útiles para especificar la integración de servicios cloud. |
| UP4 | Creo que utilizar un Modelo de la Arquitectura del Incremento independiente del Modelo de la Arquitectura de la Aplicación facilita la identificación de los cambios arquitectónicos producidos por la integración de nuevos servicios. |
| UP5 | Creo que la generación automática de los scripts de reconfiguración facilita la reconfiguración dinámica de arquitecturas de servicios cloud producida por la integración de nuevos servicios |
| UP6 | El uso de este método mejoraría mi rendimiento en las actividades relacionadas con la reconfiguración dinámica de arquitecturas de servicios cloud. |
| UP7 | De manera general, pienso que este método proporciona una manera eficaz para integrar y reconfigurar arquitecturas de servicios cloud. |
| IU1 | En caso de necesitar soporte para las actividades de reconfiguración dinámica de arquitecturas de servicios cloud, tendría la intención de utilizar este método en el futuro |
| IU2 | Pienso que sería fácil ser hábil usando este método. |
| IU3 | Sí recomendaría el uso de este método de reconfiguración dinámica e incremental de arquitecturas de servicios cloud. |

- *Eficiencia*: calculada como la ratio entre el número de cambios arquitectónicos correctamente especificados dividido por el tiempo total empleado en su especificación.

### 7.2.4 Hipótesis

Se formularon tres hipótesis a constatar, en donde de acuerdo a nuestra operacionalización de TAM, la probabilidad de aceptación de un método de reconfiguración dinámica de arquitecturas que soporte la integración de nuevos servicios se puede predecir constatando las siguientes hipótesis:

- $H1_0$: El método DIARy se percibe como difícil de utilizar. $H1_1 = \neg H1_0$.
- $H2_0$: El método DIARy no es percibido como útil. $H2_1 = \neg H2_0$.
- $H3_0$: No existe intención de utilizar el método DIARy en el futuro. $H3_1 = \neg H3_0$.

### 7.2.5 Otros factores a ser controlados

Otros factores, también denominados cofactores, pueden influir en los resultados. En particular, la experiencia de los participantes en tópicos relacionados a modelado de software y desarrollo de servicios Web pueden tener un efecto indeseable en la percepción de los participantes durante la aplicación del método de reconfiguración dinámica, y este efecto puede confundirse con el efecto del método. Se definieron tres factores con el fin de analizar la influencia que puede tener la experiencia del participante en la percepción del método, estos factores fueron: i) experiencia en modelado (*ExpModeling*); ii) experiencia en desarrollo de servicios Web (*ExpServ*); y iii) experiencia utilizando UML (*YearsUML*).

El cuestionario que se muestra en el Apéndice B.1 fue diseñado para recoger información de la experiencia de los sujetos del experimento. *ExpModeling* y *ExpServ* fueron recogidas a través de preguntas cuyas respuestas fueron codificadas empleando una escala Likert de 5 puntos; en donde los posibles valores fueron: 1 si es que el conocimiento del sujeto es únicamente teórico; 2 si es que el sujeto necesita ayuda para aplicar principios básicos de los factores; 3 si el sujeto no necesita ayuda para aplicar principios básicos de los factores; 4 si el sujeto necesita ayuda para aplicar principios avanzados de los factores; y 5 si el sujeto no necesita ayuda para aplicar principios avanzados de los factores. *ExpModeling* y *ExpServ* son factores SI/NO, calculados como la media aritmética de las respuestas a las preguntas asociadas a cada factor. Consideramos que el sujeto tiene experiencia si el valor calculado de la media aritmética de su experiencia es mayor o igual a 2.5. Con respecto a *YearsUML*, este factor fue medido haciendo uso de

una pregunta abierta a la cual los sujetos responden indicando el número de años que han trabajado con UML. Utilizamos el valor 3 como umbral para este factor; por lo tanto, el factor *YearsUML* fue analizado haciendo uso de las expresiones booleanas *YearsUML*< 3 y *YearsUML*>=3.

## 7.3 Preparación y ejecución del experimento

El experimento fue planificado para ser realizado en dos sesiones. En el primer día, una sesión de entrenamiento de 150 minutos fue realizada antes de la sesión experimental. El objetivo de la sesión de entrenamiento fue presentar los tópicos descritos en la Tabla 7-3. La ejecución del experimento tomo lugar unos días después en la sesión experimental. La sesión experimental tuvo un tiempo esperado de duración de 45 minutos; sin embargo, se permitió que los sujetos finalizarán el experimento incluso luego de los 45 minutos con el fin de mitigar el posible efecto techo (Sjøberg y otros, 2003). La sesión experimental consistió en dos tareas, cuyo detalle es resumido en la Tabla 7-3.

**Tabla 7-3 Calendario del experimento**

| Sesión | Tarea |
|---|---|
| Sesión de Entrenamiento (150 min) | Conceptos de arquitecturas de software, características de aplicaciones cloud y métodos de reconfiguración dinámica. |
| | Notaciones empleadas para describir arquitecturas de aplicaciones basadas en servicios y SoaML. |
| | Visión general del método DIARy. |
| | Entrenamiento en las actividades del método DIARy relacionadas en la reconfiguración dinámica en tiempo de ejecución: Especificación de la integración del incremento y Reconfiguración dinámica. |
| Sesión Experimental (45 min) | Cuestionario Demográfico y de Experiencia |
| | Tarea 1: Especificación de la integración del incremento. |
| | Tarea 1.1: Identificar los elementos arquitectónicos de la arquitectura actual de la aplicación que serán afectados por la integración. |
| | Tarea 1.2: Especificar los cambios arquitectónicos que la integración producirá en la arquitectura actual de la aplicación. Por cada elemento del Modelo de la Arquitectura del Incremento los sujetos especifican como su integración cambiará el Modelo de Arquitectura de la Aplicación. |
| | Tarea 1.3: Especificar los requisitos de aprovisionamiento. Especificar como se espera que sean gestionados los cambios en la carga de trabajo de los servicios. |

| | |
|---|---|
| | Tarea 2: Reconfiguración dinámica |
| | Tarea 2.1: Actualizar modelos con información de los puntos de acceso (End Points) utilizados por los servicios de orquestación para exponer sus operaciones. Los participantes acceden al portal de gestión de Microsoft Azure© [11] para obtener información de despliegue del nuevo servicio de orquestación del Sistema de Reservas; luego actualizan los modelos con información de puntos de acceso al servicio de orquestación (Exposed EndPoints del servicio de orquestación). |
| | Tarea 2.2: Actualizar los modelos con información acerca de los puntos de acceso utilizados por los servicios para invocar operaciones de sus servicios relacionados. Los participantes acceden al portal de gestión de Microsoft Azure© para obtener información de despliegue del servicio que inicia la interacción (servicio que cumple el rol de consumidor) y actualizan los modelos con la información correspondiente (Invoked Endpoints). |
| | Tarea 2.3: Generar scripts de reconfiguración dinámica y, en base a las especificación de la integración del incremento (Tarea 1) e información de despliegue (Tarea 2.2 y Tarea 2.3), verificar su correcta generación. |
| | Tarea 2.4: Modificar la arquitectura de la aplicación cloud en el entorno cloud ejecutando el script de reconfiguración generado. Reconfigura la arquitectura del Sistema de Reservas, mientras se encuentra en ejecución, mediante la actualización de enlaces entre servicios. |
| | Cuestionario de evaluación |

Múltiples documentos fueron diseñados como instrumentación para el experimento (disponible para descargas en http://thediarymethod.azurewebsites.net). La documentación de las tareas experimentales incluyó: i) un boletín que contiene la descripción del escenario de integración del Sistema de Reservas y las tareas a ser realizadas por los sujetos del experimento, ver Apéndice B.2; ii) un anexo con el Modelo de Arquitectura de la Aplicación del Sistema de Reservas actual, ver Apéndice B.2.1; iii) un anexo con las Guías para la especificación de la integración, ver Apéndice A.1; iv) una Hoja de Respuestas que contiene el Modelo de Arquitectura del Incremento luego de haber aplicado el perfil UML (el *perfil de especificación DIARY*) construido para la validación del método, la cual debía ser completada por los sujetos durante la *Especificación de la integración del incremento* en la Tarea 1; ver Apéndice 0; y v) el prototipo experimental del editor

[11] https://manage.windowsazure.com/

del *Modelo de Artefactos Cloud del Incremento*, construido para la validación del método, el cual es utilizado por los sujetos para actualizar el *Modelo de Artefactos Cloud del Incremento* y generar los scripts de reconfiguración en la Tarea 2, ver Figura 7-2.

El material de entrenamiento consistió en un conjunto de diapositivas conteniendo la descripción del método DIARy y el uso del perfil de especificación DIARy. Antes de la ejecución del experimento realizamos un estudio piloto con un experto en arquitecturas de software y sus observaciones fueron tomadas en consideración para mejorar tanto el diseño del experimento, así como el material experimental.

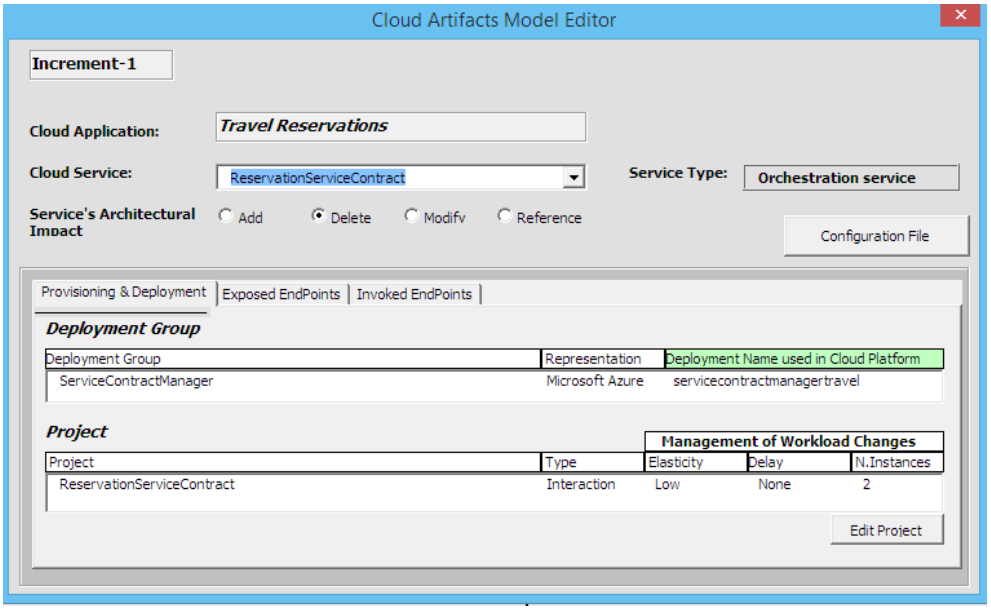



**Figura 7-2 Prototipo experimental del editor del *Modelo de Artefactos Cloud del Incremento***

El experimento se llevó a cabo con un grupo de 20 estudiantes en Ciencias de la Computación: 9 estudiantes de PhD con perfil PERF1, 6 estudiantes de PhD con perfil PERF2, 2 estudiantes de maestría con perfil PERF3, y 3 estudiantes de pregrado con perfil PERF4. La Figura 7-3 muestra la distribución de los sujetos del experimento de acuerdo a su perfil. El experimento se realizó en solo un aula, y no se permitió la interacción entre sujetos. Durante la sesión experimental los conductores del experimento respondieron a preguntas y clarificaron dudas de los sujetos.

Los datos del experimento fueron recogidos durante la ejecución de las actividades del método DIARy. El Boletín del experimento y la Hoja de Respuestas

fueron utilizados en la adquisición de datos durante la Tarea 1; mientras que el Boletín del experimento, el prototipo experimental del editor del *Modelo de Recursos Cloud del Incremento*, y la plataforma Microsoft Azure© fueron utilizados en la Tarea 2. Con el fin de evitar que los cambios en la arquitectura del Sistema de Reservas producidos por la ejecución de scripts de reconfiguración dinámica por parte de un sujeto afecten la realización de las tareas experimentales de otros sujetos, cada sujeto trabajó con instancias de servicios diferentes. Los sujetos utilizaron el boletín para registrar la hora de inicio y fin de cada tarea.

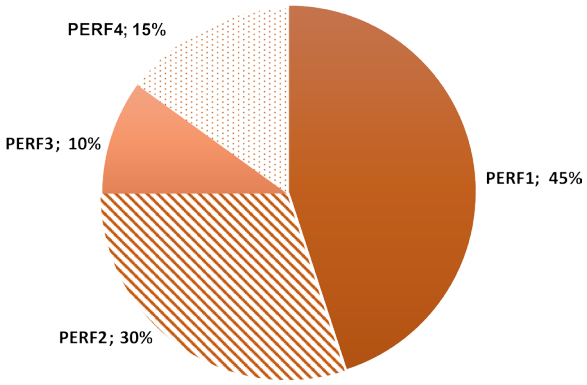



**Figura 7-3 Perfil de participantes**

## 7.4 Análisis de datos

El análisis de datos fue realizado utilizando SPSS 2.0 (IBM Corp. Released 2011. IBM SPSS Statistics for Windows, Version 20.0. Armonk, NY: IBM Corp.) con un $\alpha$=0.05. En este análisis utilizamos estadística descriptiva y pruebas estadísticas para analizar los datos recogidos.

Primero, realizamos un análisis de las respuestas al cuestionario (Apéndice B.1) previo al experimento concerniente a la experiencia de los sujetos (conocimiento y experiencia en modelado y desarrollo de servicios Web). Casi todos los sujetos (90% de los sujetos) tenían conocimientos acerca de UML y también conocimientos básicos acerca de modelado de arquitecturas basadas en servicios. Con respecto a los conocimientos en desarrollo de software mediante la aplicación de enfoques de desarrollo relacionados al cloud, 75% de los sujetos han desarrollado servicios Web y solo el 13% de ellos han desarrollado servicios Web para ser desplegados en entornos cloud. La Figura 7-4 muestra la distribución entre factores de la experiencia de los participantes.


**Figura 7-4 Experiencia sujetos**

### 7.4.1 Análisis cualitativo

Con el propósito de analizare el efecto que *ExpModeling*, *ExpServ*, y *YearsUML* tienen en las variables dependientes, realizamos la prueba no paramétrica de Mann-Whitney (Conover, 1998) ya que las tres variables (p. ej., *PEOU*, *PU* e *ITU*) son medidas utilizando la agregación de datos ordinales (p. ej., la media aritmética de varias escalas Likert). La Tabla 7-4 muestra los resultados de esta prueba, los cuales nos permiten verificar que los factores *ExpModeling*, *ExpServ*, y *YearsUML* no tienen efectos estadísticamente significativos en las variables subjetivas bajo estudio.

**Tabla 7-4 Efecto de *ExpModeling*, *ExpServ*, y *YearsUML* en variables cualitativas**

| | *Factor* | | |
|---|---|---|---|
| *Variable* | *ExpModeling* | *ExpServ* | *YearsUML* |
| *FUP* | 0.238 | 0.358 | 0.261 |
| *UP* | 0.624 | 0.440 | 0.131 |
| *IU* | 0.238 | 0.358 | 0.080 |

La Tabla 7-5 muestra un resumen de los resultados globales de las variables subjetivas por factor. Se han utilizado la media y la desviación de la varianza como estadísticos descriptivos para las variables subjetivas cualitativas *FUP*, *UP*, e *IU*. Se puede ver que todas las variables son en promedio mayores a 3, el valor neutral de la escala Likert. Esto significa que, bajo las condiciones del experimento, el método DIARy es percibido como fácil de usar, útil y los participantes muestran una cierta intención de emplearlo en el futuro. Se puede ver, además, que para los factores *ExpModeling* y *YearsUML* los valores de cada variable (*PEOU*, *PU* e *ITU*) son mejores para los sujetos con mayor experiencia; mientras que para el factor *ExpServ*, observamos mejores valores para los sujetos con menor experiencia. La Figura 7-5 muestra los diagramas de caja (boxplot) para las variables *FUP*, *UP* e *IU* para *ExpModeling*, *ExpServ*, y *YearsUML*.

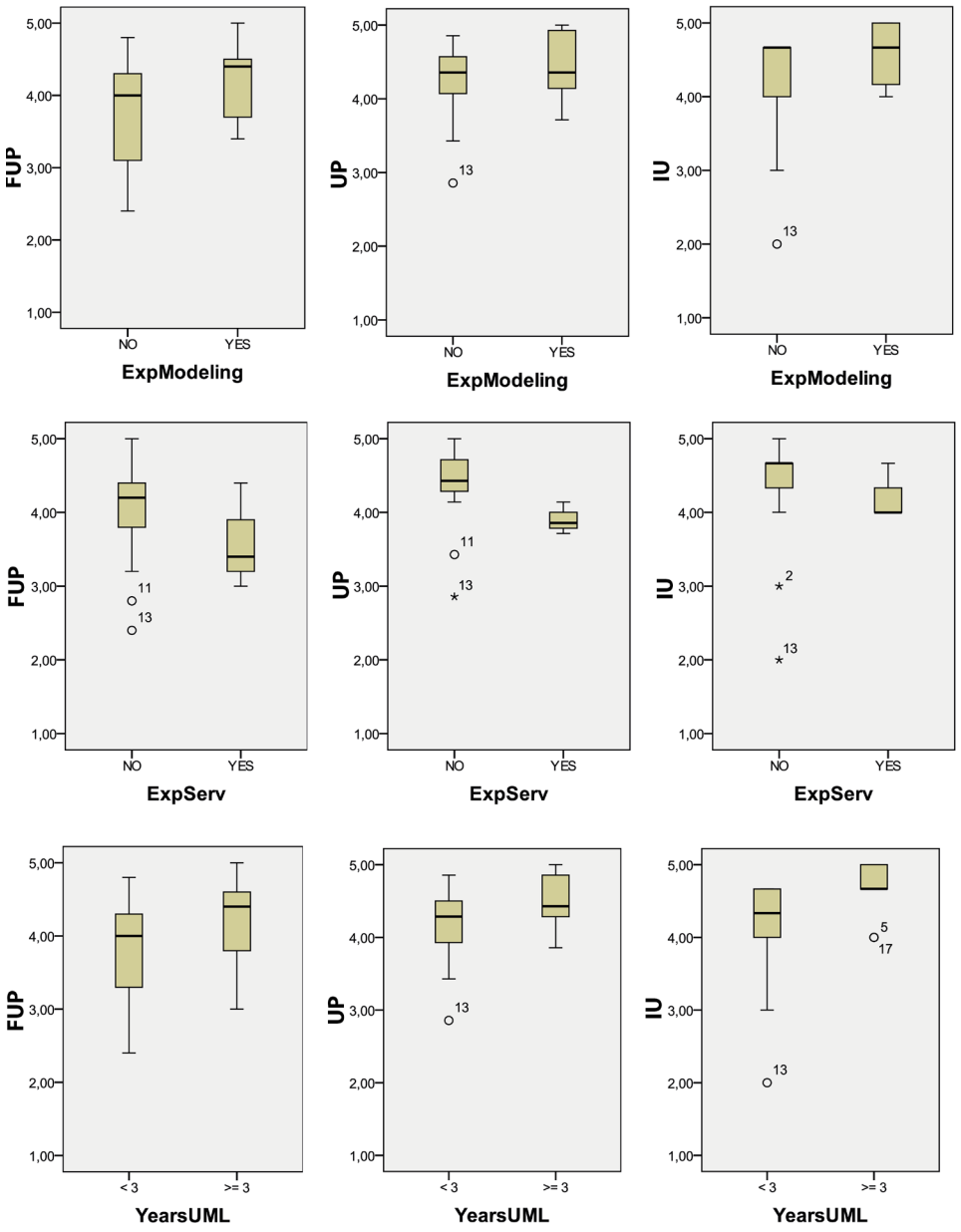



**Figura 7-5 Diagramas de caja de las variables subjetivas FUP, UP e IU**

**Tabla 7-5 Resultados descriptivos para ExpModeling, ExpServ, y YearsUML**

| | | *Variable* | | | | | |
|---|---|---|---|---|---|---|---|
| | | *FUP* | | *UP* | | *IU* | |
| *Factor* | *Nivel* | $\tilde{X}$ | $\sigma^2$ | $\tilde{X}$ | $\sigma^2$ | $\tilde{X}$ | $\sigma^2$ |
| *ExpModeling* | *No Exp.* | 4.000 | 0.578 | 4.357 | 0.328 | 4.667 | 0.717 |
| | *Exp. in UML* | 4.400 | 0.297 | 4.357 | 0.218 | 4.667 | 0.183 |
| *ExpServ* | *No Exp.* | 4.200 | 0.487 | 4.429 | 0.296 | 4.667 | 0.604 |
| | *Exp. in Serv.* | 3.400 | 0.520 | 3.857 | 0.048 | 4.000 | 0.148 |
| *YearsUML* | *<3* | 4.000 | 0.556 | 4.286 | 0.337 | 4.333 | 0.735 |
| | *>=3* | 4.400 | 0.378 | 4.429 | 0.158 | 4.667 | 0.151 |

Para comprobar la significancia estadística de estos resultados, se llevó a cabo el test paramétrico unilateral de Wilcoxon para una muestra contra el valor de prueba v = 3 para cada (ver Tabla 7-6). Por cada variable, los resultados fueron estadísticamente significantes en cada sub-grupo, excepto para los sujetos con experiencia en el desarrollo de servicios. Esto es probablemente debido el hecho de que el número de sujetos en este grupo es bajo (p. ej., N=3). Se ha aplicado el test paramétrico unilateral de Wilcoxon para una muestra con todo el conjunto de sujetos (N=20); esto con el fin de verificar si es que los datos pueden ser considerados significativos ya que no existieron diferentes estadísticas entre los sub-grupos (o perfiles). Los resultados de las pruebas fueron p-value=0.000 para las tres variables; por lo tanto, los resultados pueden ser considerados estadísticamente significativos para la población como un todo.

**Tabla 7-6 Test paramétrico unilateral de Wilcoxon para una muestra**

| | *ExpModeling* | | *ExpServ* | | *YearsUML* | |
|---|---|---|---|---|---|---|
| *Variable* | *NoExp. N=12* | *Exp. N=8* | *NoExp. N=17* | *Exp. N=3* | *<3 N=11* | *>=3 N=9* |
| *FUP* | 0.011 | 0.011 | 0.001 | 0.180* | 0.014 | 0.012 |
| *UP* | 0.003 | 0.012 | 0.000 | 0.109* | 0.004 | 0.008 |
| *IU* | 0.004 | 0.011 | 0.001 | 0.102* | 0.008 | 0.007 |

* No significante estadísticamente (p-value > 0.05)

El análisis de las respuestas a las preguntas abiertas reveló que la mayoría de los sujetos resaltaron la reducción de los detalles técnicos en el método de reconfiguración dinámica como razones para la adopción de DIARy en el futuro. Sin embargo, los sujetos expusieron cuestiones importantes que nos permitirán me-

jorar el método DIARy. Los participantes señalaron por ejemplo la falta de soporte automatizado en ciertas tareas; recomendaron que se incluyan mecanismos que permitan extraer automáticamente de las plataformas de proveedores cloud la información de aprovisionamiento que se requiere actualizar en los modelos del método DIARy y generar scripts de reconfiguración dinámica. Recomendaron también, la inclusión de mecanismos automatizados para la identificación de elementos arquitectónicos que ya existen en la aplicación actual, ayudando en la tarea de especificación del impacto arquitectónico.

### 7.4.2 Análisis cuantitativo

Similar al proceso llevado a cabo para analizar las variables cualitativas, en primer lugar, analizamos el efecto que los factores *ExpModeling*, *ExpServ* o *YearsUML* pueden tener en las variables cuantitativas de rendimiento. Para seleccionar la prueba a aplicar, y dado de que el número de muestras N es menor que 50, primero ejecutamos el test de Shapiro-Wilk con el fin de analizar si las variables se distribuyen normalmente. Como se muestra en la Tabla 7-7, los datos de Eficiencia y Efectividad se ajustan a la distribución normal (Shapiro-Wilk p-value ≥ 0.05) por cada combinación de grupos; y por lo tanto, el efecto de *ExpModeling*, *ExpServ* y *YearsUML* puede ser analizado utilizando el test de ANOVA de tres factores.

**Tabla 7-7 Test de Shapiro-Wilk para las variables cuantitativas**

| *Variable* | *ExpModeling* | | *ExpServ* | | *YearsUML* | |
|---|---|---|---|---|---|---|
| | *No Exp. N=12* | *Exp. N=8* | *No Exp. N=17* | *Exp. N=3* | *<3 N=11* | *>=3 N=9* |
| *Efectividad* | 0.229 | 0.055 | 0.064 | 0.253 | 0.251 | 0.223 |
| *Eficiencia* | 0.058 | 0.253 | 0.316 | 0.439 | 0.335 | 0.117 |

Ejecutamos una prueba de ANOVA de tres factores, incluyendo *ExpModeling*, *ExpServ* y *YearsUML* como variables de bloqueo para verificar que no existieron efectos significativos en las variables Efectividad y Eficiencia. La Tabla 7-8 muestra un extracto del test del efecto de *ExpModeling*, *ExpServ* y *YearsUML*, en donde no existe efecto de interacción significantes sobre las variables bajo estudio.

La Tabla 7-9 resume los resultados de cada variable por factor. Ya que todas las variables se ajustan a una distribución normal (p. ej., Shapiro-Wilk p-value>0.05) éstas son descritas por la media y desviación estándar ($\bar{x}$ y $\sigma$). Se puede notar que, en promedio, los participantes con experiencia en modelado (*ExpModeling*

y *YearsUML*>=3 years) obtuvieron mejores resultados en términos de efectividad y eficiencia. La efectividad para los participantes con *ExpModeling* y *YearsUML* fue 84% y 85% respectivamente (medidos por la exactitud de sus respuestas).

**Tabla 7-8 Efecto de *ExpModeling*, *ExpServ y YearsUML* en variables cuantitativas**

| *Factor* | *Variable* | |
|---|---|---|
| | *Efectividad* | *Eficiencia* |
| *ExpModeling* | 0.214 | 0.106 |
| *ExpServ* | 0.472 | 0.225 |
| *YearsUML* | 0.938 | 0.094 |
| *ExpModeling* ExpServ* | - | - |
| *ExpModeling* YearsUML* | 0.873 | 0.746 |
| *ExpServ* YearsUML* | - | - |
| *ExpModeling* ExpServ* YearsUML* | - | - |

Con respecto a la experiencia en desarrollo de software (p. ej., *ExpServ*) se puede observar que en promedio los sujetos que no tienen experiencia obtuvieron mejores resultados en términos de efectividad y eficiencia que los sujetos con experiencia; aunque estas diferencias no fueron estadísticamente significantes. Esto puede deberse a que los sujetos trabajan con modelos a un alto nivel de abstracción lo que no requiere altas habilidades técnicas para ejecutar las tareas experimentales.

**Tabla 7-9 Estadisticas descriptivas de variables cuantitativas**

| | | **Variable** | | | |
|---|---|---|---|---|---|
| | | **Efectividad** | | **Eficiencia** | |
| **Factor** | **Nivel** | **$\bar{x}$** | **σ** | **$\bar{x}$** | **σ** |
| *ExpModeling* | No Exp. | 0.770 | 0.185 | 0.032 | 0.017 |
| | ExpModeling | 0,839 | 0.171 | 0.035 | 0.013 |
| *ExpServ* | No Exp. | 0.818 | 0.175 | 0.035 | 0.016 |
| | ExpServ. | 0.683 | 0.180 | 0.023 | 0.001 |
| *YearsUML* | <3 | 0.758 | 0.193 | 0.036 | 0.019 |
| | >=3 | 0.847 | 0.154 | 0.030 | 0.010 |

## 7.5 Amenazas a la validez

En esta sección, se explican los principales problemas que pueden poner en pe-

ligro la validez del cuasi-experimento. Consideramos los cuatro tipos de amenazas que proponen Cook y Campbell (1979):

### 7.5.1 Validez interna

Las principales amenazas a la validez interna fueron: efectos de persistencia, experiencia de los participantes y cuán entendibles son los documentos.

- *Efectos de persistencia*: Con el fin de evitar efectos de persistencia, el dominio al que pertenece la aplicación cloud utilizada como parte del escenario de integración planteado en la sesión experimental, fue diferente al dominio de la aplicación cloud utilizada durante la sesión de entrenamiento. De igual manera, los cambios arquitectónicos que sufrieron estas aplicaciones debido a la integración fueron diferentes.

- *Experiencia de los participantes:* Esta amenaza fue mitigada mediante la definición de un cuestionario pre-experimento para analizar las posibles diferencias entre sujetos con experiencia y sujetos sin experiencia. Esto nos permitió rechazar el posible impacto que el factor experiencia puede tener en las variables bajo estudio. Aunque los participantes fueron estudiantes al momento del experimento, bajo ciertas condiciones, no existe gran diferencia entre estudiantes y profesionales (Höst y otros, 2000), y podrían ser considerados como la próxima generación de profesionales (Kitchenham y otros, 2002); por lo tanto, creemos que su habilidad para entender modelos arquitectónicos, principios de desarrollo de servicios, y evaluar métodos de reconfiguración arquitectónica dinámica pueden ser comparables a la habilidad de profesionales noveles.

- *Entendibilidad del material*: Esta amenaza fue minimizada aclarando todos los malentendidos que aparecieron en la sesión experimental; adicionalmente, el material fue revisado tanto por un miembro del grupo de investigación que no participó en el experimento, así como, por un experto en arquitecturas del software quién participó del estudio piloto realizado antes de las sesiones experimentales.

### 7.5.2 Validez externa

La validez externa se refiere a la verdad aproximada de las conclusiones que implican generalizaciones en diferentes contextos. La principal amenaza a la validez externa es la representatividad de los resultados, que puede verse afectada por el diseño de la evaluación, la selección de participantes y el tamaño y complejidad de las tareas experimentales.

- *Diseño de la evaluación*: Para aliviar esta amenaza, incluimos en el experimento únicamente las actividades más representativas como soporte a la reconfiguración arquitectónica dinámica. Las actividades fueron seleccionadas teniendo en cuenta que los cambios en tiempo de ejecución se aplican durante la integración y despliegue de nuevos servicios; esto es en la actividad *Despliegue y reconfiguración dinámica* del método DIARy. Adicionalmente, la complejidad del ejercicio fue ajustada de tal manera que los sujetos apliquen todas las reglas de las guías al menos una vez.
- *Selección de los participantes*: Con respecto a la experiencia de los sujetos que participaron en el experimento, el cuasi-experimento fue conducido con estudiantes con cierto conocimiento acerca de modelamiento de sistemas y desarrollo de servicios web. Los resultados preliminares obtenidos sólo podrían considerarse representativos de un entorno de desarrollo dirigido por modelos y con poblaciones compuestas de practicantes principiantes. Sin embargo, como trabajo futuro, pretendemos conducir experimentos que involucren profesionales, grupos más grandes, y mayor homogeneidad entre ellos.
- *El tamaño y complejidad de las tareas experimentales*: Propusimos un conjunto de tareas experimentales con suficiente nivel de complejidad, dado las restricciones de tiempo de las sesiones. También proveímos herramientas prototipo a fin de facilitar la ejecución de las tareas; sin embargo, para hacer frente completamente este problema deberíamos proveer herramientas integradas a un entorno de desarrollo y a plataformas de gestión específicos para proveedores cloud.

### 7.5.3 Validez de constructo

La principal amenaza a la validez de constructo es reflejar como las métricas que han sido estudiadas representan el objetivo de los investigadores. Las variables subjetivas son basadas en el Technology Acceptance Method (TAM), el cual es un modelo para la evaluación de tecnologías de la información bien conocido y validado empíricamente (Davis, 1989). Por lo tanto, la principal amenaza es la fiabilidad del cuestionario de evaluación. La fiabilidad del cuestionario fue analizada mediante la aplicación del test Alfa de Cronbach (Maxwell, 2002). Los valores de los resultados del test de fiabilidad Cronbach, ver Tabla 7-10, fueron mayores que el umbral mínimo de aceptación $\alpha > 0.70$ (Maxwell, 2002). Como resultado de este análisis, podemos concluir que los elementos del cuestionario de evaluación son medidas confiables y válidas de los constructores basados en percepción/intención subyacente del marco de validación propuesto.

**Tabla 7-10 Resultados del análisis de fiabilidad del cuestionario**

| *Variable* | *Alfa de Cronbach* |
|---|---|
| *FUP* | 0.844 |
| *UP* | 0.781 |
| *IU* | 0.747 |

### 7.5.4 Validez de las conclusiones

La principal amenaza a la validez de las conclusiones es la validez de las pruebas estadísticas aplicadas. Reducimos esta amenaza aplicando un conjunto de pruebas comúnmente aceptadas en la comunidad de la ingeniería del software empírico (Maxwell, 2002).

## 7.6 Conclusiones

En este capítulo se han presentado los resultados de la validación empírica con el objetivo de analizar el método DIARy con el propósito de evaluar respecto a su facilidad de uso, utilidad percibida, intensión de uso y rendimiento desde el punto de vista de un conjunto de arquitectos de software noveles.

Los resultados principales indican que bajo las condiciones del experimento el método DIARy es percibido como fácil de usar, útil y los participantes muestran una cierta intención de emplearlo en el futuro, independientemente de la experiencia de los participantes. Participantes con experiencia en modelado de sistemas y aquellos con menor experiencia en desarrollo de servicios Web perciben el método DIARy como fácil de usar y útil, y mostraron mayor intensión de utilizarlo en el futuro. Adicionalmente se realizó un análisis cuantitativo de la efectividad y eficiencia, que nos han permitido constatar que, en general, la aplicación del método permitió especificar la integración del incremento y reconfigurar la arquitectura de la aplicación correctamente. Por último, se analizaron las respuestas a las preguntas abiertas del cuestionario de evaluación para obtener retroalimentación sobre el método DIARy. El análisis de las respuestas a las preguntas abiertas reveló que la mayoría de los sujetos resaltaron la reducción de los detalles técnicos en el método de reconfiguración dinámica como razones para la adopción de DIARy en el futuro. Sin embargo, los sujetos expusieron cuestiones importantes que nos permitirán mejorar el método DIARy.

# Capítulo 8. Conclusiones y trabajos futuros

En este capítulo se presentan las principales conclusiones obtenidas durante el desarrollo de esta tesis y el análisis de en qué medida se han alcanzado los objetivos de investigación. Por otra parte, se presentan las contribuciones derivadas de esta investigación y los trabajos futuros.

La sección 8.1 presenta las conclusiones del trabajo realizado.

La sección 8.2 presenta los resultados obtenidos de este trabajo y las publicaciones realizadas en el contexto de la tesis.

La sección 8.3 presenta las becas obtenidos por el doctorando.

La sección 8.4 presenta los trabajos futuros que plantea el trabajo de tesis.

.

## 8.1 Conclusiones

La reconfiguración de arquitecturas de servicios producida por la integración de nuevos servicios en entornos cloud debido a un desarrollo incremental introduce algunos retos importantes. En particular, la alta disponibilidad requerida de las aplicaciones cloud implica que la reconfiguración debe realizarse mientras la aplicación se encuentra en ejecución. Con el objetivo de dar respuesta a esta problemática de una forma completa y sistemática, se ha propuesto y validado empíricamente un método para la reconfiguración dinámica e incremental de arquitecturas de aplicaciones cloud. A continuación se analiza el grado de cumplimiento de cada una de las metas definidas previamente en la Sección 1.4.

### 8.1.1 Definición de un método de reconfiguración dinámica de arquitecturas de servicios cloud

Respecto a esta meta se ha definido DIARy, un método basado en una estrategia de DSDM que da soporte al desarrollo incremental de aplicaciones cloud y a la reconfiguración dinámica de la arquitectura de la aplicación producida durante el despliegue de incrementos.

El método define un proceso que incluye un conjunto complementario de actividades. Durante la primera actividad, *Especificación de la integración del incremento*, los arquitectos realizan un razonamiento sistemático a cerca de la lógica de integración, del impacto que la integración de los servicios incluidos en un incremento producirá sobre la arquitectura actual de la aplicación cloud, y de la naturaleza del trabajo que realizarán los servicios incluidos en el incremento. En esta actividad, en contraste con otras propuestas para el soporte al desarrollo de aplicaciones cloud, nuestra propuesta considera a la arquitectura del incremento como una arquitectura de software *parcial* e *independiente* cuya integración reconfigurará la arquitectura actual de la aplicación. Desde nuestro punto de vista, considerar la arquitectura del incremento como un artefacto de diseño independiente de la arquitectura global de la aplicación, el cual será completado con: i) información del impacto de su integración en la arquitectura global de la aplicación y, ii) descripciones de alto nivel de la naturaleza y requisitos de los servicios incluidos en la arquitectura, permite:

- Proveer a los arquitectos de software de una solución que les ayuda a entender y a especificar los cambios que producirá la integración de un incremento en la arquitectura actual de la aplicación. Este entendimiento es importante debido a que la complejidad arquitectónica tiende a au-

mentar a medida que se introducen cambios, lo que probablemente resultara en un incremento en el número de errores introducidos (Svahnberg y Wohlin 2005), (Lam y Shankararaman 1999).

- Identificar los cambios que producirá la integración de incrementos, descritos durante la especificación de la integración, y propagarlos automáticamente hacia artefactos de software generados durante la implementación y despliegue de los servicios incluidos en un incremento: artefactos que implementan el diseño arquitectónico, y automatizan el despliegue del incremento y la reconfiguración dinámica de la aplicación.
- Mantener la trazabilidad de la evolución que ha tenido la arquitectura de la aplicación durante el desarrollo incremental.
- Otorgar independencia, a diferentes equipos de trabajo, para diseñar implementar y desplegar los servicios incluidos en cada incremento.

Las aplicaciones basadas en servicios son generalmente desarrolladas en una manera *ad hoc*, sin tomar en cuenta técnicas de ingeniería de software (Perepletchikov y otros, 2008).

Durante la segunda actividad*, Implementación del incremento*, los desarrolladores toman decisiones acerca de la estructura de empaquetado de los artefactos de software, la tecnología de implementación y los parámetros de los servicios que serán actualizados en tiempo de ejecución. A diferencia de otros enfoques de reconfiguración, el método DIARy promueve durante la implementación la aplicación de principios y patrones que favorecen tanto la interoperabilidad entre servicios del incremento con los de la aplicación actual, así como la reconfiguración dinámica. Consideramos que los modelos y transformaciones de modelos propuestas en esta actividad, permiten:

- Desacoplar los artefactos de software que implementan el protocolo de interacción o integración (orquestación de servicios) de aquellos que implementan la lógica de negocio; facilitando la integración y reconfiguración dinámica mediante el despliegue/re-despliegue de servicios en diferentes plataformas cloud. Permitiendo, además, ofrecer la implementación de la orquestación de servicios como servicios que pueden ser reutilizados.
- Encapsular el conocimiento acerca de la manera correcta de implementar el diseño arquitectónico, y cumplir principios y patrones que favorecen la reconfiguración dinámica.

- Consolidar en proyectos los artefactos que implementan los servicios incluidos en el diseño arquitectónico, de acuerdo a decisiones tomadas durante el proceso de desarrollo (p. ej., tecnología de implementación, impacto arquitectónico, naturaleza de trabajo de los servicios, términos SLA). Los artefactos consolidados son empaquetados en un único artefacto de despliegue y serán desplegados en una unidad computacional común, compartiendo sus recursos.

- Implementar, en varias tecnologías, los servicios descritos en la arquitectura del incremento. Mediante la ejecución de transformaciones el desarrollador es capaz de generar tantos artefactos de implementación como tecnologías especificadas para los proyectos relacionados a cada servicio.

- Propagar el impacto arquitectónico descrito durante la especificación de la integración; creando, modificando o eliminando artefactos de software que implementan el diseño arquitectónico de acuerdo al impacto arquitectónico.

En la última actividad, *Despliegue y reconfiguración dinámica*, los especialistas en operaciones toman decisiones acerca de los recursos cloud necesarios para satisfacer los requisitos descritos durante la actividad especificar la integración del incremento. Para hacer frente a la heterogeneidad en la oferta tanto de recursos cloud, así como de mecanismos de aprovisionamiento, despliegue, y reconfiguración dinámica; proponemos modelos y transformaciones de modelos que permiten a los especialistas en operaciones:

- Centrarse en las características de los recursos cloud que esperan de un proveedor, independientemente de los conceptos empleados por los proveedores para ofertarlos, y de acuerdo a los requisitos de uso y demanda especificados para cada servicio. Esto debido a que los especialistas en operaciones utilizan un lenguaje de descripción de recursos cloud independiente de la plataforma, el cual les permite no solo describir los entornos cloud que deben ser aprovisionados para el despliegue de los servicios que conforman un incremento, sino reutilizar descripciones existentes.

- Incluir en las descripciones de entornos cloud nuevos tipos de recursos cloud o características de recursos que en la actualidad no son ofertados por proveedores cloud. Por ejemplo, la descripción de instancias predefinidas de entornos cloud basada en modelos de características brinda la posibilidad, en el futuro, de extenderlas incluyendo nuevas ofertas de recursos cloud.

- Desplegar, en varios entornos cloud, los servicios descritos en la arquitectura del incremento. Mediante la ejecución de transformaciones el especialista en operaciones es capaz de generar tantos scripts de aprovisionamiento, despliegue o reconfiguración dinámica como proveedores cloud, modelos de servicio cloud o entornos cloud especificados.
- Propagar el impacto arquitectónico descrito durante la especificación de la integración, generando scripts de aprovisionamiento, despliegue o repliegue de acuerdo al impacto arquitectónico.

### 8.1.2 Definición de una aproximación tecnológica de soporte

Para cumplir con este objetivo se ha implementado una infraestructura software diseñada para soportar las actividades del método DIARy. Como parte del soporte a la estrategia de DSDM, la infraestructura software es construida en base a estándares y está compuesta por: i) un conjunto de DSLs que soportan la creación y mantenimiento de modelos en las diferentes actividades del proceso del método DIARy; y ii) un motor de transformaciones que incluye transformaciones que generan los modelos y artefactos propuestos como salida de las diferentes actividades del método DIARy.

Los DSLs incluyen la definición de los metamodelos, conforme a los cuales se crearán los modelos propuestos en el método DIARy, y editores gráficos que facilitan su creación. Transformaciones de modelo a código propagan el impacto arquitectónico desde la especificación de alto nivel de la integración del incremento hasta artefactos de implementación, despliegue y reconfiguración dinámica en la cloud. Por otro lado, transformaciones de modelo a modelo actualizan, luego de la integración, los artefactos de diseño de la aplicación actual (p. ej., modelo de la arquitectura de la aplicación). Adicionalmente, mantienen la trazabilidad y consistencia entre los artefactos generados.

La infraestructura software provee los siguientes aportes:

- Flexibilidad para especificar la integración de arquitecturas de servicios complejas. Esto debido a que la especificación de la integración se ejecuta utilizando un DSL que extiende lenguajes ampliamente utilizados y que han probado ser eficientes (p. ej., UML y SoaML); permitiendo hacer frente a necesidades de modelado futuras.
- Flexibilidad y capacidad de incorporar transformaciones que generan artefactos en nuevos lenguajes de implementación, o scripts de aprovisionamiento, despliegue de servicios y reconfiguración arquitectónica para nuevos proveedores cloud. Esto debido a que la implementación de los

componentes de la infraestructura está basada en plug-ins que se incorporan a la plataforma de Eclipse; proveyendo a los desarrolladores de un entorno de desarrollo común.

- Minimiza la discontinuidad entre las actividades de desarrollo, despliegue y reconfiguración dinámica, evitando tareas manuales y permitiendo entregar nuevas funcionalidades a los clientes de manera ágil.
- Encapsular en las transformaciones el conocimiento de los expertos del dominio. Una vez que una transformación ha sido definida, esta puede ser reutilizada repetidamente por desarrolladores quienes no necesitan conocer las especificidades técnicas de las plataformas de despliegue ni el proceso de transformación.

### 8.1.3 Validación del método mediante experimentos

De acuerdo a (Jamshidi y otros, 2013) las propuestas de reconfiguración centradas en la arquitectura incluyen algunas pruebas como parte de la evaluación; en donde la evidencia es a menudo obtenida aplicando el enfoque a casos de estudio pequeños. Pocos estudios empíricos han sido realizados, sin embargo, hasta donde conocemos, ninguno de ellos está relacionado con entornos cloud. Con el objetivo de suplir esta carencia se ha definido como meta la validación empírica del método DIARy. Para ello, se ha llevado a cabo un cuasi-experimento para comprobar la facilidad de uso percibida, la utilidad de uso percibida y la intención de uso en el futuro del método de reconfiguración dinámica e incremental de arquitecturas de aplicaciones cloud.

El experimento consistió en un escenario de integración, en donde la funcionalidad de una aplicación cloud, "Sistema de Reservas", que permite a agencias de viajes gestionar las reservas de sus clientes, evolucionó incrementalmente a través de integrar nuevos servicios cloud, reconfigurando la arquitectura actual de la aplicación. Los sujetos especificaron la integración del incremento y luego utilizaron un prototipo del editor de modelos para generar los scripts de reconfiguración dinámica de la aplicación de acuerdo al impacto arquitectónico especificado. La aplicación cloud fue implementada en el IDE Visual Studio 2013 como servicios WCF Web Roles (C#) y desplegada en Microsoft Azure©; los usuarios utilizaron los scripts generados para reconfigurar la arquitectura de la aplicación.

El experimento involucró a 20 estudiantes en ciencias de la computación de la Universitat Politècnica de València (España): 15 estudiantes de doctorado, 2 estudiantes de maestría y 3 estudiantes de pregrado. Los resultados del análisis cualitativo indican que bajo las condiciones del experimento el método DIARy

es percibido como fácil de usar, útil, y los participantes muestran una cierta intención de emplearlo en el futuro. Adicionalmente, se realizó un análisis cuantitativo de la efectividad y eficiencia, que nos ha permitido constatar que, en general, la aplicación del método DIARy permitió especificar la integración del incremento y reconfigurar la arquitectura de la aplicación correctamente. Por último, se analizaron las respuestas a las preguntas abiertas del cuestionario de evaluación para obtener retroalimentación sobre DIARy.

Desde el punto de vista investigador, el experimento ha resultado ser un medio valioso para obtener retroalimentación de mejora del método DIARy (por ejemplo, para mejorar la definición de los pasos a seguir para la *Especificación de la integración del incremento*, e identificar aspectos de usabilidad de las herramientas de soporte). Por lo que sabemos, éste es el primer estudio empírico que proporciona evidencia de la utilidad de un método reconfiguración dinámica de arquitecturas de aplicaciones cloud que soporta la implementación e integración incremental de incrementos de software en una aplicación cloud existente.

Desde un punto de vista práctico, somos conscientes de que este estudio solo proporciona resultados preliminares sobre la utilidad de DIARy como método de reconfiguración dinámica que soporta la naturaleza incremental del desarrollo de aplicaciones cloud. Si bien los resultados experimentales proporcionan buenos resultados en cuanto al rendimiento de nuestro método, estos resultados deben ser interpretados con cautela, ya que sólo son válidos dentro del contexto establecido en este experimento. Ahora es necesario analizar si los mismos resultados se producen con participantes más experimentados, utilizando nuevos objetos experimentales, y con un conjunto más amplio y más complejo de escenarios de integración.

Además de la validación empírica hemos mostrado la factibilidad de nuestro enfoque a través del desarrollo de un ejemplo que considera un escenario de desarrollo incremental (ver Capítulo 6). Este escenario, plantea que la reconfiguración dinámica de una aplicación cloud es provocada por la integración de servicios implementados con diferentes lenguajes de programación y desplegados en plataformas aprovisionadas en diferentes proveedores cloud.

## 8.2 Difusión de resultados

El trabajo descrito en esta tesis ha producido ***una*** publicación en una revista indexada en el Journal Citation Reports (JCR) *Software: Practice and Experience* con los resultados del diseño y evaluación del método DIARy, ***dos*** publicaciones en conferencias internacionales catalogadas como Nivel A según el ranking ERA-CORE (ISD 2015 e ISD 2016). Se han realizado ***dos*** publicaciones divulgativas,

una en un magazine y otra en una revista sin índice de calidad relativo: ERCIM-News y Maskana, respectivamente. Adicionalmente, ***una*** publicación en una conferencia internacional (MESOCA 2017), ***dos*** publicaciones en eventos internacionales (poster en MODELS 2014 y workshop ME 2014), y ***dos*** publicaciones en las Jornadas Nacionales de Ciencia e Ingeniería de los Servicios (JCIS 2015 y JCIS 2016). Finalmente, ***un*** capítulo de libro en Lecture Notes in Information Systems and Organisation de la editorial Springer, correspondiente a la versión extendida de uno de los mejores artículos de la conferencia (ISD 2016).

Las siguientes subsecciones describen dichas publicaciones con mayor detalle.

#### *8.2.1.1 Revistas con índice de calidad relativo*

- Miguel Zuñiga-Prieto, J. Gonzalez-Huerta, S. Abrahão, and E. Insfran, "*Dynamic reconfiguration of cloud application architectures*". Software: Practice and Experience, Special Issue on *Design Methods for Software Architectures in the Service-Oriented Computing and Cloud Paradigms*, Wiley, Impact Factor 0.652 (**JCR 2015**), 2016.

#### *8.2.1.2 Otras revistas sin índice de calidad relativo*

- M. Zuñiga-Prieto, L. Solano-Quinde, E. Insfran, and Y. Cabrera, "*Automatización del Proceso de Despliegue de Servicios en la Nube*" in Revista **Maskana** (indexada en Latindex), vol. 7, pp. 195–201, 2017.
- M. Zuñiga-Prieto, P. Cedillo, J. González-Huerta, E. Insfrán, and S. Abrahão, "*Monitoring Services Quality in the Cloud*" **ERCIM News**, no. 99, Special Theme on *Software Quality*, October 2014, pp. 19–20.

#### *8.2.1.3 Actas de congresos y workshops*

- M. Zúñiga-Prieto, E. Insfran, S. Abrahão, and C. Cano, "*Incremental Integration of Microservices in Cloud Applications*" in 25th International Conference on Information Systems Development (**ISD 2016**), August, 2016, Katowice, Poland. **ERA CORE Tier A**.
- M. Zuñiga-Prieto, S. Abrahão, and E. Insfrán, "*An incremental and model driven approach for the dynamic reconfiguration of cloud application architectures*" in 24th International Conference on Information Systems Development (**ISD 2015**), August, 2015, Harbin, China, pp. 198-209. **ERA CORE Tier A**.
- M. Zúñiga-Prieto, E. Insfran, and S. Abrahão, *"Architecture Description Language for Incremental Integration of Cloud Services Architectures"* in IEEE 10th Symposium on the Maintenance and Evolution of Service-Oriented Systems and Cloud-Based Environments (**MESOCA 2016**) co-located with the 32nd

IEEE International Conference on Software Maintenance and Evolution (ICSME), October, 2016, Raleigh, USA, pp. 19–20.

- M. Zuñiga-Prieto, J. Gonzalez-Huerta, S. Abrahão, and E. Insfran, "*Towards a Model-Driven Dynamic Architecture Reconfiguration Process for Cloud Services Integration*" in 8th International Workshop on Models and Evolution (**ME 2014**) co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2014, Valencia, Spain, pp. 52–61.
- M. Zuñiga-Prieto, J. Gonzalez-Huerta, S. Abrahão, and E. Insfran, "*Framework for Dynamic Architecture Reconfiguration of Cloud Services*" in Joint Proceedings of **MODELS 2014 Poster Session** and the ACM Student Research Competition SRC co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2014, Valencia, Spain, pp. 46–50.
- J. Sandobalín-Guamán, M. Zúñiga-Prieto, E. Insfran, S. Abrahão, and C. Cano, "*Una aproximación DevOps para el Desarrollo Dirigido por Modelos de Servicios Cloud*," in XII Jornadas de Ciencia e Ingeniería de Servicios (**JCIS 2016**), Salamanca, Spain.
- M. Zuñiga-Prieto, S. Abrahão, and E. Insfran, "*Perfil UML para el Modelado de la Integración de Servicios Cloud en Procesos de Desarrollo Incremental*" in XI Jornadas de Ciencia e Ingeniería de Servicios (**JCIS 2015**), Santander, Spain.

#### *8.2.1.4 Capítulos de libro*

- M. Zúñiga-Prieto, E. Insfran, S. Abrahão, and C. Cano, *"Automation of the Incremental Integration of Microservices Architectures"* Capítulo del Libro "*Complexity in Information Systems Development: Proceedings of the 25th International Conference on Information Systems Development* (Lecture Notes in Information Systems and Organisation, 22), Springer, 2016.

  Esta publicación es una versión extendida del artículo publicado en el ISD 2016 y seleccionado como uno de los mejores artículos de la conferencia.

## 8.3 Becas

Beca pre-doctoral (2011-2015) provista por la Secretaría de Educación Superior, Ciencia, Tecnología e Innovación de Ecuador (SENESCYT-Ecuador).

## 8.4 Trabajos futuros

En este documento se ha presentado el trabajo realizado durante la tesis doctoral. Sin embargo, esto no significa el fin del trabajo de investigación en el área. Como objetivo a corto y medio plazo se deben de afrontar las tareas pendientes hasta liberar la solución definitiva de acuerdo con el modelo de investigación y transferencia tecnológica aplicado en la tesis. En esta tesis se han cubierto las seis primeras fases de la metodología de investigación (Gorschek y otros 2006), quedando abierta la validación realista. Como trabajo futuro se deben de realizar experimentos para realizar validaciones en un contexto industrial con el objetivo de lanzar finalmente la solución final.

La reconfiguración arquitectónica es realizada mediante el despliegue (o re-despliegue) de servicios y la actualización de enlaces entre servicios; sin embargo, no siempre es posible re-desplegar servicios cuyas instancias se encuentra en ejecución. Esta tarea representa un reto, especialmente si las instancias de un servicio se ejecutan en plataformas aprovisionadas en diferentes proveedores cloud, quienes ofrecen diferentes mecanismos para la gestión de instancias. La gestión de instancias de servicios en el método propuesto se encuentra en sus etapas iniciales; por lo tanto, en un trabajo futuro se hará un análisis de los mecanismos de gestión de instancias provistos por los proveedores cloud y se proporcionará un lenguaje que permita su especificación, y transformaciones de modelos que generen scripts o servicios para su gestión.

Adicionalmente algunas características de recursos cloud que son específicas de la oferta de un proveedor podrían estar siendo utilizadas ineficientemente (o no utilizadas), debido a que éstas no son documentadas en nuestros modelos. El enfoque DSDM aplicado en nuestro método nos permite tanto abstraer la descripción de los recursos cloud generalmente ofrecidos por proveedores cloud, así como describir las características avanzadas de los recursos cloud ofrecidos por un proveedor específico. En un trabajo futuro, nuestros modelos serán extendidos con constructores que permitan especificar las características avanzadas de las ofertas de proveedores cloud. Esto permitirá sacar ventaja de las ofertas especializadas de los proveedores cloud mediante la generación de scripts de aprovisionamiento que las implementen.

La generación de artefactos de software que propagan cambios arquitectónicos de actualización y eliminación está aún en etapas iniciales en este trabajo de tesis. Como trabajo futuro, planeamos definir transformaciones de modelos que propaguen cambios arquitectónicos de tipo actualización y eliminación hacia artefactos previamente generados, entornos cloud previamente aprovisionados, y servicios previamente desplegados. Pretendemos también, proveer mecanismos

que permitan gestionar la consistencia incremental, evitando que se pierdan los cambios introducidos en los artefactos generados (p. ej., cambios en el código generado de implementación de interfaces).

El paradigma DevOps permitirá pasar de una integración incremental a una integración continua mediante la colaboración entre las actividades de desarrollo con las actividades de operación, permitiendo entregas de software más rápidas (Wettinger y otros 2015). En este contexto, en un trabajo futuro planeamos adaptar el método propuesto en este trabajo de tesis con el propósito de satisfacer practicas DevOps, las cuales promueven la automatización de los procesos de entrega de software y la automatización de los cambios en infraestructura.

Hemos analizado el impacto de aplicar el método DIARy a estilos arquitectónicos diferentes a SOA; tal es el caso del estilo arquitectónico microservices, cuyos resultados fueron presentados en (Zúñiga-Prieto y otros 2016). Como trabajo futuro planeamos analizar el impacto de aplicar el método DIARy en escenarios en los que la arquitectura del incremento y la arquitectura actual de la aplicación están compuestas por servicios ofrecidos en entornos de Internet de las Cosas y en entornos Fog Computing (Cisco Systems, 2016), dadas sus características intrínsecas y similares a la interoperación en entornos cloud.

Con respecto a la infraestructura software implementada en este trabajo de tesis, planeamos incorporar plug-ins de transformación para generar scripts de aprovisionamiento, despliegue y reconfiguración dinámica en otros proveedores cloud, tanto públicos como privados. También planeamos integrar y empaquetar de mejor manera los diferentes plug-ins de Eclipse que conforman la infraestructura software, proveyendo una aplicación independiente.

Con respecto a la validación empírica, con el propósito de mejorar la representatividad de los resultados, se pretende realizar réplicas de la validación considerando grupos homogéneos con un número mayor de sujetos, objetos experimentales diferentes, y haciendo uso de la infraestructura software implementada. También pretendemos conducir otros estudios empíricos con casos de estudio industriales que permitirán obtener evidencia empírica de la factibilidad del enfoque propuesto según profesionales.

# Bibliografía


Amini, M., Sadat, N., 2013. Cloud Computing Transform The Way of IT Delivers Services to The Organizations 1, 1–5.

Ardagna, D., Di Nitto, E., Casale, G., Petcu, D., Mohagheghi, P., Mosser, S., Matthews, P., Gericke, A., Ballagny, C., D'Andria, F., Others, 2012. MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds, in: Proceedings of the 4th International Workshop on Modeling in Software Engineering. pp. 50–56. doi:10.1109/MISE.2012.6226014

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2010. A View of Cloud Computing. Commun. ACM 53, 50–58.

ATL, 2014. ATL - A Model Transformation Language [WWW Document]. URL http://www.eclipse.org/atl/

Bai, X., Xie, J., Chen, B., Xiao, S., 2007. Dresr: Dynamic Routing in Enterprise Sservice Bus, in: E-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on. IEEE, pp. 528--531.

Basili, Victor R and Rombach, H.D., 1988. The TAME Project: Towards Improvement-oriented Software Environments. Softw. Eng. IEEE Trans. 14, 758--773.

Basili, V.R., Caldiera, G., Rombach, H.D., 1994. The Goal Question Metric Approach, in: Encyclopedia of Software Engineering. Wiley, pp. 1–10.

Basili, V.R., Shull, F., Lanubile, F., 1999. Building Knowledge through Families of Software Studies : Softw. Eng. IEEE Trans. 25, 456--473.

Belaunde, M., Burt, C., Casanave, C., DSouza, D., Duddy, K., El Kaim, W., Kennedy, A., Frank, W., Frankel, D., Hauch, R., Hendryx, S., 2003. MDA Guide Version 1.0, Object Management Group.

Bergmayr, A., Troya, J., Neubauer, P., Wimmer, M., Kappel, G., 2014. UML-based Cloud Application Modeling with Libraries, Profiles, and Templates, in: In CloudMDE@ MoDELS. pp. 56–65.

Berre, A.J., 2008. Service Oriented Architecture Modeling Language (SoaML)-Specification for the UML Profile and Metamodel for Services (UPMS). Object Manag. Gr.

Bézivin, J., 2005. On the Unification Power of Models. Softw. Syst. Model. 4, 171–188. doi:10.1007/s10270-005-0079-0

Binz, T., Breitenbücher, U., Kopp, O., Leymann, F., 2014. TOSCA: Portable Automated Deployment and Management of Cloud Applications, in: Advanced Web Services. pp. 527–549. doi:10.1007/978-1-4614-7535-4_22

Bodart, F., Patel, A., Sim, M., Weber, R., 2001. Should Optional Properties Be Used in Conceptual Modelling? A Theory and Three Empirical Tests. Inf. Syst. Res. 12, 384–405. doi:10.1287/isre.12.4.384.9702

Brambilla, M., Cabot, J., Wimmer, M., 2012. Model-Driven Software Engineering in Practice, Model-Driven Software Engineering in Practice. doi:10.2200/S00441ED1V01Y201208SWE001

Brandtzæg, E., Mohagheghi, P., Mosser, S., 2012a. Towards a Domain-specific Language to Deploy Applications in the Clouds. CLOUD Comput. 2012 Third Int. Conf. Cloud Comput. GRIDs, Virtualization Towar. 213–218.

Brandtzæg, E., Mosser, S., Mohagheghi, P., 2012b. Towards CloudML, a Model-based Approach to Provision Resources in the Clouds, in: 8th European Conference on Modelling Foundations and Applications (ECMFA). pp. 18–27.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D., 2012. Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 416–424. doi:10.1007/978-3-642-33606-5_25

Breiter, G., Behrendt, M., 2009. Life Cycle and Characteristics of Services in the World of Cloud Computing. IBM J. Res. Dev. 53, 1–8. doi:10.1147/JRD.2009.5429057

Breivold, H.P., Crnkovic, I., Radosevic, I., Balatinac, I., 2014. Architecting for the Cloud: A Systematic Review, in: 17th International Conference on Computational Science and Engineering. IEEE, pp. 312–318. doi:10.1109/CSE.2014.85

Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., D'Andria, F., 2014. SeaClouds: A European Project on Seamless Management of Multi-Cloud Applications. ACM SIGSOFT Softw. Eng. Notes 39, 1–4.

Brown, P.F., Metz, R., Hamilton, B.A., 2006. Reference Model for Service Oriented Architecture 1.0 1–31.

Brucker, A.D., Doser, J., 2007. Metamodel-based UML Notations for Domain-specific Languages. 4th Int. Work. Lang. Eng. (atem 2007) 1–15.

Buckley, J., Mens, T., Zenger, M., Rashid, A., Kniesel, G., 2005. Towards a Taxonomy of Software Change. J. Softw. Maint. Evol. Res. Pract. 17, 309--332.

Campbell, D.T., Stanley, J.C., 1982. Diseños Experimentales y Cuasi Experimentales en la Investigación Social. Editorial Ammorrortu.

Chou, D., 2010. Microsoft Cloud Computing Platform.

Cisco Systems, 2016. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Www.Cisco.Com 6.

Clark, T., Sammut, P., Willans, J., 2008. Applied Metamodelling: A Foundation for Language Driven Development, Book To Be Published. doi:10.1016/j.jbusres.2014.06.013.The

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J., 2011. Documenting Software Architectures: Views and Beyond, 2nd ed. Addison–Wesley Professional.

Clements, P., Kazman, R., Klein, M., 2003. Evaluating Software Architectures. Addison-Wesley.

Cockburn, A., Highsmith, J., 2001. Agile Software Development, the People Factor. Computer (Long. Beach. Calif). 34, 131–133.

Conover, W.J., 1998. Practical Nonparametric Statistics. John Wiley & Sons.

Cook, T.D., Campbell, D.T., 1979. Quasi-experimentation: Design & Analysis Issues for Field Settings. Houghton Mifflin Company, Boston, MA.

Costa-Soria, C., 2011. Dynamic Evolution and Reconfiguration of Software Architectures through Aspects. University of Politecnica De Valencia.

Costa-Soria, C., Pérez, J., Carsí, J.Á., 2009. Handling the Dynamic Reconfiguration of Software Architectures using Aspects, in: Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on. IEEE, pp. 263--266.

Costa, B., Pires, P.F., Delicato, F.C., Merson, P., 2014. Evaluating REST Architectures-Approach, Tooling and Guidelines. J. Syst. Softw. 112, 156–180. doi:10.1016/j.jss.2015.09.039

Davis, F.D., 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Q. 13, 319–340. doi:10.2307/249008

Di Martino, B.., Petcu, D.., Cossu, R.., Goncalves, P.., Máhr, T.., Loichate, M.., 2011. Building a Mosaic of Clouds. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 6586 LNCS, 571–578. doi:10.1007/978-3-642-21878-1_70

Díaz, J., Pérez, J., Garbajosa, J., 2014. Agile Product-line Architecting in Practice: A Case Study in Smart Grids. Inf. Softw. Technol. 56, 727–748. doi:10.1016/j.infsof.2014.01.014

Distributed Management Task Force Inc., 2010. Architecture for Managing Clouds: A White Paper from the Open Cloud Standards Incubator [WWW Document]. Distrib. Manag. Task Force Inc. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf (accessed 7.20.01).

Distributed Management Task Force Inc., 2009. Interoperable Clouds: A White Paper from the Open Cloud Standards Incubator [WWW Document]. Distrib. Manag. Task Force Inc. URL http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf (accessed 7.20.01).

DMTF, 2013. Open Virtualization Format (No. Version 2.0.0).

Eclipse, 2017a. Eclipse Modeling Framework Project (EMF) [WWW Document]. URL http://www.eclipse.org/modeling/emf/

Eclipse, 2017b. Eclipse OCL [WWW Document]. URL http://projects.eclipse.org/projects/modeling.mdt.ocl

Eclipse, 2011. The Graphical Modeling Framework [WWW Document]. URL http://www.eclipse.org/gmf/

Eclipse Foundation, 2015. Eclipse - The Eclipse Foundation Open Source Community [WWW Document]. URL http://www.eclipse.org (accessed 3.1.17).

Elvesæter, B., Berre, A., Sadovykh, A., 2011. Specifying Services Using the Service Oriented Architecture Modeling Language (SoaML)-A Baseline for Specification of Cloud-Based Services, in: CLOSER.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, Min and Newling, T., 2004. Patterns: Service-Oriented Architecture and Web Services. IBM Corporation, International Technical Support Organization.

Erl, T., 2007. SOA Principles of Service Design. doi:citeulike-article-id:2135823

Erl, T., 2005. Service-oriented Architecture: Concepts, Technology, and Design. Pearson Education India.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P., 2014. Cloud Computing Patterns: Fundamentals to Design, Build and Manage Cloud Applications. Springer.

France, R.B., Ghosh, S., Dinh-Trong, T., Solberg, A., 2006. Model-driven Development using UML 2.0: Promises and Pitfalls. Computer (Long. Beach. Calif). 39, 59--66.

Fredj, M., Georgantas, N., Issarny, V., Zarras, A., 2008. Dynamic Service Substitution in Service-oriented Architectures, in: Proceedings - 2008 IEEE Congress on Services, SERVICES 2008. pp. 101–104. doi:10.1109/SERVICES-1.2008.52

Frey, S., Fittkau, F., Hasselbring, W., 2013. Search-based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud, in: Proceedings - International Conference on Software Engineering. pp. 512–521. doi:10.1109/ICSE.2013.6606597

Frey, S., Hasselbring, W., 2011. The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. Int. J. Adv. Softw. 4, 342–353.

Frey, S., Hasselbring, W., Schnoor, B., 2013. Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms, in: Journal of Software: Evolution and Process. pp. 1089–1115. doi:10.1002/smr.582

Gartner Inc., 2001. Enterprise Applications: Adoption of E-Business and Document Technologies, 2000-2001: Worldwide Industry Study [WWW Document].

Gaynor, G.H., 1996. Handbook of Technology Management. McGraw-Hill Professional.

Gemino, A., Wand, Y., 2005. Complexity and Clarity in Conceptual Modeling: Comparison of Mandatory and Optional Properties. Data Knowl. Eng. 55, 301–326. doi:10.1016/j.datak.2004.12.009

Gómez, A., 2012. Model Driven Software Product Line Engineering: System Variability View and Process Implications. Universitat Politècnica de València.

Gómez, A., Ramos, I., 2010. Cardinality-based Feature Modeling and Model-driven Engineering: Fitting them Together. Int. Work. Var. Model. Softw. intensive Syst. 37, 61–68.

Gonçalves, G., Endo, P., Santos, M., Sadok, D., Kelner, J., Melander, B., Mångs, J.E., 2011. CloudML: An Integrated Language for Resource, Service and Request Description for D-Clouds, in: Proceedings - 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011. pp. 399–406. doi:10.1109/CloudCom.2011.60

Gorschek, T., Wohlin, C., Carre, P., Larsson, S., 2006. A Model for Technology Transfer in Practice. Software, IEEE 23, 88–95.

Guha, R., 2013. Impact of Semantic Web and Cloud Computing Platform on Software Engineering, in: Software Engineering Frameworks for the Cloud Computing Paradigm. Springer, pp. 3--24.

Guillén, J., Miranda, J., Murillo, J.M., Canal, C., 2013a. A UML Profile for Modeling Multicloud Applications, in: European Conference on Service-Oriented and Cloud Computing. pp. 180–187.

Guillén, J., Miranda, J., Murillo, J.M., Canal, C., 2013b. Developing Migratable Multicloud Applications based on MDE and Adaptation Techniques. Proc. Second Nord. Symp. Cloud Comput. Internet Technol. - Nord. '13 30–37. doi:10.1145/2513534.2513541

Haber, A., Ringert, J.O., Rumpe, B., 2012. MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Tech. Rep. AIB-2012-03 2012,3.

Haines, M.N., Rothenberger, M.A., 2010. How a Service-Oriented Architecture May Change the Software Development Process. Commun. ACM 53, 135–140. doi:10.1145/1787234.1787269

Hajjat, M., Sun, X., Sung, Y.E., Maltz, D., Rao, S., Sripanidkulchai, K., Tawarmalani, M., 2010. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. ACM SIGCOMM Comput. Commun. Rev. 40, 243–254. doi:10.1145/1851182.1851212

Halima, R. Ben, Drira, K., Jmaiel, M., 2008. A QoS-oriented Reconfigurable Middleware for Self-healing Web Services, in: Proceedings of the IEEE International Conference on Web Services, ICWS 2008. pp. 104–111. doi:10.1109/ICWS.2008.113

Hamdaqa, M., Livogiannis, T., Tahvildari, L., 2011. A Reference Model for Developing Cloud Applications, in: CLOSER. Citeseer, pp. 98–103.

Hamdaqa, M., Tahvildari, L., 2015. Stratus ML: A Layered Cloud Modeling Framework, in: Cloud Engineering (IC2E), 2015 IEEE International Conference on. pp. 96–105. doi:10.1109/IC2E.2015.42

Hertzum, M., Jacobsen, N.E., 2001. The Evaluator Effect a Chilling Fact About Usability Evaluation Methods. Int. J. Hum. Comput. Interact. 13, 421--443. doi:10.1207/S15327590IJHC1501

Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P., 2005. Generalizing a Model of Software Architecture Design from Five Industrial Approaches, in: WICSA "05: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA"05). pp. 77–88. doi:http://dx.doi.org/10.1109/WICSA.2005.36

Homer, A., Sharp, J., Brader, L., Narumoto, M., Swanson, T., 2014. Cloud Design Patterns: Prescriptive Architecture Guidance. Microsoft patterns & practices.

Höst, M., Regnell, B., Wohlin, C., 2000. Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empir. Softw. Eng. 5, 201–214. doi:10.1023/a:1026586415054

Hu, P.J., Chau, P.Y.K., Liu Sheng, O.R., Tam, K.Y., 1999. Examining the Technology Acceptance Model Using Physician Acceptance of Telemedicine Technology. J. Manag. Inf. Syst. 16, 91–112. doi:10.2307/40398433

Hull, R., Su, J., 2005. Tools for Composite Web Services. ACM SIGMOD Rec. 34, 86. doi:10.1145/1083784.1083807

Hurwitz, J., Blloor, R., Kaufman, M., Halper, F., 2009. Service Oriented Architecture for Dummies.

IEEE., 2000. Recommended Practice for Architectural Description of Software-Intensive Systems. (IEEE Std 1471-2000), IEEE Comput. Soc. doi:10.1109/IEEESTD.2000.91944

Infosys, 2011. Connecting the dots : Cloud and SOA SOA meets Cloud (White Paper).

Jamshidi, P., Ghafari, M., Ahmad, A., Pahl, C., 2013. A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research, in: 17th European Conference on Software Maintenance and Reengineering. IEEE, Genova, pp. 305–314. doi:10.1109/CSMR.2013.39

Josuttis, N.M., 2007. SOA in Practice. October. doi:10.1017/CBO9781107415324.004

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., 2008. ATL: A Model Transformation Tool. Sci. Comput. Program. 72, 31–39. doi:10.1016/j.scico.2007.08.002

Juliot, E., Benois, J.B., 2010. Viewpoints Creation using Obeo Designer or how to Build Eclipse DSM without being an Expert Developer? Obeo Des. Whitepaper.

Kamateri, E., Loutas, N., Zeginis, D., Ahtes, J., Andria, F.D., Bocconi, S., Gouvas, P., Ledakis, G., Ravagli, F., Lobunets, O., Tarabanis, K.A., 2013. Cloud4SOA : A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability, in: The European Conference on Service-Oriented and Cloud Computing. pp. 64–78.

Kandé, M., Crettaz, V., Strohmeier, A., Sendall, S., 2002. Bridging the Gap Between IEEE 1471, an Architecture Description Language, and UML. Softw. Syst. Model. 1, 113–129. doi:10.1007/s10270-002-0010-x

Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S., 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engineering Institute, Carnegie Melon University.

Kelly, S., Tolvanen, J.P., 2007. Domain-Specific Modeling: Enabling Full Code Generation, Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons. doi:10.1002/9780470249260

Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J., 2002. Preliminary Guidelines for Empirical Research in Software Engineering. Softw. Eng. IEEE Trans. 28, 721–734.

Krafzig, D., Banke, K., Slama, D., 2004. Enterprise SOA: Service-Oriented Architecture Best Practices [WWW Document]. Prentice Hall. URL http://entropy.soldierx.com/~kayin/archive/ebooks/Prentice Hall - Enterprise SOA. Service-Oriented Architecture Best Practices.pdf

Krill, P., 2009. The Cloud-SOA Connection. InfoWorld.

Kruchten, P.B., 1995. The 4+1 Viewmodel of Architecture. IEEE Softw. 12, 42–50.

Kyriazis, D., Menychtas, A., Kousiouris, G., Boniface, M., Cucinotta, T., Oberle, K., Voith, T., Oliveros, E., Berger, S., 2014. A real-time service oriented

infrastructure. J. Comput. 1, 196–204. doi:10.5176/2010-2283_1.2.60

La Manna, V.P., 2011. Dynamic Software Update for Component-based Distributed Systems. Proc. 16th Int. Work. Component-oriented Program. 1–8. doi:10.1145/2000292.2000294

Laitenberger, O., Rombach, D., 2003. (Quasi-)experimental Studies in Industrial Settings, in: Lecture Notes on Empirical Software Engineering. World Scientific Publishing Co., Inc., pp. 167–227.

Lam, W., Shankararaman, V., 1999. Requirements Change: A Dissection of Management Issues, in: Conference Proceedings of the EUROMICRO. pp. 244–251. doi:10.1109/EURMIC.1999.794787

Laplante, P.A., Zhang, J., Voas, J., 2008. What's in a Name? Distinguishing between SaaS and SOA. It Prof. 10, 46--50.

Leavitt, N., 2009. Is Cloud Computing Really Ready for Prime Time? Growth 27, 15–20.

Lehman, M.M., 1996. Laws of Software Evolution Revisited. 5th Eur. Work. Softw. Process Technol. 108–124. doi:10.1007/BFb0017737

Leymann, F., 2009. Cloud Computing: The Next Revolution in IT, in: Fritsch, D. (Ed.), Proc. 52th Photogrammetric Week. Wichmann Verlag, pp. 3–12.

Leymann, F., Fehling, C., Mietzner, R., Nowak, A., Dustdar, S., 2011. Moving Applications To the Cloud: An Approach Based on Application Model Enrichment. Int. J. Coop. Inf. Syst. 20, 307–356. doi:10.1142/S0218843011002250

Li, Y., Lu, Y., Yin, Y., Deng, S., Yin, J., 2010. Towards QoS-based Dynamic Reconfiguration of SOA-based Applications. Proc. - 2010 IEEE Asia-Pacific Serv. Comput. Conf. APSCC 2010 107–114. doi:10.1109/APSCC.2010.21

Likert, R., 1932. A Technique for the Measurement of Attitudes. Arch. Psychol. 22, 1–55. doi:2731047

Linthicum, D.S., 2009. Cloud Computing and SOA Convergence in Your Enterprise: A Step-by-Step Guide. Addison-Wesley.

Loutas, N., 2011. Cloud4SOA: Requirements Analysis Report.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A., 2013. What Industry Needs from Architectural Languages: A Survey. IEEE Trans. Softw. Eng. 39, 869–891.

Markus, Lynne, M., Keil, M., 1994. If We Build It, They Will Come: Designing Information Systems that People Want to Use. Sloan Manage, 11–25.

Maxwell, K., 2002. Applied Statistics for Software Managers, Prentice-Hall, PTR. Prentice Hall, Englewood Cliffs, NJ.

Medvidovic, N., 1996. ADLs and Dynamic Architecture Changes, Second Int. Softw. Archit, 24–27.

Medvidovic, N., Taylor, R.N., 2000. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Trans. Softw. Eng. 26, 70–93.

Medvidovic, N., Taylor, R.N., 1997. A Framework for Classifying and Comparing Architecture Description Languages. ACM SIGSOFT Softw. Eng. Notes 22, 60–76. doi:10.1145/267896.267903

Mell, P., Grance, T., 2011. The NIST Definition of Cloud Computing [WWW Document]. Spec. Publ. 800-145, Recomm. Natl. Inst. Stand. Technol. URL http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (accessed 3.2.14).

Mens, T., Van Gorp, P., 2006. A Taxonomy of Model Transformation. Electron. Notes Theor. Comput. Sci. 152, 125–142. doi:10.1016/j.entcs.2005.10.021

Metsch, T., Edmonds, A., Others, A., 2010. Open Cloud Computing Interface - Infrastructure, in: Standards Track, No. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN). p. 17.

Microsoft, 2017a. Azure Service Configuration Schema (.cscfg File) [WWW Document]. URL https://msdn.microsoft.com/library/en-us/Ee758710.aspx (accessed 7.20.01).

Microsoft, 2017b. Web.config Transformation Syntax for Web Project Deployment Using Visual Studio [WWW Document]. URL https://msdn.microsoft.com/en-us/library/dd465326(v=vs.110).aspx (accessed 7.20.01).

Miller, S.J., Scott, K., Uhl, A., Weise, D., 2004. MDA Distilled, AddisonWesley Professional. AddisonWesley Professional.

Mohagheghi, P., Berre, A.J., Henry, A., Barbier, F., Sadovykh, A., 2010. REMICS- REuse and Migration of Legacy Applications to Interoperable Cloud Services - REMICS Consortium, in: European Conference on a Service-Based Internet. Springer, pp. 195–196.

Moinuddin, M., 2007. An overview of Service-oriented Architecture in Retail. Microsoft Dev. Network2 12.

Moody, D.L., 2001. Dealing with Complexity: a Practical Method for Representing Large Entity Relationship Models. University of Melbourne.

Motta, G., Sfondrini, N., Sacco, D., 2012. Cloud Computing: An Architectural and Technological Overview, in: Service Sciences (IJCSS), 2012 International Joint Conference on. IEEE, Shanghai, pp. 23–27. doi:10.1109/IJCSS.2012.37

Muppalla, A.K., Pramod, N., Srinivasa, K., 2013. Efficient Practices and Frameworks for Cloud-Based Application Development, in: Software Engineering Frameworks for the Cloud Computing Paradigm. Springer, pp. 305–329.

Newman, S., 2015. Building Microservices. O'Reilly Media, Inc.

Nguyen, D.K., 2013. Blueprint Model and Language for Engineering Cloud Applications.

Nguyen, D.K., Lelli, F., Papazoglou, M.P., van den Heuvel, W.-J., 2012. Blueprinting Approach in Support of Cloud Computing. Futur. Internet 4, 322–346. doi:10.3390/fi4010322

Nguyen, D.K., Lelli, F., Taher, Y., Parkin, M., Papazoglou, M.P., Van Den Heuvel, W.J., 2011. Blueprint Template Support for Engineering Cloud-based Services, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 26–37. doi:10.1007/978-3-642-24755-2_3

OASIS, 2008. OASIS Security Services (SAML) TC | OASIS [WWW Document]. Sci. Technol. URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Obeo, 2005. Acceleo [WWW Document]. URL https://www.eclipse.org/acceleo/

Object Management Group Inc., 2013. Meta-Object Facility [WWW Document]. URL http://www.omg.org/spec/MOF/1.4/

Object Management Group Inc., 2012. Service oriented architecture Modeling Language (SoaML) Specification [WWW Document]. URL http://www.omg.org/spec/SoaML/

Object Management Group Inc., 2011. Knowledge Discovery Model (KDM) [WWW Document]. URL http://www.omg.org/spec/KDM/1.3/

Object Management Group Inc., 2008a. Software & Systems Process Engineering Meta-Model Specification (SPEM) v2.0 [WWW Document]. URL http://www.omg.org/spec/SPEM/2.0/

Object Management Group Inc., 2008b. Query/View/Transformation [WWW Document]. URL http://www.omg.org/spec/QVT/1.0/

Object Management Group Inc., 2006. Object Constraint Language (OCL) Specification Version 2.0 [WWW Document]. URL http://www.omg.org/spec/OCL/2.0/

Object Management Group Inc., 2001. UML 1.3 Superstructure Specification [WWW Document]. doi:10.1007/s002870050092

Oliva, G.A., de Maio Nogueira, G., Leite, L.F., Gerosa, M.A., 2012. Choreography Dynamic Adaptation Prototype.

Oriaku, C., Lami, I.A., 2012. Holistic View Aangles of Cloud Computing Services Provisions, in: 2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). IEEE, pp. 97–105. doi:10.1109/CyberC.2012.24

Palma, D., Spatzier, T., 2013. Topology and Orchestration Specification for Cloud Applications (TOSCA). Tech. report, Organ. Adv. Struct. Inf. Stand.

Parameswaran, A.V., Chaddha, A., 2009. Cloud Interoperability and Standardization. SETLabs Briefings - Infosys Technol. Ltd. 7, 19–26. doi:10.1016/j.soilbio.2010.08.001

Perepletchikov, M., Ryan, C., Frampton, K., Schmidt, H., 2008. Formalising Service-Oriented Design. J. Softw. 3, 1–14.

Perez, A.N., Rumpe, B., 2013. Modeling Cloud Architectures as Interactive Systems, in: Proceedings of the 2nd International Workshop on Model-Driven Engineering for High Performance and CLoud Computing (MDHPCL 2013) Co-Located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013). pp. 15–24.

Pérez, J., Diaz, J., Garbajosa, J., Alarcón, P.P., 2010. Flexible Working Architectures: Agile Architecting Using PPCs, in: European Conference on Software Architecture. pp. 102--117.

Perry, D.E., Wolf, A.L., 1992. Foundations for the Study of Software Architecture. ACM SIGSOFT Softw. Eng. Notes 17, 40–52. doi:10.1145/141874.141884

Rivera, J., Meulen, R. van der, 2013. Gartner Special Report Examines the Outlook for Hybrid Cloud [WWW Document].

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J., Ben-Yehuda, M., Emmerich, W., Galan, F., 2009. The Reservoir Model and Architecture for Open Federated Cloud Computing. IBM J. Res. Dev. 53, 1–11.

Ruiz, F., Verdugo, J., 2008. Guía de Uso de SPEM 2 con EPF Composer, Universidad de Castilla La Mancha: Grupo Alarcos. Ciudad Real, Spain.

Shadish, W.R.., Cook, T.D., Campbell, D.T., 2002. Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin Company, Boston, ME, USA.

Silva, E., Lopez, J.M., Pires, L.F., van Sinderen, M., 2008. Defining and Prototyping a Life-cycle for Dynamic Service Composition, Act4soc 2008: Proceedings of the 2nd International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing.

Sjøberg, D., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Vokác, M., 2003. Challenges and Recommendations When Increasing the Realism of Controlled Software Engineering Experiments, in: Springer Berlin Heidelberg (Ed.), Empirical Methods and Studies in Software Engineering, Experiences from ESERNET. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 24–38.

Stahl, T., Völter, M., Czarnecki, K., 2006. Model-Driven Software Development: Technology, Engineering, Management. John Wiley and Sons, Ltd, Chichester, England.

Sun, L., Dong, H., Ashraf, J., 2012. Survey of Service Description Languages and their Issues in Cloud Computing. Semant. Knowl. Grids (SKG), 8th Int. Conf. 128–135.

Svahnberg, M., Wohlin, C., 2005. An Investigation of a Method for Identifying a Software Architecture Candidate with Respect to Quality Attributes. Empir. Softw. Eng. 10, 149–181. doi:10.1007/s10664-004-6190-y

Talreja, Y., 2010. Agile Methodologies Accentuate Benefits of Cloud Computing (White Paper).

Tan, T., Li, Q., Boehm, B., Yang, Y., He, M., Moazeni, R., 2009. Productivity Trends in Incremental and Iterative Software Development, in: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009. pp. 1–10. doi:10.1109/ESEM.2009.5316044

Todoran, I., Hussain, Z., Gromov, N., 2011. SOA Integration Modeling: An Evaluation of how SoaML Completes UML Modeling. 2011 IEEE 15th Int. Enterp. Distrib. Object Comput. Conf. Work. 57–66.

Tsai, W., Sun, X., Balasooriya, J., 2010. Service-oriented Cloud Computing Architecture, in: Information Technology: New Generations (ITNG), 2010 Seventh International Conference on. IEEE, pp. 684–689.

Van Deursen, A., Klint, P., Visser, J., Others, 2000. Domain-specific Languages: An Annotated Bibliography. ACM Sigplan Not. 35, 26–36.

Vecchiola, C., Chu, X., Buyya, R., 2009. Aneka: A Software Platform for .NET-based Cloud Computing. Computing 30. doi:10.1.1.147.5672

Venkatesh, V., 2000. Determinants of Perceived Ease of Use : Integrating Control , Intrinsic Motivation , and Emotion into the Technology Acceptance Model. Inf. Syst. Res. 11, 342–365.

Vouk, M.A., 2008. Cloud Computing – Issues, Research and Implementations. CIT. J. Comput. Inf. Technol. 16, 235–246. doi:10.1109/ITI.2008.4588381

W3c, 2004. Web Services Architecture [WWW Document]. URL http://www.w3.org/TR/ws-arch/

Wettinger, J., Andrikopoulos, V., Leymann, F., 2015. Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments 348–358. doi:10.1007/978-3-319-26148-5

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer Heidelberg. doi:10.1016/S0065-2458(08)60338-1

Xu, D., 2010. Cloud Computing: An Emerging Technology. 2010 Int. Conf. Comput. Des. Appl. 1, V1-100-V1-104.

Yin, Y., Li, Y., Yin, J., Deng, S., Shi, W., 2009. Ensuring Correctness of Dynamic Reconfiguration in SOA Based Software. Serv. 2009 - 5th 2009 World Congr. Serv. 599–606. doi:10.1109/SERVICES-I.2009.29

Zhang, H. (Yulin), Urtado, C., Vauttier, S., Zhang, L., Huchard, M., Coulette, B., 2012. Dedal-CDL: Modeling First-class Architectural Changes in Dedal. 2012 Jt. Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit. 272–276. doi:10.1109/WICSA-ECSA.212.44

Zúñiga-Prieto, M., Insfran, E., Abrahao, S., Cano-Genoves, C., 2016. Incremental Integration of Microservices in Cloud Applications, in: 25th International Conference on Information Systems Development ISD2016.

# Apéndice A – Guías del método DIARy

En este apéndice se presenta la guía propuesta en el método DIARy para dar soporte a la actividad *Especificación de la integración del incremento*. En esta actividad el arquitecto describe cómo la arquitectura del incremento se integrará en la arquitectura actual, a través de especificar como cada uno de sus elementos arquitectónicos cambiará (impactará) a la arquitectura actual.

La guía que se presenta en este apéndice "*Guía para la Especificación de la Integración de Incrementos*" ha sido estructurada en forma de tabla. Las filas de la guía describen los pasos requeridos para completar la actividad *Especificación de la integración del incremento*, mientras que las columnas describen los posibles cambios que la integración de un elemento arquitectónico del incremento podría producir en la arquitectura actual. En la intersección entre filas y columnas la guía describe las acciones que el arquitecto debe tomar en cada paso de la actividad dependiendo del cambio arquitectónico que la integración de un elemento arquitectónico producirá en la arquitectura actual.

Para utilizar la guía, los arquitectos ejecutan cada uno de los pasos descritos en las filas de la guía. Por cada paso, y por cada elemento arquitectónico de la arquitectura del incremento, identifican en las columnas el posible cambio que el elemento arquitectónico producirá en la arquitectura actual; luego ejecutan la acción descrita en la intersección entre filas y columnas.

# A.1 Guía para la Especificación de la Integración de Incrementos

| | **Posibles Cambios a la *Arquitectura Actual*** | | | | |
|---|---|---|---|---|---|
| | **C1:** Si es que existe **un elemento** del **mismo *tipo** en la *Arquitectura Actual* | **C2:** Si es que existe **más de un elemento** del **mismo *tipo** en la *Arquitectura Actual* | **C3:** Si es que **no existen** elementos del **mismo *tipo** en la *Arquitectura Actual* | **C4:** Si es que el elemento de la *Arquitectura del **Incremento*** **reemplazará** a un elemento de ***tipo diferente** en la *Arquitectura Actual* | **C5:** Si es que la integración **eliminará** un elemento que **no** está **incluido** en la *Arquitectura del Incremento* |
| **Tarea 1.1:** Identificar los elementos arquitectónicos de la *Arquitectura Actual* que serán afectados por la integración del incremento | | | | | |
| **Paso 1:** Por cada elemento de la *Arquitectura del **Incremento*** identificar aquellos elementos que **ya existen** en la *Arquitectura Actual* | Copia el nombre de instancia del elemento de la *Arquitectura Actual* al elemento de la *Arquitectura del Incremento* | Copia el nombre de instancia del elemento de la *Arquitectura Actual* que se relacione con elementos similares a los que se relaciona el elemento arquitectónico en *la Arquitectura del Incremento* | Asignar al elemento de la *Arquitectura del Incremento* un nombre de instancia que consideres oportuno. | Copia el nombre de instancia del elemento de la *Arquitectura Actual* que será reemplazado al elemento de la *Arquitectura del Incremento*. | |
| **Paso 2:** Identificar en la *Arquitectura **Actual*** los elementos que serán **remplazados** o **eliminados** por la integración del incremento | | | | Copia el elemento arquitectónico a ser reemplazado (incluyendo su tipo, nombre de instancia y *RoleBinding* relacionados) desde la *Arquitectura Actual* a la *Arquitectura* del Incremento. | Copia el elemento arquitectónico a ser eliminado (incluyendo su tipo, su nombre de instancia y *RoleBinding* relacionados) desde la *Arquitectura Actual* a la *Arquitectura del Incremento* |
| **Tarea 1.2:** Especificar los cambios arquitectónicos (impacto) que la integración producirá en la *Arquitectura Actual*<br>**Importante:** Inicia por los elementos *Service Contract Use*, luego continua con los elementos *Participant Use* | | | | | |
| Por cada elemento de la *Arquitectura del **Incremento*** especificar el impacto que tendrá su integración en la *Arquitectura Actual* (valor de ***Architectural Impact***) | ***Delete***, si el elemento será eliminado de la Arquitectura Actual;<br>***Modify,*** si es que su implementación será modificada, y<br>***Reference*** en otro caso | | **Add** en el elemento que será añadido así como en sus *RoleBinding* relacionados | ***Delete*** en el elemento que será reemplazado así como en sus *RoleBinding* relacionados;<br>**Add** en el elemento que reemplazará al actual así como en sus *RoleBinding* relacionados | ***Delete*** en el elemento que será eliminado así como en sus *RoleBinding* relacionados |
| **Tarea 1.3:** Especificar la gestión de variaciones de demanda esperada para los servicios cloud | | | | | |
| Por cada elemento ***Service Contract Use*** de la *Arquitectura del **Incremento*** especificar la gestión de variaciones de demanda esperada | Si es que se espera que un servicio tenga altas variaciones de demanda, el valor de ***Elasticity Level*** deberá ser ***Medium*** o ***High***. Si es que no es posible retardar el procesamiento de las solicitudes un servicio el valor de ***Delay Level*** debe ser ***None***; mientras que los valores ***Low***, ***Medium*** o ***High*** se deberán utilizar dependiendo de la prioridad que se quiera dar a tiempo de espera para procesar el servicio. | | | | |

*el ***tipo*** de un elemento está ubicado luego de su nombre (luego de los dos puntos "**:**"); tanto en elementos arquitectónicos *Service Contract Use* como *Participant Use*

# Apéndice B– Material del experimento

En este apéndice se presenta el material experimental utilizado durante el experimento para validar el método DIARy. El material experimental está conformado por:

El apéndice B.1 presenta el cuestionario utilizado en una encuesta realizada antes iniciar el ejercicio del experimento.

El apéndice B.2 presenta el boletín utilizado en el experimento, el cual incluye la descripción del ejercicio, un anexo con la arquitectura actual del sistema utilizado en el ejercicio del experimento (Apéndice B.2.1), y un anexo con el modelo extendido de la arquitectura del incremento que es utilizado como hoja de respuestas del ejercicio (Apéndice B.2.2).

El apéndice B.3 presenta un cuestionario de evaluación utilizado en una encuesta realizada una vez finalizado el ejercicio del experimento.

## B.1 Encuesta pre-ejercicio: Proceso de Reconfiguración Dinámica e Incremental de Arquitecturas de Servicios – DIARy

**Responde el siguiente cuestionario:**

1. ¿Qué asignaturas relacionadas con el desarrollo de aplicaciones web/cloud y modelado de sistemas has cursado?

| Nombre | Créditos | Año (aaaa) |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

2. ¿Has modelado alguna vez utilizando diagramas UML? Sí ( ) No ( )

3. ¿Si has utilizado UML, qué tipo de diagramas has modelado?

Clases ☐
Interfaces ☐
Secuencia ☐
Máquina de estados ☐
Colaboración ☐
Componentes ☐
Arquitecturas orientadas a servicios ☐

Años de experiencia con UML: …….

4. ¿Has utilizado el Lenguaje de Modelado de Arquitectura Orientada a Servicio (SoaML) para modelar arquitecturas orientadas a servicios? Sí ( ) No ( )

5. ¿Has desarrollado servicios Web? Sí ( ) No ( )

6. ¿Has desarrollado servicios Web y los has desplegado

en el cloud? Sí ( ) No ( )

7. Valora tu nivel de experiencia en los siguientes aspectos.

1: si es que tu conocimiento únicamente teórico
2: si es que necesitas ayuda para aplicar principios básicos
3: si es que no necesitas ayuda para aplicar principios básicos
4: si es que necesitas ayuda para aplicar principios avanzados

5: si es que no necesitas ayuda para aplicar principios avanzados

| | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|
| Modelado UML | ○ | ○ | ○ | ○ | ○ |
| Modelado de Arquitecturas Software | ○ | ○ | ○ | ○ | ○ |
| SoaML | ○ | ○ | ○ | ○ | ○ |
| Integración de Servicios | ○ | ○ | ○ | ○ | ○ |
| Desarrollo de servicios Web | ○ | ○ | ○ | ○ | ○ |
| Despliegue de servicios Web en plataformas cloud | ○ | ○ | ○ | ○ | ○ |
| Reconfiguración de arquitecturas orientadas a servicio | ○ | ○ | ○ | ○ | ○ |

## B.2 Boletín

El Método DIARy

Dynamic Incremental Architectural Reconfiguration

Ejercicio para la Reconfiguración Dinámica e Incremental de Arquitecturas en la Integración de Servicios Cloud

Sistema Travel Reservations

ID _____

**Antes de Empezar**

Agradecemos tu participación en este ejercicio y te recordamos que para su correcta ejecución es importante que:

- Anotes tu identificador (ID) en la primera página de este documento y en la **Hoja de Respuestas de la Tarea 1**.
- No olvides anotar la hora de inicio y final de las tareas (hora y minutos).
- No inicies una Actividad o Tarea antes de finalizar la anterior.
- No realices cambios en las respuestas de una sección una vez que ha sido finalizada.

**Descripción del Ejercicio**

Durante el ejercicio realizarás tareas que facilitan la reconfiguración dinámica de la arquitectura de una aplicación cloud; en donde, la reconfiguración se produce por la integración de nuevos servicios cloud.

Una agencia de viajes tiene como objetivo proveer servicios de reservas que faciliten a los viajeros realizar reservas. Con este propósito, ha iniciado el desarrollo de una aplicación que incorporará funcionalidades de manera incremental. La versión inicial de la aplicación cloud incorpora funcionalidades que permiten a sus clientes gestionar sus reservas de vuelos. Para ello, la aplicación cloud de invoca servicios web provistos por las aerolíneas.

**Descripción del Incremento_1**

Ahora, la compañía requiere incorporar un servicio de reserva de taxi gratuito como valor agregado a su servicio de reserva de vuelos; por lo tanto, el Incremento_1 invocará servicios de reserva de sus socios de empresas de taxi. Luego de la integración del Incremento_1 cada vez que la agencia de viajes realice una reserva de vuelo, automáticamente generará la reserva del servicio de traslado para los pasajeros incluidos en la reserva.

El ***Anexo I*** presenta un extracto de la arquitectura de la versión inicial de la aplicación cloud donde el contrato de servicio *Reservation* (marcado con una elipse en la Fig. 1 del ***Anexo I***) será reemplazado como resultado de la integración del Incremento_1 en la arquitectura de servicios actual. La Fig. 2 ***del Anexo I*** presenta el protocolo de comunicación (orquestación) entre los participantes de un servicio de tipo *Reservation*. En la ***Hoja de Respuesta de la Tarea 1*** se presenta la arquitectura del incremento de software a integrar (*Arquitectura del Incremento*) la cual será completada durante la ejecución del ejercicio.

### Tarea 1: Especificar la Integración del Incremento

La actividad del método DIARy "*Especificar la Integración del Incremento*" es fundamental para ejecutar un proceso de reconfiguración planificado. En esta tarea especificarás cómo los elementos arquitectónicos de la Arquitectura del Incremento (documentada en la ***Hoja de Respuesta de la Tarea 1***) colaboran para cambiar (reconfigurar) la Arquitectura Actual.

**Nota:** La Hoja de Respuesta de la Tarea 1 incluye el Modelo de la Arquitectura del Incremento después de la aplicación del Perfil de Especificación DIARy.

Anota la hora de inicio (hh:mm): ____:____

**Tarea 1.1:** Identifica los elementos arquitectónicos de la *Arquitectura Actual* que serán afectados por la integración del incremento.

***Paso 1:***

- Por cada elemento de la *Arquitectura del Incremento* (Fig. 1 de la ***Hoja de Respuesta de la Tarea 1***) aplica el *Paso 1* de la *Guía para la Especificación de la Integración de Incrementos* ***(Anexo II)*** y determina su nombre de instancia (rellena los espacios **a**, **b**, **c** y **d** en la ***Hoja de Respuesta de la Tarea 1***).

  Considera que los **cambios que producirá** la integración del incremento en la *Arquitectura Actual son*:

  - ✓ **Reemplazará** al servicio (*ServiceContractUse*) *Reservation* con el servicio *FlightTaxiReservation*.

***Paso 2:***

- Identifica en la *Arquitectura Actual* (***Anexo I***) el elemento arquitectónico que será reemplazado como resultado de la integración y copia su nombre de instancia y tipo a los espacios **A** y **B** de la ***Hoja de Respuesta de la Tarea 1***.

  Considera que los **cambios que producirá** la integración del incremento en la *Arquitectura Actual son*:

  - ✓ **Reemplazará al** servicio (*ServiceContractUse*) *Reservation* con el servicio *FlightTaxiReservation*.

  **Toma en cuenta** que cuando se reemplaza un elemento arquitectónico la *Guía de Especificación de la Integración del Incremento* (Paso 2 y C4 en el ***Anexo II***) indica que se deberá copiar el elemento Service Contract Use y sus RoleBinding relacionados; sin embargo, estos ya han sido copiados (dibujados) en la ***Hoja de Respuesta de la Tarea 1***, por lo tanto, únicamente rellena los espacios **A** y **B**.

Anota la hora de fin (hh:mm): ____:____

Anota la hora de inicio (hh:mm): _____:_____

**Tarea 1.2:** Especifica los cambios arquitectónicos (impacto) que la integración producirá en la *Arquitectura Actual*.

- Por cada elemento de la *Arquitectura del Incremento* (Fig. 1 de la ***Hoja de Respuesta de la Tarea 1***) aplica la *Guía para la Especificación de la Integración de Incrementos* (Tarea 1.2 en el ***Anexo II***) y especifica cómo cambiará la *Arquitectura Actual* (rellena los espacios a*rchitecturalImpact* **i – x** en la ***Hoja de Respuesta de la Tarea 1***).

  Considera que los **cambios que producirá** la integración del incremento en la *Arquitectura Actual son*:

  - ✓ **Reemplazará** al servicio (*ServiceContractUse*) *Reservation* con el servicio *FlightTaxiReservation*.
  - ✓ **Añadirá** al participante *Taxi* (el servicio que implementa el participante *Taxi*).

Anota la hora de fin (hh:mm): _____:_____

Anota la hora de inicio (hh:mm): _____:_____

**Tarea 1.3**: Especifica la gestión de variaciones de demanda esperada para los servicios cloud.

- Aplica la *Guía para la Especificación de la Integración de Incrementos* (Tarea 1.3 en el ***Anexo II***) y especifica la gestión de variaciones de demanda esperada para el servicio ***FlightTaxiReservation*** (espacios 1 y 2 en la ***Hoja de Respuesta de la Tarea 1***)
  - ✓ ***Demanda esperada para el servicio:*** Se espera que el servicio *FlightTaxiReservation* tenga altas variaciones en su demanda y además deberá mantener los tiempos de espera para el procesamiento de la reserva de taxis.

Anota la hora de fin (hh:mm): _____:_____

**Tarea 2: Reconfigurar la Arquitectura**

Una vez que hayas especificado la integración de los incrementos de software la siguiente actividad en el método DIARy es Verificar la Compatibilidad del Incremento, es decir, verificar la compatibilidad entre las interfaces de los servicios a integrar. En este ejercicio **no incluimos tareas relacionadas con esta actividad y asumimos** que no existe incompatibilidad entre los servicios a ser integrados.

Esta tarea del ejercicio se corresponde a la actividad Reconfigurar la Arquitectura del método DIARy, en la cual se generan artefactos cloud que soportan tanto la implementación como el despliegue de los servicios cloud. **La reconfiguración dinámica de los servicios se produce durante el despliegue, por lo tanto, asumimos que los artefactos cloud de implementación ya han sido generados, completados y desplegados.**

**Descargar el material para la ejecución de la Tarea 2**

1. Descarga el fichero DIARy-HerramientasEjercicio.zip del portal Web que ha sido creado para el ejercicio *(Portal Internet del Ejercicio).*

http://thediarymethod.azurewebsites.net/Material

2. Extrae el contenido del fichero (Herramientas del Ejercicio) en la unidad c:\ u otra carpeta de tu elección.

3. Abre el fichero Editor de Artefactos Cloud.xlsm que se encuentra en la carpeta en la que has extraído las herramientas del ejercicio.

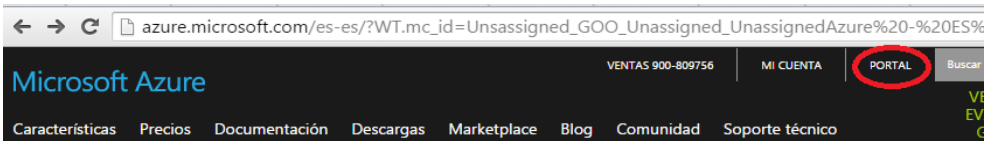
*Importante*: Permitir el uso de macros (**Habilitar edición y Habilitar contenido**). No cierres el fichero hasta finalizar todo el ejercicio.

4. Guarda el fichero Excel (guardar como en la misma carpeta del fichero original); incluye tu número ID al inicio del nombre (así: **ID***Editor de Artefactos Cloud.xlsm*).

5. Ingresa al *Portal de Administración de Microsoft Azure.*

http://azure.microsoft.com/es-es/

Clic en Portal (la clave de acceso está escrita en la pizarra)




***Importante:*** No cierres el portal hasta finalizar el ejercicio.

### Generar Scripts de Reconfiguración Dinámica

La reconfiguración dinámica de la arquitectura se obtiene modificando la configuración de conexiones entre servicios. En este paso se modifica la información de configuración de los servicios y se generan scripts (artefactos cloud) que modifican la configuración de conexiones entre servicios.

Anota la hora de inicio (hh:mm): ____:____

**Tarea 2.1:** Completa la información de los *EndPoint Expuestos* (puntos finales o puntos de acceso) por el nuevo servicio de orquestación.

***Paso 1:*** Utiliza el *Editor de Modelos de Artefactos Cloud* (fichero en Excel)

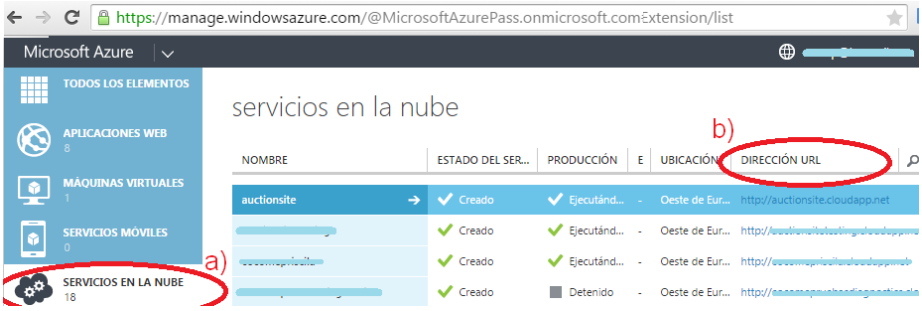
- Haz clic en el botón *Editor del Modelo de Artefactos Cloud*
- Selecciona la Aplicación Cloud ***Travel Reservations*** y haz clic el botón Aceptar.
- Selecciona el Servicio Cloud ***FlightTaxiReservationServiceContract*** (este servicio cloud implementa el nuevo servicio de orquestación).
- Haz clic en la pestaña *Aprovisionamiento y Despliegue Cloud.*
- Haz clic en el botón *Editar Proyecto* y actualiza la información del proyecto (***Nivel Elasticidad, Nivel Espera y Número de Instancias***) de acuerdo a tus respuestas en la **Tarea 1.3** (espacios 1 y 2 en la ***Hoja de Respuesta de la Tarea 1***). Luego de las modificaciones acepta los cambios.

  **Ten en cuenta** que el *Número de Instancias* debe ser **mayor a uno** para que un servicio sea capaz de soportar elasticidad.

- Anota el valor de la columna *Nombre de Despliegue en la Plataforma* (última columna cuya cabecera es de color verde). El nombre corresponde al servicio cloud utilizado para desplegar las orquestaciones de servicio en la plataforma cloud. *Nombre de despliegue*: ……..

***Paso 2:*** Utiliza el *Portal de Administración de Microsoft Azure*

- Haz clic la opción *SERVICIOS EN LA NUBE* en el menú lateral (marcado en la figura siguiente con la etiqueta **a**).




- Localiza el servicio en la nube cuyo *Nombre de Despliegue* anotaste en el paso anterior y haz clic en la DIRECCION URL (marcada en la figura anterior con la etiqueta **b**) que le corresponde.
- En la página Web que se ha abierto haz clic en el enlace BehaviorFlightTaxiReservation.xamlx el cual corresponde a la implementación en Visual Studio de la orquestación del servicio modelada en la Fig 2. de la ***Hoja de Respuesta de la Tarea 1***.
- Copia (CTRL-C) del explorador de Internet la dirección URL de la página Web que se ha abierto.

***Paso 3:*** Utiliza el *Editor de Modelos de Artefactos Cloud* (fichero en Excel)

- Haz clic en la pestaña ***EndPoints Expuestos*** y luego haz clic en el EndPoint ***epFlightTaxiReservationServiceContract***, edítalo (clic en el botón Editar) y en el campo Dirección pega (CTRL-V) la dirección que copiaste del explorador de Internet.

Completando estos pasos has actualizado el *Modelo de Artefactos Cloud* con información que indica la dirección en la que el servicio de orquestación expone sus operaciones.

Anota la hora de fin (hh:mm): ____:____

Anota la hora de inicio (hh:mm): _____:_____

**Tarea 2.2:** Actualiza la información de los ***EndPoint Invocados*** por el servicio que inicia la interacción (esto permitirá invocar al nuevo servicio de orquestación).

***Paso 1:*** Utiliza la ***Hoja de Respuesta de la Tarea 1***

- Identifica el rol que inicia la interacción (el rol que envía el primer mensaje) en el diagrama de secuencia (orquestación) de la Fig. 2.

  **Recuerda** que cada rol en el diagrama de secuencia es representado por una caja que incluye nombreRolDelParticipante **: nombreInterfaceImplementada**

- Anota el **nombre de la interface implementada** por el rol que inicia la interacción: ……..

***Paso 2:*** Utiliza el *Editor de Modelos de Artefactos Cloud* (fichero en Excel)

- Selecciona el Servicio Cloud cuyo nombre coincide con la interface que implementa el participante que inicia la interacción (el nombre lo anotaste en el paso anterior).
- Haz clic en la pestaña ***EndPoints Invocados***, haz clic en el EndPoint ***to_book_EndPoint***, edítalo (clic en el botón Editar) y actualiza la Dirección con la dirección del nuevo servicio de orquestación (CTRL-V). Si ya no tienes la dirección el en portapapeles selecciona el Servicio Cloud ***FlightTaxiReservationServiceContract*** y copia la dirección de su ***EndPoint Expuesto***, luego vuelve a seleccionar el Servicio Cloud cuyo nombre coincide con el que anotaste en el paso anterior.
- Asegurate que el impacto arquitectónico del EndPoint sea *Modificar*.

Anota la hora de fin (hh:mm): _____:_____

Anota la hora de inicio (hh:mm): ____:____

**Tarea 2.3:** Genera los scripts de Reconfiguración Dinámica.

***Paso 1:*** Utiliza el *Editor de Modelos de Artefactos Cloud* (fichero en Excel)

- Selecciona el servicio ***FlightReservationPlacer*** y haz clic en el botón ***Archivo de Configuración***.
- Selecciona el fichero de configuración actual (en Microsoft Azure es ServiceConfiguration.Cloud.cscfg). Este fichero se encuentra en la en la sub-carpeta Configuraciones/TravelAgency de la carpeta en la que has extraído las herramientas del ejercicio.
- Haz clic en el botón ***Generar***. Como resultado se creará la sub-carpeta *Generated*, la cual incluye el script de reconfiguración (transformation_ServiceConfiguration.Cloud) y el nuevo archivo de configuración (ServiceConfiguration.Cloud.cscfg). Este último fue obtenido aplicando el script de reconfiguración al archivo de configuración actual.

***Nota***: el script de reconfiguración generado no incluye operaciones que cambian el número de instancias de un servicio cloud debido a restricciones de nuestra cuenta en la plataforma cloud.

Anota la hora de fin (hh:mm): ____:____

## Reconfigurar Arquitectura de Aplicación

Luego de que los artefactos cloud de reconfiguración han sido generados la reconfiguración dinámica de la arquitectura se produce al desplegar los artefactos generados.

Anota la hora de inicio (hh:mm): _____:_____

### **Tarea 2.4:** Reconfigura la Arquitectura de la Aplicación

En esta tarea desplegarás el artefacto cloud generado en la tarea anterior y verificarás como se reconfigura la arquitectura de la aplicación.

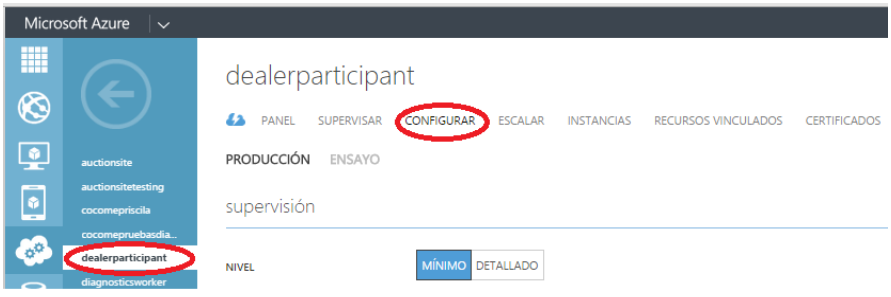
- Ejecuta la versión actual de la aplicación cloud ***Travel Reservations***. Para ello haz clic en la opción "***Aplicación: Travel Reservations***" en el portal de Internet creado para este ejercicio:

  http://thediarymethod.azurewebsites.net/worldTravels

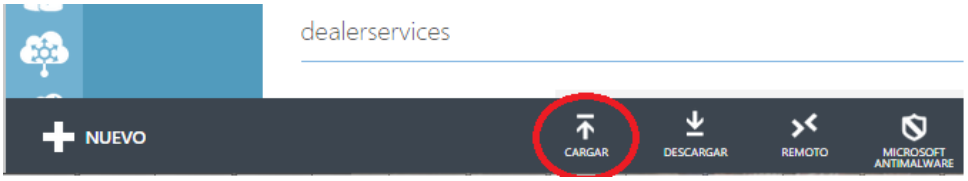
- Selecciona el *Centro de Negocios* que te hemos asignado en este ejercicio (está escrito en la pizarra). Ingresa toda la información que requiere la aplicación.

- Haz clic en ***Aceptar***, observarás que el resultado no incluye información de la reserva de taxi.

- En el *Portal de Administración de Microsoft Azure* haz clic en *SERVICIOS EN LA NUBE*, selecciona el servicio ***TravelAgencyParticipant*** correspondiente al *Centro de Negocios* al que te hemos asignado (ver pizarra) y haz clic en la opción *CONFIGURAR*.



- En la parte inferior de la pantalla haz clic en Cargar




- Carga el nuevo archivo de configuración. Utiliza el archivo de configuración Service-Configuration.Cloud.cscfg que fue generado y guardado en la sub-carpeta Configuraciones/ TravelAgency /Generated.

  Asegurate de marcar la opción "*Aplicar la configuración aunque uno o varios roles contengan una sola instancia"* en la ventana de carga. Luego haz clic en aceptar .

- Haz clic nuevamente en el botón Aceptar de la aplicación cloud y observarás que la respuesta incluye información de la reserva de taxi. Esto significa que la arquitectura de la aplicación ha sido reconfigurada e incluye el nuevo servicio de orquestación y el servicio del participante Taxi.

Anota la hora de fin (hh:mm): ____:____

## B.2.1 Anexo I: Sistema de Reservas – Modelo de Arquitectura de la Aplicación actual

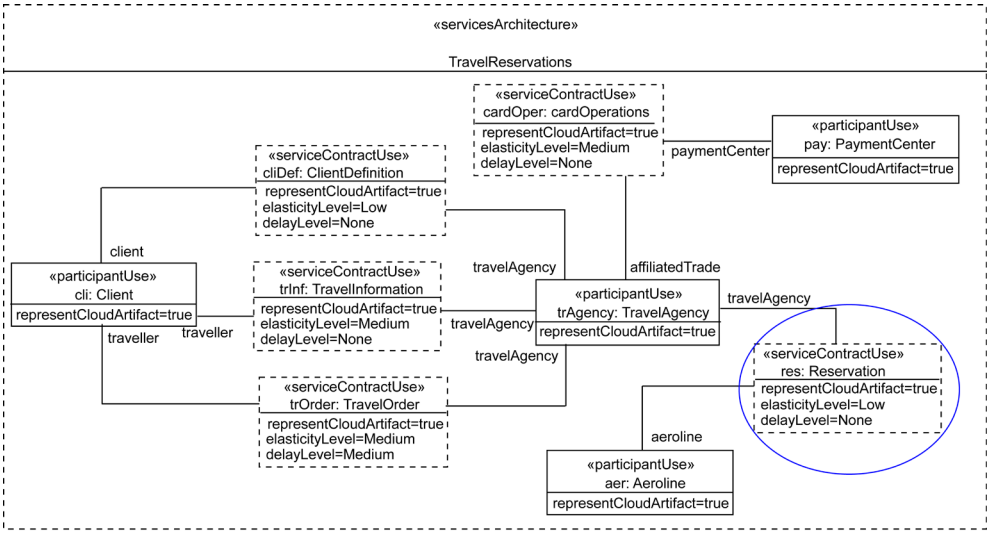



**Figura B-8-1** ***Modelo de Arquitectura de la Aplicación*** **utilizado en el experimento**

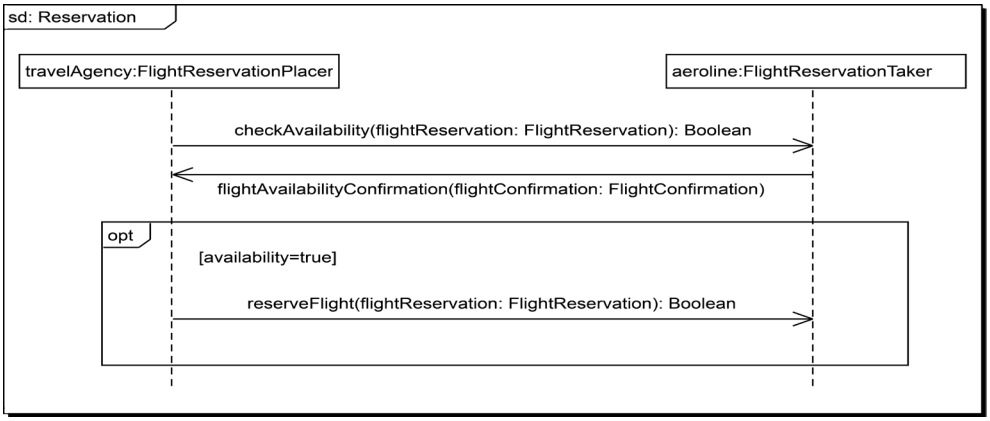



**Figura B-8-2** ***Modelo de Arquitectura de la Aplicación*** **– Protocolo de interacción utilizado en el experimento**

## B.2.2 Anexo II: Hoja de Respuestas de la Tarea 1 – Modelo de Arquitectura del Incremento


a - d: Espacios de respuestas de la Tarea 1.1, Paso 1(asignación de nombres de instancias)

A y B: Espacios de respuestas de la Tarea 1.1, Paso 2 (asignación de nombres de instancias)

i - x: Espacios de respuestas de la Tarea 1.2 (especificar impacto arquitectónico)

1 y 2: Espacios de respuestas de la Tarea 1.3 (requisitos de aprovisionamiento)

**Figura B-8-3 *Modelo Extendido de la Arquitectura del Incremento* – Hoja de respuesta del experimento**

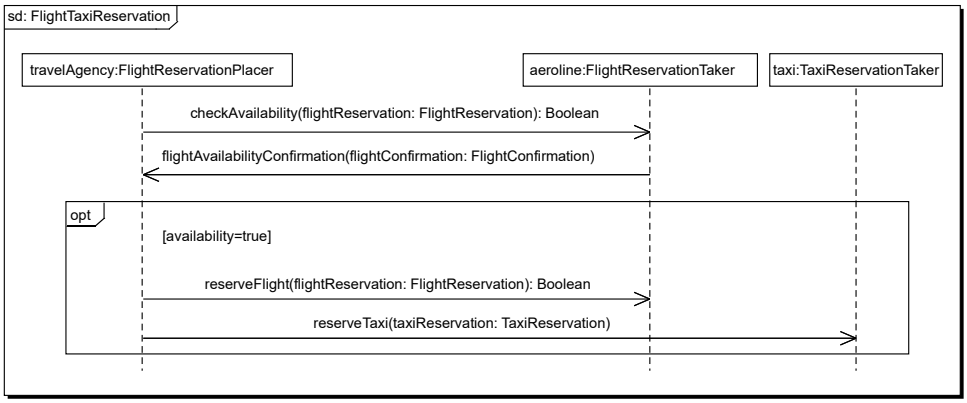



**Figura B-8-4 *Modelo Extendido de la Arquitectura del Incremento* – Protocolo de Interacción de la hoja de respuesta del experimento**

## B.3 Encuesta sobre el proceso de Reconfiguración Dinámica e Incremental de Arquitecturas de Servicios – DIARy

Para cada una de las preguntas marque, por favor, con una cruz sobre el círculo que se encuentra lo más cerca posible de su opinión.

| | | | | | | |
|---|---|---|---|---|---|---|
| **1.** El proceso para la reconfiguración dinámica e incremental de arquitecturas de servicios cloud me ha parecido complejo y difícil de seguir. | o | o | o | o | o | El proceso para la reconfiguración dinámica e incremental de arquitecturas de servicios cloud me ha parecido simple y fácil de seguir. |
| **2.** Creo que este proceso reduciría el tiempo y el esfuerzo requerido para integrar y reconfigurar dinámicamente arquitecturas de servicios cloud. | o | o | o | o | o | Creo que este proceso aumentaría el tiempo y el esfuerzo requerido para integrar y reconfigurar dinámicamente arquitecturas de servicios cloud. |
| **3.** De manera general, pienso que el proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es difícil de entender. | o | o | o | o | o | De manera general, pienso que el proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es fácil de entender. |
| **4.** No recomendaría el uso de este proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud. | o | o | o | o | o | Si recomendaría el uso de este proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud. |
| **5.** Los pasos a seguir para especificar los cambios arquitectónicos que producirá la integración de nuevos servicios y reconfigurar la arquitectura de servicios cloud actual son claros y fáciles de entender. | o | o | o | o | o | Los pasos a seguir para especificar los cambios arquitectónicos que producirá la integración de nuevos servicios y reconfigurar la arquitectura de servicios cloud actual NO son claros y son difíciles de entender. |
| **6.** De manera general, considero que el proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es útil. | o | o | o | o | o | De manera general, considero que el proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud NO es útil. |
| **7.** El proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es difícil de aprender. | o | o | o | o | o | El proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud es fácil de aprender. |

| | | | | | | |
|---|---|---|---|---|---|---|
| **8.** Pienso que sería fácil ser hábil usando este proceso. | o | o | o | o | o | Pienso que sería difícil ser hábil usando este proceso. |
| **9.** Creo que los estereotipos propuestos en este proceso para marcar los distintos elementos arquitectónicos son útiles para especificar la integración de servicios cloud. | o | o | o | o | o | Creo que los estereotipos propuestos en este proceso para marcar los distintos elementos arquitectónicos NO son útiles para especificar la integración de servicios cloud. |
| **10.** Creo que utilizar un Modelo de la Arquitectura del Incremento independiente del Modelo de la Arquitectura Actual facilita la identificación de los cambios arquitectónicos producidos por la integración de nuevos servicios. | o | o | o | o | o | Creo que utilizar un *Modelo de la Arquitectura del Incremento* independiente del *Modelo de la Arquitectura Actual* NO facilita la identificación de los cambios arquitectónicos producidos por la integración de nuevos servicios. |
| **11.** Fue fácil para mí entender cómo integrar las arquitecturas cloud utilizando este proceso. | o | o | o | o | o | Fue difícil para mí entender cómo integrar las arquitecturas cloud utilizando este proceso. |
| **12.** Creo que la generación automática de los scripts de reconfiguración dificulta la reconfiguración dinámica de arquitecturas de servicios cloud producida por la integración de nuevos servicios. | o | o | o | o | o | Creo que la generación automática de los scripts de reconfiguración facilita la reconfiguración dinámica de arquitecturas de servicios cloud producida por la integración de nuevos servicios. |
| **13.** El uso de este proceso mejoraría mi rendimiento en las actividades relacionadas con la reconfiguración dinámica de arquitecturas de servicios cloud. | o | o | o | o | o | El uso de este proceso NO mejoraría mi rendimiento en las actividades relacionadas con la reconfiguración dinámica de arquitecturas de servicios cloud. |
| **14.** De manera general, pienso que este proceso NO proporciona una manera eficaz para integrar y reconfigurar arquitecturas de servicios cloud. | o | o | o | o | o | De manera general, pienso que este proceso proporciona una manera eficaz para integrar y reconfigurar arquitecturas de servicios cloud. |
| **15.** En caso de necesitar soporte para las actividades de reconfiguración dinámica de arquitecturas de servicios cloud, tendría la intención de utilizar este proceso en el futuro. | o | o | o | o | o | En caso de necesitar soporte para las actividades de reconfiguración dinámica de arquitecturas de servicios cloud, NO tendría la intención de utilizar este proceso en el futuro. |

**Por favor, responde a las siguientes preguntas:**

¿Tienes alguna sugerencia de cómo hacer que este proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud sea más fácil utilizar?

¿Cuáles son las razones por las que tienes o no la intención de usar este proceso en un futuro?

Escribe por favor cualquier otro comentario que quieras hacer sobre el proceso de reconfiguración dinámica e incremental de arquitecturas de servicios cloud.

**Resumen**

El desarrollo de aplicaciones cloud sigue mayoritariamente un enfoque incremental, en donde la entrega incremental de funcionalidades al cliente cambia – o reconfigura – sucesivamente la arquitectura actual de la aplicación. Además, los proveedores cloud tienen sus propios estándares tanto para las tecnologías de implementación como para los mecanismos de gestión de servicios, requiriéndose soluciones que faciliten la construcción, integración y despliegue de servicios portables, la interoperabilidad entre servicios desplegados en diferentes proveedores cloud y la continuidad en la ejecución de la aplicación mientras su arquitectura es reconfigurada debido a la integración de los sucesivos incrementos.

Esta tesis doctoral pretende contribuir a cubrir las necesidades antes mencionadas proponiendo un método para la reconfiguración dinámica e incremental de arquitecturas de servicios cloud (DIARy), que se basa en los principios del desarrollo dirigido por modelos, y una infraestructura software que soporta sus actividades. El método y su infraestructura software facilitan la especificación de aspectos relacionados al impacto arquitectónico de la integración, implementación y aprovisionamiento de incrementos en entornos cloud. También permiten la automatización y la generación de artefactos que implementan la lógica de integración y orquestación de servicios, así como los scripts de aprovisionamiento, despliegue y reconfiguración dinámica específicos para distintos proveedores cloud. El método DIARy ha sido validado empíricamente mediante un experimento para evaluar su facilidad de uso, utilidad percibida, intensión de uso y rendimiento.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Tesis Doctoral



MMXI-MMXVII