Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla
Avda Reina Mercedes, s/n. 41012 SEVILLA
Fax : 95 455 71 39. **Tlf**: 95 455 71 39. **E-mail**: lsi@lsi.us.es

Una propuesta para el uso del paradigma guiado por modelos (MDE) para la definición y ejecución de procesos de negocio

TESIS DOCTORAL

Autor: D. Julián Alberto García García

Directores: Dra. Dña. María José Escalona Cuaresma
Dr. D. Manuel Mejías Risoto

Departamento de Lenguajes y Sistemas Informáticos

Universidad de Sevilla

Sevilla, Marzo 2015

A mis padres, Julián y Antonia,

por su cariño, dedicación, apoyo incondicional y altura de miras.

Si encomiendas a un hombre más de lo que puede hacer, lo hará.
Si solamente le encomiendas lo que puede hacer, no hará nada.

*— **Rudyard Kipling**, novelista y poeta.*
Gran Bretaña, 1865-1936

AGRADECIMIENTOS

La redacción de este apartado ha resultado ser una de las tareas más gratificantes y emotivas en la travesía que ha supuesto la realización de esta Tesis Doctoral; gratificante por culminar un trabajo intelectual complejo después de tanto tiempo de dedicación y esfuerzo físico-mental, y emotiva por recordar y agradecer públicamente tanto a tantas personas. En este sentido, son muchas las personas a las que quiero agradecer la realidad de esta tesis.

En primer lugar, a mi familia y con especial cariño a mis padres, Antonia y Julián, por ser los grandes pilares de mi vida y estar siempre a mi lado, por su apoyo, su ánimo, su cariño y su dedicación constante. Sin ellos, difícilmente hubiera llegado hasta donde he llegado. También a mis hermanos María Antonia y Pedro por su ayuda en lo que he necesitado. También a mi cuñado Javier y mi sobrino Francisco Javier por sus risas constantes. Además, a mi sobrina Blanca por traer, con su nacimiento, esta tesis debajo del brazo. También, a mis abuelos Antonio y Petra allá donde estén.

A mis tutores, María José y Manuel, por sus siempre acertados consejos, su confianza y su paciencia, ingredientes clave para culminar con éxito esta travesía. También por todas esas horas de dedicación, esfuerzo y entrega; algunas de ellas, sé que transcurridas durante situaciones personales adversas y difíciles de llevar en ciertos momentos. Lo hago también con una mención y con especial cariño hacia María José por todos estos años y por los venideros, por su amistad, por animarme en los momentos bajos, por haberme enseñado a caminar profesionalmente y por convertirse en más que una simple jefa, directora y amiga; por convertirse en parte de mi familia.

A todos aquellos compañeros (antiguos y actuales) del grupo de investigación Ingeniería Web y Testing Temprano (IWT2) del departamento de Lenguajes y Sistemas Informáticos con los que he compartido tantas horas de trabajo y que han estado siempre dispuestos a ofrecerme su ayuda, consejo y apoyo durante estos meses en cualquier aspecto de mi vida profesional y personal. En especial a Anabel, por transmitir su energía, simpatía y locura matutina; a Manolo, por su sentido del humor y su profesionalidad con el trabajo; a Francis por sus consejos y ánimos durante los meses de tesis; y a Alberto por su soporte durante la presentación.

A Laura y Miguel del Instituto Tecnológico de Aragón, por su apoyo y sobre todo por su ayuda para resolver cualquier problema antes y durante los viajes que hemos realizado juntos. En especial a Laura por todas esas charlas sobre nuestras tesis en las que nos hemos apoyado y motivado mutuamente.

A Jorge y Laura por esos buenos, aunque pocos, momentos en los que he podido distraer la mente en buena compañía.

A Lorena por los ánimos y consejos que me ha dado en esas largas charlas desde que nos conocemos. Además, por la amistad y cariño que hemos sabido mantener mutuamente a pesar de la distancia.

Agradecer, también, el apoyo y ánimo recibidos por parte de Beatriz y Şahin, miembros del servicio de Neurofisioterapia Biofuncional del Hospital Universitario Virgen Macarena por ayudarme a estar físicamente a la altura y por su ánimo e interés constante por conocer los avances de este trabajo.

A la fundación UNIVERSIA por haber apostado y creido en este trabajo, otorgándome por ello una de las tres nuevas becas predoctorales que ofrecía a nivel nacional (España) dentro de su programa de becas del año 2013 para la promoción y desarrollo profesional de personas con discapacidad para la obtención del título de doctor. Además, agradezco que, en 2014, la fundación UNIVERSIA me reconociera de nuevo la calidad y excelencia de los resultados derivados de esta tesis con la renovación de la beca otorgada el año anterior.

Posiblemente haya olvidado mencionar a muchos otros que aportaron directa o indirectamente en el desarrollo de este trabajo. Vaya desde aquí mi agradecimiento más sincero para todos ellos.

A todos, mil gracias, este trabajo difícilmente podría haber sido posible nunca sin vuestro apoyo.

Julián

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE EXPRESIONES

RESUMEN

La situación económica actual está originando que muchas empresas incorporen mecanismos y protocolos que hasta ahora no habían tenido en cuenta para aumentar su productividad y mejorar la calidad de sus productos y/o servicios , sin que ello suponga un incremento de sus costes de producción o desarrollo.

Una de las estrategias de gestión más utilizadas para alcanzar los objetivos anteriores es BPM (*«Business Process Management»*) [Van-der-Aalst 2004a]. A lo largo de la última década, BPM se antoja y afianza cada vez más en esta línea. Esta es una de las conclusiones obtenidas de varios estudios de investigación, como el publicado en [Malinova *et al.* 2013], en el cual los autores concluyen que implantar BPM dentro de una organización mejora el conocimiento general de la misma, de su funcionamiento interno – lo que se conoce como *«Know-how»* de la organización –, y de sus procesos de negocio.

Todo este conocimiento organizacional de la empresa permite optimizar, controlar y medir de una manera más eficaz la ejecución de cada proceso, lo que, en última instancia, supone a corto y medio plazo un incremento de su nivel de competitividad frente a su competencia [Trkman 2010].

Asimismo, multitud de entidades y organismos han motivado en la última década la aplicación de BPM como estrategia de gestión y actuación de mejora interna. Es el caso, por ejemplo, del PMI («*Project Management Institute»*) que es una organización internacional sin ánimo de lucro afincada en Estados Unidos que asocia a profesionales relacionados con la gestión de proyectos y fomenta la dirección de proyectos desde una perspectiva centrada en procesos organizacionales [PMI 2008]; la Universidad Carnegie Mellon con su propuesta CMMi (*«Capability maturity model integration»*) [Chrissis *et al.* 2011], en la que define modelos de madurez para la mejora y evaluación de procesos; y la organización ISO con algunas de sus normas, como por ejemplo la ISO 9001:2008.

Siguiendo estas recomendaciones y con el propósito de mejorar su competitividad, la industria del software está comenzando a adoptar BPM como mecanismo para controlar y definir la construcción y gestión de software.

Sin embargo, la aplicación de la gestión de procesos dentro de las empresas enmarcadas en el negocio software es difícil y costosa de implantar adecuadamente debido a las características del proceso de software, es decir, su constante evolución, incorporación de nuevos ciclos de vida, nuevas tecnologías y grandes equipos de desarrollo y en muchos casos multidisciplinares, entre otros muchos aspectos [Ruiz-González *et al.* 2004]. En consecuencia, es muy frecuente que la implantación del proceso global de mejora continua que fomenta BPM se circunscriba únicamente en la práctica a la definición formal de los procesos de software, realizando su posterior ejecución y orquestación – es decir, la gestión centralizada y coordinada de eventos durante la

ejecución del proceso– de una manera manual y desconexa por parte de cada rol que participa en ellos. Todo esto hace que el seguimiento, control y medición de los procesos de software se conciba como una tarea particularmente costosa y compleja.

El trabajo de tesis presentado en este documento, se ve motivado por los problemas planteados anteriormente dentro de las organizaciones software para ejecutar y orquestar sus procesos. Sin embargo, aunque las razones principales que han originado este trabajo la se enmarcan dentro del contexto de las organizaciones software, la flexibilidad de la propuesta aquí desarrollada ha propiciado su aplicación y extrapolación a otros ámbitos, por ejemplo, en el área de la salud y, más concretamente, en el área de gestión de procesos clínicos [García-García *et al.* 2015b].

Esta Tesis Doctoral se elabora como propuesta para resolver los problemas anteriores aprovechando las ventajas del paradigma MDE y tomándolo como vector director de la propuesta, con la finalidad de simplificar el mantenimiento de procesos software y hacerlos más efectivos. Los objetivos de la propuesta pasan por establecer los modelos o lenguajes específicos de dominio necesarios para tratar la definición y, ejecución y orquestación, de procesos software en un entorno de producción. Además, dicha propuesta contempla también la definición de una serie de protocolos sistemáticos de transformación entre los modelos.

Como finalidad final, esta tesis pretende que todos esos modelos, y sobre todo las técnicas para describirlos, sean instructivos y cognitivamente entendibles por usuarios con un perfil no técnico. El objetivo es fomentar y propiciar la participación del usuario final como entes imprescindibles, tanto para la definición como para la validación final de los modelos.

El cuerpo de la tesis pues, se cimenta sobre un conjunto de metamodelos teóricos y de mecanismos sistemáticos de derivación entre ellos. Esta estructura teórica encuentra su traslación práctica en el marco de trabajo PLM$_4$BS (*«Process Lifecycle Management for Business-Software»*): una herramienta CASE (*«Computer Aided Software Engineering»*, Ingeniería de Software Asistida por Computadora) que proporciona soporte para gestionar el ciclo de vida del proceso software en proyectos reales.

Desde la perspectiva del producto, PLM$_4$BS se ha inspirado en el paradigma PLM (*«Product Life cycle Management»*) [Stark 2011] para dar su propia visión de aplicación al contexto del software. A pesar de que el software puede ser considerado como una entidad abstracta e intangible, es un producto en sí mismo y, de forma similar a los productos industriales, también está altamente relacionado con un ciclo de vida ingenieril.

En conclusión, Tesis Doctoral plantea una solución a un problema específico: establecer dentro de las organizaciones software, mecanismos eficaces, sistemáticos y automáticos que posibiliten la ejecución y orquestación de procesos software a partir de su definición, con el propósito de gestionar el ciclo de vida del desarrollo de productos software y de los procesos que intervienen durante la gestión de estos productos.

Este problema, identificado en un estudio del estado del arte de las tendencias actuales, se resuelve en un marco teórico que se implementa posteriormente en la herramienta CASE denominada PLM$_4$BS, la cual ha sido que testada y validada en diversos proyectos reales de diferentes ámbitos de negocio.

SUMMARY

Introduction

Today's world economic situation has become an opportunity for many organizations to implement new mechanisms and protocols, unused for them before, in order to increase their productivity and quality of products and improve their services, without increasing production costs.

One of the most widely used management strategies to achieve the above goals is BPM (*«Business Process Management»*) [Van-der-Aalst 2004a]. In fact, over the last decade, BPM seems increasingly oriented towards this direction. This idea is one of the conclusions derived from the analysis of several research studies, such as [Malinova *et al.* 2013], where authors conclude that the implementation of BPM improves the overall knowledge, the inner jobs – what is known as *«Know-how»* –, as well as business processes at any organization.

All this structural knowledge allows optimizing, monitoring and measuring effectively the execution of each process, which leads in the short and medium-term to raise the company's competitiveness against competitors [Trkman 2010].

In addition, over the last decade, many international bodies and organizations have recommended that companies should formally manage their business processes in order to improve their quality and efficiency and also increase productivity and competitiveness in relation to other companies in the same business area. For instance, PMI («*Project Management Institute*», an American international nonprofit organization, which associates professionals related to project management) promotes project management from an organizational processes-based perspective [PMI 2008]; University of Carnegie Mellon with CMMi (*«Capability maturity model integration»*) [Chrissis *et al.* 2011], which defines maturity models for process assessment and improvement; and ISO organization with some of the standards, such as ISO 9001:2008.

Following these recommendations, the software industry is currently applying BPM as a mechanism to control and define how to build and manage software in order to improve competitiveness.

However, BPM in the Software Process Engineering context is not as simple as it seems to be because of the inherent properties of software processes. In [Ruiz-González *et al.* 2004], the authors identify and describe the properties that characterize software processes, such as they are constantly evolving, they usually incorporate new lifecycles and technologies, or they are strongly influenced by many work teams, among other aspects.

The aforementioned reasons have caused that software companies usually focus on defining their processes, but each involved role performs orchestration – i.e., centralized and coordinated management of events during the process execution – and execution processes in an independent and manual way. For this reason, software processes maintenance, evolution, monitoring and measurement become difficult tasks.

This PhD thesis is motivated by the problems posed above in software organizations to execute and orchestrate business processes. However, although these motivations are framed in software organizations, we have proposed a flexible solution that can be applied to other areas, for instance, to Health environment, and more particularly, to clinical processes management [García-García *et al.* 2015b].

The purpose of this work is to solve the cited problems and facilitate of software processes maintenance by taking advantage of the MDE (*«Model-Driven Engineering»*) [Schmidt 2006] paradigm. We aim to define models or domain specific languages needed to model and execute processes and establish systematic transformation protocols among those models.

In addition, we tend that all these models and techniques used to model, are intuitive and easily understandable cognitively by non-experts users without technical background. This feature allows these users may participate in the final validation of models.

In this context, this project theoretically defines a set of metamodels and systematic transformation mechanisms among them. This theoretical framework has been implemented in practice as PLM_4BS (*«Process Lifecycle Management for Business-Software»*) framework, a CASE (*«Computer Aided Software Engineering»*) tool that provides support to software process lifecycle management in real projects.

From a product perspective, PLM_4BS is inspired by the PLM (*«Product Life cycle Management»*) [Stark 2011] paradigm, as PLM_4BS provides its own view of the PLM‘s application in software context. In spite of being an intangible abstract entity, software is regarded as a product itself and it is related to an engineering lifecycle as an industrial product.

Finally, this PhD thesis provides the approach to a particular problem happening at software organizations: the need for effective, systematic and automated mechanisms to execute and orchestrate software processes from their own definition in order to manage the software-product lifecycle, that is, development and management.

This problem is based on a *state-of-the-art* study of current trends and is covered with a theoretical framework that is subsequently implemented in a CASE tool called PLM_4BS. This tool has been tested and validated in several real projects from different business areas.

Related Work

Software process management is a well-known topic in the Software Engineering research context that has been studied during the last twenty years. Researchers in this area have addressed the problem of describing, executing and using it, focusing on the idea that a software process is a specific kind of software.

As regards software companies, they often manage their processes by means of two differentiated strategies. The first one is more practical and it is based on utilizing powerful tools (known as BPMS, *«Business Process Management Suite»*) to manage the BPM lifecycle. The second one is more theoretical and aims to find the best mechanisms and techniques to define, execute, control and implement these software processes, their maintenance and applicability.

This PhD thesis faces on the second trend. Basically, it analyzes the *state-of-the-art* of the most recent model-based proposals dealing with software processes modeling, execution and orchestration in order to propose a solution. In addition, this study will allow establishing the maturity level of this research topic and stating some improvements that will be useful to define our proposal.

We have found too few literature reviews and surveys in this area [Zamli 2001; Zamli *et al.* 2004; Henderson-Sellers et al. 2005; Bendraou *et al.* 2010], although this topic has been discussed and studied for many years. Consequently, a systematic literature review [García-Borgoñón *et al.* 2013] has been carried out by two researchers; the PhD student leading this disssertation together with other member of the Web Engineering and Early Testing (IWT2) research group. We have deeply studied these languages and their abilities and we have realized that different proposals based on different technologies have appeared along the years: Petri nets, rules, programming languages, graph theory, and more recently, UML.

At present, this interest on UML is more evident than ever. In fact, the growing curiosity for model-based proposals to manage software processes has recently entailed the generation of a large number of proposals [Chou 2002; Di Nitto 2002; Riesco *et al.* 2003; Bendraou *et al.* 2005; Combemale *et al.* 2006; Wu *et al.* 2006; Martinho *et al.* 2008; Ferreira *et al.* 2011; OMG 2008a; Bendraou *et al.* 2007a; Ellner et al. 2010; Koudri *et al.* 2010; Ellner *et al.* 2011].

After carrying out this comparative study, we can conclude that, even though there are many model-based proposals, none of them is mature enough to comply with the commitment to follow. SPEM2.0 could be the solution, but it is discarded by its complexity and non-executability. Regarding executability, many researches have been conducted to solve the main drawback it involves (e.g. xSPEM). For a software process language to succeed, it will be mandatory to fill in the gap between the process description and its execution in a real environment.

Finally, we have also identified some aspects that appeared until now in most of the studied proposals, such as usability in enterprise environments, mechanisms to carry out continuous improvement, mechanisms to establish business rules and elements for software process orchestration.

Challenges and Goals

We have clearly identified the problems to solve, once the related proposals have been analyzed.

The first one deals with the characteristics of software processes. These characteristics make automation be a complex task [Ruiz-González *et al.* 2004] involving

high cost of deployment. Consequently, software companies usually centers on defining their processes, but each involved role performs orchestration and execution processes in an independent and manual way. This causes that software processes maintenance, evolution, monitoring and measurement become difficult tasks.

The second one concerns the fact that there are not many CASE (*«Computer-Aided Software Engineering»*) tools that allow the practical application of model-based BPM in real projects. Business processes management from a model-based perspective should be properly tackled, especially when processes are critical to the organization. In these cases, the cost of the project or the implementation of this solution can be significantly more expensive due to the incorporation of new protocols, techniques and models for handling business processes. It is therefore necessary to provide practical and agile-based CASE environments to systematize or automate tools in the framework.

The purpose of this PhD thesis is to solve the cited problems and facilitate software processes maintenance taking advantage of the MDE paradigm. We aim to define models or domain specific languages needed to model and execute processes and establish systematic transformation protocols among these models. However, although these motivations are framed in software organizations, we have proposed a flexible solution that can be applied to other areas, for example, to Health environment, and more particularly, to clinical processes management [García-García *et al.* 2015b].

The concrete objectives of this PhD thesis are listed below:

1. **Defining a BPM lifecycle** that provides the necessary theoretical foundations to enable the software process modeling, execution, orchestration and process measurement. This objective is addressed in Section 4 of Chapter III.

2. Defining a (*i*) **Software Process Definition Metamodel**, with which models software processes according to current standards, and (*ii*) **Software Process Execution and Orchestration Metamodel**, with which define the execution context of process. Chapter IV describes formally both metamodels.

3. Defining a **systematic derivation mechanism** with which generates the execution and orchestration model from the software process definition model. In addition, defining a mechanism with which generates the executable version from the execution and orchestration model. Chapter V addresses this objective.

4. Developing an applicable version of our metamodels and systematic derivation mechanisms. This objective entails: (*i*) defining **UMLProfiles** or domain specific languages for both metamodels; and (*ii*) implementing a **CASE tool** that supports our framework, i.e., PLM_4BS. This objective is achieved in Chapter VI.

5. **Evaluating the results** obtained in a real environment after a successful use. Chapter VI is focused on meeting this objective.

Contributions

This PhD thesis culminates in a serie of contributions that pose a novel solution to support software process definition, execution and orchestration, framed in the model-driven engineering paradigm. Below, these contributions are detailed:

Firstly, a study of the *state-of-the-art* has been carried out on based-model languages to support the software process definition, execution and orchestration. The study presented in this work is part of a full paper [García-Borgoñón *et al.* 2013], which has been carried out together with another PhD member of IWT2. This paper presents a comprehensive and systematic literature review related to software process modeling languages. We have also conducted a review of proposals to manage the business process lifecycle that has been included in our paper *«A model-based Approach for Software Processes Modelling in PLM$_4$BS»* [García-García *et al.* 2015a] and is now being considered by the *«Enterprise Information Systems»* journal.

Secondly, this work provides a complete and formal theoretical framework for the generation of executable code from the process software definition model. This framework is composed of metamodels, described formally in Chapter IV, and transformation rules, analyzed in Chapter V.

On the one hand, this dissertation identifies a metamodel for software process definition and another one for software process execution and orchestration, both defined in compliance with the guidelines and recommendations of ISO/IEC TR 24744. The former has been included in our paper *«A model-based Approach for Software Processes Modelling in PLM$_4$BS»* and the latter in *«A Model-Driven framework for Product Lifecycle Management Process and Orchestrating Executing»* [García-García *et al.* 2014c], which has been submitted to the *«ACM Transactions on Software Engineering and Methodology»* journal. In addition, these papers determine the concrete syntax of each metamodel through UML profiles.

On the other hand, we introduce systematic transformation rules [García-García *et al.* 2014c] that establish relationships among metamodels. All these relationships as well as systematic rules themselves are original contributions of this work. In particular, we have formally specified rules to generate the software process execution and orchestration model from its definition model, and WS-BPEL executable code from the process execution and orchestration model. These rules have been implemented with QVT and MOFM2T, respectively.

Thirdly, this thesis provides a CASE tool named PLM$_4$BS that supports the characterized theoretical framework. This tool allows instancing metamodels and running systematic rules in a practical, usable and friendly environment based on Enterprise Architect (EA).

Finally, our approach has advantages over the proposals studied in Chapter II, since it provides mechanisms to define business rules to support software process continuous improvement and orchestrate the process.

Results, future work and conclusions

Both effective Product Lifecycle Management (PLM) and efficient Business Processes Management (BPM) are crucial in any company in order to reduce costs, maximize profits and increase productivity and competitiveness. These concerns have been greatly promoted by good practice guides (such as PMBOK and PRINCE, among others) as well as standards (e.g., CMMI or ISO 9001:2008).

Nonetheless, PLM is not completely consolidated in software organizations, since very often this concept is related to industrial manufacturing processes. Besides, applying BPM is not a simple task because management and software development processes are characterized, among others, by the following aspects: (*i*) they are constantly evolving; (*ii*) they usually incorporate new lifecycles and technologies, they frequently comprise several iterations and produce different versions of software products; (*iii*) they are complex because they are strongly influenced by many work teams; and (*iv*) they often rely on communication, coordination and cooperation among different frameworks and development technologies as well as on the different roles they play.

For these reasons, software companies usually focus on defining their processes, but each involved role performs orchestration and execution processes in an independent and manual way. This fact makes software processes maintenance, evolution, monitoring and measurement become a difficult task.

This study proposes a MDE-based solution that would help manage the conceptual complexity of the software process lifecycle. For this purpose, it defines a model-based approach for software processes modeling, execution and orchestration at organizations and makes monitoring easier after being modeled.

Our model-based approach is inspired by PLM applied to business software and it is framed in a new continuous improvement BPM lifecycle. It is called PLM_4BS (*«Process Lifecycle Management for Business-Software»*), that is, a tool-based and a MDE-based framework for software processes modeling, orchestration, execution and monitoring in actual enterprise environments. Our metamodels are simple enough to enable using PLM_4BS in real environments. Nonetheless, they allow defining processes of considerable complexity. We have suggested a straightforward language for two reasons. On the one hand, our language is simple in order to apply PLM_4BS to different business environments to later check and validate it. This provides us with valuable feedback to improve our proposal. In this sense, we are successfully applying PLM_4BS to a present project. On the other hand, we propose a simple language in order to facilitate the insertion of MDE-based mechanisms to PLM_4BS and reduce user's cognitive overload when he/she uses our software processes modeling language (this usability aspect has a major influence on user's efficiency).

Regarding future work, it is important to define mechanisms for continuous improvement. This preoccupation has been greatly promoted by good practice guidelines (such as PMBOK and PRINCE, among others) as well as standards (e.g., CMMI and ISO 9001:2008). In addition, this interest is essentially motivated by the growing need to build reliable and complex software systems to market in a short period of time. Consequently, our future work leads towards answering how the software process monitoring can be included in PLM_4BS by means of the MDE paradigm.

Moreover, executable process models can be used with different aims: to coordinate agents, to enforce artifacts routing among the stages of the process and to ensure rules and constraints integrity and process deadlines. They can also be effectively helpful since they can be utilized for simulation and testing. Simulation results can be conceived as a basis for important improvement decisions such as resource allocation, deadlock identification, estimation of the project duration and many other aspects that have a direct

impact on the process and thus, on the quality of the delivered software. In light of this, we propose to investigate new research lines in order to extend the functionality of PLM_4BS and also explore its applicability to other contexts.

On the one hand, two examples of future lines of work we will: to include new concepts in our business process modeling metamodel in order to simulate and testing processes and to provide a wider range of logical operators for control nodes.

On the other hand, this PhD thesis is focused on providing a BPMS-based solution for applying PLM_4BS in real environments. However, it would be interesting to extend PLM_4BS to other contexts that are not purely software, for example, to aeronautics, where using existing PLM software is very common in the market (such as the Siemens's PLM solution or Dassault Systèmes's PLM solution, among others) in order to control and manage the manufacturing process of a product.

If we get to apply PLM_4BS to these environments, aeronautical companies would not be forced to use a specific commercial PLM tool. In addition, these companies might be able to migrate their manufacturing processes among different PLM solutions by means of MDE mechanisms (which will be developed in PLM_4BS). We have identified this need during a collaborative project named CALIPSOneo [Escalona *et al.* 2013]) developed between IWT2 and Airbus Military (EADS CASA).

Capítulo I INTRODUCCIÓN

Este capítulo describe cuál es el contexto del trabajo desarrollado en esta Tesis Doctoral. Para ello, en la primera sección se presenta una breve introducción para centrar el trabajo. A continuación, las secciones segunda y tercera describen cuál es la estructura de este documento y unas breves conclusiones del capítulo, respectivamente.

1. Introducción

En cualquier período o ciclo de la economía, y más en períodos de crisis económica como el actual, resulta interesante que las organizaciones implementen mecanismos para mejorar su competitividad frente a otras organizaciones de su mismo ámbito de negocio.

La competitividad en el entorno empresarial exige una capacidad significativa de asimilación, adaptación y tolerancia a los cambios, y, en este sentido, las organizaciones deben desarrollar una estructura ágil y flexible que pueda reaccionar rápidamente a los cambios para intentar garantizar su supervivencia y competitividad en sus mercados. En este contexto, la innovación y la reinvención son características clave para el éxito.

En esta línea, otro aspecto interesante para que las organizaciones sobrevivan en este clima económico, podría pasar por compatibilizar la reducción de costes con la mejora en la calidad de los productos y servicios desarrollados y ofrecidos, así como en el incremento de la productividad de sus empleados durante el desempeño al ejecutar los procesos de negocio de la organización. En el contexto de esta tesis se entiende por procesos de negocio como *aquel conjunto de tareas relacionadas de manera lógica que se deben realizar para entregar valor a los clientes o para cumplir ciertos objetivos estratégicos* [Strnadl 2006].

Esta Tesis Doctoral parte de la hipótesis de que una gestión adecuada de los procesos de negocio de una organización puede contribuir a alcanzar los objetivos organizacionales expuestos anteriormente, de tal forma que a través de un modelo de gestión de procesos de negocio maduro, las organizaciones pueden poner en práctica sus estrategias de negocio para mantener una ventaja competitiva frente a su competencia.

La gestión de procesos de negocio se conoce frecuentemente por su acrónimo inglés BPM (*«Business Process Management»*). En esta Tesis Doctoral seguiremos la definición establecida por Van-der-Aalst en [Van-der-Aalst 2004a] y completada por Netjes en [Netjes *et al.* 2006] para este concepto (consultar Tabla I.1).

Tabla I.1. Definición *«BPM»* [Van-der-Aalst 2004a] [Netjes *et al.* 2006]

«BPM («Business Process Management») incluye aquellos métodos, técnicas y herramientas de soporte al ciclo de vida de los procesos de negocio, el cual a su vez incluye el diseño, aprobación, gestión y análisis de los procesos de negocio operacionales que involucran a personas, organizaciones, aplicaciones, documentos y cualquier otra fuente de información.»

BPM se antoja y afianza cada vez más como una posible estrategia para reducir costes y mejorar, a través de un ciclo de vida de mejora continua, los procesos de negocio que vertebran cualquier organización [Hill *et al.* 2007a, b; Richardson *et al.* 2013]. De hecho, el estudio presentado en [Malinova *et al.* 2013] sobre los resultados de un conjunto de entrevistas realizadas a expertos BPM de varias compañías revela que, las razones para adoptar BPM se pueden agrupar en tres necesidades primordiales: comprender y asimilar el conocimiento intrínseco de los procesos de negocio (es decir, el *«Know-how»* de la organización); conocer el desempeño de sus trabajadores durante la ejecución de los procesos; y por último, controlar y medir los procesos.

Gestionar de manera eficaz estos aspectos posibilita, entre otros, la optimización de los procesos y la localización de redundancias, lo que podría implicar una reducción de costes, una mejora en la calidad y eficiencia durante el desempeño de dichos procesos, y un aumento en la productividad. Esto, en última instancia, redunda en una mejora del parámetro ROI (*«Return On investment»*) a través de la reducción de los costes de producción. Es importante destacar que los procesos de negocio son la clave para contribuir a la ventaja competitiva de la organización [Trkman 2010].

Por otra parte, durante los últimos años multitud de organismos internacionales proponen y fomentan que las organizaciones lleven a cabo una mejora continua de sus procesos de negocio como forma efectiva de gestión. En este contexto se pueden considerar guías internacionales de buenas prácticas para la dirección de proyectos como PMBOK [PMI 2008] (*«Project Management Body of Knowledge»*) o Prince2 [SOB 2009] (*«PRojects IN Controlled Environments 2»*); estándares de referencia en el marco de las TIC (Tecnologías de la Información y la Comunicación), como CMMi [Chrissis *et al.* 2011] (*«Capability maturity model integration»*) o ITIL [ITIL 2014] *(«Information Technology Infrastructure Library»*), así como estándares de referencia generalistas como ISO 9001:2008 [ISO/IEC 2008a].

Respecto a las organizaciones software, se hace palpable la necesidad de implementar procesos bien definidos, con el objetivo no sólo de aumentar su nivel de madurez, sino también mejorar la forma en que desarrollan sus productos y, por tanto, su calidad y competitividad [Fuggetta 2000]. En este sentido, los procesos de software y métodos de modelización de procesos software fueron y siguen siendo el epicentro de las preocupaciones dentro de este tipo de empresas. Estas preocupaciones han sido en gran medida impulsadas y promovidas por guías de buenas prácticas y estándares de referencia (como los mencionados anteriormente). Este interés también está motivado por la creciente necesidad de construir sistemas software cada vez más fiables y complejos,

dentro de un entorno con cada vez mayores restricciones de coste en cuanto a tiempo y esfuerzo.

Sin embargo, la implementación de BPM dentro de cualquier tipo de organización no es una tarea sencilla ni trivial, y de forma particular, dentro de las organizaciones software la situación no es diferente.

Los procesos de negocio relacionados con el desarrollo y la gestión de software se conocen frecuentemente como procesos software. En esta Tesis Doctoral partiremos de la definición dada por Fuggetta en [Fuggetta 2000] para este concepto (consultar Tabla I.2).

Tabla I.2. Definición de *«Proceso Software»* [Fuggetta 2000]

«El proceso software se define como un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos necesarios para concebir, desarrollar, desplegar y mantener un producto software.»

La definición de proceso software recogida en la tabla anterior, da pie a equiparar el software como el producto que es y brinda, además, la posibilidad de relacionarlo con el concepto PLM (*«Product Life cycle Management»*) [Stark 2011; Sääksvuori *et al.* 2008]. PLM puede definirse como aquél proceso global o macroproceso de negocio que gestiona y da soporte, de la manera más eficaz posible, a un producto — desde su acepción más general — durante toda su trayectoria, desde su confección o especificación, hasta su mantenimiento y posterior retirada. Esta gestión implica una recopilación de reglas de negocio, métodos, procesos y directrices sobre cómo y cuándo aplicar dichas reglas en entornos reales. Esta definición de PLM está muy relacionada con la definición de BPM recogida en la Tabla I.1 de este capítulo.

El paradigma de la gestión del ciclo de vida del producto visto desde una perspectiva basada en procesos está arraigado dentro de entornos industriales y organizaciones de manufacturación (por ejemplo, dentro de la industria aeroespacial y de automoción, entre otras). Sin embargo, el concepto de PLM y su aplicación en el área de la Ingeniería de Software no está consolidado, a pesar de que el software puede ser considerado como un producto en sí mismo — aunque sea intangible — y que está relacionado además, con un ciclo de vida ingenieril. Así como el desarrollo de productos industriales pasa en una cadena de montaje, los procesos asociados a la "producción" de software discurren a través de una serie de fases durante su ciclo de vida.

Esta tesis se inspira en las cadenas de montaje de productos industriales y en el propio paradigma de gestión PLM para definir un marco teórico — y apoyado sobre herramientas de soporte — que gestione el ciclo de vida del proceso software, desde su especificación o definición, hasta su ejecución, monitorización y mejora continua. Centrándonos en el proceso software, resulta relevante tener en cuenta que existen factores que lo hacen particularmente complejo de gestionar debido a sus características intrínsecas.

Estas características han sido estudiadas en diversos trabajos. El más reciente que hemos localizado es el artículo [Ruiz-González *et al.* 2004]. En ese trabajo, los autores identifican y describen las propiedades que caracterizan el proceso software frente a otro tipo de procesos. De las características identificadas, las más importantes a destacar son las siguientes:

1. Los procesos software son complejos debido a que se ven fuertemente condicionados por multitud de circunstancias impredecibles y por muchos equipos de trabajo. Esta característica hace que los procesos software sean completamente diferentes a los típicos procesos de producción.
2. Debido a que las fases de diseño y producción no están claramente diferenciadas, hace que los procesos software no puedan considerarse como procesos puros industriales.
3. Desde el punto de vista de la gestión, en multitud de ocasiones, resulta difícil presupuestar y planificar los procesos software con la fiabilidad y certeza suficiente como para garantizar una suficiente calidad del producto software. Uno de los factores que frecuentemente afectan a estos aspectos es el hecho de no contabilizar el coste que supone atender peticiones de cambio del producto.
4. Frecuentemente, los procesos software dependen de la comunicación, coordinación y cooperación de diferentes frameworks y tecnologías de desarrollo, así como diferentes roles (departamento de ventas, múltiples equipos desarrollo, gerentes, analistas, clientes, etc.).
5. Los procesos de desarrollo de software y los procesos asociados a su gestión están en continua y constante evolución debido a que frecuentemente incorporan diferentes ciclos de vida, constan frecuentemente de varias iteraciones y producen distintas versiones del producto software o entregables de gestión.

Por las razones expuestas anteriormente, dentro de las organizaciones software es frecuente que la implantación de BPM y de su proceso de mejora continua se circunscriba a la definición de sus procesos debido a que la automatización de los mismos (y la mayoría de sus actividades) se antoja como una tarea con un alto grado de complejidad [Ruiz-González *et al.* 2004]. Esto es un hecho que hemos podido constatar dentro del seno del grupo de investigación al que pertenece el doctorando — el grupo de Ingeniería Web y Testing Temprano (IWT2) del departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla — a partir del feedback y la experiencia que hemos obtenido desde el tejido empresarial desde el año 2000 en los que el grupo ha participado en más de 150 proyectos de transferencia tecnológica con diferentes organismos y empresas tanto públicas como privadas De todos estos proyectos aquellos que han estado relacionados con la gestión de procesos de negocio y/o el paradigma guiado por modelos, han supuesto un volumen de negocio de alrededor de 3,5 millones de euros.

En este contexto, la ejecución y orquestación de los procesos son actuaciones que frecuentemente se realizan de una manera manual y/o independiente por cada rol que participa en esos procesos. Este escenario provoca que el seguimiento, control y medición

de los procesos de negocio dentro de la organización se antoje como una tarea muy compleja.

En este contexto y como solución inicial, desde el grupo IWT2 planteamos un marco de trabajo basado en procesos para que las empresas software pudieran aplicar y ejecutar sus procesos bajo el paraguas de marcos de referencia, modelos de madurez y estándares ISO junto con la metodología NDT (acrónimo de *«Navigational Development Techniques»*) [Escalona 2004][Escalona *et-al.* 2008]. Este marco de trabajo se denomina NDTQ-Framework [Ponce et-al. 2013] (descrito en la Sección 3.2 del Capítulo III de influencias) y fue definido en el marco de un proyecto de excelencia motriz de la Junta de Andalucía en el año 2011.

NDTQ-Framework está basado en modelos y en la definición global de seis grupos de procesos software, pero su principal limitación es que sólo ofrece una visión estática del proceso software. Con esta tesis se pretende dar un soporte mejorado a la definición del proceso y un soporte completo a su ejecución y orquestación.

Llegados a este punto, es importante aclarar cuál es la definición de los términos ejecución y orquestación de un proceso de negocio, ya que a lo largo de esta tesis se hará referencia a ellos de manera recurrente. La definición de estos conceptos se recoge en la Tabla I.3 y en la Tabla I.4, respectivamente.

Tabla I.3. Definición de *«Ejecución de un Proceso de Negocio»* [Papazoglou *et al.* 2006]

«La ejecución de procesos de negocio se refiere a la capacidad de despliegue y ejecución de un proceso de negocio dentro de un motor de ejecución BPM[1], el cual, ejecuta instancias del proceso de negocio, delegando ciertas tareas a seres humanos y a aplicaciones automatizadas tal y como se especifica en el modelo de proceso definido.»

Tabla I.4. Definición de *«Orquestación de un Proceso de Negocio»* [Pedraza *et al.* 2009]

«La orquestación de un proceso de negocio se refiere a la gestión centralizada y coordinada de eventos o componentes durante la ejecución del proceso de negocio. A menudo se trata de una variedad de componentes que forman parte de una nueva aplicación o proceso.»

[1] El motor de ejecución suele formar parte de una Suite BPM conocida como BPMS (*«Business Process Management Suite»*) y es el componente software encargado de cargar la definición de los procesos y ejecutar sus instancias. BPMS puede ser definido como una nueva categoría de software empresarial que permite a las empresas modelar, implementar y ejecutar conjuntos de actividades interrelacionadas –es decir, Procesos- de cualquier naturaleza, sea dentro de un departamento o permeando la entidad en su conjunto, con extensiones para incluir a clientes, proveedores y otros agentes como participantes en las tareas de los procesos.

Como solución a la situación descrita anteriormente, Acuña propone en [Acuña *et al.* 2013] focalizar los esfuerzos en dos líneas de actuación e investigación. Por una parte, mejorar las técnicas existentes (o proponer otras nuevas) para optimizar los procesos de desarrollo de software. Por otra parte, mejorar prácticas la forma en la que se gestionan y orquestan los procesos software dentro la organización.

Para cubrir el espectro de posibilidades que pueden surgir de las líneas de trabajo descritas anteriormente, la investigación se divide en dos líneas: (*i*) el modelado de procesos software y (*ii*) ejecución, orquestación y mejora de dichos procesos. Esto último incluye todos los esfuerzos de la comunidad software relacionados con modelos de madurez y estándares como ISO 9001:2008 e ISO/IEC 15504:2004 [ISO/IEC 2004], entre otros.

Por tanto, la senda hacia una gestión efectiva de los procesos de desarrollo y gestión de software parece discurrir hacia la definición de métodos y protocolos que permitan, en primer lugar, modelar procesos software para que posteriormente puedan ser ejecutados y orquestados. Para ello, algunos autores concretaron cuáles son los aspectos necesarios a cubrir [Ellner *et al.* 2010]:

1. Definir un lenguaje de modelado de procesos con una semántica suficientemente rica como para que la definición de los procesos software sea adecuada.
2. Definir una herramienta fácil de usar con un entorno de modelado de procesos que sea lo suficientemente flexible para cubrir diferentes categorías de proyectos.
3. Definir un entorno de ejecución de procesos fácilmente integrable con las herramientas existentes de desarrollo de software.

El trabajo de Tesis Doctoral presentado en este documento, se inspira en los aspectos identificados y analizados por Ellner en [Ellner *et al.* 2010], así como en la visión que establece el paradigma PLM en la gestión del producto — en términos generales —. Además, la propuesta aquí presentada se ve motivada por los problemas planteados anteriormente dentro de las organizaciones software para gestionar el ciclo de vida del proceso software.

Por todo ello, esta tesis realiza una propuesta teórica basada en modelos y sus transformaciones para hacer más sencilla y efectiva la aplicación de BPM dentro de las organizaciones software. Tanto el entramado teórico como la herramienta CASE (*«Computer Aided Software Engineering»*, Ingeniería de Software Asistida por Computadora) que le da soporte se bautizan bajo el acrónimo PLM_4BS (*«Process Lifecycle Management for Business-Software»*, es decir, Gestión del Ciclo de Vida del Proceso para el Negocio del Software).

PLM_4BS nace además con el firme propósito de proporcionar un marco de trabajo aplicable en proyectos reales y para ello, se aprovecha también de las ventajas que proporciona un paradigma muy arraigado en el mundo del desarrollo de software [Ardagna *et al.* 2008] como es el paradigma MDE [Schmidt 2006] (*«Model-Driven Engineering»*), llegando a considerarlo como el pilar maestro de todo el trabajo de tesis.

A modo de reseña, la aplicación de MDE en el contexto de la Ingeniería del Software, consiste en desarrollar software a través de la evolución de determinados modelos desde la definición hasta el código a través de una serie de reglas de transformación bien

definidas. Este objetivo puede ser progresivamente alcanzado a través de la coherencia entre el modelado de procesos software y el paradigma de desarrollo de software [Van-Der-Straeten *et al.* 2009].

Volviendo a la descripción de la propuesta, PLM_4BS propone una serie de metamodelos y un conjunto de mecanismos de transformación entre ellos, para llevar a cabo de una manera eficaz el mantenimiento de procesos software desde la propia definición del proceso hasta su despliegue en un motor BPM, pasando por la definición del contexto de ejecución y orquestación del proceso.

De manera general, la Figura I.1 esboza la propuesta de solución para llevar a cabo la gestión del ciclo de vida de proceso de negocio dentro de las organizaciones software. La propuesta plantea tres etapas para la gestión de procesos. Aunque los aspectos teóricos de estas etapas serán desarrollados en detalle en los siguientes capítulos, a continuación, se presenta una visión preliminar e introductoria de las mismas.

Sin embargo, previamente es importante recalcar que las etapas representadas en la Figura I.1 no son secuenciales; de hecho, desde la etapa de ejecución y orquestación es posible volver a la etapa de definición para mejorar el modelo en caso de detectar un error en el mismo (esto se refleja mediante la línea punteada desde la etapa de ejecución y orquestación hasta la etapa de definición del proceso). Aclarado este punto, a continuación se describe cada una de las etapas que aparecen en la Figura I.1:

1. **Etapa de definición**. La primera etapa permitirá al ingeniero de procesos modelar el proceso de negocio deseado de acuerdo con un metamodelo diseñado para dicho propósito. Este metamodelo estará constituido por un conjunto de elementos con su semántica y restricciones asociadas, para permitir la definición de procesos de negocio.

2. **Etapa de ejecución y orquestación**. Una vez definido el proceso de negocio en la etapa anterior, el ingeniero de procesos podrá derivarlo de una manera sistemática y semiautomática mediante un conjunto de reglas de transformación hacia un modelo de ejecución y orquestación (estas transformaciones son conocidas como transformaciones *«model-to-model»* en el contexto del paradigma MDE). Este nuevo modelo recogerá aquellas propiedades específicas que contextualizan el entorno de ejecución y orquestación de procesos. Además, este modelo deberá ser construido conforme a un segundo metamodelo constituido por una serie de elementos con semántica operacional y con propiedades específicas que permitirán instanciar los procesos posteriormente.

3. **Etapa de implantación**. Finalmente, una vez que el ingeniero de procesos haya finalizado de definir y refinar el modelo de ejecución y orquestación en la etapa previa, será el momento de transformarlo en un formato de código entendible por un motor de ejecución BPM (por ejemplo, un formato que siga el lenguaje estándar WS-BPEL [OASIS 2007]). El conjunto de reglas de transformación utilizadas para obtener este formato de código desde el modelo, se conocen como transformaciones *«model-to-text»* en el contexto del paradigma MDE. Llegados a este punto, el usuario final ya puede desempeñar su trabajo y beneficiarse del

soporte que le proporcionará un proceso de negocio bien definido y desplegado en un entorno software.

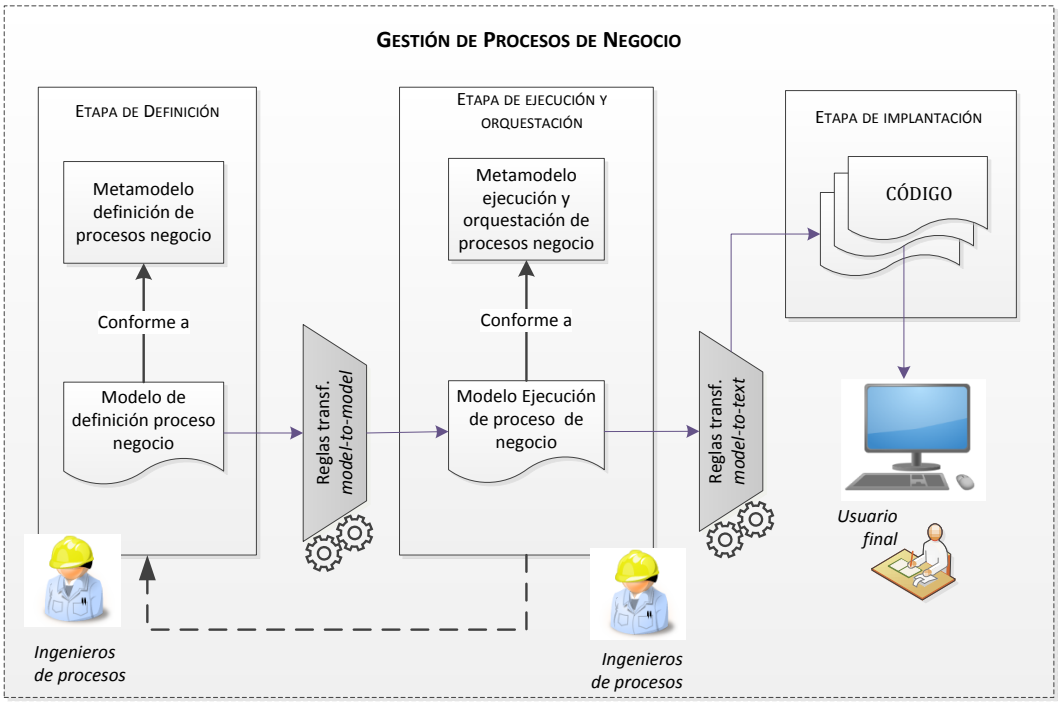


Figura I.1. Propuesta PLM$_4$BS para la gestión de procesos de negocio

Además de esto, esta tesis pretende que todos los modelos mencionados anteriormente, y sobre todo las técnicas y herramientas para describirlos, sean cercanos y cognitivamente entendibles por usuarios con un perfil no técnico, con idea de que se puedan incluir en el ciclo de vida BPM como entes necesarios, tanto para la definición como para la validación final de los modelos de proceso definidos.

2. Estructura de la Tesis Doctoral

Después de este capítulo introductorio, la tesis continúa con un estudio de la situación actual en cuanto a propuestas basadas en modelos que permitan la definición y posterior ejecución de procesos de negocio (y de manera particular, procesos software). A continuación, este trabajo sienta las bases teóricas de un marco de trabajo basado en modelos y presenta una herramienta de soporte CASE que permite la aplicación de esta base teórica en entornos prácticos y proyectos reales. Todos estos aspectos son desarrollados a lo largo de los capítulos de esta Tesis Doctoral, la cual se estructura de la siguiente forma:

El Capítulo II ofrece un estudio del estado del arte sobre cuáles son los trabajos relacionados con esta tesis, en cuanto a otras propuestas basadas en modelos para la definición de procesos de negocio y su posterior ejecución y/u orquestación. Esta visión general permitirá analizar cómo está definida, dentro de la literatura existente, la gestión de procesos software desde una perspectiva basada en modelos.

Además, como se ha hecho hincapié a lo largo de la sección anterior, la propuesta desarrollada en esta Tesis Doctoral se enmarca dentro de un ciclo de vida de mejora continua para la gestión de procesos de negocio y es en este sentido cuando se hace palpable la necesidad de estudiar otras propuestas de ciclo de vida BPM. En esta línea, el Capítulo II presenta brevemente estas propuestas.

Con la visión general que ofrece el estudio presentado en el Capítulo II de la situación actual en la gestión de procesos desde una perspectiva basada en modelos, se sientan las bases para que durante el Capítulo III se pueda hacer una definición del problema a tratar. El Capítulo III proporciona, por tanto, una descripción del planteamiento del problema que trata esta tesis, el entorno de trabajo y define, además, de manera concreta, los retos y objetivos a satisfacer. En este capítulo además serán definidos con más detalle las influencias que han impulsado la realización de este trabajo.

Planteado el problema y los objetivos a alcanzar, se comienza a trabajar en ellos en los siguientes capítulos. Por un lado, el Capítulo IV presenta nuestra propuesta sobre cuáles deberían ser los metamodelos para realizar un tratamiento adecuado del modelado, ejecución y orquestación de procesos software. Por otro lado, el Capítulo V analiza las posibles relaciones entre los metamodelos planteados y define una serie de procedimientos básicos y sistemáticos de derivación entre ellos. Estos procedimientos permitirán sentar las bases teóricas para su posterior automatización.

Los fundamentos teóricos presentados durante los dos capítulos anteriores se materializan durante el Capítulo VI en un entorno práctico a través de una herramienta CASE llamada PLM$_4$BS. Esta herramienta proporciona un perfil o lenguaje específico de dominio para utilizar de manera eficaz cada metamodelo presentados en el Capítulo IV. Además, esta herramienta automatiza los procedimientos sistemáticos de derivación definidos de forma teórica en el Capítulo V. El Capítulo VI describe además dos casos prácticos en los que la propuesta presentada en esta tesis ha sido aplicada de forma satisfactoria.

Más adelante, esta tesis ofrece, ya en el Capítulo VII, las aportaciones de esta tesis, los trabajos futuros de investigación que se han abierto en base a los resultados obtenidos, las relaciones nacidas durante la elaboración de este trabajo y unas conclusiones finales.

Para finalizar la presentación de este trabajo, este documento culmina en tres apéndices que amplían y dan soporte a los contenidos presentados. El primero (Anexo A) presenta un glosario de términos en el que se ordenan los conceptos clave utilizados a lo largo de esta tesis. El Anexo B presenta brevemente un manual de la herramienta desarrollada para dar soporte al marco teórico descrito en este trabajo. Finalmente, el tercer apéndice, Anexo C, presenta la trayectoria investigadora del autor de este trabajo de tesis.

3. Conclusiones

La Tesis Doctoral que se presenta en este trabajo viene motivada por un problema que ha sido identificado dentro de las organizaciones dedicadas al negocio del software: la

necesidad de establecer mecanismos sistemáticos y automáticos que posibiliten la gestión eficaz y eficiente del ciclo de vida del proceso software, desde la propia definición del proceso hasta su ejecución y orquestación. Todo esto con el propósito de facilitar y mejorar su mantenibilidad y calidad.

El capítulo de introducción, aparte de enmarcar y contextualizar el trabajo de la tesis, ofrece una definición del ciclo de vida para la gestión de procesos, tal como se entiende en todo este trabajo.

Además, proporciona de manera global cuál es la estructura de presentación de los resultados de investigación que han concluido en la realización de esta tesis.

Llegados a este punto resulta conveniente mencionar que este trabajo ha sido llevado a cabo en el seno del grupo de investigación Ingeniería Web y Testing Temprano (IWT2) de la Universidad de Sevilla, grupo referenciado en el Plan Andaluz de Investigación como PAIDI TIC021.

Además y de manera más concreta, la tesis se ha desarrollado dentro de un marco estratégico del grupo IWT2 en el que se pretende explorar e investigar cómo combinar de manera satisfactoria el paradigma guiado por modelos con la gestión de procesos de negocio. En este sentido, y como se describe en el Capítulo VII, son varias las Tesis Doctorales que se están llevando a cabo en este contexto y de forma paralela a la que se presenta en este documento de tesis.

Capítulo II TRABAJOS RELACIONADOS

Esta tesis está motivada por la necesidad de buscar y proponer un marco de trabajo eficaz, sistemático, basado en modelos, soportado por herramientas, enraizado y fundamentado a nivel teórico, para que las empresas software puedan gestionar de una manera eficaz y eficiente el ciclo de vida de sus procesos software. Todo esto con el propósito de mejorar el mantenimiento de estos procesos y de posibilitar la realización de una mejora continua efectiva durante la obtención del producto software. En este contexto, la propuesta que se realiza en esta tesis se circunscribe a la definición formal del proceso software y de su contexto de ejecución para la obtención de una versión ejecutable y orquestable.

Para este propósito y como primer paso natural para proponer una aportación original es necesario estudiar el conocimiento existente en cuanto a propuestas de ciclo de vida para la gestión de procesos de negocio. Estos ciclos de vida consideran diferentes aspectos de la vida del proceso, pero esta tesis se centra en el tratamiento de la definición, ejecución y orquestación del proceso. En este sentido, este capítulo tiene por objetivo estudiar también aquellas propuestas basadas en modelos para la definición, ejecución y orquestación del proceso software, para así, extraer aquellos aspectos que podrían servir de base para guiar esta tesis tanto en el planteamiento como en la culminación de sus objetivos.

Para obtener ese conocimiento, este capítulo presenta cuáles son las propuestas de ciclo de vida BPM y además, identifica las propuestas más recientes de la literatura actual que están relacionadas con la definición y ejecución de procesos de negocio desde una perspectiva basada en modelos. Respecto a las propuestas para la definición y ejecución del proceso, se resume la información más relevante de las mismas mediante un esquema de caracterización.

El capítulo concluye sintetizando los aspectos más relevantes del estado del estudio del arte presentado a lo largo del mismo.

1. Antecedentes del estudio de la situación actual

Como se ha trasladado a lo largo del capítulo anterior, uno de los pilares de esta tesis es enmarcar sus resultados dentro de un ciclo de vida que gestione el proceso de negocio — y más concretamente, el proceso software — dentro de un contexto de mejora continua.

De manera general e históricamente, ciclo de vida incluye, trata y considera diferentes aspectos en la gestión de procesos de negocio tales como la definición y ejecución monitorización, entre otros aspectos.

Sin embargo, aunque esta tesis propone una solución para determinados aspectos de esta gestión — es decir, la definición, ejecución y orquestación del proceso — resulta interesante enfocar esta sección de antecedentes de la situación actual en dos vertientes: (*i*) ciclos de vida BPM para su mejora continua; y (*ii*) estudios previos que hayan desarrollado revisiones sistemáticas o estudios del estado del arte en cuanto a propuestas basadas en modelos que posean la capacidad para modelar, ejecutar y orquestar procesos.

Por una parte y como se ha trasladado en el capítulo introductorio, BPM puede ser considerada como una estrategia o disciplina de gestión orientada a procesos [Hill *et al.* 2008] con una clara naturaleza multidisciplinar debido a que puede ser aplicada en diferentes contextos y dominios, y utilizada por distintos perfiles de usuario [Ryan *et al.* 2009]. Esto ha propiciado diferentes visiones, definiciones y perspectivas de aquellos aspectos que debería contemplar el ciclo de vida BPM para su mejora continua.

El ciclo de vida BPM puede definirse como aquel conjunto de etapas o fases – cada una con un enfoque específico – que se realizan de forma cíclica para realizar un proceso global e integral de mejora continua de los procesos de negocio de una organización. Debido al amplio abanico de dominios de aplicación, a lo largo de la última década se han propuesto diferentes visiones del ciclo de vida BPM. A continuación, se resumen las más recientes.

En [Hill *et al.* 2006], los autores proponen un proceso global de revisión de los procesos de negocio de una organización con el objetivo de crear valor en las mismas. Para ello, contemplan el modelado de procesos como primer paso para alcanzar este objetivo. De esta manera, antes de iniciar cualquier diseño o proceso de revisión, la organización debe decidir el alcance de sus actividades iniciales.

El proceso de mejora continua que proponen estos autores está basado en nueve fases (ver Figura II.1). A modo de reseña, estas fases son: *Discovery*, definida como la primera actividad del proceso y tiene por objetivo establecer la metodología seguida para evaluar el trabajo y medir el éxito del proceso de mejora continua; *Modeling*, contemplada como aquella fase en la que se definen de manera estática aquellos procesos y objetivos de negocio a mejorar; *Simulation*, fase en la que se detectan los posibles cuellos de botella que no son evidentes durante el modelado estático; *Deployment*, etapa en la que se desarrollan los scripts necesarios para integrar el proceso en los sistemas de información de la organización; *Execution*, fase en la que se instancian los procesos definidos en las fases anteriores; *Monitor*, fase en la que se recopila la información del proceso en ejecución y en tiempo real; *Analytics*, fase en la que se deben evaluar indicadores claves de rendimiento sobre los procesos en ejecución para detectar desviaciones de los objetivos organizacionales; *Optimization*, fase en la que se llevan a cabo las acciones correctivas necesarias para optimizar los procesos, mejorar sus resultados y reducir posibles riesgos para la organización; y *Refine*, que se contempla como aquella fase en la que se mejora la metodología de negocio definida en la primera fase del proceso.

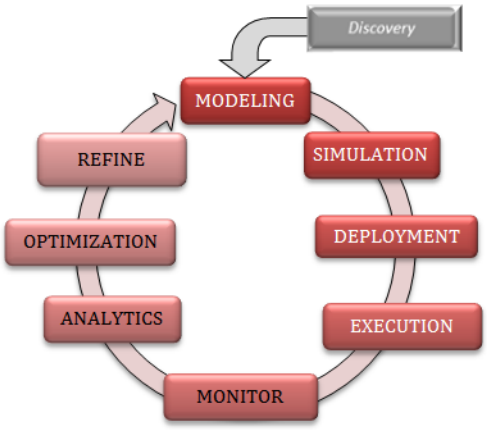


Figura II.1. Ciclo de mejora continua BPM según [Hill *et al.* 2006]

Por otra parte, en [van der Aalst 2004a, 2004b] los autores establecen un ciclo de vida BPM bastante más compacto que el presentado en la propuesta anterior. En este caso, el ciclo de vida está basado en cuatro fases y se relacionan entre sí tal y como aparece en la Figura II.2.

A modo resumen, las fases que componen este ciclo de vida BPM son las siguientes: *Process design*, fase durante la cual los modelos deben ser definidos teniendo cuenta su flujo de control, el flujo de datos, perfiles de usuario, datos de la organización y cualquier otro aspecto operativo; *System configuration*, es la fase en la que se despliega, integra y configura el modelo del proceso definido en la fase anterior dentro del sistema de información de la organización; *Process enactment*, es la fase en la que se instancian los procesos desplegados en el sistema de información de la organización; y *Diagnosis*, fase en la que se monitorizan y analizan los procesos en ejecución para identificar problemas y encontrar los posibles aspectos para mejorar el proceso de manera global.

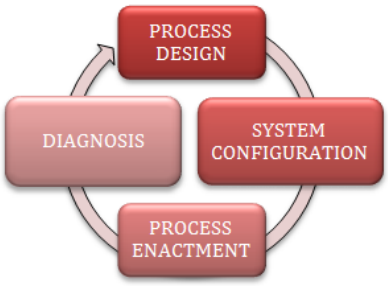


Figura II.2. Ciclo de mejora continua BPM según [van der Aalst 2004a, 2004b]

Finalmente y a modo de reseña general, Shewhart publicó en [Shewhart 1986] su visión particular de cuáles deberían ser las fases globales que debe contener cualquier ciclo de mejora continua. Aunque esta propuesta no está específicamente orientada hacia la gestión de procesos software, resulta interesante tenerla presente debido a su transcendencia y trayectoria desde su publicación.

Tal y como se puede apreciar en la Figura II.3, la propuesta de Shewhart plantea una estrategia de mejora continua basada en la realización cíclica de cuatro pasos: *Plan*, en el que se debe establecer de manera obligatoria cuáles son los objetivos a alcanzar o los

resultados a obtener tras finalizar el proceso; *Do*, en el que se debe ejecutar lo que se ha planificado para alcanzar los objetivos planteados; *Study*, en el que se evalúa el resultado obtenido con el resultado esperado; y *Act*, en el que se deben establecer las acciones correctivas necesarias –en el caso en el que el resultado obtenido no coincida con el esperado– para alcanzar el objetivo marcado en la siguiente iteración del ciclo.

Figura II.3. Ciclo de mejora continua según Shewhart [Shewhart 1986]

Por otra parte, después de analizar los ciclos de vida anteriores es posible identificar las fases de modelado o definición y ejecución como fases comunes, y es precisamente para estas fases a las que esta tesis propone una solución basada en modelos para darles soporte. Además de estas fases, incluimos el aspecto de la orquestación tal y como se describe en siguientes capítulos.

Como paso previo antes de comenzar a desarrollar el trabajo que se presenta en este documento, se buscaron estudios previos que indicaran cuál era el estado del arte en cuanto a lenguajes para modelar y ejecutar, y con capacidad de orquestar procesos. A pesar de que este tema se ha discutido y estudiado durante muchos años, se han encontrado muy pocas referencias bibliográficas que reflejen estudios en esta área. A continuación, se describen los trabajos más relevantes en este sentido.

Zamli llevó a cabo un estudio sobre el estado del arte de lenguajes de modelado de procesos correspondientes a la segunda generación de lenguajes de modelado de procesos [Zamli 2001]. De manera breve, la segunda generación de lenguajes de modelado de procesos se corresponde con aquellos lenguajes que han sido publicados después de 1996. Esta clasificación fue propuesta por Sutton [Sutton *et al.* 1997]. Zamli clasificó los lenguajes de modelado de procesos en tres categorías: (*i*) no ejecutables, es decir, aquellos lenguajes que proporcionan soporte para el modelado de procesos pero no para su ejecución; (*ii*) simulados, es decir, aquellos lenguajes que permiten simular el proceso a alto nivel pero no proporcionan mecanismos para controlar esa simulación; y (*iii*) ejecutables, es decir, aquellos lenguajes que permiten controlar el contexto de ejecución del proceso.

Además de esta clasificación, los autores introdujeron cinco aspectos que cualquier lenguaje de definición de procesos debería soportar: soporte para el modelado, soporte para la ejecución, soporte para la medición, soporte para la orquestación, y soporte para la interacción con personas.

Algunos años más tarde, Zamli publicó un nuevo estudio [Zamli *et al.* 2004] tomando como base la taxonomía y los cinco aspectos identificados en su trabajo previo.

Por otra parte, Bendraou publicó los resultados de un estudio comparativo de seis lenguajes basados en UML para el modelado de procesos software [Bendraou *et al.* 2010]. Para llevar a cabo esta comparación, el autor propone una serie de requisitos para el modelado de procesos, tales como la riqueza semántica, modularidad, ejecutabilidad, la conformidad con el estándar UML, y formalidad.

Este trabajo llevado a cabo por Bendraou demuestra que, si bien UML ofrece un alto potencial en cuanto a servir como base para la construcción de propuestas de modelado de procesos aprovechando de esta manera su comprensibilidad, UML no resulta suficiente para modelar de manera formal la ejecutabilidad de los procesos. Este trabajo también proporciona un listado de problemas que deben ser resueltos para desarrollar propuestas que realmente satisfagan las necesidades del modelado de procesos software. Entre estos problemas se encuentran por ejemplo el soporte de la propuesta con herramientas, la ejecución y la expresividad.

Finalmente, en [Henderson-Sellers *et al.* 2005] evaluaron cuatro metamodelos propuestos como base para la definición formal de metodologías de procesos. Sobre la base de este análisis, los autores proponen un nuevo metamodelo para la definición de procesos de desarrollo software.

En resumen, a lo largo de los últimos años se han llevado a cabo muy pocos estudios sistemáticos que reflejen la situación actual en cuanto a propuestas para modelar, ejecutar y orquestar procesos.

Esto ha motivado la realización de un estudio de la situación actual [García-Borgoñón *et al.* 2013] que he llevado a cabo de manera conjunta con un miembro del grupo de investigación Ingeniería Web y Testing Temprano (IWT2) cuya Tesis Doctoral se está llevando a cabo también en la actualidad y que está enmarcada también dentro del contexto BPM. Tal y como se adelanta en la Sección 3 del Capítulo I y se detalla en el Capítulo VII, el grupo IWT2 está explorando cómo combinar satisfactoriamente el paradigma guiado por modelos con una gestión eficaz de procesos de negocio.

2. Estudio del estado del arte de propuestas basadas en modelos que traten la definición y ejecución de procesos de negocios

A lo largo de esta sección se describen los trabajos relacionados más relevantes encontrados después de haber llevado a cabo un estudio del estado del arte en cuanto a propuestas basadas en modelos para la definición y ejecución con capacidad de orquestar procesos. Con el propósito de identificar características comunes y posibles puntos de mejora, las propuestas aquí presentadas han sido comparadas entre sí a través del esquema de caracterización descrito en la Sección 2.1. Asimismo, los resultados de evaluar este esquema sobre cada trabajo identificado así como una breve descripción del mismo, se describe en la Sección 2.2.

El estudio del estado del arte presentado en este apartado forma parte de un trabajo más completo llevado a cabo de manera conjunta con un miembro del grupo de investigación IWT2 y cuya tesis también se enmarca dentro de la aplicación de BPM desde la perspectiva del paradigma MDE — esta tesis se describe en el Capítulo VII. Este trabajo ha sido publicado en [García-Borgoñón *et al.* 2013] y para su realización se ha llevado a

cabo una búsqueda exhaustiva y sistemática de la literatura sobre lenguajes de modelado de procesos software. Los lenguajes estudiados abarcan aquellos basados en modelos, basados en redes de Petri, basados en lenguajes, basados en reglas y basados en lenguajes de programación, entre otros.

Debido a la temática de esta tesis, en este apartado únicamente se describen aquellas propuestas basadas en modelos para la definición y, ejecución y orquestación del proceso software.

2.1. Esquema de caracterización

Esta sección define brevemente el formato y el esquema de caracterización utilizado para evaluar las propuestas presentadas a lo largo de este capítulo. Si desea obtener información más detallada sobre estos trabajos, le invitamos a acudir a las referencias indicadas.

La Tabla II.1 define el esquema de caracterización utilizado para resumir y analizar cada uno de los estudios presentados en la próxima sección de este capítulo.

Tabla II.1. Esquema de caracterización.

Atributo	Descripción
Expresividad	Un usuario debe ser capaz de definir cualquier tipo de proceso software; incluso el más complejo de ellos. Esta característica nos permite comprobar si la propuesta evaluada proporciona conceptos de modelado como actividades, condicionales, ramas paralelas, manejadores de excepciones, productos y roles.
Granularidad	Indica si los usuarios pueden describir a sus procesos con el nivel de detalle o granularidad que estimen necesario.
Medible	Esta característica evalúa si la propuesta proporciona mecanismos para definir métricas o indicadores claves de rendimiento (o conceptos similares) durante el modelado del proceso de negocio. La definición de métricas e indicadores es esencial para la mejora continua de los procesos de negocio.
Ejecutabilidad	Indicar si la propuesta contempla y proporciona algún tipo de soporte a la ejecución del proceso. Esta característica podría ser de utilidad para que los procesos puedan ser simulados y probados de forma semi-automática.
Comprensibilidad	Es posible que el usuario que utiliza la propuesta de modelado de procesos no tenga un perfil de ingeniero software. La evaluación de esta característica evalúa si la propuesta proporciona el formato de la representación comprensible y legible basado en metáforas visuales y familiares. Las metáforas se utilizan para comunicar conceptos abstractos de una forma familiar y accesible teniendo en este sentido, un papel dominante en el diseño de las interfaces de usuarios de cualquier tipo de software o lenguaje gráfico. El uso de metáforas ayuda a construir soluciones software que puedan ser utilizadas por comunidades de usuarios más diversas.

Conforme a UML	Esta característica evalúa si la propuesta evaluada reutiliza UML, es decir, su definición viene dada como una extensión en forma de metamodelo MOF[2] [OMG 2011b], un perfil UML, o simplemente reutiliza diagramas UML. Con esta característica, cualquier organización podrá evaluar: (*i*) el costo de desarrollar una herramienta específica de soporte de UML, (*ii*) y en qué medida puede reutilizar o personalizar su herramienta favorita UML. Se ha optado por incluir esta característica debido a la amplia aceptación y reconocimiento que presenta este estándar a nivel empresarial.
Orquestabilidad	Indica si la propuesta evaluada proporciona mecanismos para la especificación de llamadas a otros componentes software o servicios Web (por ejemplo, URI, puerto de comunicación, método para ejecutar del servicio Web, entre otros).
Herramienta de Soporte	Esta característica evalúa si la propuesta proporciona herramientas de soporte, lo cual, podría ser decisivo para implantar la propuesta en un contexto empresarial y de producción.
Reglas de negocios	Esta característica evalúa si la propuesta estudiada proporciona mecanismos para definir reglas de negocio, entendidas como aquellas reglas que se crean durante el modelado de cada proceso y que permiten definir los comportamientos de ciertos elementos, como por ejemplo, las condiciones que permiten definir qué bifurcación tomar después una decisión.

A continuación, en la siguiente sección, se resumen brevemente las propuestas estudiadas y para cada una de ellas, se presenta un estudio en base al esquema de caracterización anterior.

2.2. Trabajos evaluados

Propuesta de Chou

Chou [Chou 2002] publicó su visión particular para modelar y ejecutar procesos software desde una perspectiva basada en modelos. La propuesta de Chou usa un subconjunto de los conceptos definidos en UML1.4 para este propósito; concretamente, el autor utiliza Diagramas de Actividades de UML1.4. Además, para dar soporte a la ejecución del proceso, la propuesta propone utilizar un lenguaje propio de programación orientado a objetos.

A priori, la principal desventaja de esta propuesta es la falta de reglas de derivación sistemáticas para obtener el código ejecutable a partir del diagrama UML. En este contexto, en el que el código no se deriva desde el modelo, implica una reescritura manual por parte de los desarrolladores de la lógica definida en el modelo de proceso. Esta desconexión entre ambas representaciones del mismo proceso, puede conllevar inconsistencias a lo largo del tiempo, debido al coste que supone mantener dos representaciones independientes e incomunicadas de un mismo proceso.

Para finalizar, la Tabla II.2 muestra el esquema de caracterización de esta propuesta.

[2] El estándar MOF, *«Meta-Object Facility»*, es un conjunto de interfaces estándares que pueden ser utilizadas para definir y manipular metamodelos interoperables y sus correspondientes modelos.

Tabla II.2. Esquema de caracterización Chou [Chou 2002]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
Chou	×		×	×	×	×	×	×	×	×				

Propuesta de Di Nitto

En [Di Nitto 2002], los autores presentan un marco de trabajo para ejecutar procesos software a partir de su representación UML1.3 (los autores no extienden ningún metamodelo de UML ni tampoco definen nuevos conceptos específicos para el modelado de procesos software). El flujo de trabajo se define mediante Diagramas de Actividad de UML, mientras que el ciclo de vida de cada entidad se define a través de una máquina de estado.

Además, la propuesta propone reutilizar la metaclase *«Class»* de UML –manteniendo la misma semántica y notación– como representación intermedia entre el diagrama de actividades y el formato ejecutable del proceso. En cualquier caso, no se establecen mecanismos de trazabilidad para enlazar ambas representaciones del proceso.

Finalmente, esta propuesta define reglas de transformación del modelo de clases – representación intermedia del modelo del proceso – para posibilitar su ejecución sobre la plataforma OPSS [Cugola 2001]. Como reseña, OPSS es un sistema *ad-hoc* de gestión centralizado de flujos de trabajo que ha sido desarrollado por los mismos autores. Este sistema fue concebido para dar soporte al diseño y funcionamiento de servicios basados en procesos complejos como comercio electrónico, atención al cliente, la educación a distancia, y procesos de desarrollo de software.

Para finalizar, la Tabla II.3 muestra el esquema de caracterización de esta propuesta.

Tabla II.3. Esquema de caracterización Di Nitto [Di Nitto 2002]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
Di Nitto	×	×	×		×	×	×	×	×	×		×	×	

UML-EWM: UML Ex-tended Workflow Metamodel

UML-EWM [Riesco *et al.* 2003] es una propuesta basada en UML que presenta un metamodelo de *Workflow* como una extensión de UML; concretamente, los autores de esta propuesta extienden el metamodelo de los Diagramas de Actividades de UML para incluirle nuevas metaclases para proporcionar la capacidad de modelar flujos de trabajo. La Tabla II.4 muestra el esquema de caracterización de esta propuesta.

Tabla II.4. Esquema de caracterización UML-EWM [Riesco *et al.* 2003]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
UML-EWM	×	×			×	×	×	×	×				×	

En un trabajo posterior [Debnath *et al.* 2006], los autores publican un nuevo trabajo en el que realizan un mapeo entre los conceptos definidos en su propuesta (UML-EWM) con la propuesta SPEM. Todo ello para facilitar la automatización de procesos de desarrollo software con SPEM en tecnologías de workflow.

UML4SPM: UML for Software Process Modelling

UML4SPM [Bendraou *et al.* 2005; 2006; 2009] es una propuesta para modelar procesos software. Esta propuesta se sustenta sobre el estándar MOF y se define como una extensión del lenguaje UML2.0 [OMG 2005]. Gracias a la utilización de UML como lenguaje base para definir el metamodelo de UML4SPM, los autores aprovechan varias de las ventajas que proporciona UML2.0, como son, entre otras, las siguientes: (*i*) su capacidad para incluir dentro de las actividades del proceso acciones ejecutables contribuyendo así a la orquestabilidad del proceso; (*ii*) la consolidación de UML dentro de la industria del software; y (*iii*) una notación estandarizada.

El metamodelo de UML4SPM se organiza en dos paquetes: *UML4SPM Process Structure*, que contiene el conjunto de conceptos que permiten modelar procesos software; y *UML4SPM Foundation*, que contiene el subconjunto de conceptos UML2.0 extendidos por los elementos del primer paquete. Los conceptos de modelado considerados en la propuesta UML4SPM son: *Software Activity*, que describe una unidad de trabajo que debe ser realizada durante el proceso software; *Activity Performer*, que describe la herramienta (*Tool*) o actor responsable, (*Responsible Role*) el cual además se desglosa como agentes (*Agents*) o equipos (*Teams*); y *WorkProduct*, que representa una pieza física de información generada durante la ejecución de procesos.

Por otra parte, en [Bendraou *et al.* 2007b] se propone un mecanismo para mapear los conceptos de UML4SPM en elementos del lenguaje estándar WS-BPEL [OASIS 2007] con el propósito de proporcionar ejecutabilidad a la propuesta UML4SPM.

Para finalizar, la Tabla II.5 muestra el esquema de caracterización de esta propuesta.

Tabla II.5. Esquema de caracterización UML4SPM [Bendraou *et al.* 2005]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
UML4SPM	×	×	×	×	×	×	×	×	×	×			×	

Propuesta de Combemale

En [Combemale *et al.* 2006], los autores proponen un lenguaje basado en SPEM1.1 para modelar procesos. A modo aclaratorio, SPEM1.1 (*Software & Systems Process Engineering Metamodel*) [OMG 2002] está basado en UML1.4 e introduce un conjunto de conceptos para modelar procesos de desarrollo software.

La propuesta de Combemale surge debido a la generalidad y falta de directrices de uso de SPEM1.1, lo que en última instancia dificulta su utilización. Además, la semántica de SPEM1.1 se expresa utilizando un lenguaje natural, lo que resulta ser, muy a menudo, el catalizador para definir modelos de procesos inconsistentes. La propuesta de Combemale se focaliza en esta línea en base a una especialización de SPEM1.1. La propuesta por tanto define claramente los conceptos para modelar procesos y expresa formalmente su semántica con OCL (*Object Constraint Language*) [ISO/IEC 2012].

Para finalizar, la Tabla II.6 muestra el esquema de caracterización de esta propuesta.

Tabla II.6. Esquema de caracterización Combemale [Combemale *et al.* 2006]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
Combemale	×	×			×	×	×	×	×					

UPME: Process Modeling based on UML Extension

UPME [Wu *et al.* 2006] es una propuesta basada en UML para el modelado de procesos software. En este sentido, establece además un marco de trabajo compuesto por tres fases (basadas en UML y sus mecanismos de extensión) para llevar a cabo este modelado. Estas fases son:

1. Modelar el proceso software en base al metamodelo definido en la propuesta. Estos modelos son almacenados en un repositorio central de modelos.

2. Construir un modelo de instanciación (a partir del modelo del proceso definido en la fase previa) para un dominio o proyecto concreto. Para ello, la propuesta plantea utilizar el lenguaje de script IDL (*«Instantiation Description Language»*).

3. Construir un modelo de compilación para traducir el modelo de instanciación en un esqueleto de código definido como un lenguaje orientado a objetos con el objetivo de desplegar el proceso modelado en un contexto ejecutable.

Respecto a la fase de modelado, UPME plantea una serie de extensiones de diferentes diagramas UML (por ejemplo, diagramas de clases y de Estado, entre otros) para definir sus conceptos para el modelado de procesos software. Estos conceptos son: *«Activity»*, *«Artifact»*, *«Agent»* (que se desglosa en *«HumanAgent»*, *«SoftwareAgent»* o *«Group»*) y *«Resource»*.

Para finalizar, la Tabla II.7 muestra el esquema de caracterización de esta propuesta.

Tabla II.7. Esquema de caracterización UPME [Wu *et al.* 2006]

	Expresividad													
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
UPME	×				×	×	×		×	×			×	

FlexUML: A UML Profile for Flexible Process Modelling

FlexUML [Martinho *et al.* 2008] es otra propuesta basada en UML; concretamente, esta propuesta se plantea como una extensión de los Diagramas de Actividades de UML.

En esta propuesta los autores parten de la idea de que durante el modelado de procesos se lleva a cabo una elicitación y captura informal de la descripción del proceso. Esto implica que el proceso, y concretamente sus elementos (actividades, productos, roles, entre otros), es susceptible de variar con el tiempo según las necesidades del usuario. Por ser motivo, los autores consideran esencial reflejar, de una manera controlada, la flexibilidad de los elementos del proceso. Para ellos, los autores proponen etiquetar cada elemento de procesos como *flexible* o como *no constante*.

Para finalizar, la Tabla II.8 muestra el esquema de caracterización de esta propuesta.

Propuesta de Ferreira

Ferreira [Ferreira *et al.* 2011] presenta su propuesta para diseñar e implementar procesos software. Para ello, la propuesta considera que el desarrollo adecuado de un modelo de proceso software debe contemplar una fase inicial de diseño o modelado y una fase final de implementación. La fase de implementación consiste en especificar los detalles de bajo nivel del proceso con el fin de desplegarlo en un entorno ejecutable. De esta manera, los autores proponen un metamodelo para soportar la fase de diseño y un conjunto de reglas de transición para hacer un mapeo entre la fase de diseño y la fase de implementación.

Tabla II.8. Esquema de caracterización FlexUML [Martinho *et al.* 2008]

	Expresividad													
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
FlexUML	×	×	×		×	×	×	×					×	

Respeto a la fase de diseño, Ferreira propone un metamodelo basado en Diagramas de Componentes de UML. En este sentido, los conceptos definidos son: *«ProcessArchitecture»*, el cual está compuesto de *«Components»* (metaclase de UML); *«Component»*, que se especializa como *«SoftComponents»* (utilizado para declarar componentes arquitecturales sin una semántica definida) y *«HardComponent»* (que modela componentes con semántica propia por medio del concepto *«ModuleLibrary»*); y *«ModuleLibrary»*, que proporciona un repositorio de procesos con el fin de apoyar la gestión de las descripciones textuales de los métodos aplicados en el desarrollo de software mediante la definición de módulos de proceso.

Respecto a la fase de implementación, los autores definen reglas de transición y sistemáticas para mapear cada concepto del Diagrama de Componentes de UML a constructores del lenguaje Little-JIL [Wise 2006]: un lenguaje de programación para la coordinación de agentes. A modo introductorio, Little-JIL es un lenguaje ejecutable de programación de procesos a alto nivel que permite definir procesos de negocio y gestionar la coordinación de agentes y recursos. Este lenguaje define una sintaxis formal y rigurosa sustentada sobre una semántica operacional [Wise *et al.* 2000].

Para finalizar, la Tabla II.9 muestra el esquema de caracterización de esta propuesta.

Tabla II.9. Esquema de caracterización Ferreira [Ferreira *et al.* 2011]

	Expresividad													
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
Ferreira	×						×		×	×		×		

SPEM2.0

Recientemente, desde el punto de vista de la estandarización se ha presentado una nueva versión de una propuesta definida para modelar procesos software. Esta propuesta se denomina SPEM2.0 [OMG 2008a].

SPEM2.0 es un lenguaje que toma MOF como punto de partida y proporciona un lenguaje para definir procesos de desarrollo de software y es conocido por proponer una división entre el uso y contenido de este tipo de procesos. SPEM2.0 se define además por medio de un perfil UML, lo que facilita el desarrollo de herramientas de soporte.

La desventaja que presenta SPEM2.0, a priori, es su complejidad debido a las dependencias y relaciones que existen entre los conceptos que contempla su especialización. Otro aspecto importante a tener en cuenta es que no aborda la ejecutabilidad de los procesos.

Para finalizar, la Tabla II.10 muestra el esquema de caracterización de esta propuesta.

Tabla II.10. Esquema de caracterización SPEM2.0 [OMG 2008a]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
SPEM2.0	×	×	×		×	×	×	×	×		×		×	

xSPEM: eXecutable SPEM

La propuesta xSPEM [Bendraou *et al.* 2007a] contempla una posible definición para hacer ejecutable el estándar SPEM2.0. Esta propuesta añade una serie de conceptos para modelar las características de ejecución de un proceso (como es el caso de su estado interno). Para ello, los autores se centran en validar el modelo de proceso traduciéndolo a una red de Petri.

Para finalizar, la Tabla II.11 muestra el esquema de caracterización de esta propuesta.

Tabla II.11. Esquema de caracterización xSPEM [Bendraou *et al.* 2007a]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
SPEM2.0	×	×	×		×	×	×	×	×		×		×	

eSPEM: SPEM Extension for Enactable Behavior Modelling

La propuesta eSPEM [Ellner *et al.* 2010] es una extensión de SPEM para modelar el comportamiento actualizable de los procesos. Esta propuesta sustituye los conceptos de

interfaz de comportamiento por conceptos más granulados y específicos basados en conceptos de UML para el modelado de comportamiento, así como máquinas de estados de UML como soporte al modelado del ciclo de vida de los procesos.

Aunque hasta ahora no ha sido abordado, los autores de esta propuesta establecen como trabajo futuro la implementación completa de eSPEM por medio de una sintaxis abstracta y el diseño e implementación de herramientas de soporte que mejoren la usabilidad de eSPEM.

Para finalizar, la Tabla II.12 muestra el esquema de caracterización de esta propuesta.

Tabla II.12. Esquema de caracterización eSPEM [Ellner *et al.* 2010]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
eSPEM	×	×	×		×	×	×	×	×		×		×	

MODAL: Modal Oriented Development Application Language

MODAL [Koudri *et al.* 2010] es una propuesta basada en SPEM2.0 que persigue introducir conceptos adicionales para explotar el potencial del paradigma MDE en el modelado de procesos. MODAL se centra principalmente en la ejecución de los modelos del proceso, proporcionando una definición más clara de sus componentes y proporcionando un método para construir instancias ejecutables del modelo de proceso.

Para finalizar, la Tabla II.13 muestra el esquema de caracterización de esta propuesta.

Tabla II.13. Esquema de caracterización MODAL [Koudri *et al.* 2010]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
MODAL	×	×	×		×	×	×	×	×	×	×		×	

Propuesta de Ellner

Hace relativamente pocos años, el consorcio OMG[3] (*«Object Management Group»*) dedicado al cuidado y el establecimiento de diversos estándares, publicó un trabajo en el que se

[3] Sitio web: http://www.omg.org/

definía la semántica operacional de un subconjunto de actividades y acciones de UML2.0. Este trabajo fue llamado FUML (*«Foundational Subset for Executable UML Models»*) y su última versión ha sido publicada en [OMG 2013a]. A modo introductorio, FUML define un modelo UML de ejecución que actúa como almacén de información en el que se guarda la instancia de aquél modelo que se esté ejecutando. Además, FUML establece un mapeo entre la metaclase *«Activity»* de UML y el lenguaje de programación Java, posibilitando la instanciación y simulación de modelos UML.

Sin embargo, FUML resulta insuficiente para ejecutar modelos de procesos software que presentan una complejidad relativamente alta, en donde los actores del proceso trabajan de manera deslocalizada. Como solución, Ellner presenta una propuesta basada en FUML con la que pretende definir una máquina de ejecución distribuida para ejecutar modelos de procesos software [Ellner *et al.* 2011].

Para finalizar, la Tabla II.14 muestra el esquema de caracterización de esta propuesta.

Tabla II.14. Esquema de caracterización Ellner [Ellner *et al.* 2011]

	Expresividad														
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UM L	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
Ellner	×	×	×	×	×	×	×	×	×	×				

3. Conclusiones

Este capítulo resume el conocimiento previo adquirido que ha resultado necesario para proponer una aportación original en el ámbito de la gestión del ciclo de vida del proceso software y más concretamente en sus fases de definición, ejecución y orquestación del proceso.

En esta línea, este capítulo abarca dos aspectos bien diferenciados. Por un lado, este capítulo presenta cuáles son las propuestas de ciclo de vida BPM y por otro lado, identifica las propuestas más recientes de la literatura actual que están relacionadas con la definición y ejecución de procesos de negocio desde una perspectiva basada en modelos.

Respecto a las propuestas de ciclo de mejora continua BPM y como se menciona a lo largo del documento, la propuesta desarrollada en esta tesis se enmarca dentro de un ciclo de vida de mejora continua del proceso software y es por este motivo por el que surge la necesidad de estudiar qué propuestas existen en este ámbito y qué contemplan. De este estudio concluimos que, si bien la tendencia ha sido la de simplificar su definición y número de fases con el claro objetivo de facilitar su aplicación en diferentes contextos, existe un aspecto que no ha sido considerado de una manera clara y cuya necesidad se hace patente hoy en día, debido a la necesidad de interactuar con multitud de sistemas de información durante la ejecución de los procesos de una organización. Este aspecto es el

de establecer mecanismos que permitan especificar y llevar a cabo la orquestación de estas herramientas cuando los procesos software son ejecutados.

La necesidad de hacer una mención más clara de la orquestación del proceso se justifica porque a lo largo de la última década, cada vez más empresas ejecutan sus procesos de negocio de una manera colaborativa y global, para lo cual utilizan diferentes herramientas interconectadas entre sí [Bosch 2009]. Por ejemplo, en el ámbito de la industria software, es necesario interactuar con sistemas de control de versiones, sistemas de gestión documental, bases de datos, sistemas de información generales, frameworks y tecnologías de desarrollo, entre otros, cuando se ejecutan procesos asociados al desarrollo de software y su gestión. La capacidad de comunicación y coordinación del proceso con múltiples sistemas es lo que se conoce como orquestación del proceso.

Por otra parte, este capítulo resume un estudio del estado del arte sobre las propuestas basadas en modelos para la definición y ejecución de procesos software. Como se ha mencionado anteriormente, este estudio ha sido elaborado de manera conjunta con un miembro del grupo de investigación IWT2 y ha servido como punto de partida para plantear ambas Tesis Doctorales [García-Borgoñón *et al.* 2013]. Ambos trabajos de tesis se enmarcan dentro de una estrategia de actuación del grupo IWT2 (Capítulo VII) con la que trata de investigar cómo combinar satisfactoriamente el paradigma guiado por modelos con diferentes aspectos de la gestión de procesos de negocio.

Volviendo al estudio presentado en este capítulo, para comparar las propuestas más recientes que abordan la definición y ejecución de procesos software desde una perspectiva basada en modelos se ha utilizado el esquema de caracterización descrito en la Sección 1.3. La Tabla II.15 muestra de manera agrupada los esquemas de caracterización presentados a lo largo de este capítulo. Tras el estudio de estas propuestas y con la panorámica que proporciona esta tabla, se ponen de manifiesto varios aspectos importantes que resumimos a continuación.

En primer lugar, aunque existe un número significativo de propuestas, aspectos tales como la estandarización (es decir, conformidad a UML), ejecutabilidad y orquestabilidad parecen ser unos de los próximos retos a abordar por la comunidad investigadora para el futuro. Respecto a los estándares, SPEM2.0 podría ser la solución, pero su complejidad y falta de ejecutabilidad no lo permiten actualmente.

En segundo lugar, las propuestas identificadas y estudiadas a lo largo de este capítulo son definidas de manera teórica y no se han encontrado referencias publicadas que evidencien la aplicación práctica de estas propuestas. Es probable que existan, pero no se han podido localizar.

El hecho de que las propuestas basadas en modelos para la gestión de procesos no sean muy conocidas en contextos reales podría ser justificado argumentando que la práctica totalidad de las herramientas que permiten gestionar procesos de negocio,

conocidas normalmente por el acrónimo BPMS[4] (*«Business Process Management Suite»*), solo proporcionan soporte a BPMN (*«Business Process Model and Notation»*) [OMG 2011a], el cual es considerado como el estándar de facto para modelar procesos de negocio [Mendling *et al.* 2010]. Además, estas herramientas son poco flexibles y no suelen soportar otros tipos de lenguajes de modelado de procesos.

Tabla II.15. Resumen de la evaluación del esquema de caracterización de propuestas

	Expresividad													
	Actividad	Condicional	Ramas Paralelas	Manejador Excepciones	Productos	Roles	Conforme A UML	Comprensibilidad	Granularidad	Ejecutabilidad	Medible	Orquestabilidad	Herramienta Soporte	Regla Negocio
Chou	×		×	×	×	×	×	×	×	×				
Di Nitto	×	×	×		×	×	×	×	×	×		×	×	
UML-EWM	×	×			×	×	×	×	×				×	
UML4SPM	×	×	×	×	×	×	×	×	×	×			×	
Combemale	×	×			×	×	×	×	×					
UPME	×				×	×	×		×	×			×	
FlexUML	×	×	×		×	×	×	×					×	
Ferreira	×						×		×	×		×		
SPEM2.0	×	×	×		×	×	×	×	×		×		×	
xSPEM	×	×	×		×	×	×	×	×	×	×		×	
eSPEM	×	×	×		×	×	×	×	×		×		×	
MODAL	×	×	×		×	×	×	×	×	×	×		×	
Ellner	×	×	×	×	×	×	×	×	×	×				

Sin embargo, existen trabajos [Mazanek *et al.* 2011; Hlaoui *et al.* 2011; Geambaşu 2012] en los que se concluyen que es posible describir modelos equivalentes con BPMN y propuestas basadas en modelos UML; por ejemplo, utilizando diagramas de actividades de UML. De hecho, se han realizado evaluaciones por medio de patrones de workflow entre BPMN y diagramas de actividades de UML que han revelado que ambos lenguajes de modelado de procesos de negocio son equivalentes y proporcionan soluciones similares para la mayoría de los patrones definidos [Russell *et al.* 2004a; 2004b; 2006a; 2006b]. Por tanto, es posible establecer reglas de mapeo entre ambas representaciones [Geambaşu 2012].

En tercer lugar, se pone de manifiesto que estas propuestas no proporcionan mecanismos que: (*i*) posibiliten una mejora continua de los procesos (por ejemplo, a

[4] BPMS puede ser definido como una nueva categoría de software empresarial que permite a las empresas modelar, implementar y ejecutar conjuntos de actividades interrelacionadas – es decir, Procesos- de cualquier naturaleza, sea dentro de un departamento o permeando la entidad en su conjunto, con extensiones para incluir los clientes, proveedores y otros agentes como participantes en las tareas de los procesos.

través de la definición de indicadores o métricas); (*ii*) permitan definir cómo orquestar el proceso; y (*iii*) contemple la posibilidad de definir reglas de negocio para controlar el flujo de datos y ejecución del proceso bajo determinadas condiciones.

Por último, un aspecto clave y fundamental que ha motivado la realización de esta tesis es proporcionar soporte CASE (*«Computer Aided Software Engineering»*, Ingeniería de Software Asistida por Computadora) para hacer posible la aplicabilidad en entornos reales del marco teórico presentado en este trabajo. En este sentido, el grupo de investigación en cuyo seno se desarrolla esta tesis — grupo Ingeniería Web y Testing Temprano (IWT2) — tiene una amplia trayectoria en la ejecución de proyectos I+D y proyectos de transferencia. Gracias a esta experiencia, hemos podido constatar que el uso de herramientas de soporte mejora la aplicación y adaptación satisfactoria de resultados de investigación en entornos reales.

Respecto a este soporte, la mayoría de las propuestas descritas a lo largo de este capítulo proporcionan herramientas CASE que dan soporte durante la aplicación de la propuesta en cuestión.

La necesidad de ofrecer procesos sistemáticos o incluso automatizados de definición, generación o incluso validación de resultados es relevante en este tipo de entornos. De la misma forma se requiere la necesidad de facilitar el trabajo de los ingenieros de procesos con la ayuda de herramientas CASE.

Capítulo III PLANTEAMIENTO DEL PROBLEMA

La presentación del estudio del estado del arte del capítulo anterior ha servido para enmarcar cómo la definición y ejecución de procesos de negocio es tratada en la actualidad desde un punto de vista basado en modelos.

Gracias al punto de partida que proporciona las conclusiones del capítulo anterior, este capítulo comienza con una descripción detallada y precisa de cuáles son los problemas a resolver. Una vez conocido, por el capítulo anterior, el estado del arte en cuanto a propuestas basadas en modelos que presentan su particular visión respecto a la definición y ejecución de procesos, el capítulo continúa planteando los objetivos de la tesis.

Con los objetivos marcados y el problema definido, el capítulo continúa describiendo brevemente cuáles han sido los trabajos que han influenciado la realización de la propuesta presentada en esta Tesis Doctoral. Posteriormente, este capítulo presenta la estructura de la aproximación de nuestra propuesta y concluye con una serie de consideraciones finales que cierran la definición del alcance del problema.

1. Planteamiento del problema

De las conclusiones obtenidas durante el capítulo anterior, se plantean los problemas que se quieren solventar en este trabajo.

1. El primer problema está fundamentalmente relacionado con el hecho de que, debido a las características intrínsecas de los procesos software (identificadas en el Capítulo I), su automatización se antoja como una tarea con un alto grado de complejidad [Ruiz-González *et al.* 2004] y un coste de implantación a tener en cuenta. Por este motivo, es muy común que dentro de las organizaciones software la aplicación de BPM se limite a la definición de procesos software siendo su posterior ejecución y orquestación realizada manual e independientemente por cada rol que participa en esos procesos. Además, el estudio presentado en el capítulo anterior aflora el hecho de que la capacidad de ejecutar y orquestar los modelos de procesos es, aún hoy, un reto a alcanzar por parte de la comunidad investigadora y empresarial.
2. El contexto descrito en el punto anterior acarrea muy a menudo una dificultad añadida para llevar a cabo un seguimiento, control y medición adecuado de los procesos, lo que, en última instancia, redunda en la dificultad para efectuar una mejora continua dentro de un marco concreto que gestione el ciclo de vida de los procesos de negocio.

3. El tercer gran problema que se plantea, consiste en la necesidad de dar soporte CASE (*«Computer Aided Software Engineering»*) a todas estas ideas [Yu *et al.* 1997; Bandara *et al.* 2005], es decir, un soporte basado en herramientas. El tratamiento de la gestión de procesos de negocio desde una perspectiva basada en modelos debe hacerse de manera adecuada, más aún cuando los procesos que se desean modelar son críticos para la organización. En estos casos, el coste del proyecto o implantación de esta solución puede verse seriamente encarecido por la incorporación de nuevos protocolos, técnicas y modelos para la gestión de procesos de negocio. Por ello, es necesario ofrecer entornos prácticos y ágiles basados en herramientas CASE que permitan sistematizar o incluso automatizar el marco de trabajo.

Con todo esto, por tanto, se hace palpable la necesidad de establecer dentro de las organizaciones software, mecanismos eficaces y automáticos que posibiliten una gestión eficaz del ciclo de vida del proceso software. Con esta Tesis Doctoral se pretende dar soporte a las fases de definición, ejecución y orquestación de este ciclo de vida. Todo ello con el propósito facilitar el desarrollo de productos software y su gestión.

Este problema, fundamentado sobre el estudio del estado del arte de las tendencias actuales expuesto en el Capítulo II, es resuelto en el desarrollo de esta tesis con un marco teórico (definido bajo el paraguas del paradigma MDE) cuyo máximo exponente se traduce en una herramienta CASE: el marco de trabajo PLM_4BS – «*Process Lifecycle Management for Business-Software*» –, un marco para gestionar el ciclo del proceso en el negocio software.

Además, tal y como se ha reflejado a lo largo de capítulos anteriores, este trabajo de tesis se ve inspirado en cierta medida por la estrategia de gestión del ciclo de vida de producto – estrategia PLM, *«Product Life cycle Management»* –, estrategia muy aplicada en el sector industrial. Con todo ello, el marco de trabajo PLM_4BS presenta una posible solución para acercar la estrategia de gestión del producto software hacia las organizaciones del sector.

Finalmente, la herramienta PLM_4BS nace con el firme propósito de facilitar su aplicación en proyectos y entornos reales, ya que la necesidad de ofrecer procesos sistemáticos o incluso automatizados de definición, generación o incluso validación de resultados es muy importante en este tipo de entornos. De la misma forma se requiere la necesidad de facilitar el trabajo de los ingenieros de procesos con la ayuda de herramientas CASE.

2. Objetivos

Después de concretar cuál es el alcance del problema, llega el momento de describir de manera concreta los objetivos que se pretenden alcanzar con esta Tesis Doctoral. En este sentido, se establecen los siguientes objetivos a alcanzar:

1. **Definir un ciclo de vida BPM** que siente las bases teóricas necesarias que posibiliten el modelado, la ejecución, la orquestación y la medición de procesos software, todo ello enmarcado dentro de un ciclo de mejora continua de los mismos. Este objetivo queda resuelto en la Sección 4 de este capítulo.

2. Definir un **Metamodelo de Definición de Procesos Software** para poder modelar procesos software de acuerdo a los estándares actuales y un **Metamodelo de Ejecución y Orquestación de Procesos Software** con el que sea posible definir el contexto de ejecución del proceso. El Capítulo IV describe de manera formal, completa y exhaustiva ambos metamodelos.

3. Definir un conjunto de **Mecanismos de Derivación Sistemáticos** que permitan generar el metamodelo de ejecución y orquestación a partir del metamodelo de definición de procesos software. El contenido teórico que resuelve este objetivo se describe en el Capítulo V de este documento.

4. Desarrollar una versión aplicable y generalista de los metamodelos y de los mecanismos de derivación mencionados en los dos objetivos anteriores. Este objetivo conlleva: (*i*) definir **perfiles** o lenguajes específicos de dominio adecuados para que sea posible instanciar ambos metamodelos; e (*ii*) implementar una **herramienta CASE** que dé soporte a dicho entorno (esta herramienta es bautizada con el nombre de PLM$_4$BS «*Process Lifecycle Management for Business-Software*»). Este objetivo queda resuelto en el Capítulo VI.

5. **Evaluar los resultados obtenidos** en un entorno de producción basado en la instanciación de la solución en un entorno metodológico real cuya aplicación haya sido considerada como satisfactoria. Este objetivo queda resuelto en el Capítulo VI.

3. Influencias

Los resultados de investigación expuestos en los siguientes capítulos, están influenciados por diferentes trabajos.

Por una parte, existe una influencia directa desde las propuestas existentes sobre la definición, ejecución y orquestación de procesos de negocio desde una perspectiva basada en modelos que han sido estudiados durante el capítulo anterior. Además, otra influencia importante y relacionada con la anterior ha sido el paradigma de ingeniería guiada por modelos.

Por otro lado, una de las influencias más relevantes que han propiciado y motivado la realización de este trabajo de tesis ha sido la última evolución que ha experimentado la metodología NDT («*Navigational Development Techniques*») [Escalona 2004; Escalona *et al.* 2008]. Esta influencia se justifica por el hecho de que el desarrollo de esta Tesis Doctoral se gesta dentro del seno del grupo de investigación IWT2, uno de cuyos principales activos es la metodología NDT y el conjunto de herramientas asociadas.

Tal y como se describe a continuación en la Sección 3.2, la metodología NDT ha evolucionado recientemente hacia una definición basada en procesos lo que ha servido como catalizador principal para el planteamiento y elaboración de esta tesis.

Una vez introducidas, esta sección describe con más detalle las propuestas más influyentes que han servido como base para obtener los resultados que se presentan a lo largo de capítulos posteriores.

3.1. Ingeniería dirigida por modelos (MDE)

Durante las últimas cinco décadas, investigadores y desarrolladores de software han ido creando lenguajes cada vez más abstractos para ayudarles a programar sobre componentes hardware cada vez más complejos. Esta tendencia a elevar el nivel de abstracción en los lenguajes de programación supuso la aparición de bastantes paradigmas de programación a lo largo de este periodo.

A pesar de estos avances, sigue existiendo un problema de fondo, y es que el crecimiento de la complejidad de las plataformas sobre la que trabaja el desarrollador es mucho más rápido que la capacidad de los lenguajes de uso general para ocultar dicha complejidad. Por ejemplo, algunas plataformas populares, tales como J2EE (*«Java Enterprise Edition»*) [Rubinger *et al.* 2014] y .NET (*«.NET Framework API de Microsoft»*) [Millas 2013], entre otras, contienen miles de clases y métodos con un complejo intrincado de dependencias que pueden provocar efectos secundarios inesperados. Para mitigar esto, se requiere un esfuerzo considerable por parte de los desarrolladores para conocerlas en profundidad y así, evitar errores. El problema se acrecienta cuando estas mismas plataformas crecen rápidamente y además aparecen otras nuevas, o nuevas versiones, con el consiguiente esfuerzo de migración de unas a otras.

En este último caso, cuando se produce la evolución tecnológica de las plataformas o de los sistemas, es importante conservar el mismo modelo conceptual del negocio, es decir, que la lógica del dominio del problema debe ser la misma, sea cual sea la plataforma o lenguaje que implementa dicha lógica.

La ingeniería guiada por modelos (llamada MDE por sus siglas en inglés, «*Model-Driven Engineering*») [Schmidt 2006] surgió para abordar la complejidad de los sistemas software con el objetivo de expresar los conceptos del dominio del problema de una manera efectiva. En esta línea, el principio básico de MDE es «*Everything is a model*» [Bézivin 2005].

La principal idea de MDE es utilizar un conjunto de modelos para ir disminuyendo el nivel de abstracción [Fondement *et-al.* 2004]. Así, en las primeras etapas del desarrollo se elaboran modelos con un alto nivel de abstracción para representar un aspecto relevante del sistema y, a medida que el desarrollo avanza, se van desarrollando nuevos modelos con un nivel de detalle más alto y más cercanos a la implementación final del sistema. Sin embargo, al trabajar con modelos de esta forma, surgen dos necesidades principales que hay que satisfacer [Fondement *et-al.* 2004]: (*i*) disponer de un conjunto de elementos comunes para que todos los modelos se desarrollen de la misma forma; y (*ii*) definir mecanismos que hagan posible la obtención de unos modelos a partir de otros modelos

diferentes, de una manera sistemática y susceptible de automatización de forma parcial o total.

Para resolver la primera necesidad, surge el concepto de metamodelo. El objetivo de un metamodelo es definir las relaciones entre conceptos de un dominio de problema y definir de manera precisa la semántica asociada a dichos conceptos del dominio [Schmidt, 2006]. Así, un metamodelo engloba el conjunto de constructores que representan los conceptos del metamodelo, las asociaciones válidas entre los constructores, las restricciones existentes y la definición del significado [Giachetti *et-al.* 2008]. Otra terminología utilizada, nombra a los metamodelos como sintaxis abstracta y a los modelos como sintaxis concreta.

Para resolver la necesidad de derivar nuevos modelos a partir de modelos ya existentes, se usa el mecanismo de transformaciones. Una transformación entre dos modelos representa una relación entre dos sintaxis abstractas o metamodelos y se define mediante un conjunto de relaciones entre los elementos del correspondiente metamodelo [Thiry *et-al.* 2009]. Una transformación es una expresión que relaciona los elementos de un metamodelo de origen con los elementos de un metamodelo de destino, de tal manera que cuando la transformación es realizada proporcionando un modelo de entrada conforme al metamodelo de origen, la transformación sabe qué hacer con dichos elementos y cómo utilizarlos para construir los elementos del modelo de salida conforme al metamodelo de destino.

MDE puede ser aplicado en la práctica de varias formas siendo una de ellas la definida por el estándar MOF («*Meta-Object Facility*») [OMG 2011b]. MOF es considerado como un metametamodelo, es decir, una herramienta para construir metamodelos (lenguajes de modelado) o incluso lenguajes de transformación que utilizan aquellos como base para especificar transformaciones.

De manera general, la Figura III.1 muestra cómo se combinan todos estos conceptos para la generación de nuevos modelos a partir de otros modelos ya existentes. Esta figura se irá completando a lo largo de capítulos posteriores a medida que se concrete qué lenguaje de definición de metamodelos y de transformaciones será utilizado.

La ingeniería guiada por modelos aparece como una posible guía para plantear una solución a los objetivos planteados al principio de este capítulo. Esta elección viene justificada por varias razones:

1. El paradigma MDE trabaja con modelos para representar la información de un determinado dominio. En el contexto de esta tesis, con el uso de modelos se hace posible la formalización de la información de los procesos software.
2. El paradigma MDE también trabaja con transformaciones que son una herramienta posible para formalizar el proceso de transformación entre modelos.
3. Otras razones que también justifican la elección del desarrollo guiado por modelos son la reutilización que se hace de trabajos plenamente asentados en el ámbito del software – y de forma particular, su uso exitoso en la metodología NDT, tal y como se expone en la siguiente sección.

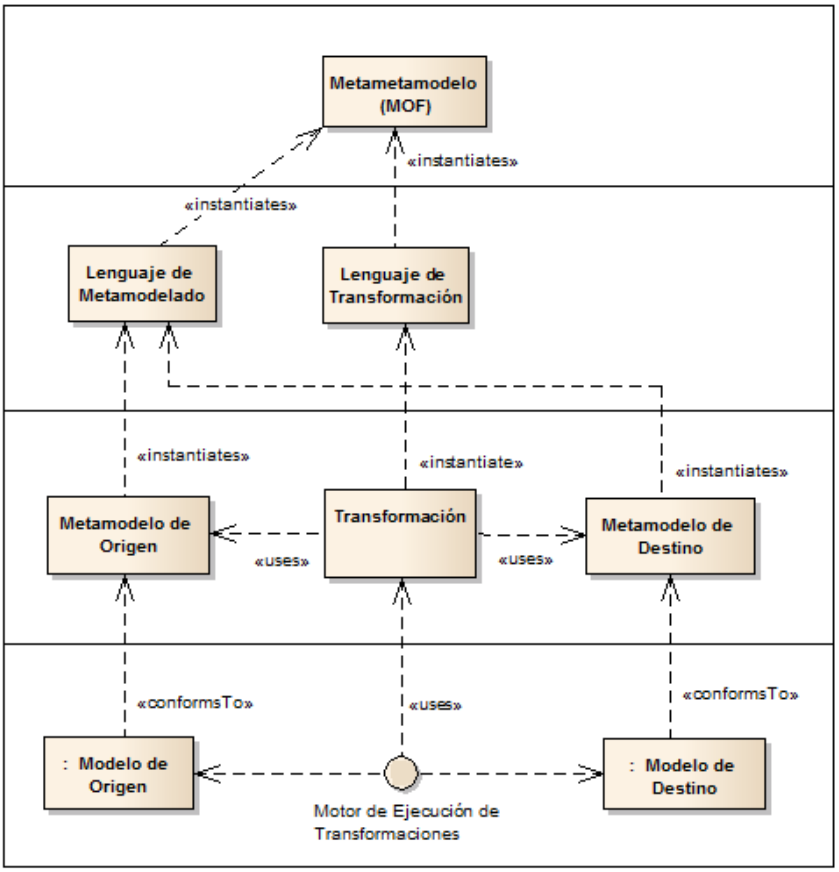


Figura III.1. Proceso de transformación del paradigma MDE

3.2. Metodología NDT (Navigational Development Techniques)

Como se ha comentado anteriormente, la última evolución que ha experimentado la metodología NDT (acrónimo de *«Navigational Development Techniques»*) [Escalona 2004], [Escalona *et-al.* 2008] ha sido el detonante para la elaboración de esta tesis.

La propuesta metodológica NDT comienza su gestación en el año 2002, pero no es hasta el año 2004 cuando cristaliza como base de un trabajo de tesis presentado en el seno del departamento de Lenguajes y la Sistemas Informáticos de la Universidad de Sevilla. Dicha propuesta estaba centrada en potenciar el aspecto navegacional en sistemas web e hipermedia, sin embargo incorporaba un proceso y un conjunto completo de artefactos para abordar gran parte del ciclo de vida de desarrollo de un sistema.

Inicialmente, la metodología NDT se centraba en la fase de Requisitos y la fase de Análisis, prestando especial énfasis en potenciar los requisitos, su trazabilidad con el análisis y la ejecución de transformaciones, con el objetivo de proporcionar el mayor soporte posible a esta etapa de desarrollo. Para ello, NDT definió de manera formal, a través de un conjunto de metamodelos, todos los elementos de la fase de Requisitos y de Análisis. Además, definió un conjunto de reglas de trasformación para generar los modelos de análisis desde los modelos de requisitos.

La Figura III.2 esquematiza el proceso de transformación para generar los modelos de la fase de Análisis desde la fase de Requisitos.

Como se puede apreciar en la figura, la fase de requisitos de NDT está sustituida por tres actividades principales: captura, en la que se utilizan diferentes técnicas de captura de requisitos tales como entrevistas o «*brainstorming*» para averiguar los objetivos y requisitos del sistema; definición, actividad en la que se formaliza la información capturada a partir de las distintas entrevistas y reuniones con el cliente; y validación, en la que se contempla la utilización de técnicas de valuación y conciliación de requisitos.

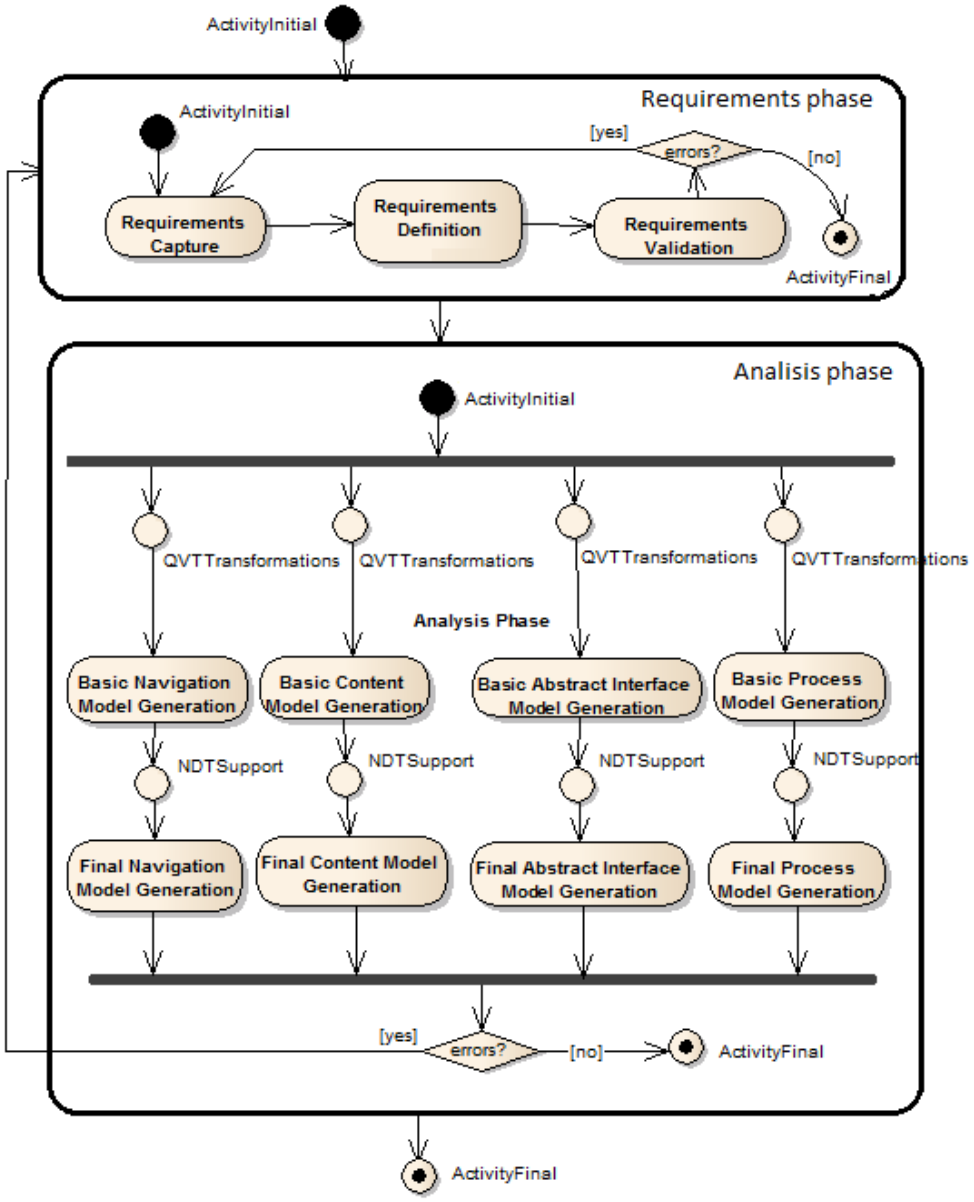


Figura III.2. Generación NDT de la fase de Análisis desde la fase de Requisitos

Una vez que la fase de Requisitos ha finalizado, es posible generar los modelos básicos de la fase de Análisis de NDT aplicando diferentes reglas de transformación; todas ellas representadas en la Figura III.2 mediante el estereotipo «*QVTTransformations*». NDT contempla cuatro modelos en la fase de análisis: el modelo de navegación, en el que se refleja cómo puedes navegar los usuarios a través del sistema; el modelo de contenido, que representa la estructura estática del sistema; el modelo de procesos, que representa la

estructura funcional del sistema; y el modelo de interfaz abstracta, en el que se representa un conjunto de prototipos de la interfaz del sistema.

A continuación, una vez que se han definido los modelos básicos, el equipo de analistas puede realizar transformaciones controladas con el objetivo de completar y enriquecer los modelos para obtener el modelo definitivo. Este paso no es automático y requiere la experiencia del analista. Estas transformaciones están representadas en la Figura III.2 mediante el estereotipo *«NDTSupport».*

Hasta ahora se ha descrito el planteamiento inicial de la metodología, pero NDT continúa evolucionando hasta el punto de que, actualmente, proporciona soporte a todas las fases del ciclo de vida software (Estudio de Viabilidad, Ingeniería de Requisitos, Análisis, Diseño, Implementación, Pruebas, y Mantenimiento) contemplando para ello, un conjunto de metamodelos para cada una de ellas. Además, define nuevas reglas de transformación para generar modelos de una fase de manera sistemática a partir de otros modelos de fases anteriores. Los metamodelos definidos se formalizan mediante perfiles de UML, con el aporte de OCL [ISO/IEC 2012] para la definición de las restricciones sobre los conceptos de los metamodelos, y el soporte de la herramienta Enterprise Architect (EA) [SparxSystems 2014] como herramienta de modelado UML de referencia de la metodología. Las transformaciones se formalizan mediante el lenguaje QVT [OMG 2008b].

Toda esta definición teórica de la metodología se antojaba impracticable en contextos empresariales debido al uso de una terminología demasiado abstracta (metamodelos, transformaciones, conceptos, etc.).

Por todo ello, se hizo necesario desarrollar herramientas software que ocultasen esta terminología teórica para mejorar la aplicabilidad de NDT en estos entornos y proyectos reales. Esto se logró con el desarrollo de NDT-Suite [IWT2 2014; Garcia-Garcia *et al.* 2012a; Garcia-Garcia *et al.* 2014], un paquete de herramientas Java cuyas herramientas son:

- **NDT-Profile** [Garcia-Garcia *et al.* 2014]. Cuando una metodología es definida de manera formal utilizando metamodelos –como es el caso de NDT– es posible garantizar uniformidad, una terminología formal y una correcta definición de objetos. Sin embargo, si no se dispone de una herramienta de soporte, la aplicación de dicha metodología puede resultar demasiado compleja y costosa. En el caso de NDT, es NDT-Profile la herramienta de soporte que soluciona este aspecto gracias a su integración dentro de Enterprise Architect (EA). NDT-Profile no sólo proporciona un proyecto base en EA, sino que además implementa cada uno de los metamodelos de NDT utilizando los mecanismos de extensión de UML junto con la funcionalidad MDG (*«Model Driven Generation»*) de Enterprise Architect.
- **NDT-Quality** [Garcia-Garcia *et al.* 2013a]. Aunque la herramienta NDT-Profile proporciona un entorno formal para utilizar la metodología NDT y gestionar todos sus artefactos, es posible que los analistas comentan errores e inconsistencias metodológicas cuando especifican sus proyectos. Esta es la razón de ser de la herramienta NDT-Quality. Esta herramienta cubre dos objetivos principales:

1 Garantizar la calidad del uso de la metodología NDT en cada una de sus fases del ciclo de vida software. Para ello, esta herramienta comprueba la consistencia de todas las restricciones OCL definidas en los diferentes metamodelos de NDT. Además, la herramienta controla otros aspectos tales como la completitud en las definiciones de elementos, la correcta identificación de artefactos siguiendo los patrones lingüísticos establecidos por la metodología, definiciones incorrectas de restricciones de artefactos, la existencia de ciclos en diagramas de actividades, etc.

2 Garantizar la trazabilidad entre artefactos cuando se utiliza las reglas de transformación entre modelos definidas por la metodología. Este control es necesario porque NDT permite a los analistas llevar a cabo modificaciones de forma manual para enriquecer y completar sus modelos. Por tanto, es necesario asegurar la trazabilidad entre modelos (según las reglas definidas por la propia metodología).

— **NDT-Driver** [Garcia-Garcia *et al.* 2012b]. Esta herramienta es la encargada de implementar cada uno de las reglas de transformación QVT definidas por la metodología NDT. Uno de las características a destacar de esta herramienta, es que es posible su aplicación tanto en proyectos que siguen un ciclo de vida secuencial como evolutivo.

— **NDT-Prototype** [Saenz *et al.* 2010]. Esta herramienta permite generar un conjunto de prototipos a XHTML a partir de los modelos de navegación descritos en la fase de Análisis de un proyecto especificado con la herramienta NDT-Profile. Además de proporcionar un punto de partida para la fase de Implementación, esta herramienta brinda también la posibilidad de validar con el cliente los requisitos del proyecto de una manera gráfica e interactiva.

— **NDT-Merge** [Garcia-Garcia *et al.* 2012c]. Esta herramienta está relacionada íntimamente con el aseguramiento de la calidad cuando en el proyecto interactúan multitud de equipos de trabajo en la especificación del mismo. En este contexto, es posible que surjan inconsistencias cuando diferentes equipos analizan la misma necesidad del cliente. NDT-Merge se encarga de analizar y detectar inconsistencias sintácticas y semánticas en diferentes visiones del mismo catálogo de requisitos.

— **NDT-Report** [IWT2 2014]. Esta herramienta está integrada dentro de NDT-Profile y permite generar un entregable (en diferentes formatos ofimáticos) de la documentación contenida en un proyecto especificado con NDT-Profile.

— **NDT-Glossary** [Garcia-Garcia *et al.* 2011]. Esta herramienta define e implementa mecanismos MDE para generar un glosario terminológico a partir del catálogo de requisitos de un proyecto especificado con NDT-Profile. Disponer de un glosario común de términos presenta diferentes ventajas en un proyecto. Por ejemplo, entre otros aspectos, permite reducir riesgos en el proyecto debido a malentendidos y facilita la comunicación entre todos los interesados.

— **NDT-Counter** [Armario *et al.* 2012]. Esta herramienta proporciona una estimación del esfuerzo o coste necesario para desarrollar un proyecto. Para llevar a cabo esta estimación, la herramienta aplica la técnica de puntos de casos de uso [Karner 1993] sobre el catálogo de requisitos del proyecto definido con NDT-Profile.

Aunque el uso de metodologías – como es el caso de NDT– ayuda a asegurar la calidad de los resultados durante el desarrollo software, en el día a día de las empresas software muy a menudo acaecen muchos problemas que no deberían darse que provocan en última instancia, que la aplicación de las fases metodológicas sean consideradas como una mera formalidad sin utilidad. A veces incluso, proyectos enmarcados dentro de una metodología experimentan retrasos, cambios o parches de código que causan inconsistencias entre la documentación y el sistema final.

En este sentido, NDT ha evolucionado recientemente para proporcionar un marco de trabajo que motive el uso de nuevos enfoques, normas y paradigmas para el desarrollo de software de calidad. Todo ello desde una perspectiva basada en la definición de procesos software. A día de hoy, NDT cubre y define un conjunto de procesos categorizados en seis grupos:

1. Procesos de Desarrollo de Software, basados en el ciclo de vida software que define la propia metodología NDT.
2. Procesos de Mantenimiento de Software, basados en ITIL [ITIL 2014] y los modelos de madurez de CMMi (*«Capability maturity model integration»*) [Chrissis *et al.* 2011].
3. Procesos de Pruebas Software, basados en la reciente norma ISO/IEC 29119 [ISO/IEC 2014].
4. Procesos de Calidad del Software, basados en la norma ISO 9001:2008 [ISO/IEC 2008a] y en los modelos de madurez de CMMi.
5. Procesos de Gestión de Proyecto Software, basados en la guía de dirección de proyectos PMBOK [PMI 2008] y en los modelos de madurez de CMMi.
6. Procesos de Seguridad, basados en la norma ISO de seguridad ISO/IEC 27001:2009 [ISO/IEC 2009].

Todos estos procesos están definidos de manera formal y completa en la herramienta NDTQ-Framework [Ponce *et-al.* 2013] para que puedan ser aplicados en entornos reales. Para ello, NDTQ-Framework se sustenta sobre un metamodelo básico de definición procesos software (Figura III.3). Este metamodelo ofrece un mecanismo adecuado y simple para la definición de procesos de negocio, siendo, además, compatible con la norma ISO/IEC TR 24774 [ISO/IEC TR 2007a] la cual, proporciona una guía para la descripción de procesos.

Como se puede apreciar en la Figura III.3, el metamodelo está constituido por siete metaclases, de las cuales, la metaclase *«Process»* es la principal. Esta metaclase representa un conjunto de acciones ordenadas que deben ser ejecutadas para obtener un determinado objetivo o resultado de negocio. Cada una de estas acciones se representa

mediante la metaclase *«Activity»*. Además de una serie de atributos específicos, la metaclase *«Activity»* contempla una relación recursiva sobre sí misma para permitir segmentar la actividad en un conjunto de subactividades. El estándar ISO/IEC TR 24774 establece explícitamente que debe existir esta relación en cualquier metamodelo de definición de procesos que siga el estándar.

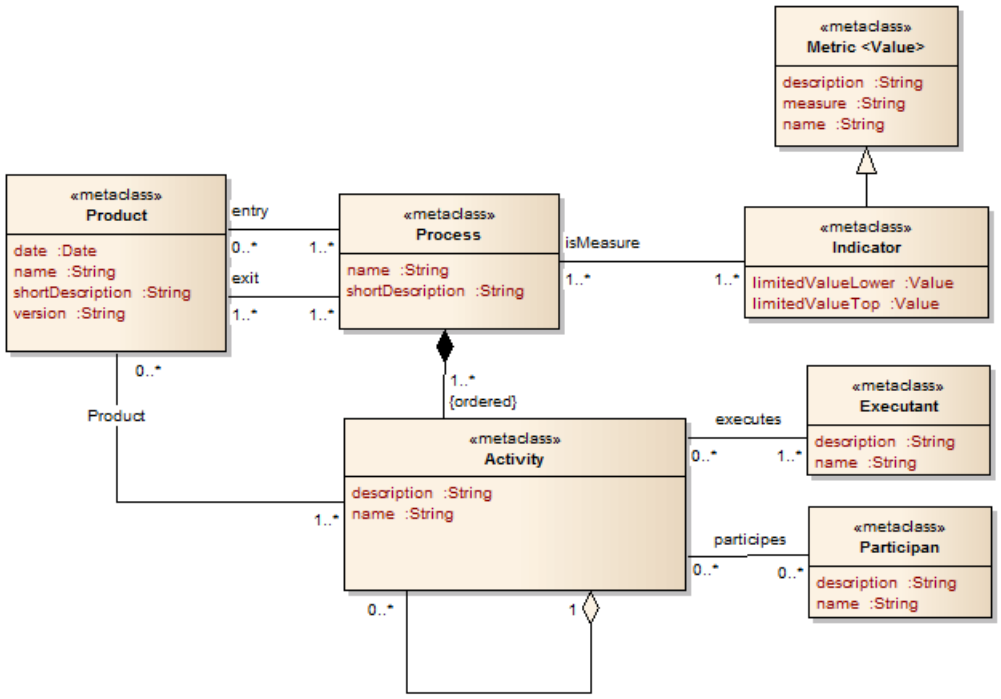


Figura III.3. Metamodelo de procesos en NDTQ-Framework [Ponce *et-al.* 2013]

Por otra parte, el metamodelo diferencia entre dos tipos de interesados o *«stakeholders»* que puede desempeñar una actividad. Esta diferenciación está basada en el grado de involucración del interesado en la ejecución de la actividad. Estos tipos se representan mediante las metaclases *«Executant»* y *«Participant»*. La diferencia entre ambos radica en que la metaclase *«Executes»* representa al actor responsable de llevar a cabo la actividad mientras que la metaclase *«Participes»* representa cualquier interesado que participa en la actividad sin ser directamente el responsable de su consecución.

Desde el punto de vista del producto, el metamodelo define la metaclase *«Product»* como entidad que representa los productos (al menos uno) obtenidos como resultado de ejecutar el proceso – relación con rol *«exit»* entre las metaclases *«Product»* y *«Process»* – y los productos considerados como datos de entrada al proceso – relación con rol *«entry»* entre las metaclases *«Product»* y *«Process»*.

Finalmente, la metaclase *«Metric»* se contempla en el metamodelo para definir métricas que permitan medir el desempeño del proceso. En esta línea, también se ha definido la metaclase *«Indicator»* con el objetivo de medir una métrica concreta contra un rango posible de valores – atributos *«limitedValueTop»* y *«limitedValueLower»* de la metaclases *«Indicator»*.

Actualmente y al igual que ocurría en sus inicios, la metodología NDT define de manera formal y teórica un marco de trabajo – denominado NDTQ-Framework –. En este

caso, el marco de trabajo está basado en procesos software especificados conforme a un metamodelo de definición de procesos.

Sin embargo, aunque NDTQ-Framework proporciona la guía necesaria para el desarrollo de software con calidad, no deja de ser una definición estática de los procesos. Es necesario por tanto, que la ejecución de estos procesos se pueda efectuar de manera dinámica en entornos reales. Ésta ha sido la principal influencia que ha servido como catalizador para elaborar la Tesis Doctoral presentada en este documento y que ha cristalizado en el marco de trabajo PLM_4BS para la gestión del ciclo de vida del proceso en el negocio software.

Es interesante mencionar un posible valor añadido que podrá apreciarse cuando se combine el uso de la metodología NDT con la propuesta PLM_4BS en entornos reales. Como se mencionaba anteriormente, NDT potencia la fase de Requisitos como fuente de información para el resto de fases del ciclo de vida software. Entre los diferentes tipos de requisitos considerados por NDT se encuentran los Requisitos de Interacción, con los que es posible capturar y especificar las necesidades navegacionales e interfaces de usuario del sistema a desarrollar.

Por último, puede ser interesante mencionar que NDT se ha utilizado con éxito en diferentes proyectos I+D a lo largo de la última década. Entre ellos, se pueden mencionar los siguientes: el proyecto Mosaico llevado a cabo en el seno de la Consejería de Cultura y deporte de la Junta de Andalucía [Escalona *et-al.* 2008]; el proyecto AQUA-WS [Cutilla *et-al.* 2011] llevado a cabo en EMASESA (Empresa Metropolitana de Abastecimiento y Saneamiento de Aguas de Sevilla S.A.); y más recientemente el proyecto de CALIPSOneo [Escalona *et-al.* 2013; Salido *et-al.* 2014] llevado a cabo de forma conjunta con Airbus EADS|CASA.

4. Estructura de la aproximación

Una vez descritas las influencias desde las que emerge este trabajo de tesis así como planteado ya el problema y los objetivos, esta sección presenta una propuesta de solución para dar soporte a las primeras fases del ciclo de vida de gestión de procesos de negocio; más concretamente a las fases de definición, ejecución y orquestación de procesos software.

Teniendo en cuenta que la tesis se apoya sobre el ciclo de vida de gestión del proceso, es necesario estudiar y plantear cuáles son las fases de las que consta dicho ciclo de vida BPM para localizar en cuáles de ellas se debe realizar la definición, ejecución y orquestación de los procesos.

4.1. Propuesta de ciclo de vida y mejora continua BPM

En el primer capítulo de esta tesis se hace hincapié en la necesidad de enmarcar la gestión de procesos dentro de un marco de mejora continua para mejorar la productividad y la competitividad de las empresas que utilizan dicho marco como estrategia de negocio. Es por este motivo por el que el trabajo desarrollado en esta tesis se engrana dentro de un ciclo de vida de mejora continua para BPM y en este sentido, a lo largo de Capítulo II se

han mostrado las diferentes propuestas y visiones de lo que debería abarcar un ciclo de vida BPM [Van-der-Aalst 2004a, 2004b; Hill *et al.* 2006] y otras propuestas más generalistas como el ciclo de mejora continua de Shewhart [Shewhart 1986].

Sin embargo, tal y como concluye el capítulo anterior, en estas propuestas no se aborda clara y específicamente la necesidad de orquestar los procesos. Esta necesidad se fundamenta en el hecho de que a lo largo de la última década, la orquestación de procesos ha llegado a ser un aspecto importante dentro de las organizaciones – y en particular dentro de las organizaciones software – debido a que cada vez más, los procesos de negocio se interconectan y coordinan entre sí mediante diferentes herramientas y tecnologías para alcanzar un determinado objetivo de negocio y organizacional [Bosch 2009].

Por este motivo, esta tesis – inspirada en las propuestas de Shewhart [Shewhart 1986] y van der Aalst [van der Aalst 2004a; 2004b] – propone una visión particular y una redefinición de las fases que debería contener un ciclo de vida para la gestión de procesos de negocio dentro de un marco de mejora continua.

El ciclo de vida BPM propuesto, se muestra en Figura III.4 y sirve como base sobre la que se sustente toda la propuesta de solución presentada en esta tesis.

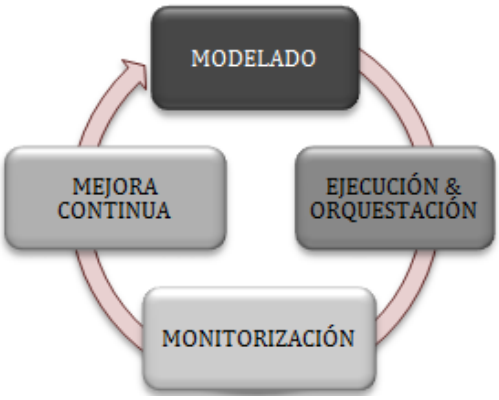


Figura III.4. Ciclo de vida y mejora continua para la gestión de procesos software

El ciclo de vida BPM que se propone en la figura anterior está compuesto por las siguientes cuatro fases:

1. **Modelado**. En esta fase, el ingeniero de procesos puede modelar y describir los procesos software de la organización de una manera estructurada. Durante esta fase, será posible definir roles funcionales, actividades a desempeñar, entregables a generar durante el proceso, entregables necesarios para completar cada actividad, reglas de proceso para guiar el flujo de control – es decir, el orden en el cual cada actividad es ejecutada– del proceso, definir métricas e indicadores para establecer cómo controlar – durante la fase de monitorización– el proceso de negocio definido.

2. **Ejecución y Orquestación**. Esta es la fase en la que se debe especificar cuáles son los parámetros del contexto de ejecución del proceso y la forma en la que se van a orquestar todos aquellos procesos que lo requieran. Siguiendo la definición proporcionada en la Tabla I.4, la orquestación implica la definición de todos aquellos eventos de coordinación necesarios para la ejecución del proceso, posibilitando la gestión bajo demanda de los servicios que surja durante la ejecución. Toda esta especificación hará posible la instanciación de los procesos modelados.
3. **Monitorización**. Dentro de cualquier tipo de organización, sin importar su tamaño, es esencial evaluar el desempeño de sus procesos, ya que esta información puede llegar a proporcionar una visión detallada de la productividad global de cada unidad de negocio de la organización. Esta fase del ciclo de vida se focaliza precisamente en este aspecto: realizar un control y seguimiento del desempeño de los procesos a partir de la información obtenida durante su ejecución. Para llevar a cabo este control es necesario previamente haber definido – durante la fase de modelado – aquellas métricas e indicadores clave de rendimiento con los que medir el proceso.
4. **Mejora continua**. Una vez evaluado el rendimiento de los procesos, es conveniente –y recomendable– que la organización analice los datos obtenidos y considere la posibilidad de optimizar sus procesos software como estrategia de gestión para mejorar su competitividad. Esto es lo que se conoce como mejora continua y es un aspecto muy recomendado por diferentes normas internacionales; entre ellas la norma ISO 9001:2008.

4.2. Arquitectura de la propuesta de solución

Una vez definido y concretado en la sección anterior el ciclo de mejora continua BPM, se presenta en la Figura III.5 una posible arquitectura MDE para una solución que dé soporte a las fases de definición, ejecución y orquestación del proceso software que forman parte del ciclo de vida BPM de la Figura III.4.

Como puede apreciarse en la figura, la arquitectura MDE de la propuesta de solución se ha divido en los mismos niveles que la Figura III.1. De izquierda a derecha, estos niveles son: (*i*) lenguaje MOF; (ii) estándares sobre los que se apoya la propuesta; (*iii*) metamodelos y reglas de transformación que componen la solución; y (*iv*) un entorno real donde se establece una correspondencia de cada metamodelo definido con las fases del ciclo de vida BPM mostrado en la Figura III.4.

Como se ha mencionado, la propuesta está compuesta por dos metamodelos: el primero, para modelar procesos software y el segundo, para definir las propiedades asociadas con la ejecución y orquestación del proceso. Ambos están basados en el estándar de metametamodelado MOF («*Meta-Object Facility*») [OMG 2011b] propuesto por la OMG. Este estándar es el usado como base para definir UML y lenguajes de transformación tales como QVT («*Query/View/Transformation*») [OMG 2008b] y MOFM2T («*MOF Model to Text Transformation Language*») [OMG 2008c], entre otros.

Tanto el metamodelo de definición como el de orquestación y ejecución de procesos están basados en UML. Por este motivo, en la Figura III.5 se ha establecido una relación con el estereotipo «*instantiate*» desde cada uno de estos metamodelos hacia MOF.

Tal y como se adelantó en la Figura I.1 del capítulo introductorio, la propuesta deberá contemplar dos tipos de reglas de transformación: reglas para derivar un modelo respecto a otro y reglas para generar código en un formato entendible por un motor BPM. Para definir el primer conjunto de reglas se utilizará la especificación QVT – este lenguaje permite definir las transformaciones «*model-to-model*» –, mientras que para definir el segundo conjunto de reglas se utilizará la especificación MOFM2T – este lenguaje permite definir las transformaciones «*model-to-text*» –. De igual modo que en el caso de los metamodelos, estas relaciones están recogidas en la Figura III.5 mediante enlaces con estereotipo «*instantiate*».

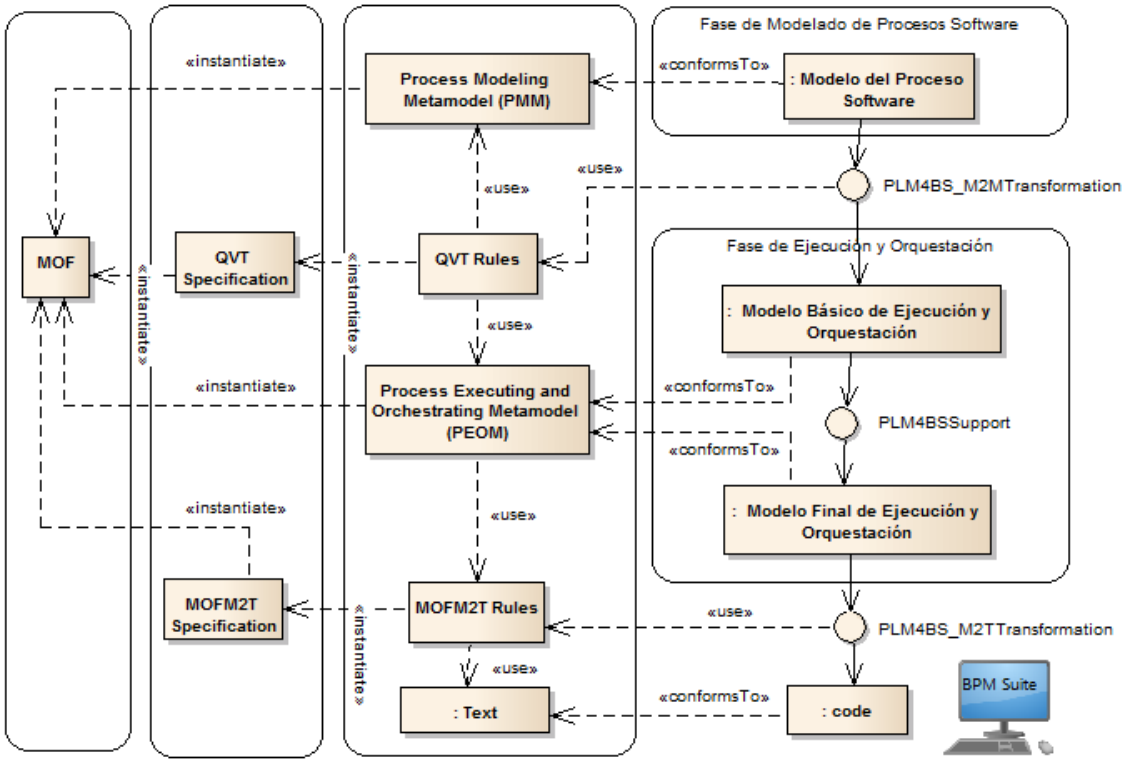


Figura III.5. Arquitectura MDE de la propuesta de solución PLM$_4$BS

Finalmente, desde el punto de vista del usuario, cualquier usuario podrá ser capaz de modelar sus procesos de negocios instanciando los metamodelos anteriores. Una vez que la fase de modelado ha sido completada, será posible ejecutar en un motor de ejecución las reglas de derivación QVT. De esta manera, una primera versión del modelo de ejecución y orquestación será generado. El proceso de construcción de este modelo estará formado por dos etapas diferenciadas:

1. La primera etapa se corresponde con una derivación sistemática desde el modelo con la definición del proceso para generar un modelo básico de orquestación y ejecución del proceso. Esta etapa está representada en la Figura III.5 con el estereotipo «*PLM4BS_M2MTransformation*».

2. La segunda etapa dependerá del conocimiento y habilidad del ingeniero de procesos para ajustar el modelo del proceso al contexto de ejecución concreto. Esta derivación no sistemática generará la versión final del modelo de orquestación y ejecución. Esta etapa está representada en la Figura III.5 con el estereotipo «*PLM4BSSupport*».

Una vez finalizada la fase de orquestación y ejecución, el usuario podrá generar una versión ejecutable del modelo aplicando sobre el propio modelo las reglas de transformación «*PLM4BS_M2TTransformation*». La salida de estas reglas de transformación será código ejecutable WS-BPEL que podrá ser desplegado en un motor BMP concreto.

5. Conclusiones

A lo largo de este capítulo se ha descrito el problema concreto que se desea resolver. Para ello, el capítulo ha comenzado centrando el alcance del problema a abordar analizando las conclusiones obtenidas tras el estudio del estado del arte desarrollado en el Capítulo II.

Planteado el alcance del problema, el capítulo define específicamente cuáles son los objetivos a alcanzar y las premisas que hay que seguir para alcanzarlos.

A partir de dichos objetivos, se han presentado las influencias que han resultado ser fuente de inspiración para la elaboración de este trabajo. Por último, se ha definido una propuesta sobre un ciclo de vida de mejora continua BPM así como una primera aproximación – con una arquitectura basada en el paradigma MDE – del marco de trabajo PLM_4BS (*«Process Lifecycle Management for Business-Software»*) que sentará los fundamentos teóricos necesarios para satisfacer los objetivos planteados.

Como se ha comentado a lo largo de los capítulos anteriores, el marco de trabajo PLM_4BS pretende dar soporte a la gestión del ciclo de vida del proceso software para mejorar la calidad de la gestión y disminuir el coste que supone el mantenimiento de estos procesos. Además, con el propósito de facilitar su aplicación en proyectos y entornos reales, PLM_4BS basa sus pilares teóricos sobre el paradigma guiado por modelos.

Con todo esto, se han cerrado los marcos de introducción de la tesis y se han sentado las bases adecuadas para comenzar a presentar, en los siguientes capítulos, la aproximación de nuestra propuesta.

Capítulo IV DEFINICIÓN DE METAMODELOS

Como se ha comentado en capítulos anteriores, a la hora de plantear una adecuada gestión del ciclo de vida del proceso software (y de forma específica sus fases de definición, ejecución y orquestación), resulta necesario definir cuáles son los metamodelos que influyen en dicha gestión en las etapas del ciclo de vida BPM. Éste es precisamente el objetivo de este capítulo.

A lo largo del capítulo anterior, se ha descrito el alcance de la tesis y la estructura de la propuesta de solución, la cual además, se sustenta sobre la definición de un ciclo de vida BPM. Este capítulo describe los lenguajes y conceptos necesarios que permitan la definición de procesos software y el modelado posterior de las propiedades asociadas con su contexto de ejecución y orquestación. Los metamodelos planteados son definidos formalmente y representados formalmente por medio de diagramas de clases UML.

La definición formal de los lenguajes indicados en el párrafo anterior es tomada como base en capítulos posteriores para plantear las reglas de derivación con las que generar el modelo de ejecución y orquestación a partir del modelo de definición del proceso.

Finalmente, este capítulo sintetiza una serie de conclusiones.

1. Introducción

Durante el capítulo anterior, se han sentado las bases teóricas de un ciclo de vida de mejora continua BPM (Figura III.4) y se han definido las fases que podría contemplar dicho ciclo de vida para tratar de forma adecuada la gestión de procesos software. Estas fases son, como se presentaron en la Sección 4.1 del Capítulo III, las siguientes:

- Modelado
- Ejecución y Orquestación
- Monitorización
- Mejora Continúa

Una vez definido este ciclo de vida BPM, resulta necesario definir cómo se va a brindar soporte a cada una de sus fases. Tal y como se concreta en la Sección 2 del Capítulo III, esta Tesis Doctoral se centra en el soporte a las dos primeras fases del ciclo de vida BPM dentro del marco y contexto de la propuesta PLM$_4$BS (*«Process Lifecycle Management for Business-Software»*). El soporte al resto de las fases del ciclo de vida BPM será definido por medio de la consecución de una serie de trabajos futuros, concretados todos ellos en el Capítulo VII.

Este capítulo, por tanto, inicia la descripción formal de la propuesta de trabajo PLM_4BS; descripción que será completada con los resultados desarrollados en los siguientes capítulos. Como se describe, estos resultados están amparados por el paradigma MDE.

En lo que se refiere a este capítulo, se definen formalmente los metamodelos que dan soporte a las fases de modelado y, ejecución y orquestación de procesos software. Estos metamodelos se representan con la notación del diagrama de clases de UML.

Durante la definición de estos metamodelos se ha intentado primar su simplicidad, pero sin perder el horizonte de una aplicabilidad efectiva. Esta decisión ha sido tomada por dos razones:

1. La definición de lenguajes simples posibilita su aplicación y validación dentro de diferentes contextos de negocio. El propósito es mejorar estos lenguajes después de su aplicación y validación en estos entornos. En este sentido, el grupo de investigación Ingeniería Web y Testing Temprano (IWT2) de la Universidad de Sevilla – en cuyo seno se gesta y desarrolla esta tesis – tiene una dilatada experiencia (actual y pasada) en proyectos I+D y transferencia tecnológica. Estas oportunidades y capacidades posibilitan la aplicación y validación de los resultados de esta tesis con el objetivo de obtener información valiosa para su mejora y evolución.
2. La definición de lenguajes simples facilita también la aplicación de protocolos de derivación MDE y además, reduce la sobrecarga cognitiva del usuario cuando utiliza dichos lenguajes.
    a. Desde la perspectiva del paradigma MDE, esta simplicidad posibilita una aplicación satisfactoria en entornos empresariales. La experiencia del grupo IWT2 en proyectos tecnológicos demuestra que la aplicación de soluciones simples hace más probable su éxito dentro del contexto empresarial, porque resulta más sencillo y eficaz diseñar e implementar reglas de derivación entre modelos.
    b. Desde la perspectiva de la usabilidad, existen estudios [Moody 2010] [Goodman 1968] que han consolidado teorías en relación con la sobrecarga cognitiva de varios lenguajes, como por ejemplo, i* [Moody *et al.* 2009a], BPMN [Genon 2011], WebML [Granada *et al.* 2013] o UML [Moody *et al.* 2009b], entre otros. Una de las conclusiones de estos estudios es que la complejidad de los lenguajes de modelado (es decir, con muchos elementos, muchas posibilidades de interconexión entre dichos elementos, una extensa simbología, etc.) incrementa la sobrecarga cognitiva del usuario y hace más difícil que éste pueda entender con facilidad los modelos definidos en base a dichos lenguajes. La usabilidad tiene, por tanto, una mayor influencia sobre la eficiencia y eficacia en el uso de los lenguajes de modelado.

Finalmente, decir que para la definición formal de los metamodelos presentados en las subsecciones de este capítulo, se han tenido en cuenta diferentes marcos de referencia

para la definición de procesos. Concretamente, los estándares ISO/IEC TR 24774:2007 [ISO/IEC TR 2007a] e ISO/IEC 12207:2008 [ISO/IEC 2008b] en los que se establece que la definición de cualquier proceso de negocio debe cubrir determinados aspectos tales como aquellas entradas y salidas que recibe el proceso, métricas e indicadores que permiten identificar la correcta ejecución del proceso, descripción de los procesos mediante mapas de procesos, entre otros. Estas normas han sido publicadas como una guía para la descripción de procesos tal y como concluye el estudio que realizamos y fue publicado en [García-Borgoñón *et al.* 2013].

Siguiendo las pautas expuestas anteriormente, la Sección 2 y Sección 3 de este capítulo formaliza, respectivamente, la definición del metamodelo de definición de procesos y la definición del metamodelo de ejecución y orquestación de procesos.

2. Metamodelo para la definición de procesos software

Esta sección describe cómo se ha resuelto en este trabajo de doctorado el segundo objetivo enumerado en la Sección 2 del Capítulo III Capítulo III, es decir, la definición de un metamodelo para el modelado de procesos software.

Para este propósito, esta sección se divide en dos apartados. El primero de ellos describe el contexto y planteamiento previo, mientras que en el segundo apartado se define amplia, formal y rigurosamente la solución propuesta.

2.1. Contexto y planteamiento previo

El lenguaje de definición de procesos software presentado en esta sección, viene descrita en forma de metamodelo MOF y presentado mediante la notación de diagramas de clases de UML. Además, incluye los atributos necesarios para cumplir con el estándar ISO/IEC TR 24744.

Durante el diseño y especificación del metamodelo para la definición de procesos software, se han tenido en cuenta los siguientes aspectos esenciales:

1) **Semántica rica y simple**. La solución planteada debe ser lo suficientemente rica desde el punto de vista semántico como para permitir la definición de procesos con un relativo grado de complejidad y a la vez, debe ser lo suficientemente sencilla como para facilitar su aplicación en entornos empresariales reales de una forma satisfactoria.
2) **Comprensibilidad y legibilidad**. El número de usuarios que pueden aplicar la propuesta desarrollada puede ser potencialmente considerable y es posible que no todos ellos posean las competencias y conocimientos propios de un perfil de ingeniero software. En este sentido, para mejorar la usabilidad de la propuesta se ha establecido un formato de representación comprensible y legible basado en metáforas visuales.
3) **Granularidad**. El lenguaje de definición de procesos software debe posibilitar la definición de modelos de procesos que pueden incluir otros procesos. Con esta

característica, los usuarios pueden describir sus procesos con el nivel de detalle necesario.

4) **Medibilidad**. En cualquier organización, es importante mejorar continuamente sus procesos de negocio para lograr una gestión eficaz de los mismos. En este sentido, la definición de indicadores clave de rendimiento – conocidos por su acrónimo inglés KPI o «*Key Performance Indicators*» – es uno de los mecanismos que se puede utilizar para analizar el desempeño de los procesos y actuar en consecuencia para mejorarlos. Este análisis se acompaña frecuentemente con el uso de herramientas externas, tales como herramientas de minería de datos y cuadros de mando. La propuesta de definición de procesos software presentada en esta tesis debe incluir mecanismos alternativos para establecer estos indicadores.

5) **Orquestabilidad**. Durante la etapa de modelado de procesos, resulta necesario definir en qué momento del proceso se requiere información desde otros componentes o sistemas de información ajenos al proceso. Es por tanto necesario incluir elementos en el metamodelo con el propósito de ejecutar y coordinar eventos automáticos necesarios para lograr esta orquestación durante la etapa siguiente del ciclo de vida BPM, es decir, la fase de ejecución.

2.2. Definición del metamodelo

Los conceptos definidos en este metamodelo están agrupados en el paquete *«Software Processes Modeling Structural»*. La Figura IV.1 muestra este metamodelo y las relaciones entre sus elementos.

Antes de continuar, es importante aclarar que la sintaxis utilizada para representar el metamodelo de definición de procesos software no es lo suficientemente rica como para reflejar semánticamente las restricciones necesarias. En consecuencia, se han definido restricciones semánticas (tal y como recomienda la OMG, «*Object Modeling Group*») mediante el lenguaje de restricciones OCL [ISO/IEC 2012]. Estas restricciones limitan las posibilidades de instanciación con el propósito de construir modelos de procesos válidos.

En la siguiente sección, se describen en detalle los elementos del metamodelo, siguiendo el estilo y la plantilla de organización de documentación utilizado por el OMG para la documentación de sus estándares y normas (como es el caso del lenguaje UML).

Este estilo se describe a continuación. En primer lugar se ofrece una breve descripción del elemento. A continuación, se indican sus generalizaciones, si existen, y las restricciones asociadas a dichas generalizaciones. A continuación se describen sus asociaciones y sus atributos. Posteriormente, se describen las restricciones a los valores de los atributos y asociaciones si las hubiera.

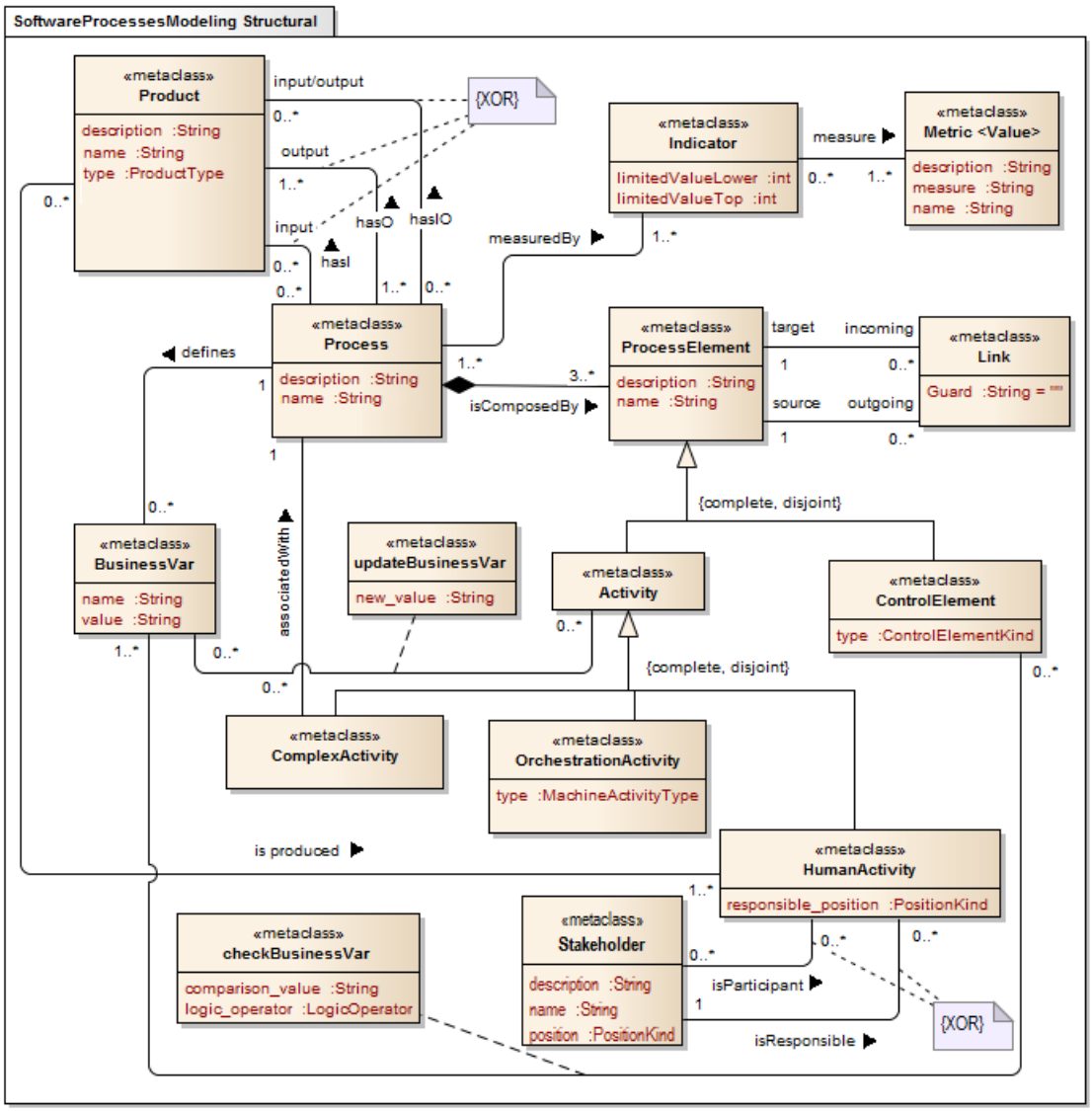


Figura IV.1. Metamodelo de definición de procesos software

2.2.1. Metaclase «Process»

Descripción: La metaclase *«Process»* es el epicentro del metamodelo y es el elemento alrededor del cual orbitan el resto de elementos del metamodelo. Con esta metaclase es posible representar cualquier proceso software (por ejemplo, procesos de aseguramiento de la calidad, procesos de desarrollo de software, procesos de seguridad o procesos de gestión de proyectos software, entre otros).

Generalización: Ninguna.

Atributos:

- *name: String [1]*
  Descripción corta y concisa con la que los usuarios podrán identificar de manera unívoca al proceso.
- *description: String [1]*

Descripción detallada sobre cuáles son los objetivos que debe cumplir el proceso, los productos que tiene que generar o qué entradas necesita, etc.

Operaciones: Ninguna.

Asociaciones:

- *defines: BusinessVar [0..*]*
  Conjunto de variables del proceso con las que controlar su flujo de ejecución e información. El concepto de variables del proceso ha sido modelado mediante la metaclase *«BusinessVar»*. La Sección 2.2.2 describe en detalle esta metaclase.
- *isComposedBy: ProcessElements [3..*]*
  Conjunto ordenado de, al menos, tres elementos estructurales – metaclase *«ProcessElements»*; descrita en la Sección 2.2.3 – del proceso secuenciados en un orden particular para producir un determinado resultado de negocio.

  Respecto al conjunto mínimo de elementos del proceso, éste debe estar compuesto por dos instancias de la metaclase *«ControlElement»* (consultar Sección 2.2.4) que actúen como punto de entrada y salida del proceso, respectivamente, y un tercer elemento que sea una instancia de la metaclase *«Activity»* (consultar Sección 2.2.5).
- *measuredBy: Indicator [1..*]*
  Conjunto de indicadores con los que será posible definir qué aspectos del proceso es necesario medir con el objetivo de llevar a cabo un ciclo de mejora continua del mismo. El concepto de indicador ha sido modelado mediante la metaclase *«Indicator»*. La Sección 2.2.15 describe en detalle esta metaclase.
- *associatedWith: ComplexActivity [0..*]*
  Conjunto de actividades, denominadas complejas, con las que un proceso está enlazado. Mediante esta relación ha sido posible cumplir con el objetivo de granularidad expuesto en la Sección 2.1. La Sección 2.2.8 describe en detalle la metaclase *«ComplexActivity»*.
- *hasIO: Product [0..*]*
  Conjunto de productos de entrada/salida del proceso. Como se describe en detalle en la Sección 2.2.13, un producto – metaclase *«Product»* – se entiende como aquella información generada o consumida durante el proceso.
- *hasI: Product [0..*]*
  Conjunto de productos que el proceso necesita como información de entrada para poder desempeñar sus tareas.
- *hasO: Product [1..*]*
  Conjunto de productos que el proceso va a generar durante su desempeño. Es importante tener en cuenta que un proceso debe generar al menos un producto con el propósito de disponer de al menos una evidencia, ante cualquier evaluación o auditoría, de que el proceso ha sido realizado.

Restricciones:

Respecto a la asociación *«associatedWith»*, es necesario definir una restricción en el modelo para controlar que un proceso no se contenga a sí mismo con el propósito de evitar que un usuario pueda definir de manera indeterminada y recursiva un proceso. La Expresión IV-1 describe esta restricción en OCL.

Expresión IV-1. Restricción OCL sobre la metaclase *«Process»* (I)

```
context Process inv:
  let complexActColl : SortedSet = self.ProcessElement→select(oclIsTypeOf(ComplexActivity)) in
  self.ComplexActivity->forAll (act1 : ComplexActivity |
                                    not complexActColl→exist(act2 : ComplexActivity | act1 = act2)))
```

Por otra parte, es necesario controlar que cada uno de los productos de salida del proceso debe haber sido generado por alguna de las actividades humanas (metaclase *«HumanActivity»*) del proceso. La Expresión IV-2 describe esta restricción en OCL.

Expresión IV-2. Restricción OCL sobre la metaclase *«Process»* (II)

```
context Process inv:  let totalProducts  : Integer   = self.output→size();
                      let activitiesColl : SortedSet = self.ProcessElement→select(oclIsTypeOf(HumanActivity)) in

  -- cada producto debe haber sido generado por una de las HumanActivity que forman parte del proceso

  self.output→forAll (p1 : Product |
          activitiesColl→exist(a : HumanActivity | a.Product→exist (p2 : Product | p1 = p2)))
  and

  -- el número de productos de salida del proceso debe coincidir con el número total de productos
  -- generados por todas las HumanActivity que son partes del proceso

  totalProducts = activitiesColl→iterate(a: HumanActivity; acc : Integer = 0 | acc + a.Product→size())
```

Como se ha adelantado durante la especificación de los atributos de la metaclase *«Process»*, ésta debe estar relacionada con al menos tres elementos: dos instancias de la metaclase *«ControlElement»* que actúen como punto entrada y salida del proceso, respectivamente, y un tercer elemento que sea una instancia de la metaclase *«Activity»*. La Expresión IV-3 describe esta restricción en OCL.

Expresión IV-3. Restricción OCL sobre la metaclase *«Process»* (III)

```
context Process inv: self.ProcessElement→size() >= 3 and
                     self.ProcessElement→select(oclIsTypeOf(Activity))→size() >= 1 and
                     self.ProcessElement→exist(c : ControlElement | c.type = "InitialElement") and
                     self.ProcessElement→exist(c : ControlElement | c.type = "FinalElement")
```

Por último, para construir modelos bien formados es necesario controlar que el proceso sólo contenga un punto de entrada y un punto de salida. La Expresión IV-4 describe esta restricción en OCL.

Expresión IV-4. Restricción OCL sobre la metaclase *«Process»* (IV)

```
context Process inv:  self.ProcessElement→count (c : ControlElement |
                                                  c.type = "InitialElement" or c.type = "FinalElement") = 2
```

2.2.2. Metaclase «BusinessVar»

Descripción: El concepto de variable de negocio resulta fundamental para establecer cuál es la siguiente actividad a ejecutar durante la instanciación del proceso porque: (*i*) permite establecer caminos alternativos en el flujo de datos y ejecución del proceso; y (*ii*) son los bloques de construcción mínimos para definir reglas de negocio en el proceso. Este es el propósito de la metaclase *«BusinessVar»*.

Generalización: Ninguna.

Atributos:

- *name: String [1]*
  Descripción corta y concisa con la que identificar de manera unívoca a la variable.
- *value: String [1]*
  Valor inicial de la variable.

Operaciones: Ninguna.

Asociaciones:

- *defines: Process [1]*
  Especifica cuál es el proceso en el que la variable está siendo definida.
- *checkBusinessVar: Conditional [0..*] (Metaclase asociación)*
  Conjunto de elementos *«ControlElement»* de tipo *«Conditional»* en los que se trabaja con la variable. El tipo de elemento *«ControlElement»* es controlado mediante una restricción OCL (definida a continuación).
- *updateBusinessVar: Activity [0..*] (Metaclase asociación)*
  Conjunto de elementos *«Activity»* con cuya consecución se actualiza en valor de la variable por un nuevo valor (atributo de la metaclase *«updateBusinessVar»*; consultar Sección 2.2.10).

Restricciones:
Respecto a la metaclase asociación *«checkBusinessVar»*, es necesario definir una restricción en el metamodelo para controlar que una variable sólo pueda ser consultada por elementos *«ControlElement»* de tipo *«Conditional»*. La Expresión IV-5 describe esta restricción en OCL.

Expresión IV-5. Restricción OCL sobre la metaclase *«BusinessVar»*

```
context BusinessVar inv: self.ControlElement→forAll (c : ControlElement | c.type = "Conditional")
```

2.2.3. Metaclase «ProcessElements»

Descripción: La estructura del proceso se construye a través de un conjunto ordenado de elementos estructurales, los cuales son representados mediante la metaclase *«ProcessElements»*.

Generalización: Ninguna.

Atributos:

- *name: String [1]*
  Descripción corta y concisa con la que el usuario puede identificar de manera clara y unívoca al elemento.
- *description: String [1]*
  Descripción detallada sobre cuál es el cometido del elemento.

Operaciones: Ninguna.

Asociaciones:

- *incoming : Link [0..*]*
  Conjunto de enlaces de entrada al elemento *«ProcessElements»*. Estos enlaces se representan en el metamodelo mediante la metaclase *«Link»* (Sección 2.2.9). Utilizando estos enlaces se construye la estructura del proceso, pero hay que garantizar que se satisfacen cuenta ciertas restricciones para garantizar la definición de un modelo de proceso bien formado. Estas restricciones son definidas en la Sección 2.2.9 utilizando expresiones OCL.
- *outgoing : Link [0..*]*
  Conjunto de enlaces de salida al elemento *«ProcessElements»*. Estos enlaces se representan en el metamodelo mediante la metaclase *«Link»* y al igual que en el caso anterior, este tipo de enlaces permiten construir la estructura del proceso.
- *isComposedBy: Process [1] (Asociación de composición)*
  Establece cuál es el proceso en el que se está definiendo este elemento.

Restricciones: Ninguna.

2.2.4. Metaclase «ControlElement»

Descripción: Representa los elementos de control del proceso con los que es posible definir su estructura y los diferentes caminos desde el punto de entrada del proceso hasta el punto de salida.

Generalización: Metaclase *«ProcessElements»*.

Atributos:

- *type : ControlElementKind [1]*
  Establece cuál es el tipo del elemento de control. Los posibles valores han sido modelados por medio del enumerado *«ControlElementKind»*, cuya lista de valores es la siguiente: *«InitialElement»*, que representa el punto de entrada al proceso y a partir del cual se secuencia el resto de elementos estructurales; *«FinalElement»*, que representa el punto de salida del proceso; *«Conditional»*, con el que es

posible crear ramas excluyentes en el flujo de control del proceso; y *«Fork»* y *«Join»*, que permiten comenzar y finalizar, respectivamente, ramas paralelas en el flujo de control del proceso.

Con la lista de valores que proporciona el enumerado *«ControlElementKind»*, un ingeniero de procesos es capaz de modelar procesos con una complejidad considerable. Con el propósito de no limitar la futura extensión del metamodelo de definición de procesos software, este enumerado puede ser ampliado en cualquier momento.

Operaciones: Ninguna.

Asociaciones:

— *checkBusinessVar: BusinessVar [1..*] (Metaclase asociación)*
Conjunto de, al menos, una variable de tipo *«BusinessVar»* para las que se consulta el valor actual. Esta asociación sólo está permitida en caso de que el elemento *«ControlElement»* sea de tipo *«Conditional».* Esta restricción se controla mediante la expresión OCL descrita en la Expresión IV-5.

Restricciones:

Respecto a las posibles relaciones que se puedan establecer entre la metaclase *«ControlElement»* y otros elementos del proceso, es necesario llevar a cabo un control con el propósito de construir modelos consistentes y bien definidos. En este sentido, se establece la restricción OCL descrita en la Expresión IV-6, con la que se controla, por ejemplo y entre otros, que un elemento *«ControlElement»* de tipo *«InitialElement»* no puede tener ningún enlace de entrada y sólo puede tener un enlace de salida, o que un elemento *«ControlElement»* de tipo *«Conditional»* debe tener al menos un enlace de entrada y al menos dos de salida.

Expresión IV-6. Restricción OCL sobre la metaclase «*ControlElement»*

```
context ControlElement inv :
if (self.type = "Conditional") then self.oclAsType(ProcessElement).incoming→size() >= 1  and
                                    self.oclAsType(ProcessElement).outgoing→size() >= 2
else if (self.type = "Fork")   then self.oclAsType(ProcessElement).incoming→size() >= 1  and
                                    self.oclAsType(ProcessElement).outgoing→size() >= 1
else if (self.type = "Join")   then self.oclAsType(ProcessElement).incoming→size() >= 1  and
                                    self.oclAsType(ProcessElement).outgoing→size() = 1
else if (self.type = "InitialElement") then self.oclAsType(ProcessElement).incoming→size() = 0  and
                                            self.oclAsType(ProcessElement).outgoing→size() = 1
else
    -- if (self.type = "FinalElement") then
    self.oclAsType(ProcessElement).incoming→size() > 0  and
    self.oclAsType(ProcessElement).outgoing→size() = 0
endif endif endif endif
```

2.2.5. Metaclase «Activity»

Descripción: Esta metaclase representa cualquier tipo de acción que debe ser ejecutada para alcanzar los objetivos del proceso.

Generalización: Metaclase *«ProcessElements»*.

Atributos: Ninguna.

Operaciones: Ninguna.

Asociaciones:

— *updateBusinessVar: BusinessVar [0..*] (Metaclase asociación)*
Conjunto de variables del proceso que pueden ser actualizadas por la actividad que se está definiendo.

Restricciones:
Respecto a las posibles relaciones que se puedan establecer entre la metaclase *«Activity»* (y por extensión sus subtipos) con otros elementos del proceso, es necesario llevar a cabo un control con el propósito de asegurar la construcción de modelos consistentes y bien definidos.

En este sentido, se establece la restricción OCL descrita en la Expresión IV-7 en la que se establece que un elemento de tipo *«Activity»* debe tener exclusivamente un enlace de salida y al menos uno de entrada.

Expresión IV-7. Restricción OCL sobre la metaclase «*Activity*»

```
context Activity inv : self.oclAsType(ProcessElement).incoming→size() >= 1 and
                       self.oclAsType(ProcessElement).outgoing→size() = 1
```

2.2.6. Metaclase «HumanActivity»

Descripción: Esta metaclase representa una actividad que debe ser desempeñada por un agente humano.

Generalización: Metaclase *«Activity»*.

Atributos:

— *responsible_position: PositionKind [1]*
Esta propiedad representa el perfil de la persona responsable de llevar a cabo y completar la actividad. Los posibles valores están controlados por el enumerado *«PositionKind»* y cuya lista de valores es la siguiente: *«Project Manager»*, que representa el jefe de proyecto software encargado de organizar y dirigir de trabajo; *«Developer»*, que representa al equipo de desarrollo; *«Analyst»*, que representa al equipo de analistas; y *«Manager»*, que representa al responsable del proyecto.

Con el propósito de no limitar la futura extensión del metamodelo de definición de procesos software, este enumerado puede ser ampliado en cualquier momento.

Operaciones: Ninguna.

Asociaciones:

- *produces: Product [0..*]*
  Conjunto de elementos de productos que genera la actividad humana que se está definiendo.
- *isResponsible: Stakeholder [1..*]*
  Conjunto de, al menos, un interesado que actúa como responsable de la actividad. La Sección 2.2.12 que contiene una descripción completa de la metaclase *«Stakeholder»*.
- *isParticipant: Stakeholder [0..*]*
  Conjunto de interesados que actúan como participantes de la actividad y por tanto, no tienen capacidad para completarla. La Sección 2.2.12 que contiene una descripción completa de la metaclase *«Stakeholder»*.

Restricciones: Ninguna.

2.2.7. Metaclase «OrchestrationActivity»

Descripción: Esta metaclase representa una actividad de orquestación, es decir, una actividad que debe ser desempeñada por un sistema software.

Generalización: Metaclase *«Activity»*.

Atributos:

- *type: MachineActivityType [1]*
  Establece cuál es el tipo de actividad de orquestación. Los posibles valores están controlados por el enumerado *«MachineActivityType»* y cuya lista de valores es la siguiente: *«execute script»*, para especificar la ejecución de scripts; *«invoke service»*, para especificar llamadas a servicios externos; y *«send message»* para indicar el envío de un mensaje. Con el propósito de no limitar la futura extensión del metamodelo de definición de procesos software, este enumerado puede ser ampliado en cualquier momento.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

2.2.8. Metaclase «ComplexActivity»

Descripción: Esta metaclase permite incluir un proceso dentro de otro. De esta forma, cumplimos el objetivo de granularidad expuesto en la Sección 2.1 de este capítulo.

Generalización: Metaclase *«Activity»*.

Atributos: Ninguna.

Operaciones: Ninguna.

Asociaciones:

- *associatedWith: Process [1]*
  Proceso con el que está vinculada la metaclase *«ComplexActivity».*

Restricciones: Ninguna.

2.2.9. Metaclase «Link»

Descripción: Esta metaclase posibilita la definición de la estructura interna del proceso ya que permite establecer relaciones entre los elementos estructurales del mismo.

Generalización: Ninguna.

Atributos:

- *Guard: String [1]*
  Representa el valor que debe darse en el flujo del proceso. Por defecto, este atributo no tendrá valor. Sin embargo, en el caso de que el elemento origen de la asociación sea una instancia de la metaclase *«ControlElement»* de tipo *«Conditional»*, este atributo debe tener un valor.

Operaciones: Ninguna.

Asociaciones:

- *source : ProcessElements [1]*
  Representa el elemento *«ProcessElements»* origen de la asociación.
- *target : ProcessElements [1]*
  Representa el elemento *«ProcessElements»* destino de la asociación.

Restricciones:
Como se ha adelantado, el atributo *«Guard»* debe tener un valor asociado en el caso de que el elemento origen del enlace sea un elemento *«ControlElement»* de tipo *«Conditional»*. En cualquier otro caso, el atributo *«Guard»* no debe tener valor. Además, es necesario asegurar la unicidad del atributo *«Guard»* entre todas las instancias de la metaclase *«Link»* que sean enlaces de salida de un elemento *«ControlElement»* de tipo *«Conditional»*. La Expresión IV-8 describe esta restricción en OCL.

Expresión IV-8. Restricción OCL sobre la metaclase *«Link»* (I)

```
context Link inv :
  if (self.source.oclIsTypeOf(ControlElement) and self.source.type = "Conditional") then
       not self.Guard = "" and
       self.source.outgoing→forAll (link : Link | not link.Guard = self.Guard)
  else
       self.Guard = ""
endif
```

Por otra parte, como se adelanta en la Sección 2.2.3, para construir un modelo bien formado para la definición del proceso, la metaclase *«ProcessElements»* se debe asociar con la metaclase *«Link»*.

Sin embargo, durante la construcción del modelo hay que controlar que el número de instancias de la metaclase *«Link»* sea el correcto en función del subtipo de la metaclase *«ProcessElements»*. Este control se realiza mediante la restricción OCL descrita en la Expresión IV-9.

Expresión IV-9. Restricción OCL sobre la metaclase *«Link»* (II)

```
context Link inv:
if self.source.oclIsTypeOf(ControlElement) then
        if self.source.oclAsType(ControlElement).type="InitialElement" then self.source.outgoing→size() = 0
        else if self.source.oclAsType(ControlElement).type = "FinalElement" then false
        else if self.source.oclAsType(ControlElement).type = "Conditional" then true
        else if self.source.oclAsType(ControlElement).type = "Join" then self.source.outgoing→size() = 0
        else -- if self.source.oclAsType(ControlElement).type = "Fork" then
            true
       endif endif endif endif
else if.target.oclIsTypeOf(ControlElement) then
        if self.target.oclAsType(ControlElement).type="InitialElement" then false
        else if self.target.oclAsType(ControlElement).type = "FinalElement" then true
        else if self.target.oclAsType(ControlElement).type = "Conditional" then true
        else if self.target.oclAsType(ControlElement).type = "Join" then true
        else -- if self.target.oclAsType(ControlElement).type = "Fork" then
           true
       endif endif endif endif
else if self.source.oclIsTypeOf(Activity) then self.source.outgoing→size() = 0
     else if self.target.oclAsType(Activity) then true
     else false
endif endif
```

2.2.10. Metaclase «updateBusinessVar»

Descripción: Es posible que en determinadas ocasiones, surja la necesidad de actualizar el valor de una (o varias) variable del proceso cuando una actividad ha sido completada. Este requisito se materializa a través de la metaclase asociación *«updateBusinessVar»* establecida entre las metaclases «*BusinessVar*» y «*Activity*».

Generalización: Ninguna.

Atributos:

— *new_value: String [1]*

Esta propiedad representa el nuevo valor que debe tomar la variable involucrada en la asociación. Cuando el proceso de negocio haya sido instanciado y se esté ejecutando en un motor de procesos, el valor de cada variable de negocio se irá actualizando conforme se ejecuten aquellas actividades relacionadas con la variable mediante un enlace de tipo *«updateBusinessVar»*.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

2.2.11. Metaclase «checkBusinessVar»

Descripción: El uso de variables proporciona caminos indirectos en el flujo de datos desde el punto donde un valor es asignado a una variable hasta todos aquellos puntos en los que dicho valor es consultado. La comprobación o chequeo de los valores de [1..*] variable/s se realiza en los nodos *«ControlElement»* de tipo *«Conditional»* del proceso. Este requisito se materializa a través de la metaclase asociación *«checkBusinessVar»* establecida entre las metaclases «*BusinessVar*» y «*ControlElement*».

Cuando el modelo del proceso software esté en ejecución y un nodo «*Conditional*» sea alcanzado, el resultado de la siguiente expresión lógica determinará qué rama tomará el proceso y por tanto, el siguiente nodo del proceso a ejecutar. Como para ello, se comparará el resultado de la expresión como una de las guardas de los enlaces de salida del nodo *«ControlElement».*

$$Result\ (C) = \wedge_{i=1}^{n}\{C_i(initValueVar)\ C_i(logicOperator)\ C_i(comparisonValue)\}$$

Donde:

- C es el conjunto de variables que comprueba el nodo *«ControlElement»* de tipo *«Conditional»* y C_i es la i-esima variable chequeada.
- C_i *(initValueVar)* es el valor de la variable C_i.
- C_i *(logicOperator)* es el operador lógico seleccionado para evaluar la variable C_i. Los operadores vienen establecidos en el enumerado *«LogicOperator».*
- C_i *(comparisonValue)* es el valor con el cual es comparado el valor inicial de la variable C_i.

La Figura IV.2 ilustra un ejemplo de la expresión lógica anterior a través de la instanciación de las metaclases involucradas. En esta figura se modela una instancia de *«ControlElement»* de tipo *«Conditional»* y cómo ésta consulta dos instancias de la metaclase *«BusinessVar»* y cómo se enlaza con dos instancias de la metaclase *«ProcessElement»* por medio de dos instancias de la metaclase *«Link»*. En este ejemplo, el conjunto de variables consultadas, *«C»*, es *«{BusinessVar₁, BusinessVar₂}»* y la expresión lógica que será evaluada durante la fase de ejecución para decidir cuál es el camino a tomar dentro del flujo del proceso es la siguiente:

$$Result\ (C) = valueVar_1 EQUAL\ compValue_{\ 1} \wedge valueVar_2\ NONEQUAL\ compValue_{\ 2}$$

El siguiente nodo a ejecutar – es decir, *«ProcessElement1»* o *«ProcessElement2»* – será aquél cuyo enlace con el nodo *«ControlElement»* contenga la misma guarda que el resultado de la expresión anterior. Por ejemplo, si la expresión anterior devuelve como resultado *«true»*, el siguiente nodo a ejecutar será *«ProcessElement2»*.

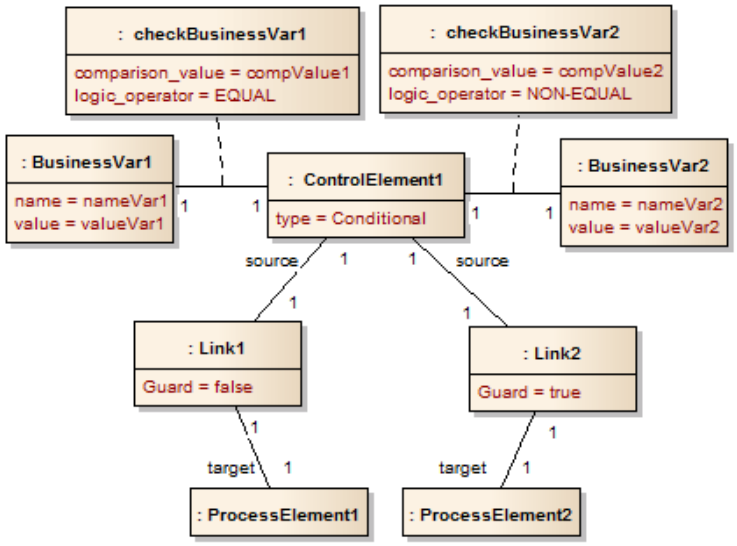


Figura IV.2. Evaluación de variables en un nodo condicional

Generalización: Ninguna.

Atributos:

- *comparison_value: String [1]*
  Esta propiedad representa el valor contra el que se va a comparar el valor que tenga la variable.
- *logic_operator: LogicOperator [1]*
  Representa el operador lógico a aplicar durante la comparación. Los posibles operadores lógicos contemplados actualmente son *«equal»* y *«non-equal».* Ambos, modelados a través del enumerado *«LogicOperator».* Este enumerado puede ser ampliado en cualquier momento con el propósito de soportar una posible extensión del metamodelo de definición de proceso software.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

2.2.12. Metaclase «Stakeholder»

En el ámbito de las organizaciones software, muy a menudo se diferencian dos tipos de interesados en función de su grado de involucración en la consecución de una determinada actividad *«HumanActivity»*, ya que es común que existan muchos equipos de trabajo con multitud de miembros, y cada uno de ellos coopera con el resto para desarrollar actividades.

La propuesta de modelado de procesos software presentada en esta tesis diferencia si un interesado es participante o responsable de una actividad. La diferencia entre ambos radica en que el primero puede completar ciertos aspectos de los productos de salida de la actividad, pero no puede completar la misma sin la autorización del miembro responsable.

Descripción: La metaclase *«Stakeholder»* representa aquellos agentes humanos que podrían desempeñar actividades humanas del proceso.

Generalización: Ninguna.

Atributos:

- *name: String [1]*
  Descripción corta y concisa con la que es posible identificar al interesado.
- *description: String [1]*
  Descripción detallada sobre cuáles son las competencias del interesado.
- *position: PositionKind [1]*
  Esta propiedad representa el perfil de la persona dentro de la organización. Los posibles valores están controlados por el enumerado *«PositionKind»* y cuya lista de valores es la siguiente: *«Project Manager»*, que representa el jefe de proyecto software encargado de organizar y dirigir de trabajo; *«Developer»*, que representa al equipo de desarrollo; *«Analyst»*, que representa al equipo de analistas; y *«Manager»*, que representa al responsable del proyecto. Con el propósito de no limitar la futura extensión del metamodelo de definición de procesos software, este enumerado puede ser ampliado en cualquier momento.

Operaciones: Ninguna.

Asociaciones:

- *isResponsible: HumanActivity [0..*]*
  Conjunto de actividades humanas del proceso en las que el interesado actúa como responsable.
- *isParticipant: HumanActivity [0..*]*
  Conjunto de actividades humanas del proceso en las que el interesado actúa como un mero participante.

Restricciones:
Sobre la metaclase *«Stakeholder»* es conveniente establecer que un mismo interesado no puede ser responsable y participante de la misma actividad. Esta restricción se incluye en el metamodelo por medio del operador lógico *«XOR»* de UML entre las asociaciones *«isParticipant»* y *«isResponsible»* descritas anteriormente.

Además, es importante controlar un aspecto en cuanto a la relación de la metaclase *«Stakeholder»* interesado con la metaclase *«HumanActivity»*. En concreto, en caso de que un interesado interesado – *«Stakeholder»* – sea responsable de una actividad humana – *«HumanActivity»* – es necesario verificar que el puesto que ocupa dicho interesado en la organización sea igual con el puesto mínimo responsable para llevar a cabo dicha actividad. La Expresión IV-10 describe esta restricción en lenguaje OCL.

Expresión IV-10. Restricción OCL sobre la metaclase «*Stakeholder*»

```
context Stakeholder inv :  self.isResponsible->forAll (act: HumanActivity |
                                                act.responsible_position = self.position)
```

2.2.13. Metaclase «Product»

Como se ha comentado en el capítulo introductorio de esta tesis y concretado en la Tabla I.2, un proceso software debe definirse, entre otros aspectos, para desarrollar o mantener productos software. Resulta, por tanto, esencial contemplar el concepto de producto dentro de cualquier lenguaje de modelado de procesos orientado a cualquier ámbito de negocio, y en particular, orientado al ámbito del software. Siguiendo esta necesidad, a continuación se describe el concepto de producto dentro del metamodelo de la Figura IV.1.

Descripción: La metaclase *«Product»* representa cualquier tipo de producto obtenido como resultado de un proceso o como entrada al mismo.

Generalización: Ninguna.

Atributos:

- *name: String [1]*
  Descripción corta y concisa con la que es posible identificar al producto.
- *description: String [1]*
  Descripción detallada del producto.
- *typeProduct: ProductType [1]*
  Establece cuál es el tipo del producto. Los posibles valores están controlados por el enumerado *«ProductType»* y cuya lista de valores es la siguiente: *«code»*, *«multimedia»*, *«document»* y *«other»*. Este enumerado puede ser ampliado en cualquier momento con el propósito de soportar una posible extensión del metamodelo de definición de proceso software.

Operaciones: Ninguna.

Operaciones: Ninguna.

Asociaciones:

- *produces: HumanActivity [1..*]*
  Conjunto de, al menos, una actividad humana encargada de generar el producto que se está definiendo.
- *hasIO: Process [0..*]*
  Conjunto de procesos en los que el producto actúa como un producto de entrada/salida al proceso.
- *hasI: Process [0..*]*
  Conjunto de procesos en los que el producto actúa como un producto de entrada al proceso.
- *hasO: Process [1..*]*
  Conjunto de, al menos, un proceso en el que el producto actúa como un producto de salida del proceso.

Restricciones:
Sobre la metaclase *«Product»* resulta pertinente controlar que un mismo producto sólo puede estar vinculado con un proceso mediante alguna de las relaciones *«hasI»*, *«hasO»* y *«hasIO»*. Esta restricción está controlada en el metamodelo a través del operador lógico *«XOR»* de UML entre dichas asociaciones.

2.2.14. Metaclase «Metric»

Para cumplir con el objetivo de medibilidad estipulado en la sección de introducción de este capítulo, es esencial establecer mecanismos que permitan la medición del proceso con el objetivo de llevar a cabo un ciclo de mejora continua del mismo. Para este propósito, se han incluido las metaclases *«Metric»* e *«Indicator»* – ambas, descritas en esta sección y en la siguiente, respectivamente.

Descripción: La metaclase *«Metric»* define una medida cuantitativa para evaluar un determinado atributo en un proceso.

Generalización: Ninguna.

Atributos:

- *name: String [1]*
  Descripción corta y concisa con la que es posible identificar a la métrica definida.
- *description: String [1]*
  Descripción textual detallada de la métrica.
- *measure: String [1]*
  Establece aquella propiedad del proceso que debe ser medida.

Operaciones: Ninguna.

Asociaciones:

- *measure: Indicator [0..*]*
  Conjunto de indicadores que miden el estado de la métrica dentro de un rango de valores.

Restricciones: Ninguna.

2.2.15. Metaclase «Indicator»

Descripción: Esta metaclase se define como una especialización de la metaclase anterior y permite comparar una métrica con un valor objetivo dado por un rango posible de valores sobre los que la métrica es evaluada. Un indicador permite la medida objetiva de una métrica concreta y es la base para definir cuadros de mandos.

Generalización: Ninguna.

Atributos:

- *limitedValueTop: String [1]*
  Establece el límite superior del rango posible de valores sobre el que aplica el indicador que se están definiendo.

— *limitedValueLower: String [1]*
Establece el límite inferior del rango posible de valores sobre el que aplica el indicador que se están definiendo.

Operaciones: Ninguna.

Asociaciones:

— *measuredBy: Process [1..*]*
Conjunto de, al menos, un proceso sobre el que actúa el indicador definido.

— *measure: Metric [1..*]*
Conjunto de, al menos, una métrica que es medida por el indicador dentro de los rangos de valores que éste establece.

Restricciones: Ninguna.

3. Metamodelo de ejecución y orquestación de procesos software

Esta sección describe uno de los pilares fundamentales de la propuesta que presenta este trabajo de doctorado. De forma específica, este apartado se centra en definir una propuesta de solución para resolver el tercer objetivo enumerado en la Sección 2 del Capítulo III, es decir, la definición de un metamodelo de ejecución y orquestación de procesos software.

Para este propósito, esta sección se divide en dos apartados. El primero de ellos describe el contexto y planteamiento previo a la especificación de la propuesta, mientras que en el segundo apartado se define amplia, formal y rigurosamente la solución elaborada.

3.1. Contexto y planteamiento previo

El metamodelo de ejecución y orquestación de procesos software es una representación de la estructura y contexto de ejecución que permite especificar cómo son ejecutados los modelos de procesos. Este metamodelo debe ofrecer una especificación lo más completa y abstracta posible que permite la ejecución del comportamiento del modelo dentro de cualquier motor de ejecución de procesos así como proporcionar una descripción de la semántica de tal ejecución [OMG 2013a].

El contexto de ejecución puede ser considerado como un aspecto crítico durante la gestión del proceso de negocio. Por este motivo es necesario buscar mecanismos y técnicas que garanticen la calidad del mismo. El metamodelo de ejecución va a representar básicamente tres aspectos:

1. Qué elementos del modelado del proceso son representativos como para que aparezcan en el contexto de ejecución y orquestación del proceso.
2. Cuál es la definición de las propiedades estáticas y dinámicas requerida para establecer dicho contexto [OMG 2013a]. Ambos términos son explicados más adelante.

3. Cuáles son las relaciones que deben existir entre los distintos elementos representativos de la ejecución y orquestación del proceso.

Como se ha adelantado, es necesario definir los elementos del metamodelo de ejecución y orquestación desde dos perspectivas semánticas: estática y dinámica. Es importante discernir la semántica de ejecución de lo que se denomina semántica estática, un término que surge de las teorías de compiladores de lenguajes de programación.

El término de *propiedades estáticas* hace referencia a la especificación de la información estática – es decir, propiedades o atributos específicos – de cada elemento del modelo de ejecución y orquestación así como a la definición de restricciones y reglas semánticas para construir modelos bien formados. Todas estas reglas y restricciones serán formalizadas a lo largo de esta sección mediante el lenguaje de especificación de restricciones OCL.

Una vez que se ha construido el modelo del proceso y concretada toda la información estática para cada uno de sus elementos, un sistema computacional debe ser capaz de realizar operaciones sobre dichos elementos del modelo y su información, con el objetivo de instanciar cada elemento y permitir la interacción con el usuario y otros sistemas.

Para ello, es necesario dotar a los elementos del modelo con las propiedades dinámicas suficientes. El término *propiedades dinámicas*, también conocido como *propiedades de ejecución*, hace referencia a cuál es la información que se genera y gestiona de manera dinámica y, cómo y cuándo se pueden instanciar y orquestar cada elemento del modelo para producir un comportamiento de ejecución específico.

Con esta semántica, cada elemento también tiene la capacidad de reaccionar y evolucionar en tiempo de ejecución a lo largo de su ciclo de vida.

3.2. Definición del metamodelo

A la hora de definir el metamodelo de ejecución y orquestación de procesos software se han seguido las mismas directrices que en el metamodelo de definición de procesos software.

En primer lugar, el metamodelo de ejecución y orquestación ha sido definido en forma de metamodelo MOF y presentado mediante la notación de diagramas de clases de UML. Además, incluye los atributos necesarios para cumplir con el estándar ISO/IEC TR 24744.

En segundo lugar, para mantener cierta homogeneidad y organización respecto a la propuesta de definición de procesos, los conceptos definidos en el metamodelo de ejecución y orquestación han sido agrupados en un nuevo paquete, denominado «*Software Processes Executing Structural*».

La Figura IV.3 muestra los conceptos del metamodelo de ejecución y orquestación junto con las relaciones que existen entre ellos.

Antes de continuar, es conveniente aclarar un aspecto relevante de la representación del metamodelo de ejecución. Como se puede apreciar en la Figura IV.3, se han utilizado diferentes estereotipos para catalogar las propiedades y operaciones de cada uno de los elementos del metamodelo.

En primer lugar, respecto a las propiedades, se han utilizado los estereotipos *«Static»* y *«Dynamic»* para denotar cuáles son las aquellas propiedades estáticas y dinámicas del elemento ejecutable del proceso.

En segundo lugar, respecto a las operaciones, se han utilizado los siguientes estereotipos: *«Constr»*, abreviatura de *«Construct»*), que agrupa aquellas operaciones cuyo propósito es el de instanciar cada metaclase del metamodelo; y *«SOP»* (abreviatura de *«Status Operation»*), que cataloga todas aquellas operaciones que permiten actualizar o consultar el estado interno del elemento ejecutable.

Aclarado el aspecto anterior, en las siguientes secciones se describen con detalle los elementos que conforman el metamodelo de la imagen anterior. Se ha seguido en todo momento la estructura de OMG para la documentación de UML y especificaciones relacionadas, la cual es la misma estructura utilizada para especificar los elementos del metamodelo de definición de procesos software visto con anterioridad en la Sección 2.2 de este capítulo.

Respecto a la especificación de las operaciones contempladas en los metamodelos definidos en esta Tesis Doctoral, se ha optado por utilizar la misma notación que la empleada en la especificación del documento *«Semantics of a Foundational Subset for Executable UML Models (fUML)»* [OMG 2013a]. Este documento de OMG selecciona un subconjunto de elementos del metamodelo de UML que proporcionan los fundamentos necesarios para modelar conceptos de alto nivel y establece una definición concreta de la semántica de ejecución de los elementos de UML contenidos en dicho subconjunto. Todo ello con el propósito de sentar las bases teóricas para la posible definición de una máquina virtual con la que ejecutar modelos UML.

Ahondando en el objetivo del documento anterior, con la inclusión de operaciones en el metamodelo de ejecución y orquestación de esta tesis se persigue dar soporte operacional a uno de los trabajos futuros con mayor proyección futura de esta tesis; concretamente, la definición de un motor de ejecución de procesos para la propuesta desarrollada en este trabajo de tesis. Para proporcionar este soporte, la semántica de todas las operaciones está formalmente especificada con un lenguaje de orientado a objetos cuya sintaxis es similar a la de cualquier lenguaje de programación orientado a objetos de propósito general.

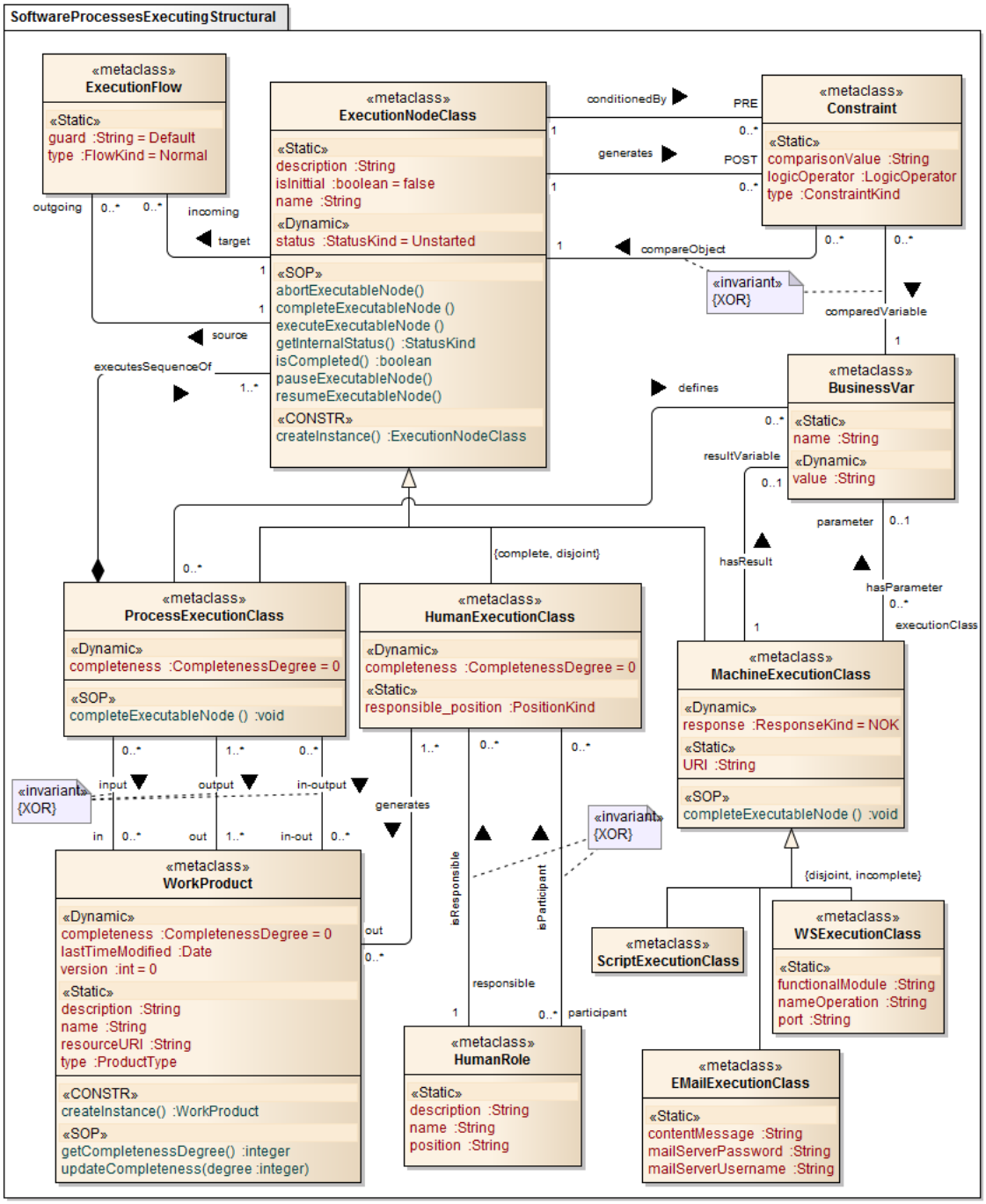


Figura IV.3. Metamodelo de ejecución y orquestación de procesos software

3.2.1. Metaclase «ExecutionNodeClass»

Descripción: La metaclase *«ExecutionNodeClass»* constituye el epicentro del metamodelo de ejecución y orquestación, y representa la versión ejecutable de cualquier elemento del proceso que requiere de la definición de un contexto de ejecución y orquestación.

La definición formal de una instancia de la metaclase *«ExecutionNodeClass»* consistirá en la definición de sus propiedades estáticas y dinámicas, así como el inicio de su ciclo de vida asociado. Este ciclo de vida se muestra en la Figura IV.4 y con él, es posible controlar

los estados internos por los que el elemento ejecución puede transitar. En el apartado de comentarios de esta misma sección se describe en detalle este ciclo de vida.

Generalización: Ninguna.

Atributos:

- *«Static» name: String [1]*
  Descripción corta y concisa con la que es posible identificar al proceso.
- *«Static» description: String [1]*
  Descripción detallada sobre cuáles son los objetivos que debe satisfacer el nodo de ejecución en cuestión.
- *«Static» isInittial: boolean [1]*
  Valor lógico que establece si el nodo de ejecución es el nodo inicial del proceso.
- *«Dynamic» status: StatusKind [1] = «Unstarted»*
  Refleja cuál es la situación o estado interno del elemento ejecutable durante su ciclo de vida a lo largo de la ejecución del proceso. Por defecto, toma el valor *«Unstarted»*.

  Los posibles estados por los que puede transitar un elemento ejecutable del proceso viene modelado por el enumerado *«StatusKind»* y se limitan a los siguientes valores: *«Unstarted»*, estado inicial del elemento ejecutable que representa el estado pre–instanciable; *«Running»*, con el que es posible identificar que un elemento ha sido instanciado y está en ejecución; *«Stand_by»*, con el que es posible situar cualquier elemento en un estado de espera hasta que ocurra un determinado evento; *«Completed»*, con el que es posible etiquetar cada uno de los elementos que han finalizado su ejecución; y *«Aborted»*, con el que es posible identificar aquellos elementos que han finalizado de forma inesperada.

Operaciones:

- *«Constr» createInstance (String name, String descp, boolean isInittial)*
  Esta operación define la semántica necesaria para poder instanciar un elemento de ejecución – metaclase *«ExecutionNodeClass»* – en un posible motor de ejecución de procesos (los cuales, han debido ser definidos mediante el metamodelo de ejecución de estación descrito en este capítulo).

  La Expresión IV-11 especifica esta operación con un lenguaje orientado a objetos.

Expresión IV-11. Especificación de la operación *«createInstance»*

```
createInstance (String name, String descp, boolean isInittial)  {
        this.name = name;
        this.description = descp;
        this.isInittial = isInittial;
        this.status = "Unstarted";
}
```

- *«SOP» getInternalStatus () : String*
  Esta operación retorna el estado interno en el que se encuentra el elemento ejecutable. Los posibles valores del estado (descritos en detalle en el apartado de

comentarios de esta sección) están restringidos a los definidos en el enumerado *«StatusKind»* y cuyos valores son: *«Unstarted»*, *«Running»*, *«Stand_by»*, *«Completed»*, y *«Aborted»*. La Expresión IV-12 especifica formalmente esta operación.

Expresión IV-12. Especificación de la operación *«getInternalStatus»*

```
getInternalStatus () : String  { return this.status ;  }
```

— *«SOP» executeExecutableNode () : void*

Esta operación permite iniciar la ejecución de un elemento y para ello, cambia el estado *«Unstarted»* a *«Running».* La Expresión IV-13 muestra el código orientado a objetos que formaliza esta operación.

Esta operación sigue fielmente la máquina de estado que formaliza el ciclo de vida (Figura IV.4) de una instancia de la metaclase *«ExecutionNodeClass».*

Expresión IV-13. Especificación de la operación *«executeExecutableNode»*

```
executeExecutableNode ()  : void  {
      if (this.status.equals ("Unstarted") ||  this.status.equals ("Stand_by"))
          this.status = "Running";
}
```

— *«SOP» abortExecutableNode () : void*

Esta operación permite cambiar el estado interno desde *«Running»* to *«Aborted»*. La Expresión IV-14 muestra el pseudocódigo orientado a objetos de esta operación, y al igual que la anterior, sigue fielmente el ciclo de vida mostrando en la Figura IV.4.

Expresión IV-14. Especificación de la operación *«abortExecutableNode»*

```
abortExecutableNode ()  : void  {  if (this.status.equals ("Running")) this.status = "Aborted";  }
```

— *«SOP» pauseExecutableNode () : void*

Esta operación permite cambiar el estado interno del elemento al estado *«Stand_by»* si y sólo si, el estado previo es *«Running»*. La especificación de esta operación se recoge en la Expresión IV-15.

Expresión IV-15. Especificación de la operación *«pauseExecutableNode»*

```
pauseExecutableNode ()  : void  {  if (this.status.equals ("Stand_by"))  this.status = "Running";  }
```

— *«SOP» resumeExecutableNode () : void*

Esta operación – Expresión IV-16 – permite volver a activar un elemento que se encontraba suspendido.

Expresión IV-16. Especificación de la operación *«resumeExecutableNode»*

```
resumeExecutableNode () : void { if (this.status.equals ("Running")) this.status = "Stand_by"; }
```

— *«SOP» completeExecutableNode () : void*
Esta operación permite iniciar la ejecución de un elemento y para ello, cambia el estado *«Running»* a *«Complete»*.

Expresión IV-17. Especificación de la operación *«completeExecutableNode»*

```
completeExecutableNode () : void { if (this.status.equals ("Running")) this.status = "Complete"; }
```

— *«SOP» isCompleted () : Boolean*
Esta operación permite conocer si el elemento ha sido completado.

Expresión IV-18. Especificación de la operación *«resumeExecutableNode»*

```
isCompleted () : Boolean { return this.status.equals ("Complete"); }
```

Asociaciones:

— *incoming : ExecutionFlow [0..*]*
Conjunto de enlaces de entrada al elemento *«ExecutionNodeClass»*. Estos enlaces se representan en el metamodelo mediante la metaclase *«ExecutionFlow»* (Sección 3.2.12). Utilizando estos enlaces se construye la estructura de ejecución del proceso.

— *outgoing : ExecutionFlow [0..*]*
Conjunto de enlaces de salida al elemento *«ExecutionNodeClass»*. Estos enlaces se representan en el metamodelo mediante la metaclase *«ExecutionFlow»* y son los que permiten construir la estructura de ejecución del proceso.

— *executesSequenceOf: ProcessExecutionClass [1] (Asociación de composición)*
Establece cuál es el proceso en el cual se está definiendo este elemento.

— *conditionedBy: Constraint [1..*]*
Conjunto de restricciones que actúan como precondiciones indispensables para que el nodo pueda comenzar su ejecución.

— *generates: Constraint [0..*]*
Conjunto de restricciones que actúan como postcondiciones que deben ser satisfechas una vez que el nodo del proceso ha finalizado su ejecución.

— *comparedObject: Constraint [0..*]*
Conjunto de restricciones en las que se evalúa el estado actual del nodo de ejecución. La sección 3.2.8 describe la metaclase *«Constraint»*.

Restricciones:

Mientras se construye la estructura y flujo de ejecución del proceso es necesario asegurar que el modelo esté bien definido y para ello, es conveniente controlar que no exista

ninguna metaclase *«ExecutionNodeClass»* aislada dentro de la estructura. La Expresión IV-19 define esta restricción en lenguaje OCL.

Expresión IV-19. Restricción OCL sobre la metaclase «*ExecutionNodeClass*»

```
context ExecutionNodeClass inv :  self.incoming→size() + self.outgoing→size() >= 1
```

Comentarios:
Como se ha comentado a lo largo de esta sección, el metamodelo de ejecución y orquestación define – a través del enumerado *«StatusKind»* – cuáles son los estados por los que puede transitar un elemento del proceso durante su ejecución.

Sin embargo, la definición de estos estados no resulta suficiente para definir la situación interna por la que puede fluctuar cada elemento ejecutable; resulta necesario también especificar cuáles son las posibles transiciones entre los estados establecidos.

Para suplir este handicap, la Figura IV.4 formaliza el ciclo de vida de un elemento ejecutable y para ello, se utiliza la notación de máquina de estados en la que se concreta todas las posibles transiciones entre estados, siendo cualquier otra transición una transición no está permitida.

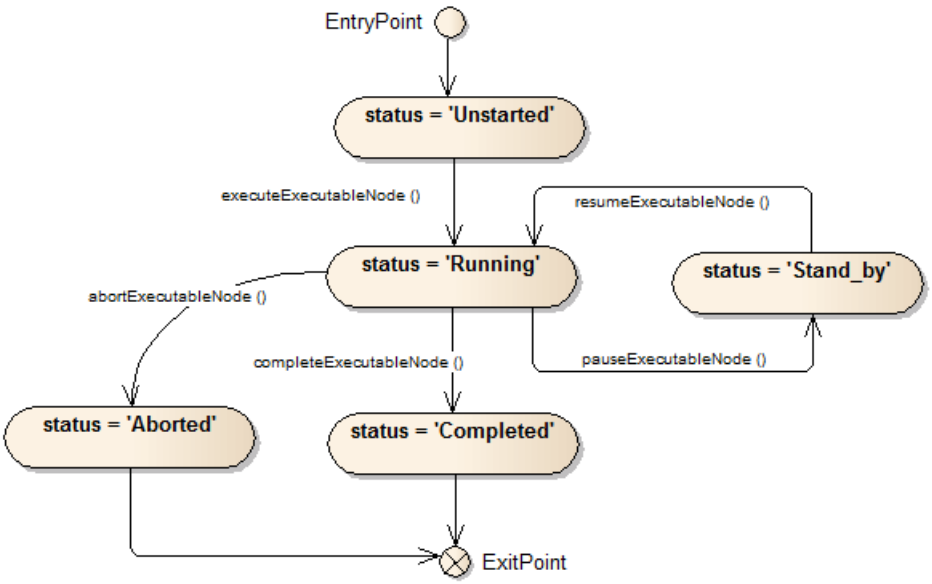


Figura IV.4. Máquina de estados asociada a cada elemento ejecutable

Como se puede apreciar en la máquina de estados mostrada en la figura anterior, ésta está íntimamente relacionada con las operaciones definidas anteriormente, ya que cada posible transición contemplada en la máquina de estados es activada por medio de la invocación a una de las operaciones de la metaclase *«ExecutionNodeClass»*. Por ejemplo, para comenzar o reanudar la ejecución de un elemento ejecutable (es decir, situar al elemento ejecutable en estado *«Running»*), éste debe encontrarse en estado *«Unstarted»* o en estado *«Stand_by»*, respectivamente. En este caso, estos cambios de estado se deben

realizar mediante las operaciones *«executeExecutableNode»* y *«resumeExecutableNode»*, respectivamente.

3.2.2. Metaclase «ProcessExecutionClass»

Descripción: Esta metaclase representa la entidad de ejecución de mayor nivel y es con la que se define la estructura del proceso y el flujo de ejecución de las actividades (humanas o máquinas) que deben ser ejecutadas. En este sentido, la metaclase *«ProcessExecutionClass»* está compuesta por un conjunto de elementos ejecutables. Esta relación de composición está modelada en la Figura IV.3 mediante una relación de composición entre las metaclases *«ProcessExecutionClass»* y *«ExecutionNodeClass»*.

Generalización: Metaclase *«ExecutionNodeClass»*.

Atributos:

— *«Dynamic» completeness: CompletenessDegree [1]*
Representa el grado de completitud del proceso en ejecución. Los posibles valores que puede tomar esta propiedad está regulada por el enumerado *«CompletenessDegree»* que representa la siguiente escala de valores: 0, 20, 40, 60, 80, 90, y 100.

Operaciones:

— *«SOP» completeExecutableNode () : void*
La metaclase *«ProcessExecutionClass»* redefine este método definido anteriormente en la metaclase *«ExecutionNodeClass»*.

El motivo de esta redefinición radica en que para completar un elemento en ejecución (es decir, transitar su estado interno hacia el estado *«Complete»*) es necesario que se cumplan las condiciones: (*i*) su estado actual debe ser el estado *«Running»*; y (*ii*) su grado de completetitud debe ser del 100%.

La Expresión IV-20 describe la especificación de esta operación.

Expresión IV-20. Especificación de la operación *«completeExecutableNode»*

```
completeExecutableNode () : void
{
    if (this.completeness.equals ("100"))
        super.completeExecutableNode ();
}
```

Asociaciones:

— *executesSequenceOf: ExecutionNodeClass [1..*] (Asociación de composición)*
Conjunto de elementos en ejecución, al menos uno, de tipo *«ExecutionNodeClass»* que componen la estructura que flujo de ejecución del proceso.

— *in-output: WorkProduct [0..*]*
Conjunto de productos de trabajo del proceso que tienen un carácter de entrada/salida. En la Sección 3.2.10 se describe en detalle la metaclase *«WorkProduct»*.

- *input: WorkProduct [0..*]*
  Conjunto de productos de trabajo que el proceso necesita como información de entrada para poder desempeñar sus tareas.
- *output: WorkProduct [1..*]*
  Conjunto de productos que el proceso va a generar durante su desempeño. Al menos, un proceso debe generar un producto de trabajo con la evidencia de su desempeño.
- *defines: BusinessVar [0..*]*
  Conjunto de variables del proceso con las que controlar su flujo de ejecución. El concepto de variables del proceso ha sido modelado mediante la metaclase *«BusinessVar»* (Sección 3.2.9).

Restricciones:

Respecto a las restricciones que se aplican sobre este elemento del metamodelo de ejecución y orquestación, es necesario controlar varios aspectos relevantes para asegurar una construcción bien formada del modelo de ejecución: (*i*) es necesario controlar que un proceso no se contenga asimismo con el objetivo de no permitir una definición recursiva e indefinida del modelo de ejecución; (*ii*) es necesario comprobar que sólo existe un elemento del proceso establecido como punto inicial del mismo; y (*iii*) cada producto debe haber sido generado por una distancia de la metaclase *«HumanExecutionClass»*. Todas estas restricciones se expresan en OCL en la Expresión IV-21.

Expresión IV-21. Restricción OCL sobre la metaclase «*ProcessExecutionClass*»

```
context ProcessExecutionClass inv:
  let activitiesColl : SortedSet = self.ExecutionNodeClass →select(oclIsTypeOf(HumanExecutionClass)) in

  //un proceso no se puede contener a sí mismo
  self.ExecutionNodeClass→select(oclIsTypeOf(ProcessExecutionClass))→count(self) = 0      and

  //sobre de existir un único nodo catalogado como 'isInittial'
  self.ExecutionNodeClass→iterate(node: ExecutionNodeClass; acc:Integer=0 |
                                        if node.isInittial then acc + 1 else acc endif) = 1   and

  //cada producto debe haber sido generado por una de las HumanExecutionClass del proceso
  self.out→forAll (p1 : WorkProduct | activitiesColl→
          exist(a : HumanExecutionClass | a.WorkProduct →exist (p2 : Product | p1 = p2))) and

  //el número de productos de salida del proceso debe coincidir con el número total de productos
  //generados por todas sus HumanExecutionClass
  self.out→size() = activitiesColl→iterate(a: HumanExecutionClass;
                                        acc : Integer = 0 | acc + a.Product→size());
```

3.2.3. Metaclase «MachineExecutionClass»

Descripción: Esta metaclase representa, de manera general, un elemento de ejecución con el que definir la coordinación de eventos durante la ejecución del proceso. Con este tipo de elementos es posible realizar peticiones o invocaciones a otros sistemas de gestión o de información para solicitar o enviar información.

Generalización: Metaclase *«ExecutionNodeClass»*.

Atributos:

- *«Static» URI: String [1]*
  Establece cuál es la ubicación unívoca del recurso o servicio al cual se accede durante la actividad de orquestación. Este dato es conocido como *Uniform Resource Identifier* o más comúnmente por su acrónimo URI.
- *«Dynamic» response: ResponseKind [1] = NOK*
  Representa cuál es el estado de la actividad de orquestación después de llevar a cabo su cometido. Los posibles valores que para tomar esta propiedad están recogidos y regulados por el enumerado *«ResponseKind»* cuyos valores son *«OK»* y *«NOK»*.

Operaciones:

- *«SOP» completeExecutableNode () : void*
  Por los mismos motivos que los proporcionados para la metaclase anterior, la metaclase *«MachineExecutionClass»* redefine este método definido anteriormente en la metaclase *«ExecutionNodeClass»* para tener en cuenta el estado de la respuesta proporcionada. La Expresión IV-22 muestra el pseudocódigo orientado a objetos de esta operación.

Expresión IV-22. Especificación de la operación *«MachineExecutionClass»*

```
completeExecutableNode () :  void
{
    if (this.response.equals ("OK"))
        super.completeExecutableNode ();
}
```

Asociaciones:

- *hasResult: BusinessVar [0..1]*
  Variable que almacena el valor devuelto por la actividad de orquestación. Esta variable está representada por la metaclase «*BusinessVar*» (Sección 3.2.9).
- *hasParameter: BusinessVar [0..1]*
  Parámetro de entrada necesario para desempeñar la actividad de orquestación. Estos parámetros están representados por la metaclase *«BusinessVar»*.

Restricciones:
Respecto a las restricciones que se aplican sobre este elemento del metamodelo de ejecución y orquestación, es necesario controlar que todos los parámetros tengan valor. Esta restricción se expresa en OCL en la Expresión IV-23.

Expresión IV-23. Restricción OCL sobre la metaclase «*MachineExecutionClass»*

```
context MachineExecutionClass inv : not self.parameter→exist(param : BusinessVar | param.value = null);
```

3.2.4. Metaclase «HumanExecutionClass»

Descripción: Esta metaclase representa aquel elemento ejecutable que debe ser llevado a cabo por intervención de un agente humano.

Generalización: Metaclase *«ExecutionNodeClass»*.

Atributos:

- *«Dynamic» completeness: CompletenessDegree [1]*
  Representa el grado de avance que el agente humano le ha dado a este elemento en ejecución. Los posibles valores que puede tomar esta propiedad está regulada por el enumerado *«CompletenessDegree»* que representa la siguiente escala de valores: 0, 20, 40, 60, 80, 90, y 100.
- *responsible_position: PositionKind [1]*
  Esta propiedad representa el perfil de la persona responsable de completar la actividad. Los posibles valores están controlados por el enumerado *«PositionKind»* (descrito en la Sección 2.2.6) aunque puede ser ampliado para soportar una extensión del modelo. A modo de recordatorio, los posibles valores controlados por este enumerado son: *«Project Manager»*; *«Developer»*; y *«Manager»*.

Operaciones:

- *«SOP» completeExecutableNode () : void*
  Al igual que para la metaclase *«ProcessExecutionClass»*, la metaclase *«HumanExecutionClass»* redefine este método de la metaclase *«ExecutionNodeClass»*.

  Los motivos de esta redefinición coinciden con los proporcionados en el caso de la metaclase *«ProcessExecutionClass»* (Sección 3.2.2) y su especificación coincide con la descrita en la Expresión IV-22.

Asociaciones:

- *generates: WorkProduct [0..*]*
  Conjunto de productos de trabajo generados por la actividad humana que se está definiendo.
- *isResponsible: HumanRole [1..*]*
  Agente humano que actúa como responsable de la actividad. La Sección 3.2.11 contiene una descripción completa de la metaclase *«HumanRole»*.

— *Con: HumanRole [0..*]*
Conjunto de agentes humanos que actúan como participantes de la actividad y por tanto, no tienen capacidad para completarla. La Sección 3.2.11 contiene una descripción completa de la metaclase *«HumanRole»*.

Restricciones:
Respecto a esta metaclase, es necesario asegurar que los productos generados por ella sean productos del proceso al que pertenece. Esta restricción se expresa en OCL en la Expresión IV-24.

Expresión IV-24. Restricción OCL sobre la metaclase *«HumanExecutionClass»*

```
context HumanExecutionClass inv:
  self.generates →forAll (product : WorkProduct | product.ProcessExecutionClass→
                                  exist(self.oclIsTypeOf(ExecutionNodeClass).ProcessExecutionClass));
```

3.2.5. Metaclase «ScriptExecutionClass»

Descripción: Esta metaclase representa, de manera específica, un elemento de orquestación con el que definir la ejecución del script. La ubicación de este script se proporciona a través de la propiedad URI de la supermetaclase.

Generalización: Metaclase *«MachineExecutionClass»*.

Atributos: Ninguna.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

3.2.6. Metaclase «WSExecutionClass»

Descripción: Esta metaclase representa, de manera específica, un elemento de orquestación con el que definir la petición o llamada a un servicio web.

Generalización: Metaclase *«MachineExecutionClass»*.

Atributos:

— *«Static» functionalModule: String [1]*
Siguiendo los principios de la arquitectura REST (acrónimo de *Representational State Transfer*) para el acceso a información a través de invocaciones a servicios web, este atributo representa el tipo de recurso al que se pretende acceder.

— *«Static» nameOperation: String [1]*
En este caso, siguiendo también el principio de la arquitectura REST, este atributo representa el nombre de la operación que se pretende ejecutar del servicio web.

— *«Static» port: String [1]*
Este atributo representa el puerto sobre el que se realiza la comunicación del servicio web.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

3.2.7. Metaclase «EMailExecutionClass»

Descripción: Esta metaclase representa un elemento de ejecución con el que definir el envío de un e-mail durante el proceso.

Generalización: Metaclase *«MachineExecutionClass»*.

Atributos:

- *«Static» mailServerUsername: String [1]*
  Esta propiedad representa las credenciales del usuario (en este caso, el nombre de usuario) para poder acceder al servidor de correo electrónico indicado. La dirección del servidor de correo electrónico deberá estar indicada en el atributo *«URI»* de la metaclase padre *«MachineExecutionClass».*
- *«Static» mailServerPassword: String [1]*
  Esta propiedad representa las credenciales del usuario (en este caso, la contraseña de usuario) para poder acceder al servidor de correo electrónico.
- *«Static» contentMessage: String [1]*
  Representa el contenido del mensaje que se desea enviar por e-mail.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

3.2.8. Metaclase «Constraint»

El desencadenamiento de la ejecución de un elemento del proceso y las posibles implicaciones posteriores a su consecución, están altamente condicionadas por ciertas situaciones. En el primer caso, esas situaciones son denominadas *pre-condiciones* afectando al inicio de la ejecución del nodo *«ExecutionNodeClass»*. En el segundo caso, esas situaciones son denominadas *post-condiciones* para reflejar condiciones que deben ser satisfechas después de ejecutar el elemento.

Respecto a las *pre-condiciones*, éstas son usadas para capturar una conjunción de eventos y condiciones que conducen a la ejecución de una actividad (humana o máquina) en un proceso de negocio. Por ejemplo, podría ser necesario que una actividad no comience su ejecución hasta que las actividades previas no hayan sido completadas o hasta que una determinada regla de negocio sea cierta.

Por su parte, las *post-condiciones* permiten definir condiciones tales como: *después de ejecutar la actividad actual, el producto generado debe haber sido completado*.

Descripción: La metaclase *«Constraint»* identifica una condición que debe ser satisfecha, ya sea considerada como una pre-condición o una post-condición.

Generalización: Ninguna.

Atributos:

- *«Static» type: ConstraintKind [1]*
  Representa el tipo de restricción. Los posibles tipos de enlace están regulados en el enumerado «*ConstraintKind*» cuyos valores se limitan a «*Pre-condition*» o «*Post-condition*».
- *«Static» comparisonValue: String [1]*
  Esta propiedad representa el valor contra el que se va a comparar el valor actual que tenga la variable o el objeto asociado.
- *«Static» logicOperator: LogicOperator [1]*
  Representa el operador lógico a aplicar durante la comparación. Los posibles operadores lógicos contemplados actualmente son *«equal»* y *«non-equal».* Ambos, modelados a través del enumerado *«LogicOperator».* Este enumerado puede ser ampliado en cualquier momento con el propósito de soportar una posible extensión del metamodelo.

Operaciones: Ninguna.

Asociaciones:

- *comparedVariable: BusinessVar [1]*
  Instancia de tipo *«BusinessVar»* para la que se consulta su valor actual. La Sección 3.2.9 describe en detalle la metaclase asociación *«BusinessVar».*
- *comparedObject: ExecutionNodeClass [1]*
  Instancia de tipo *«ExecutionNodeClass»* para la que se consulta su estado actual. La Sección 3.2.1 describe en detalle la metaclase asociación *«ExecutionNodeClass».*
- *conditionedBy: ExecutionNodeClass [1]*
  Elemento de ejecución sobre el que actúa la restricción de tipo precondición.
- *generates: ExecutionNodeClass [1]*
  Elemento de ejecución que genera la restricción de tipo postcondicion.

Restricciones:
Cuando se crea una instancia de esta metaclase es necesario comprobar que sus atributos tienen un valor asociado de manera obligatoria antes de efectuar la comparación. Esta restricción se formaliza mediante la Expresión IV-25 utilizando OCL.

Expresión IV-25. Restricción OCL sobre la metaclase *«Constraint»*

```
context Constraint inv : not self.comparisonValue = null and  not self.logicOperator = null;
```

Además, es importante recalcar que una instancia de esta metaclase sólo puede comparar un nodo de ejecución o una variable, pero no ambos a la vez. Esta restricción se controla mediante el operador *«XOR»* entre los enlaces con roles *«comparedVariable»* y *«comparedObject»*.

3.2.9. Metaclase «BusinessVar»

Descripción: El concepto de variable de negocio resulta fundamental para almacenar valores temporales durante la ejecución del proceso, con el objetivo de consulta a posteriori el estado de las mismas en diferentes puntos del proceso. Este concepto resulta esencial para construir las condiciones que actuarán como desencadenantes de la ejecución de un elemento del proceso.

Generalización: Ninguna.

Atributos:

- *«Static» name: String [1]*
  Descripción corta y concisa con la que identificar de manera unívoca a la variable dentro del contexto del proceso en el que se decide.
- *«Dynamic» value: String [1]*
  Valor actual de la variable.

Operaciones: Ninguna.

Asociaciones:

- *comparedVariable: BusinessVar [0..*]*
  Conjunto de restricciones – es decir, instancias de la metaclase *«Constraint»* – en las que se evalúa el valor de la variable
- *hasResult: MachineExecutionClass [1]*
  Elemento de ejecución que tras su desempeño almacena su resultado en la variable en cuestión.
- *hasParameter: MachineExecutionClass [0..*]*
  Conjunto de elementos de ejecución en los que se utiliza la variable definida como parámetro que almacena información de entrada.
- *defines: ProcessExecutionClass [1]*
  Especifica cuál es el proceso en el que la variable está siendo definida.

Restricciones: Ninguna.

3.2.10. Metaclase «WorkProduct»

Descripción: Como se ha mencionado a lo largo de este capítulo, resulta necesario tener en cuenta el concepto de producto cuando se modela un proceso software. El producto debe ser entendido como resultado de la ejecución del proceso o como una entrada necesaria para su desempeño. Es importante disponer de este concepto en el metamodelo de ejecución y orquestación como evidencia, ante cualquier evaluación o auditoría, de que el proceso ha sido realizado.

Generalización: Ninguna.

Atributos:

- *«Static» name: String [1]*
  Descripción corta y concisa con la que es posible identificar al producto.
- *«Static» description: String [1]*
  Descripción detallada del producto.

— *«Static» type: ProductType [1]*
Establece cuál es el tipo del producto. Los posibles valores están controlados por el enumerado *«ProductType»* (descrito en la Sección 2.2.13 de este capítulo). Los valores contemplados por este enumerado son: *«code»*, *«multimedia»*, *«document»* y *«other»*. Este enumerado puede ser ampliado en cualquier momento con el propósito de soportar una posible extensión del metamodelo.

— *«Static» resourceURI: String [1]*
Establece cual es la ubicación unívoca del producto durante la ejecución del proceso.

— *«Dynamic» version: String [1]*
Representa la versión actual del producto del trabajo.

— *«Dynamic» lastTimeModified: String [1]*
Indica cuál es la última fecha de modificación del producto.

— *«Dynamic» completeness: CompletenessDegree [1]*
Representa el grado de completitud del producto de trabajo. Los posibles valores que puede tener este atributo están regulados por el enumerado *«CompletenessDegree»* (descrito en la Sección 3.2.2 de este capítulo).

Operaciones:

— *«Constr» createInstance (String name, String descp, String resourceURI, String type)*
Esta operación permite crear una nueva instancia de la metaclase *«WorkProduct»* e inicializa todas sus propiedades.

La Expresión IV-26 especifica esta operación con un lenguaje orientado a objetos.

Expresión IV-26. Especificación de la operación *«createInstance»*

```
createInstance (String name, String descp,  String resourceURI, String type)  {
        this.completeness = 0;        this.lastTimeModified = "";
        this.version = 0;             this.resourceURI = resourceURI;
        this.name  = name;            this.description = descp;
        this.type = type;
}
```

— *«SOP» updateCompleteness (Integer degree) : void*
Esta operación permite actualizar el grado de completitud de un producto de trabajo. La Expresión IV-27 especifica formalmente esta operación.

Expresión IV-27. Especificación de la operación *updateCompleteness»*

```
updateCompleteness (Integer degree) : void { this.completeness = degree;  }
```

— *«SOP» getCompletenessDegree () : Integer*
Esta operación (Expresión IV-28) permite consultar el grado de completitud de un producto de trabajo.

Expresión IV-28. Especificación de la operación *updateCompleteness»*

```
getCompletenessDegree () : Integer { return completeness; }
```

Asociaciones:

- *in-output: ProcessExecutionClass [0..*]*
  Conjunto de procesos en ejecución que toman como entrada/salida el producto de trabajo que se están definiendo.
- *input: ProcessExecutionClass [0..*]*
  Conjunto de procesos en ejecución que toman como entrada el producto de trabajo que se están definiendo.
- *output: ProcessExecutionClass [1..*]*
  Conjunto de, al menos, un proceso en ejecución que genera el producto de trabajo que se están definiendo.
- *generates: HumanExecutionClass [1..*]*
  Conjunto de, al menos, un elemento de ejecución ejecutado por un agente humano que genera el producto de trabajo que se está definiendo.

Restricciones:
Al igual que en el metamodelo de definición de procesos definido en la sección anterior, es conveniente comprobar que un mismo producto de trabajo no esté vinculado, al mismo tiempo, con un proceso en ejecución como un producto de trabajo de entrada, salida o entrada/salida. Esta restricción está controlada en el metamodelo de ejecución y orquestación a través del operador lógico *«XOR»* de UML entre las asociaciones *«output»*, *«input»* y *«in-output»*.

3.2.11. Metaclase «HumanRole»

Descripción: Representa los agentes humanos que podrían desempeñar actividades humanas del proceso.

Generalización: Ninguna.

Atributos:

- *«Static» name: String [1]*
  Descripción corta y concisa con la que es posible identificar al agente humano.
- *«Static» description: String [1]*
  Descripción detallada sobre cuáles son las competencias del agente humano.
- *«Static» position: PositionKind [1]*
  Esta propiedad representa el perfil de la persona dentro de la organización. Los posibles valores están controlados por el enumerado *«PositionKind»* (descrito en la Sección 2.2.6) aunque puede ser ampliado para soportar una extensión del modelo. A modo de recordatorio, los posibles valores controlados por este enumerado son: *«Project Manager»*; *«Developer»*; y *«Manager»*.

Operaciones: Ninguna.

Asociaciones:

- *isResponsible: HumanExecutionClass [0..*]*
  Conjunto de elementos en ejecución en los que el agente participa como responsable de completar dicho elemento en ejecución.
- *isParticipant: HumanExecutionClass [0..*]*
  Conjunto de elementos en ejecución en los que el agente actúa como un simple participante en la ejecución del elemento; sin tener capacidad para dar por completada la ejecución.

Restricciones:
Resulta conveniente establecer que un mismo agente no puede ser responsable y participante en la ejecución de un mismo elemento del modelo. Esta restricción se incluye en el metamodelo por medio del operador lógico *«XOR»* de UML entre las asociaciones *«isResponsible»* y *«isParticipant»* descritas anteriormente.

3.2.12. Metaclase «ExecutionFlow»

Descripción: Esta metaclase posibilita la definición de la estructura interna del proceso en base a su flujo de ejecución ya que permite establecer relaciones entre los elementos en ejecución.

Generalización: Ninguna.

Atributos:

- *«Static» Guard: String [1]*
  Representa el valor que debe tomar una variable para poder tomar ese camino del flujo y que defecto toma el valor *«Default»* (el valor almacenado en este atributo permite determinar el siguiente nodo de ejecución después de ejecutar un nodo condicional).
- *«Static» type: FlowKind [1]*
  Representa el tipo de enlace. Los posibles tipos de enlace están regulados en el enumerado *«FlowKind»* cuyos valores se limita a los siguientes: *«Normal»*, valor por defecto que define un flujo de ejecución que no tiene que ser exactamente capturado por ninguna excepción; y *«Exception»*, es que establece un único flujo de ejecución cuando una excepción ha ocurrido. Este enumerado puede ser ampliado en cualquier momento con el propósito de soportar una posible extensión del metamodelo.

Operaciones: Ninguna.

Asociaciones: Ninguna.

Restricciones: Ninguna.

4. Conclusiones

A lo largo de este capítulo se han presentado de una manera formal los metamodelos necesarios para abordar la definición de procesos software, así como la definición de su contexto de ejecución y orquestación dentro de un marco de trabajo PLM_4BS (*«Process*

Lifecycle Management for Business-Software») cuyo objetivo es el de gestionar el ciclo de vida del proceso software desde la perspectiva del paradigma MDE.

Esta definición y representación formal, materializada por medio de la notación de diagramas de clases UML, permite establecer relaciones de derivación entre los dos metamodelos definidos que, implementadas en una herramienta CASE, pueden incluso automatizar parte de la tarea de modelado. Estas reglas de derivación son definidas en el Capítulo V.

Además, el Capítulo VI describe una sintaxis concreta – basada en perfiles UML – para los metamodelos presentados en este capítulo. Un perfil de UML es una extensión formal del propio lenguaje UML con el objetivo de definir nuevos conceptos a partir de constructores ya existentes en UML, con el fin de dotarles de una semántica más precisa y concreta.

Capítulo V DERIVACIÓN ENTRE MODELOS

En los capítulos anteriores se han definido y formalizado los conceptos que deben contemplar el lenguaje de definición y el lenguaje de ejecución y orquestación del proceso software mediante metamodelos. El objetivo de este capítulo es especificar formalmente un proceso sistemático basado en dos etapas para obtener código ejecutable a partir del modelado del proceso software.

Por una parte, la primera etapa consiste en generar el modelo de ejecución y orquestación a partir del modelo de definición del proceso software. Para este propósito es necesario definir un conjunto de reglas de derivación entre modelos que permitan sistematizar esta etapa. Estas reglas son especificadas con el lenguaje estándar QVT –*«Query/View/Transformation*» – [OMG 2008b] y dentro del argot del paradigma guiado por modelos son denominadas comúnmente como reglas *«model-to-model*».

Por otra parte, la segunda etapa consiste en generar código ejecutable desde el modelo de ejecución y orquestación del proceso software, utilizando para ello un conjunto de reglas de transformación. En este caso, para la especificación de estas reglas se utilizará el lenguaje estándar «*MOF Model to Text Transformation Language*» conocido por su acrónimo MOFM2T [OMG 2008c]. Este tipo de reglas se referencian frecuentemente por la expresión reglas *«model-to-text*».

Para conseguir el objetivo marcado, este capítulo se ha organizado en varias secciones principales. En primer lugar, la Sección 1 expone la visión global del proceso sistemático de generación mencionado anteriormente y además, presenta una descripción general de los lenguajes de transformación QVT y MOFM2T.

A continuación, la Sección 2 formaliza las transformaciones QVT necesarias para generar el modelo de ejecución y orquestación a partir del modelo de definición del proceso software y la Sección 3 describe las reglas de transformación MOFM2T para generar código ejecutable que pueda ser interpretado por un motor de procesos convencional.

Finalmente, el capítulo concluye con una serie de consideraciones finales.

1. Introducción

Antes de especificar las reglas de transformación necesarias para obtener, en última instancia, código ejecutable desde el modelo de definición del proceso software, resulta conveniente establecer la estrategia a seguir y sentar las bases teóricas y técnicas necesarias. Éste es el propósito de esta primera sección del Capítulo V, y para alcanzarlo, la

Sección 1.1 presenta de manera concreta el proceso sistemático global de obtención de código ejecutable desde el modelo de ejecución y orquestación del proceso, el cual, a su vez, es derivado desde el modelo de definición del proceso.

En segundo lugar, las Secciones 1.2 y 1.3 describen brevemente los lenguajes de transformación QVT y MOFM2T, con los que son especificadas, respectivamente, las reglas de transformación «*model-to-model*» y «*model-to-text*» necesarias.

La justificación por la que se ha optado por utilizar QVT y MOFM2T en detrimento de otros lenguajes como pudiera ser ATL – *«ATL Transformation Language»* – [Jouault *et al.* 2008] son básicamente dos: (*i*) QVT y MOFM2T son lenguajes estándares; y (*ii*) todos los trabajos con los que se prevé integrar la propuesta desarrollada en esta tesis – por ejemplo, la metodología NDT descrita en el Capítulo III – están especificados con QVT, así que utilizar esta opción podrá garantizar la compatibilidad entre propuestas.

1.1. Proceso sistemático de derivación de la propuesta

En primer lugar, un proceso global de generación del modelo de ejecución y orquestación del proceso software a partir del modelo con su definición debe contemplar qué información se va a tomar desde el modelo de definición del proceso, cómo se va a manipular y transformar dicha información, qué dependencias existen con otras informaciones y qué información se va a obtener al final.

La pregunta que se puede plantear, por tanto, es de qué manera se podría definir con precisión los aspectos anteriores y, en concreto, cómo se podría definir el proceso de generación del modelo de ejecución y orquestación a partir del modelo de definición del proceso. También se plantea si dicho proceso podría ser automático y hasta qué nivel puede serlo.

Durante todo este capítulo, y basándose en la definición formal de los metamodelos que se ha realizado durante el capítulo anterior, se van a estudiar las relaciones semánticas que se establecen entre ellos, mediante las cuales se puede conseguir el modelo de ejecución y orquestación del proceso. Esta primera versión de este modelo puede ser denominado como *modelo básico* de ejecución y orquestación del proceso software.

Sin embargo, el proceso con el que se lleva a cabo el modelado del proceso software y su contexto de ejecución y orquestación no se puede limitar a este aspecto. En ese sentido, es conveniente establecer un mecanismo de transformación – basado en reglas – con el que sea posible obtener este modelo básico. Sin embargo, es necesario también permitir al ingeniero de procesos la capacidad de aplicar su conocimiento experto para hacer las modificaciones oportunas que permitan adecuar, de la mejor manera posible, el contexto de ejecución y orquestación a la realidad del entorno software final de ejecución, consiguiendo así lo que se podría denominar como el *modelo final* de ejecución y orquestación.

En la Sección 4.2 del Capítulo III en la que se expone el planteamiento del problema, la Figura III.5 muestra una primera aproximación de la solución. Esta solución presenta de manera general el proceso para obtener código ejecutable a partir del modelo de ejecución

y orquestación del proceso software, y el procedimiento de obtención de éste a partir del modelo de definición del proceso software. Tomando como base esta figura, a continuación se describe con mayor grado de detalle el proceso de derivación en su totalidad.

El proceso sistemático de derivación que se ha planteado en este trabajo de tesis se muestra en la Figura V.1 y, como se puede apreciar, está fuertemente influenciado por la metodología NDT – descrita en la Sección 3 del Capítulo III –. Esta influencia se ve justificada principalmente por el éxito que ha tenido y está teniendo a nivel práctico el método de trabajo de NDT y su proceso global de transformación entre modelos. Este método de trabajo ha permitido extender la aplicabilidad de la metodología NDT hacia múltiples y diferentes ámbitos de negocio y contextos de utilización – por ejemplo, hacia entornos colaborativos con equipos multidisciplinares.

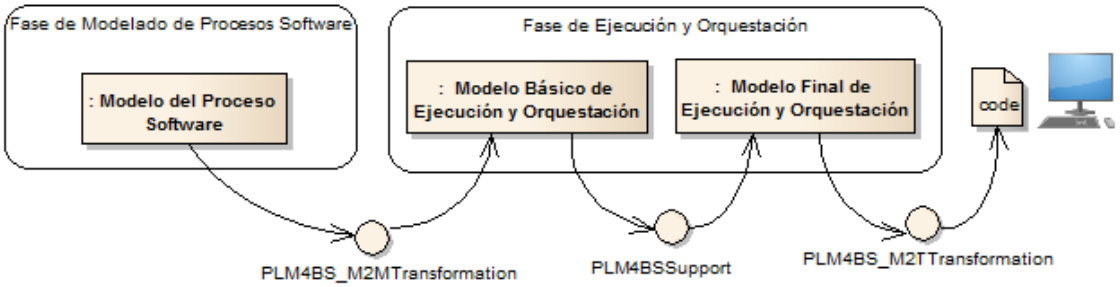


Figura V.1. Descripción del proceso de derivación dentro del ciclo de vida BPM

Como puede apreciarse en la imagen anterior, la hoja de ruta para generar un código ejecutable y orquestable desde el modelo de definición de proceso software, está constituida de tres etapas de transformación bien diferenciadas.

En primer lugar, el modelo básico de ejecución y orquestación se genera de manera sistemática y automática a partir del modelo de definición del proceso software. Para ello, se utilizan un conjunto de reglas de derivación *«model-to-model».* Esta etapa está representada en la Figura V.1 con el estereotipo «*PLM4BS_M2MTransformation*».

Una vez derivado el modelo básico de ejecución y orquestación, el grupo de ingenieros de procesos podría realizar cambios manualmente en el modelo básico para conseguir los modelos finales. Ésta es la segunda etapa del procedimiento y depende en gran medida del conocimiento y habilidad del ingeniero de procesos para ajustar el modelo del proceso al contexto de ejecución concreto. Esta etapa está representada en la Figura V.1 con el estereotipo «*PLM4BSSupport*».

Esta derivación no sistemática generará la versión final del modelo de orquestación y ejecución. Ahora bien, es necesario controlar y definir claramente cuáles son los cambios que puede realizar el ingeniero sin contradecir las reglas semánticas y la trazabilidad entre los modelos. Por todo ello, en la Sección 3 de este capítulo se identifican y estudian todas las relaciones que permiten derivar los modelos básicos y se catalogan también todos los cambios que el usuario puede realizar. De esta forma, cuando el ingeniero de proceso detecte la necesidad de hacer un cambio en un modelo básico, podría ver si ese cambio es, o no, acorde a las relaciones semánticas establecidas entre los modelos y

detectar así, o bien un error en el modelo de ejecución o bien un error en el modelo de definición del proceso.

Inicialmente, esta solución inicial se presupone viable en el contexto de esta tesis tras las buenas experiencias cosechadas por la metodología NDT en este tema. De hecho, esta solución se ve justificada por la influencia que NDT tiene sobre el proceso sistemático de derivación descrito en esta sección. En este sentido, NDT dispone de su herramienta de soporte NDT-Quality y gracias a ella, ha sido posible aplicar la metodología dentro de entornos reales y controlar automáticamente la calidad de los modelos definidos por el usuario (así como las modificaciones que éste pudiera haber realizado sobre los modelos) sin incurrir por ello en un coste excesivo durante tareas de aseguramiento de la calidad.

Una vez finalizada la fase de orquestación y ejecución, el ingeniero de procesos podrá generar una versión ejecutable del modelo aplicando sobre el propio modelo final un conjunto de reglas de derivación *«model-to-text».* Esta etapa está representada en Figura V.1 con el estereotipo «*PLM4BS_M2TTransformation*». La salida de estas reglas de derivación será código ejecutable WS-BPEL [OASIS 2007] que podrá ser desplegado en un motor de ejecución de procesos concreto.

Es importante mencionar que el propio lenguaje WS-BPEL presenta limitaciones para representar toda la semántica del metamodelo de ejecución y orquestación (descrito en la Sección 3 del Capítulo IV). Por este motivo, se hace necesaria la intervención manual del ingeniero de procesos o analista para completar todos los aspectos técnicos necesarios con el fin de desplegar y ejecutar el código WS-BPEL generado en un motor de procesos. Estas limitaciones son descritas en la Sección 2.1 de este capítulo. Esta etapa de mejora manual del código generado se lleva a cabo después de la transformación «*PLM4BS_M2TTransformation*» antes mencionada.

Respecto al código de salida de la propuesta aquí presentada, se hace palpable la clara dependencia con el lenguaje de ejecución de procesos WS-BPEL. Esta dependencia se debe a que actualmente no existen motores de ejecución de procesos que permitan desplegar y ejecutar procesos que estén definidos mediante lenguajes basados en UML. Por este motivo, ha sido necesario obtener código que pueda ser ejecutado en la mayoría de motores de ejecución de procesos (por ejemplo, IBM Websphere, Suite BPM de Oracle, jBPM, BonitaOS y Activiti, entre otros). En cualquier caso, se plantea como trabajo futuro la posible investigación de un motor de procesos que pueda interpretar y ejecutar modelos UML.

Gracias al proceso de transformación descrito anteriormente, se establece un vínculo o trazabilidad entre el metamodelo de definición de procesos software y el metamodelo de ejecución y orquestación. Sin embargo, hay un aspecto relacionado con las modificaciones a posteriori que es conveniente tener en cuenta. Este respecto se refleja en la Figura V.2.

Como se ha indicado anteriormente, el estereotipo «*PLM4BS_M2MTransformation*» refleja que el proceso para la obtención del modelo básico de ejecución y orquestación es un proceso sistemático y automático, mientras que el estereotipo «*PLM4BSSupport*» es un proceso manual que depende de la experiencia del analista. Sin embargo, como diferencia respecto la Figura V.1, la Figura V.2 añade una nueva línea discontinua: la línea de retorno.

Desde la realización del modelo final de ejecución y orquestación puede darse la necesidad de realizar cambios en el modelo de definición del proceso. Este proceso de retorno no es automático ya que depende de las decisiones y la experiencia del grupo de ingenieros de procesos o analistas.

Sin embargo, tras estas modificaciones resulta necesario garantizar la calidad de los modelos y su trazabilidad. Aunque todas las posibles modificaciones están catalogadas en la Sección 3 de este capítulo, esta tesis no abarca su automatización. El diseño y desarrollo de una herramienta CASE que dé soporte a este aspecto, es uno de los trabajos futuros planteados en esta tesis.

Además, tal y como se puede apreciar, la Figura V.2 añade una línea discontinua que representa una retroalimentación en el código WS-BPEL generado. Esta línea refleja las modificaciones que el ingeniero de procesos o analista debe realizar de manera manual para completar algunos aspectos necesarios para la correcta ejecución del código en un motor de procesos.

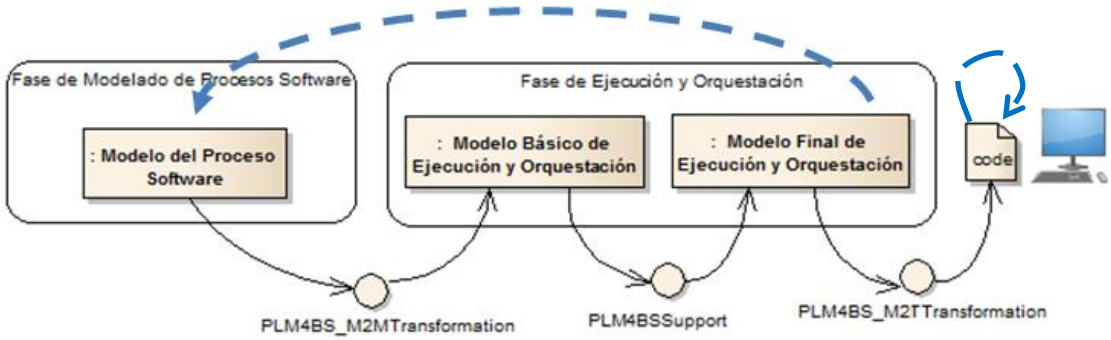


Figura V.2. Descripción del proceso de derivación con vueltas atrás

1.2. QVT: lenguaje de especificación formal de transformaciones model-to-model

«Query/View/Transformation», QVT [OMG 2008b], es un lenguaje para definir reglas de transformación entre modelos UML. Una transformación se puede definir como un conjunto de pasos para construir un modelo de salida a partir de un modelo de entrada.

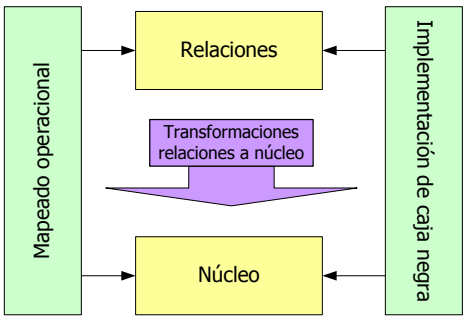


Figura V.3. Descripción general de QVT [OMG 2008b]

Tal y como refleja la Figura V.3, QVT define tres lenguajes específicos de dominio: (*i*) un lenguaje relacional y declarativo, basado en el concepto de *pattern matching*, según el cual, se define un conjunto de patrones que el entorno de ejecución intentará hacer coincidir con elementos del modelo; (*ii*) un núcleo o *core*, también definido como un

lenguaje declarativo; y (*iii*) un lenguaje operacional, que es una extensión y complemento de los lenguajes relacional y *core*, para incorporar construcciones imperativas como condicionales, bucles, etc.

QVT, además de los lenguajes y mecanismos mencionados, integra el lenguaje de definición de restricciones OCL y también introduce el concepto de *BlackBox* o caja negra, mediante el cual es posible incorporar funciones auxiliares desarrolladas en otros lenguajes (como por ejemplo XQuery o Java, entre otros) a las transformaciones.

Las transformaciones QVT se especifican en base a los elementos definidos en un metamodelo y se aplican sobre las instancias de dicho metamodelo.

En cuanto su definición, una regla de transformación QVT contiene una cabecera, la especificación de las claves y la especificación de todos los mapeos y funciones auxiliares necesarios. La cabecera indicará además cuáles son los metamodelos de entrada y salida. Estos metamodelos permiten especificar de qué tipo serán los modelos sobre los que se pueda aplicar la transformación. Como ejemplo, si una transformación especifica como entrada el metamodelo A y como salida el metamodelo B, será una transformación que se pueda aplicar sobre cualquier modelo conforme al metamodelo A, para obtener como resultado un modelo conforme al metamodelo B.

Además de transformaciones, el lenguaje QVT también permite definir funciones auxiliares que pueden ser referenciadas desde cualquier mapeo. Las funciones auxiliares utilizadas en la especificación formal de las transformaciones de este capítulo se dividen en dos tipos: *helpers* y *queries* (utilizamos los términos originales en inglés).

La principal diferencia entre ambas es que una función *helper* no define el retorno de ningún valor mientras que una función *query* sí.

1.3. MOFM2T: lenguaje de especificación formal de transformaciones model-to-text

MOFM2T [OMG 2008c] – *«MOF Model to Text Transformation Language*» – es un lenguaje estándar propuesto por el OMG, *«Object Management Group*». Este estándar aborda cómo traducir un modelo en una representación textual linealizada (código, documentos, etc.).

Para generar la aproximación textual de modelos, MOFM2T propone la definición de reglas de derivación basadas en plantillas mediante las cuales, el texto que se genere a partir de los modelos se especifica como un conjunto de plantillas de texto parametrizadas con los elementos del modelo.

Además de transformaciones, el lenguaje MOFM2T también permite definir funciones auxiliares que pueden ser referenciadas desde cualquier plantilla.

2. Generación del modelo básico de ejecución y orquestación desde el modelo de definición del proceso software

En el capítulo anterior se ha definido un lenguaje basado en UML para la definición del proceso software. Sin embargo, es necesario definir un protocolo sistemático que haga posible la definición del contexto de ejecución y orquestación del mismo.

Esta sección tiene como propósito definir el contexto y planteamiento necesario para establecer las reglas de derivación oportunas que permitan obtener el modelo de ejecución y orquestación del proceso software a partir del modelo con su definición.

2.1. Contexto y planteamiento previo

Como paso previo al diseño y planteamiento de las reglas de transformación QVT para derivar los elementos del modelo de ejecución y orquestación desde el modelo de definición del proceso software, resulta conveniente e interesante establecer cuál es el mapeo entre los elementos de ambos metamodelos.

La Tabla V.1 muestra esta correspondencia – junto con unos comentarios preliminares – como resumen previo para afrontar las reglas de derivación expuestas a lo largo de esta sección.

Tabla V.1. Correspondencia entre elementos de los metamodelos de soporte a BPM

Elemento metamodelo definición de proceso	Elemento metamodelo ejecución y orquestación	Comentario
«Process»	*«ProcessExecutionClass»*	—
«ProcessElement»	*«ExecutionNodeClass»*	—
«Activity»	—	La traducción de esta metaclase en un elemento del metamodelo de ejecución se circunscribe a sus metaclases específicas.
«ControlElement»	*«Constraint»*	La metaclase *«ControlElement»* se traduce en la metaclase *«Constraint»* si su atributo «type» es igual a *«Conditional»*, *«Fork»* o *«Join»*. Los nuevos elementos *«Constraint»* permiten condicionar la ejecución de los elementos enlazados con el elemento *«ControlElement»*. Estas restricciones implican la consulta del estado interno de los nodos de ejecución relacionados con el elemento de control del proceso.
	—	En el caso de que la metaclase *«ControlElement»* sea de tipo *«InitialElement»* o *«FinalElement»*, ésta no tiene traducción en el metamodelo de ejecución y orquestación. Sin embargo, en el caso particular de que sea de tipo *«InitialElement»* su siguiente elemento – en la secuencia de pasos – será considerado el primer elemento del proceso. Para ello, se informará la propiedad *«isInittial»* de la metaclase *«ExecutionNodeClass»*.
«ComplexActivity»	*«ProcessExecutionClass»*	—
«OrchestrationActivity»	*«ScriptExecutionClass»* *«EMailExecutionClass»* *«WSExecutionClass»*	Esta correspondencia se establece en función del valor que contenga la propiedad *«type»* de la metaclase *«OrchestrationActivity»*.
«HumanActivity»	*«HumanExecutionClass»*	—

«Stakeholder»	*«HumanRole»*	—
«BusinessVar»	*«BusinessVar»*	—
«Link»	*«ExecutionFlow»*	—
«checkBusinessVar»	*«Constraint»*	Al igual que la metaclase *«Conditional»*, esta metaclase es traducida en precondiciones – metaclase *«Constraint»* – con las que condicionar la ejecución de los elementos enlazados con cualquier *«Conditional»* en el que se evalúa una variable de negocio.
«updateBusinessVar»	*«Constraint»*	Esta metaclase es traducida como una postcondición – en la que está involucrada la variable actualizada – que debe ser cierta después de la ejecución de un determinado elemento del proceso – metaclase *«ExecutionNodeClass»*.
«Product»	*«WorkProduct»*	—
«Indicator»	—	Estas metaclases no disponen de traducción en elementos del metamodelo de ejecución.
«Metric»	—	

Una vez planteada la tabla de correspondencia anterior, las siguientes subsecciones especifican las principales reglas QVT necesarias para llevar a cabo las transformaciones indicadas.

2.2. Especificación de la transformación *«PLM4BS_M2MTransformation»*

Esta sección específica la primera transformación – transformación estereotipada con «*PLM4BS_M2MTransformation*» en la Figura V.1 – del procedimiento sistemático de derivación propuesto en esta tesis. Esta transformación es la encargada de generar el modelo de ejecución y orquestación de procesos software desde el modelo de definición del proceso software. Esta transformación toma como entrada los elementos vistos en el metamodelo de definición de procesos software y genera todos los elementos necesarios para construir el modelo básico de ejecución y orquestación definido conforme a su metamodelo. Ambos metamodelos son descritos completa y formalmente en el capítulo anterior.

La Expresión V-1 especifica en lenguaje QVT la cabecera de esta transformación – mediante la directiva *«transformation»* de QVT –, así como los mapeos y funciones auxiliares necesarias. Una vez presentada, esta transformación es descrita en detalle.

Tal y como es posible apreciar, la transformación toma como entrada – directiva *«in»* – un modelo conforme al metamodelo de definición de procesos software y devuelve como salida – directiva *«out»* – un modelo conforme al metamodelo de ejecución y orquestación del proceso software.

En primer lugar, en el cuerpo principal – directiva *«main»* – de esta transformación se especifica un bloque de código QVT con el que se define el conjunto de propiedades que, para cada objeto instanciado del modelo destino, actuará como identificadores unívocos

de la nueva instancia. Esto se consigue mediante la directiva *«key»* de QVT. En este caso, se ha decidido incluir identificadores en los elementos conceptuales representados en los metamodelos del capítulo anterior. Estos nuevos atributos permitirán la posterior identificación y resolución de los elementos que intervienen en todas las que son posible establecer entre elementos.

Expresión V-1. Transformación QVT para derivar el modelo de ejecución y orquestación

```
transformation PLM4BS_M2MTransformation (in spm : SPDefinitionMetamodel,
                                         out spexem : SPExecutionMetamodel) {
    main() {
        /* Se define el conjunto de propiedades que actúan como identificadores unívocos del nuevo objeto. */

        key ProcessExecutionClass (id);   key EMailExecutionClass  (id);   key HumanExecutionClass  (id);
        key WSExecutionClass      (id);   key ScriptExecutionClass (id);   key ExecutionFlow (id);
        key HumanRole  (name);            key WorkProduct   (id);
        key BusinessVar  (name, process); key Constraint  (id, ExecutionNodeClass);

        // Se realiza las llamadas a las funciones de mapeo.
        spm.objectsOfType(Stakeholder)→map toHumanRole ();
        spm.objectsOfType(Product)→map toWorkProduct ();
        spm.objectsOfType(BusinessVar)→map toBusinessVar ();
        spm.rootObjects()[Process]→map toExecutableProcess ();
        spm.objectsOfType(Link)→map toExecutionFlow ();
    }
}
```

Una vez planteados los identificadores de los objetos, la regla de transformación realiza una serie de llamadas a varias funciones de mapeo que se irán describiendo en las siguientes secciones. El orden de invocación de estas funciones de mapeo es importante ya que de esta forma es posible consultar y enlazar elementos generados previamente.

De entre estas llamadas, cabe destacar la función de mapeo que permite transformar el elemento raíz del metamodelo de entrada, es decir, el elemento *«Process»*. Este elemento es obtenido mediante la instrucción *«rootObjects»* de QVT. El nombre de esta función de mapeo es *«toExecutableProcess»* y su especificación se detalla a continuación en la Sección 2.3. El objetivo de esta función es generar el elemento *«ProcessExecutionClass»* (elemento raíz del metamodelo de ejecución y orquestación) y comenzar con la transformación de todos los elementos que componen el proceso.

Una vez que todos los elementos del modelo de entrada han sido mapeados a sus respectivos elementos del modelo de salida, es el momento de invocar la función de mapeo para derivar los enlaces entre dichos elementos. El nombre de esta función de mapeo es *«toExecutionFlow»* y los detalles de su especificación se describen en la Sección 2.6.

A continuación, las siguientes secciones describen de manera concreta todas las reglas de mapeo que conforman el protocolo de derivación del modelo de ejecución y orquestación a partir del modelo con la definición del proceso software.

2.3. Especificación del mapeo de la metaclase *«Process»*

Para especificar cualquier regla de mapeo se utiliza la directiva *«mapping»* de QVT. Utilizando esta directiva, la Expresión V-2 describe de manera específica cómo se deriva cada *«ProcessExecutionClass»* (elemento raíz del modelo de ejecución y orquestación) a partir de un *«Process»* (elemento raíz del modelo de definición de procesos).

El propósito de esta expresión es inicializar las propiedades del nuevo elemento *«ProcessExecutionClass»* así como servir como punto de entrada para ejecutar las funciones de mapeo de todos los elementos que componen el modelo con la definición proceso software.

Como se puede apreciar, después de inicializar las propiedades de la nueva instancia *«ProcessExecutionClass»* comienza la generación de los elementos del modelo de ejecución y orquestación del proceso a partir de los elementos que componen el modelo de definición del proceso.

Según recoge el metamodelo de definición del proceso software descrito en la Sección 2.2 del Capítulo IV y según muestra – a modo parcial y como recordatorio – la Figura V.4, se establecen diferentes relaciones sobre la metaclase *«Process»*: (*i*) una composición para definir los elementos estructurales del proceso; (*ii*) tres asociaciones con la metaclase *«Product»*; y (*iii*) una asociación con las variables de negocio que afectan al flujo del proceso. Estas dos relaciones se transforman mediante la llamada a las funciones de mapeo *«toExecutionNodeClass»* y *«toBusinessVar»*, descritas en la Sección 2.5 y Sección 2.4, respectivamente.

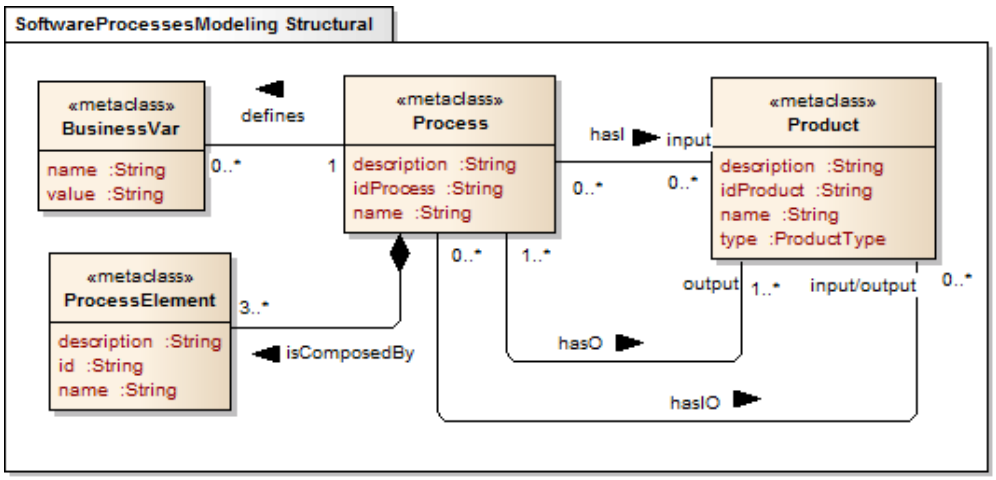


Figura V.4. Relaciones sobre la metaclase *«Process»* (fragmento del metamodelo)

Teniendo en cuenta las posibles relaciones que presenta una instancia de la metaclase *«Process»*, la Expresión V-2 describe finalmente su regla de mapeo.

Expresión V-2. Mapeo QVT de la metaclase *«Process»* a *«ProcessExecutionClass»*

```
mapping SPDefinitionMetamodel::Process::toExecutableProcess :
                                                    SPExecutionMetamodel::ProcessExecutionClass ()
{
    // Se inicializan las propiedades del elemento ProcessExecutionClass
    completeness := 0;          isInittial := false;                status  := "Unstarted";
    name  := self.name;         description := self.description;

    // Se transforma cada elemento relacionado con el proceso a su homólogo dentro del modelo de ejecución
    defines += self.defines→forAll (var: BusinessVar | var.resolveone(SPExecutionMetamodel::BusinessVar);

    input += self.hasI→forAll (p: Product | p.resolveone(SPExecutionMetamodel::WorkProduct);
    output += self.hasO→forAll (p: Product | p.resolveone(SPExecutionMetamodel::WorkProduct);
    in-output += self.hasIO→forAll (p: Product | p.resolveone(SPExecutionMetamodel::WorkProduct);

    executesSequenceOf += self.isComposedBy→map toExecutionNodeClass ();
}
```

Por último, respecto a la función de mapeo anterior cabe destacar un detalle: el uso de la directiva ***«resolveone»*** de QVT. Cuando un objeto del modelo fuente es transformado en un objeto del modelo destino, y más adelante el mismo objeto fuente aparece como una referencia, éste no debería ser transformado de nuevo. En vez de esto, la referencia debe ser resuelta con el objeto recién creado en el modelo destino.

Para obtener esta referencia el lenguaje QVT proporciona varias directivas, pero en nuestro caso se ha optado por utilizar ***«resolveone»***, la cual devuelve un único objeto destino y en caso de que el mapeo del objeto en el modelo fuente resultase en múltiples objetos en el modelo destino, esta directiva sólo devuelve el último de los objetos creados.

2.4. Especificación del mapeo de la metaclase *«BusinessVar»*

Este mapeo se materializa con la función *«toBusinessVar»* en la Expresión V-3. Mediante esta regla de mapeo es posible transformar cada instancia de la metaclase *«BusinessVar»* en una instancia de la metaclase homóloga en el modelo de ejecución y orquestación.

Expresión V-3. Mapeo QVT de la metaclase *«BusinessVar»*

```
mapping SPDefinitionMetamodel::BusinessVar::toBusinessVar : SPExecutionMetamodel::BusinessVar () {
    name  := self.name;       value := self.value;
}
```

2.5. Especificación del mapeo de la metaclase *«ProcessElement»* y sus subtipos

La regla de transformación que mapea cada instancia de la metaclase *«ProcessElement»* en una instancia de la metaclase en el modelo de ejecución y orquestación se lleva a cabo mediante la función *«toExecutionNodeClass»*, la cual, viene descrita en la Expresión V-4.

Expresión V-4. Mapeo QVT de la metaclase *«ProcessElement»* a *«ExecutionNodeClass»*

```
mapping  SPDefinitionMetamodel::ProcessElement::toExecutionNodeClass :
         SPExecutionMetamodel::ExecutionNodeClass ()
disjuncts SPDefinitionMetamodel::Activity::transformActivity
{   /* Sin instrucciones de mapeo */   }
```

Al igual que en las anteriores funciones de mapeo, se ha utilizado la directiva *«mapping»* de QVT. Sin embargo, tal y como se puede apreciar, esta directiva se ha combinado con el uso de la directiva *«disjuncts»* de QVT.

La directiva *«disjuncts»* compara el tipo de objeto de origen de la función de mapeo con los tipos de objetos de origen de las asignaciones que se indican después de la palabra clave *«disjuncts»*. De esta manera, cuando se recibe uno de los objetos de origen indicados, se ejecuta la función de transformación asociada. En otras palabras, la semántica del "mapeo disjunto" es comprobar el tipo del objeto de entrada y trasmitirlo a la primera función de mapeo capaz de transformar dicho objeto.

Una vez conocida la semántica de la directiva *«disjuncts»* resulta más sencillo explicar que la regla de mapeo expresada en la Expresión V-4 refleja que todos los objetos del modelo de entrada que sean del tipo *«ProcessElement»* y subtipo *«Activity»* serán transformados mediante la función de mapeo *«transformActivity»*. Sin embargo, existe un caso especial. Si se consulta el metamodelo de definición del proceso software descrito en el Capítulo IV, se puede observar que la metaclase *«ProcessElement»* tiene también por subtipo a la metaclase *«ControlElement»*. La estrategia seguida para transformar la metaclase *«ControlElement»* pasa por utilizar una función QVT auxiliar conforme se mapean las instancias de la metaclase *«Activity»*. A continuación, se describe todo este procedimiento y la estrategia de transformación.

Respecto a la función de mapeo *«transformActivity»*, ésta se describe en la Expresión V-5 y como se puede apreciar tiene una estructura análoga a la función de mapeo de la metaclase *«ProcessElement»* (descrita anteriormente en la Expresión V-4).

Expresión V-5. Mapeo QVT de la metaclase *«Activity»*

```
mapping  SPDefinitionMetamodel::Activity::transformActivity () : SPExecutionMetamodel::ExecutionNodeClass ()
disjuncts SPDefinitionMetamodel::HumanActivity::toHumanExecutionClass,
          SPDefinitionMetamodel::ComplexActivity::toExecutableProcessFromComplexActivity,
          SPDefinitionMetamodel::OrchestrationActivity::toScriptExeClass,
```

```
        SPDefinitionMetamodel::OrchestrationActivity::toEMailExeClass,
        SPDefinitionMetamodel::OrchestrationActivity::toWSExeClass
{  /* Sin instrucciones de mapeo */  }
```

A continuación, se describe cada una de las reglas de mapeo referenciadas en la Expresión V-5.

En primer lugar, la función de mapeo *«toHumanExecutionClass»* transforma una instancia de la metaclase *«HumanActivity»* del metamodelo de definición del proceso en una instancia de la metaclase *«HumanExecutionClass»* del metamodelo de ejecución y orquestación.

La Expresión V-6 recoge la especificación de esta regla QVT y como se puede apreciar, se han utilizado varias funciones auxiliares. La primera, la función *«isInittialActivity»*, permite comprobar si la metaclase *«HumanActivity»* que se está transformando es la primera actividad del modelo de definición del proceso software. Para definir esta función auxiliar se ha utilizado la directiva *«query»* de QVT con la que es posible utilizar y combinar expresiones QVT y OCL para recopilar datos desde el objeto de entrada, y a partir de estos datos devolver una salida.

Expresión V-6. Mapeo QVT de la metaclase *«HumanActivity»* a *«HumanExecutionClass»*

```
mapping SPDefinitionMetamodel::HumanActivity::toHumanExecutionClass () :
        SPExecutionMetamodel::HumanExecutionClass  {
    // Se inicializan los atributos de la 'HumanExecutionClass'
    status  := "Unstarted";              name := self.name;          description  := self.description;
    isInittial := isInittialActivity ();  responsible_position := self. responsible_position;

    // Se asocia la 'HumanExecutionClass' con los productos que genera y con
    // los roles de usuario establecidos en el modelo de definición
    generates   += self.Product→forAll (p : Product | p.resolveone(SPExecutionMetamodel::WorkProduct);
    responsible  := self.isResponsible.resolveone(SPExecutionMetamodel::HumanRole);
    participant   += self.isParticipant→forAll (rol : Stakeholder |
                                                    rol.resolveone(SPExecutionMetamodel::HumanRole);
    PRE  := createPreConditions ();
    POST := createPosConditions ();
}
```

La Expresión V-7 describe la función *«isInittialActivity»*, pero se recomienda consultar el metamodelo de definición de proceso software mostrado en la Figura IV.1 del Capítulo IV para una mejor comprensión. Brevemente, esta función permite consultar si la instancia de la metaclase *«ProcessElement»* es la primera en el modelo de definición del proceso. Para ello, se comprueba si tiene un enlace con la instancia de la metaclase *«ControlElement*» cuando ésta es de tipo *«InitialElement»*, siendo esta última además el elemento origen de dicho enlace.

Además, es importante destacar que esta función auxiliar ha sido definida a nivel de la metaclase *«ProcessElement»* con el propósito de que sea accesible desde cualquiera de las funciones de mapeo con las que transformar sus subtipos.

Expresión V-7. Función QVT auxiliar *«isInittialActivity»*

```
query SPDefinitionMetamodel::ProcessElement::isInittialActivity () : Boolean {
    return self.incoming→exist (obj | obj.oclIsTypeOf (ControlElement) and obj.type = InitialElement);
}
```

Además de la función auxiliar anterior, la regla de mapeo de una instancia de la metaclase *«HumanActivity»* utiliza las funciones auxiliares *«createPreConditions»* y *«createPosConditions»* para crear en la instancia *«HumanExecutionClass»* las precondiciones y postcondiciones necesarias para su ejecución, respectivamente. Tanto las precondiciones como las postcondiciones se modelan con la metaclase *«Constraint»* del metamodelo de ejecución y orquestación del proceso.

Respecto a las precondiciones, éstas establecen condiciones indispensables para comenzar la ejecución de una instancia de la metaclase *«ExecutionNodeClass»* – la cual, puede venir representada por alguno de sus subtipos, como es el caso de la metaclase *«HumanExecutionClass»* –. Concretamente, para poder comenzar dicha ejecución, todas las instancias *«ExecutionNodeClass»* anteriores – y directamente enlazadas – deben haber sido completadas. Para comprobar que una instancia de la metaclase *«ExecutionNodeClass»* ha sido completada basta con crear una *«Constraint»* que evalúe si su estado interno es *«Completed»* – para más información sobre el ciclo de vida de la metaclase *«ExecutionNodeClass»* consultar la Sección 3.2.1 del Capítulo IV.

Durante la transformación del modelo de definición del proceso y la generación de las precondiciones, puede darse el caso de que esté involucrada una instancia de la metaclase *«ControlElement»*. Tal y como se justifica en la Tabla V.1 (Sección 2.1), las instancias *«ControlElement»* de tipo *«Conditional»*, *«Fork»* o *«Join»* se traducen en instancias de la metaclase *«Constraint»*. Para una mejor comprensión de la regla QVT asociada, es importante tener en cuenta el procedimiento a seguir y para explicarlo nos basaremos en el ejemplo mostrado en la Figura V.5. La representación de este ejemplo ha sido simplificada para facilitar su propia representación, comprensión y explicación.

En la parte izquierda de la Figura V.5 se especifica el modelo de definición del proceso *«Proceso $_1$»*. Como se puede observar, este proceso está compuesto de cinco instancias de la metaclase *«HumanActivity»* y cinco instancias de la metaclase *«ControlElement»*. Estas últimas actúan como *«InitialElement»*, *«Fork»*, *«Conditional»*, *«Join»* y *«FinalElement»*, respectivamente.

En este modelo interviene también la variable *«BusinessVar1».* Esta variable es consultada por el elemento *«Conditional»* para comprobar si su valor es igual a *«compValue»* – tanto el operador lógico utilizado en la comparación como el propio valor con el que se compara la variable, son atributos que vienen informados en la instancia de

«checkBusinessVar» –. Además, se ha indicado que el valor de esta variable sea actualizado por la instancia *«HumanActivity $_5$»* cuando ésta haya sido completada. El nuevo valor es *«value2»* y vendrá dado por el atributo *«new_value»* de la instancia *«updateBusinessVar»*.

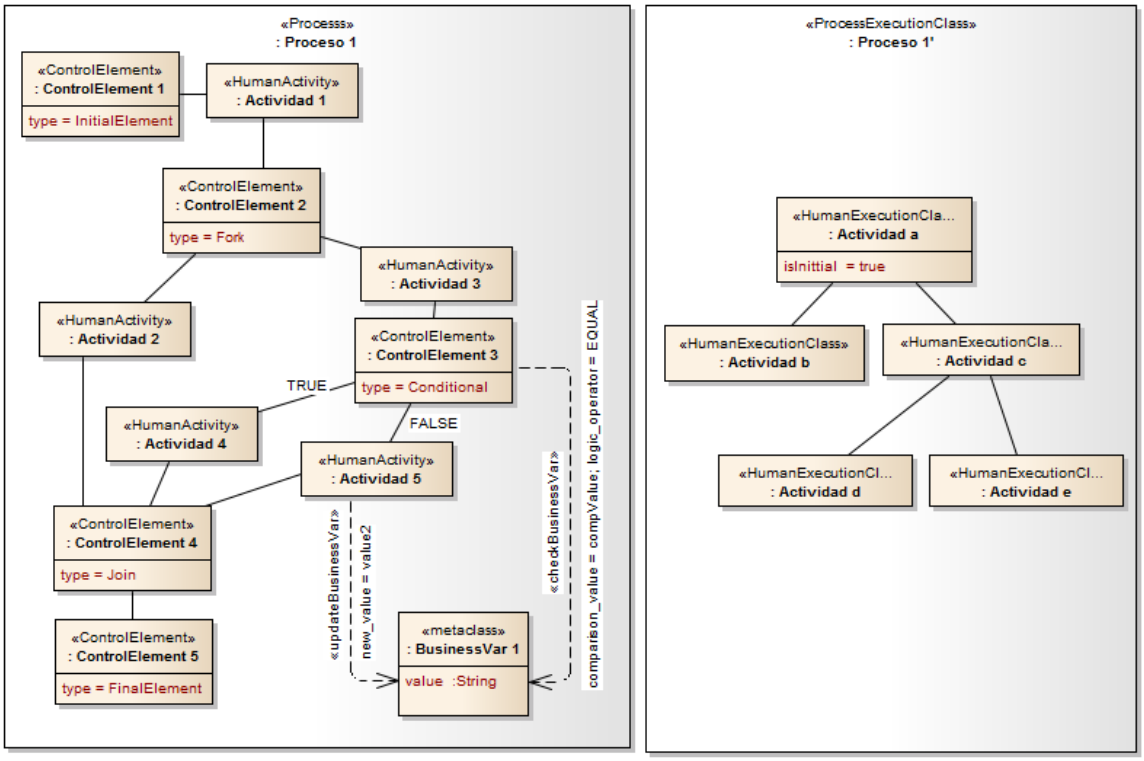


Figura V.5. Ejemplo ilustrativo con el que mostrar la generación de *«Constraint»*

En la parte derecha de la Figura V.5 se representa el modelo de ejecución y orquestación asociado al proceso *«Proceso $_1$»*. Desde el modelo de definición del proceso, cada instancia de la metaclase *«HumanActivity»* se transforma en una instancia de la metaclase *«HumanExecutionClass»* en el modelo de ejecución y orquestación y, además, se crean las siguientes precondiciones y postcondiciones. Éstas son, en lenguaje natural, las siguientes:

- *«Actividad $_a$»* (generada a partir de «*Actividad $_1$*»): no requiere precondiciones ni postcondiciones.
- *«Actividad $_b$»* (generada a partir de «*Actividad $_2$*»):
  - *«Constraint $_{b.1.PRE}$»*: *«Actividad $_a$»* debe haber sido completada.
- *«Actividad $_c$»* (generada a partir de «*Actividad $_3$*»):
  - *«Constraint $_{c.1.PRE}$»*: *«Actividad $_a$»* debe haber sido completada.
- *«Actividad $_d$»* (generada a partir de «*Actividad $_4$*»):
  - *«Constraint $_{d.1.PRE}$»*: *«Actividad $_c$»* debe haber sido completada.
  - *«Constraint $_{d.2.PRE}$»*: El valor de la variable debe ser igual a *«compValue»*.

— *«Actividad $_e$»* (generada a partir de «*Actividad $_5$*»):

- *«Constraint $_{e.1.PRE}$»*: *«Actividad $_c$»* debe haber sido completada.
- *«Constraint $_{e.2.PRE}$»*: El valor de la variable no debe ser igual a *«compValue»*.
- *«Constraint $_{e.3.POST}$»*: El valor de la variable debe cambiar a *«value2»*.

Como se puede apreciar, las instancias de la metaclase *«ControlElement»* no tienen una traducción explícita en forma de *«Constraint»* en el modelo de ejecución y orquestación.

Realmente, la generación de una *«Constraint»* se realiza teniendo en cuenta qué elemento del modelo de definición del proceso esta relacionado mediante un enlace de entrada hacia el *«ControlElement»* de tal manera que se crean tantas precondiciones como elementos *«OrchestrationActivity»*, *«HumanActivity»*, y *«ComplexActivity»* participen en un enlace de entrada hacia el *«ControlElement»* en cuestión.

Explicado el procedimiento general en base al ejemplo anterior, resulta más sencillo comprender la función auxiliar *«createPreConditions»* descrita en la Expresión V-8. Resulta conveniente destacar un par de aspectos.

En primer lugar, esta función ha sido definida a nivel de la metaclase *«Activity»* con el propósito de que sea accesible desde cualquiera de sus subtipos, y en segundo lugar, se ha utilizado una segunda función auxiliar – denominada *«createPreCondsREC»* – para realizar un tratamiento recursivo sobre los enlaces de entrada de una instancia de la metaclase *«ControlElement»* hasta encontrar algunos de los tipos indicados anteriormente, es decir, elementos *«OrchestrationActivity»*, *«HumanActivity»*, o *«ComplexActivity»*.

Expresión V-8. Función QVT para crear precondiciones a partir de *«ControlElement»*

```
query SPDefinitionMetamodel::Activity::createPreConditions () : Set(SPExecutionMetamodel::Constraint)     {
    var preConditionSet  : Set (SPExecutionMetamodel::Constraint);
    var preCondition       : SPExecutionMetamodel::Constraint;

    self.oclAsType(ProcessElement).incoming→forEach(link : Link) {
        preCondition := object Constraint {     type := “Pre-condition”;                logicOperator   := “equal”;
                                                comparisonValue := “Completed”;    comparedObject  := null;
                                                comparedVariable := null;    };
        switch {
          case (link.source.oclIsTypeOf (OrchestrationActivity))
             precondition.comparedObject :=
                                link.source.resolveone(SPExecutionMetamodel::ExecutionNodeClass);
          case (elem.oclIsTypeOf (HumanActivity))
             precondition.comparedObject  :=
                                link.source.resolveone(SPExecutionMetamodel::HumanExecutionClass);
          case (elem.oclIsTypeOf (ComplexActivity))
             precondition.comparedObject :=
                                link.source.Process.resolveone(SPExecutionMetamodel::ProcessExecutionClass);
          else
              // si no es ninguno de los casos anteriores, ‘link.source’ es de tipo  ‘ControlElement’
              preConditionSet += createPreCondsREC (elem);
        };
```

```
            preConditionSet += preCondition;
      }
      return preConditionSet;
}

query SPDefinitionMetamodel::ControlElement::createPreCondsREC () : Set(SPExecutionMetamodel::Constraint) {
      var preConditionSet  : Set (SPExecutionMetamodel::Constraint);
      var preCondition       : SPExecutionMetamodel::Constraint;

      self.oclAsType(ProcessElement).incoming→forEach(link) {
            preCondition := object Constraint {    type := "Pre-condition";                  logicOperator   := "equal";
                                                   comparisonValue := "Completed";    comparedObject  := null;  };
            switch {
              case (link.source.oclIsTypeOf (ControlElement))   preConditionSet += createPreCondsREC (elem);
              else  precondition.comparedObject := link.source.resolveone(SPExecutionMetamodel::Activity);
            };
            preConditionSet += precondition;
      }
      return preConditionSet;
}
```

Respecto a las postcondiciones y siguiendo el procedimiento descrito anteriormente, la Expresión V-9 describe formalmente la semántica de la operación *«createPosConditions»* con la que se crean las instancias de la metaclase *«Constraint»* que actúan como postcondiciones en el modelo. Al igual que en el caso de las precondiciones, esta función ha sido definida a nivel de la metaclase *«Activity»* con el propósito de que sea accesible desde cualquiera de sus subtipos.

Expresión V-9. Función QVT para crear postcondiciones

```
query SPDefinitionMetamodel::Activity::createPosConditions () : Set(SPExecutionMetamodel::Constraint) {
      var postConditionSet  : Set (SPExecutionMetamodel::Constraint);
      var postCondition       : SPExecutionMetamodel::Constraint;

      self.updateBusinessVar→forEach(link : updateBusinessVar) {
            postCondition := object Constraint {
                  type := "Post-condition";              logicOperator   := "equal";
                  comparisonValue := link.value;       comparedObject  := null;
                  comparedVariable :=  link.BusinessVar.resolveone(SPExecutionMetamodel::BusinessVar); };
            postConditionSet  += postCondition;
      }
      return preConditionSet;
}
```

Por otra parte y retomando la descripción de la regla de transformación de la metaclase *«Activity»* (descrita en Expresión V-5), la siguiente función de mapeo es la función *«toExecutableProcessFromComplexActivity»*.

Esta regla se encarga de transformar la instancia *«Process»* que está relacionada con la instancia de la metaclase *«ComplexActivity»* – ambas del metamodelo de definición del proceso software – en una instancia de la metaclase *«ProcessExecutionClass»* del metamodelo de ejecución y orquestación del proceso. En este caso hay que tener en cuenta que la instancia *«Process»* debe ser transformada si no se ha sido transformada previamente. La Expresión V-10 recoge la especificación de esta regla.

Expresión V-10. Mapeo QVT de la metaclase *«ComplexActivity»* a *«ProcessExecutionClass»*

```
mapping SPDefinitionMetamodel::ComplexActivity::toExecutableProcessFromComplexActivity () :
        SPExecutionMetamodel::ProcessExecutionClass {
    var processExexClass: SPExecutionMetamodel::ProcessExecutionClass;

    processExexClass := self.Process.resolveone(SPExecutionMetamodel::ProcessExecutionClass);
    if (processExexClass == null) then
      self.Process→map toExecutableProcess ();
}
```

Finalmente, respecto a las últimas tres funciones de mapeo invocadas en la Expresión V-5, éstas tienen una definición afín. Estas funciones transforman una instancia de la metaclase *«OrchestrationActivity»* en función de su tipo. En este caso, se ha optado por especificar únicamente la regla de mapeo *«toScriptExeClass»* a modo de ejemplo. En cualquier caso, conforme se describa esta función se irá indicando cuáles son los aspectos que varían respecto a las otras dos reglas (*«toWSExeClass»* y *«toEMailExeClass»*).

Tal y como adelanta en la Tabla V.1 (Sección 2.1 de este capitulo), la metaclase *«OrchestrationActivity»* se debe transformar en una de las tres metaclases de orquestación que define el metamodelo de ejecución y orquestación (Sección 3 del Capítulo IV). La elección para realizar una u otra transformación radica en el valor de la propiedad *«type»* de la metaclase *«OrchestrationActivity»* (Sección 2.2.7 del Capítulo IV). A modo de recordatorio, la propiedad *«type»* establece cuál es el tipo de actividad de orquestación y sus posibles valores están controlados por el enumerado *«MachineActivityType»*, cuya lista de valores es: *«execute script»*, *«invoke service»*, y *«send message»*.

En el caso que nos ocupa, para transformar una instancia de la metaclase *«OrchestrationActivity»* en una instancia de la metaclase *«ScriptExecutionClass»* en el metamodelo de ejecución y orquestación, es necesario que se dé una condición a priori: la actividad de orquestación debe definir la ejecución de un script en la propiedad *«type»* de la metaclase *«OrchestrationActivity»*.

Desde el punto de vista de la regla de transformación QVT, la precondición anterior se controla mediante la directiva *«when»*. Esta directiva QVT permite condicionar la ejecución de una regla de mapeo de tal forma que sólo se activa para aquellos objetos que verifiquen las precondiciones indicadas. En el caso de las funciones de mapeo

«toWSExeClass» y *«toEMailExeClass»* en la precondición de la cláusula *«when»* se debe expresar que la instancia de la metaclase *«OrchestrationActivity»* debe ser de tipo *«invoke service»* o *«send message»*, respectivamente.

Finalmente, la Expresión V-11 recoge la especificación de la regla de transformación de la metaclase *«OrchestrationActivity»* cuando ésta refleja la ejecución de un script. Aunque tanto el código QVT de la función de mapeo como el de las funciones auxiliares invocadas está documentado, se describen brevemente para facilitar su comprensión.

Expresión V-11. Mapeo QVT de la metaclase *«OrchestrationActivity»* a *«ScriptExecutionClass»*

```
mapping SPDefinitionMetamodel::OrchestrationActivity::toScriptExeClass () :
        SPExecutionMetamodel::ScriptExecutionClass   when   { self.type = "execute script" }
{
    var varResultado : SPExecutionMetamodel::BusinessVar;

    // Se inicializan los atributos de la 'ScriptExecutionClass'
    response    := "NOK";          URI     := "";              description  := self.description;
    name        := self.name;      status  := "Unstarted";     isInittial   := isInittialActivity ();

    // Se crea la variable donde será almacenado el resultado tras la ejecución del script
    varResultado   := object BusinessVar {
                name  := "Result_Variable_OrchestrActv_" + self.id;      value := "";
              };
    resultVariable := varResultado;



    // La nueva variable creada debe ser asociada también con el proceso
    self.oclAsType (MachineExecutionClass).oclAsType (ExecutionNodeClass).
                                                    ProcessExecutionClass.defines += varResultado;
    PRE  := createPreConditions ();
    POST := createPosConditions ();
}
```

Respecto a la función expresada anteriormente, primero se inicializan todos los atributos de la nueva instancia de la metaclase *«ScriptExecutionClass»*. Este primer bloque de código es similar para las funciones de mapeo *«toWSExeClass»* y *«toEMailExeClass»*.

A continuación, se crea la variable de negocio donde será almacenado el resultado de la operación de ejecución del script. Este paso también es común para las funciones de mapeo *«toWSExeClass»* y *«toEMailExeClass»*. Esta nueva variable debe ser asociada también con el proceso al que pertenece la nueva instancia de la metaclase *«ScriptExecutionClass»* con el objetivo de que el modelo de ejecución y orquestación sea consistente y esté bien formado.

Finalmente, se crea el conjunto de condiciones que deben ser ciertas antes (precondiciones) y después (postcondiciones) de ejecutar la instancia de la metaclase *«ScriptExecutionClass»*. Para este propósito, se ha utilizado funciones auxiliares descritas anteriormente.

2.6. Especificación del mapeo de la metaclase *«Link»*

Tal y como se adelanta en la Sección 2.2, esta regla de mapeo se materializa con la función *«toExecutionFlow»*. La especificación de esta regla se proporciona en la Expresión V-12 y su objetivo es transformar cada instancia de la metaclase *«Link»* del metamodelo de definición del proceso software (metamodelo de entrada) en una instancia de la metaclase homóloga en el modelo de ejecución y orquestación (metamodelo de salida), es decir, la metaclase ***«****ExecutionFlow****»***.

Expresión V-12. Mapeo QVT de la metaclase *«Link»* a *«ExecutionFlow»*

```
mapping   SPDefinitionMetamodel::Link::toExecutionFlow : SPExecutionMetamodel::ExecutionFlow () {
      type  := "Normal";
      guard := ( if (0 = self.Guard.size ()) then "Default"  else self.Condition endif; );
      target := self.target.resolveone(SPExecutionMetamodel::ExecutionNodeClass);
      source := self.source.resolveone(SPExecutionMetamodel::ExecutionNodeClass);
}
```

2.7. Especificación del mapeo de la metaclase *«Stakeholder»*

La regla de mapeo que se encarga de transformar una instancia de la metaclase *«Stakeholder»* del metamodelo de definición del proceso en una instancia de la metaclase *«HumanRole»* del metamodelo de ejecución y orquestación es la función *«toHumanRole»*.

La Expresión V-13 describe la especificación de esta regla de mapeo. Resulta importante señalar que durante el mapeo de una instancia de la metaclase *«Stakeholder»* no es necesario no es necesario volver a resolver sus enlaces con otros elementos del proceso de que dichos enlaces ya han sido resueltos en funciones de mapeo anteriores.

Expresión V-13. Mapeo QVT de la metaclase *«Stakeholder»* a *«HumanRole»*

```
mapping   SPDefinitionMetamodel::Stakeholder::toHumanRole : SPExecutionMetamodel::HumanRole ()
{
      name  := self.name;       description := self.description;         position := self.position;
}
```

2.8. Especificación del mapeo de la metaclase *«Product»*

La regla de mapeo que se encarga de transformar una instancia de la metaclase *«Product»* del metamodelo de definición del proceso en una instancia de la metaclase *«WorkProduct»* del metamodelo de ejecución y orquestación es la función *«toWorkProduct»*.

La Expresión V-14 describe la especificación de esta regla de mapeo. Resulta importante señalar que durante el mapeo de una instancia de la metaclase *«Product»* no es

necesario volver a resolver sus enlaces con otros elementos del proceso de que dichos enlaces ya han sido resueltos en funciones de mapeo anteriores.

Expresión V-14. Mapeo QVT de la metaclase *«Product»* a *«WorkProduct»*

```
mapping SPDefinitionMetamodel::Product::toWorkProduct : SPExecutionMetamodel::WorkProduct ()
{
    completeness := 0;      lastTimeModified := "";      version := 0;
    resourceURI := "";      name := self.name;           description := self.description;
    type := self.type;
}
```

3. Generación del modelo final de ejecución y orquestación

Como se ha comentado a lo largo de la sección anterior, las relaciones entre el modelo de definición del proceso software y su modelo de ejecución y orquestación posibilita la obtención de éste (en su versión básica) desde aquél.

Este proceso sistemático de transformación ha sido implementado en una herramienta CASE denominada PLM$_4$BS, herramienta estudiada en profundidad en el siguiente capítulo. Esta herramienta automatiza todo este proceso sistemático generando una versión del modelo que es cercana a su versión final. Sin embargo, es necesario revisar algunos aspectos del modelo básico de ejecución y orquestación para adecuarlos lo máximo posible a la realidad del sistema. A lo largo de este apartado, se presentan los aspectos que hay que estudiar y modificar para conseguir el modelo final de ejecución y orquestación del proceso.

Para obtener el modelo final de ejecución y orquestación es necesario que el ingeniero de procesos proporcione información relativa al contexto de ejecución, es decir, la información sobre puertos de comunicaciones y direcciones para el acceso a servicios webs en el caso de una actividad de la invocación de servicios, dirección o URI del script a ejecutar en caso de una actividad de ejecución de scripts, etc.

Mientras que el proceso de derivación es totalmente sistemático, la construcción del modelo final no lo es y de hecho, se requiere la participación de un ingeniero de procesos o analista.

Sin embargo, lo que sí se puede normalizar son las revisiones que se deben realizar y los cambios que éstas puedan ocasionar, así como las consecuencias que esos cambios pueden tener en el modelo de definición del proceso software. A lo largo de este apartado, se estudian las revisiones que debería hacer el ingeniero de procesos o analista tras conseguir el modelo básico de ejecución y orquestación del proceso. En cada revisión se catalogan los posibles cambios que se podrían plantear y las repercusiones que éstos podrían tener.

3.1. Revisión de los nodos de ejecución del proceso

El ingeniero de procesos o analista debe revisar los nodos generados en la estructura del modelo básico de ejecución y orquestación que se han obtenido para, de esta forma, comprobar que dicha estructura se adecúa a la realidad final de ejecución cuando el proceso se despliegue en el sistema real. Durante esta revisión se puede presentar la necesidad de realizar los siguientes cambios:

1. **Añadir un nuevo nodo de ejecución**. Si el ingeniero de procesos detecta la necesidad de añadir un nuevo nodo en la estructura de ejecución. Los nodos del modelo de ejecución y orquestación provienen de elementos que conforman la estructura de definición del proceso. En este caso, por tanto, hay que analizar por qué no ha sido contemplado dicho elemento en el modelo de definición del proceso y una vez encontrada la justificación, modificar el modelo de definición para volver a generar el nodo de ejecución en cuestión.
2. **Borrar un nodo de ejecución**. Igual que en el cambio anterior, cuando se detecta esta necesidad hay que prescindir de un nodo del modelo de ejecución y orquestación, lo que se está cambiando realmente es la definición del proceso que se definió durante la etapa anterior, por lo que es necesario revisar dicho modelo para eliminar aquel elemento del que surge el nodo de ejecución a eliminar.

3.2. Revisión de los roles y productos del proceso

Al igual que los nodos de la estructura de ejecución y orquestación del proceso, el ingeniero de procesos o analista debe revisar que, tanto los roles como los productos incluidos en el modelo básico de ejecución son realmente necesarios y adecuados para la ejecución final del proceso. Durante esta revisión se puede presentar la necesidad de realizar los siguientes cambios:

1. **Añadir un nuevo rol/producto**. Si se detecta la necesidad de añadir un nuevo rol/producto en el modelo de ejecución y orquestación, hay que analizar por qué no ha sido contemplado dicho elemento en el modelo de definición del proceso y una vez encontrada la justificación, modificar el modelo de definición para volver a generar el elemento en cuestión.
2. **Borrar un rol/producto**. Igual que en el cambio anterior, cuando se detecta esta necesidad hay que prescindir del rol/producto en el modelo de ejecución. Sin embargo, lo que se está cambiando realmente es la definición del proceso que se definió durante la etapa anterior. Por este motivo es necesario revisar dicho modelo para eliminar aquel elemento del que surge el rol/producto a eliminar.

3.3. Revisión de las restricciones de los nodos de ejecución del modelo básico

Otro aspecto relevante que puede suscitar modificaciones durante la revisión del modelo básico de ejecución y orquestación es la existencia de restricciones (precondiciones y postcondiciones) en los nodos de ejecución del proceso. Durante esta tarea de revisión se pueden dar los siguientes cambios:

1. **Añadir una nueva restricción en un nodo de ejecución**. En el caso de que surja la necesidad de añadir una nueva restricción (metaclase *«Constraint»* en el metamodelo de ejecución y orquestación) hay que tener en cuenta dos posibles situaciones:

   (a) **La nueva restricción implica la consulta o actualización de una variable**. En el caso de que surja la necesidad de añadir una restricción de tipo precondición en la que se evalúa una variable no contemplada hasta ahora, es conveniente reconsiderar y estudiar la definición del proceso y más concretamente, el uso del enlace *«checkBusinessVar»* entre el nodo *«Conditional»* (que precede al elemento al que se desea añadir la restricción) y una variable del proceso o *«BusinessVar»*. En caso de que la restricción sea de tipo postcondición habrá que considerar y estudiar si el elemento del modelo de definición tiene los enlaces *«updateBusinessVar»* adecuados con las variables necesarias.

   (b) **La nueva restricción implica la consulta del estado interno de un nodo de ejecución**. En este caso, añadir una nueva restricción de este tipo implica reconsiderar el flujo de elementos establecido en el modelo de definición del proceso. Según se comentó en la Tabla V.1 de correspondencia entre elementos del metamodelo de definición y el metamodelo de ejecución y orquestación, este tipo de restricciones están relacionadas con los nodos de control del proceso que son de tipo *«Join»*, *«Fork»* y *«Conditional»*. Por tanto, la adición de una nueva restricción implica reconducir el flujo del proceso en el modelo de definición y, hecho lo cual, volver a generar el modelo básico para tener en cuenta la nueva restricción.

2. **Borrar una restricción en un nodo de ejecución**. En el caso de que surja la necesidad de borrar una restricción en un nodo de ejecución, el modus operandi es análogo al punto anterior en el sentido de que su eliminación implica la reconsideración y el estudio del modelo con la definición del proceso para estudiar los enlaces *«updateBusinessVar»* y *«checkBusinessVar»* de los elementos involucrados.

3.4. Revisión de las asociaciones en el modelo básico

El ingeniero de procesos o el analista también debe realizar la revisión de las asociaciones generadas en el modelo básico de ejecución y orquestación. Durante esta tarea puede surgir la necesidad de realizar los siguientes cambios:

1. **Añadir o borrar asociaciones en las que interviene productos**. Durante la revisión del modelo básico de ejecución y orquestación es posible que surja la necesidad de modificar (añadir o borrar) los productos de entrada, salida o entrada/salida de un nodo de ejecución del proceso, es decir, cambiar las asociaciones entre productos y nodos de ejecución. En este caso, esta modificación implicaría modificar el modelo de definición de procesos para volver a generar o eliminar el enlace en cuestión.

2. **Añadir o borrar asociaciones en las que interviene roles**. Al igual que el caso anterior, la modificación (adición o eliminación) de participantes o responsable de un nodo de ejecución a desempeñar por un humano implica cambiar el modelo con la definición del proceso para, en última instancia, derivar la modificación en cuestión en el modelo de ejecución y orquestación.

3.5. Revisión de nombres y descripciones de los elementos del proceso

Todos los nombres y descripciones del modelo básico de ejecución y orquestación, se derivan de manera sistemática desde el modelo de definición del proceso tal y como se ha descrito en la sección anterior de este capítulo. Sin embargo, las descripciones y nombres, tanto de los nodos de ejecución como de los roles y productos, pueden ser modificados por el analista ingeniero de procesos para que sean más completas y descriptivas en un contexto de ejecución. Cualquier cambio en estos campos puede hacerse con total libertad pues no afecta a la relación o vínculo que se establece entre elementos de ambos modelos tras el proceso de derivación sistemática. Esta trazabilidad se garantiza gracias al uso de los identificadores únicos de cada elemento.

Para finalizar la sección, a modo de resumen, la Tabla V.2 recoge los pasos a realizar, los cambios y si afecta a o no al modelado del proceso. En cualquier caso, es necesario recalcar que cualquier cambio que se produzca en el resultado de la etapa de definición anterior, debe realizarse con la aceptación del mismo por parte del grupo de clientes y de usuarios.

La simbología usada en la tabla tiene el siguiente significado: ×, el cambio no afecta al modelo de definición del proceso; y √, el cambio afecta al modelo de la definición del proceso.

Tabla V.2. Cambios en el modelo de ejecución y orquestación, y repercusión

Paso	Cambios		Afecta a la etapa de definición del proceso
1	Añadir un nuevo nodo de ejecución.		√
	Borrar un nodo de ejecución.		√
2	Añadir un nuevo rol/producto.		√
	Borrar un rol/producto.		√
3	Añadir una nueva restricción en un nodo de ejecución.	Implica la consulta del valor de una variable.	√
		Implica implica la consulta del estado interno de un nodo de ejecución.	√
	Borrar una restricción en un nodo de ejecución.		√
4	Añadir o borrar asociaciones en las que interviene productos.		√
	Añadir o borrar asociaciones en las que interviene roles.		√
5	Cambios en nombres y descripciones		×

4. Generación del código ejecutable desde el modelo de ejecución y orquestación del proceso software

En el capítulo anterior se ha definido un lenguaje basado en UML para definir el contexto de ejecución y orquestación del proceso software. Como se ha argumentado a lo largo del capítulo anterior, UML proporciona compresibilidad y expresividad, ambas basadas en su representación gráfica y sus constructores de alto nivel para la extensión del propio lenguaje UML.

UML también proporciona conceptos con semántica de ejecución tales como sus metaclases *«Activity»* y *«Action»*. Sin embargo, actualmente existe una laguna tecnológica que imposibilita la ejecución de lenguajes de procesos basados en UML (como el que se ha definido en esta tesis) debido a que no existen máquinas virtuales, intérpretes o compiladores UML.

Por este motivo, para poder efectuar una transferencia del marco teórico presentado en esta tesis hacia entornos reales y de producción, es necesario buscar mecanismos alternativos que permitan ejecutar los modelos que instancien el metamodelo de ejecución y orquestación del proceso software visto en el Capítulo IV.

Esta sección tiene como propósito definir cuál es el mecanismo elegido, así como definir el contexto y planteamiento necesario para establecer las reglas de derivación oportunas, que permitan ejecutar el modelo de ejecución y orquestación en un motor de procesos convencional.

4.1. Contexto y planteamiento previo

Actualmente, existen normas y estándares que definen lenguajes de ejecución con los que es posible describir procesos de negocio para su ejecución en herramientas BPMS (*«Business Process Management Suite»*). Las normas más importantes en este ámbito son: BPML (*«Business Process Modeling Language»*) [Havey 2005] y WS-BPEL [OASIS 2007].

BPML fue propuesta por la iniciativa BPMI (*«Business Process Management Initiative»*) y es un metalenguaje basado en XML que describe la representación estructural de un proceso de negocio y la semántica de su ejecución, apoyándose para ello en el concepto de máquina transaccional de estados finitos.

A pesar de que BPML es un lenguaje estándar completo y formal para la ejecución de procesos de negocio, fue una línea de trabajo abandonada después de que su organización fundadora, BPMI, se fusionara con la OMG en 2005. Este abandono fue a favor de WS-BPEL estandarizado por la *«Organization for the Advancement of Structured Information Standards (OASIS)»*.

WS-BPEL es en la actualidad el estándar de ejecución de mayor influencia en el mercado y el estándar de facto en la industria software para la orquestación de procesos mediante el uso de servicios web de manera combinada con actividades del flujo de control de un proceso [Mateo *et al.* 2012; Zeng *et al.* 2004]. Está basado en XML y permite la especificación de los procesos de negocio utilizando WSDL (*«Web Service Definition Language»*) de tal forma que es posible definir la forma en que el proceso de negocio se

construye a partir de las invocaciones de servicios web existentes y la interacción del proceso con otros participantes externos.

Sin embargo, a pesar de esta potencia, el lenguaje WS-BPEL tiene un problema: no proporciona soporte para interacciones humanas durante el proceso. Este aspecto es un handicap importante para muchos procesos de negocio del mundo real. En el caso particular de las organizaciones software este hecho es una limitación importante ya que el factor humano es clave y esencial en la ejecución de los procesos software – por ejemplo, el factor humano es esencial para la toma de decisiones en la gestión de proyectos, en las tareas de revisión técnica llevadas a cabo por miembros de oficinas técnicas de calidad y en tareas de codificación, entre otros.

Para llenar este vacío, la especificación BPEL4People (*«WS-BPEL Extension for People»*) [OASIS 2010] extiende WS-BPEL tomando como base la especificación WS-HumanTask [OASIS 2012; Russell *et al.* 2007] para tener en cuenta la orquestación de actividades humanas dentro de la especificación del contexto de ejecución de procesos de negocio.

De los estándares de ejecución de procesos de negocio mencionados, WS-BPEL (con su extensión BPEL4People) es el más soportado por la mayoría de motores de ejecución de procesos (por ejemplo, IBM Websphere, Suite BPM de Oracle, jBPM, BonitaOS y Activiti, entre otros). Además, WS-BPEL es un lenguaje ampliamente utilizado y reconocido por el sector empresarial, y además proporciona soporte para la definición de tareas que deben ser desempeñadas por roles humanos.

Éstas son las razones por las que se ha optado por utilizar WS-BPEL (con su extensión BPEL4People) como lenguaje de ejecución hacia el que derivar el metamodelo de ejecución y orquestación definido en el Capítulo IV.

Además, para conseguir que la propuesta PLM_4BS sea ejecutable, se ha decidido explorar la posibilidad de combinar PLM_4BS con WS-BPEL y su extensión BPEL4People. De esta manera, se utiliza por una parte PLM_4BS para la definición de alto nivel del proceso software y su contexto de ejecución, y por otra, se aprovecha las ventajas y la trayectoria de un lenguaje de ejecución consolidado en el sector empresarial para la ejecución de procesos.

Sin embargo, es importante aclarar que durante la transformación del metamodelo de ejecución y orquestación hacia WS-BPEL y BPEL4People, es posible que se pierda parte de la potencia semántica de aquél.

Una vez concretado el lenguaje de ejecución hacia el que derivar el modelo de ejecución y orquestación y como paso previo a establecer cuál es el mapeo entre ambos lenguajes, para posteriormente especificar las reglas de transformación MOFM2T necesarias, resulta conveniente e interesante mencionar algunos trabajos relacionados que proponen un mapeo entre UML y WS-BPEL como [Korherr *et al.* 2006; Faleh *et al.* 2011; Bendoukha *et al.* 2012] entre otros.

La Tabla V.3 muestra la correspondencia entre los elementos del metamodelo de ejecución y orquestación de la propuesta PLM_4BS, y WS-BPEL/BPEL4People – junto con

unos comentarios preliminares – como resumen previo para afrontar las reglas de derivación MOFM2T expuestas a lo largo de esta sección.

Tabla V.3. Correspondencia entre el metamodelo de ejecución y orquestación con WS-BPEL y BPEL4People

Elemento metamodelo ejecución y orquestación	Etiqueta XML de BPEL4People/WS-BPEL		Comentario
«ProcessExecutionClass»	(a)	*«bpel:process»*	A la hora de mapear esta metaclase a WS-BPEL hay que tener en cuenta dos posibles situaciones. La primera, (a), hace referencia a la transformación de un proceso de negocio hacia WS-BPEL. La segunda, (b), hace referencia a cómo se mapea en WS-BPEL la invocación de un proceso desde otro proceso distinto. Este último caso sólo es posible si el proceso invocado ya está definido como un servicio web (por ejemplo, en forma de servicio WSDL). La etiqueta *«bpel:partnerLink»* permite especificar la interacción del proceso con otros servicios webs. En cualquier caso, será necesaria la intervención manual de ingeniero de procesos o analista para completar toda la información de este servicio web (URL, operación, puerto).
	(b)	*«bpel:partnerLink»* & *«bpel:invoke»*	
«ExecutionNodeClass»	—		La traducción de esta metaclase hacia WS-BPEL se circunscribe a sus metaclases específicas.
«WSExecutionClass»	*«bpel:partnerLink»* & *«bpel:invoke»*		En este caso, y al igual que en la situación (b) de la metaclase *«ProcessExecutionClass»*, la traducción de la metaclase *«WSExecutionClass»* pasa por utilizar las etiquetas WS-BPEL mencionadas.
«EMailExecutionClass»	*«bpel:partnerLink»* & *«bpel:invoke»*		El lenguaje WS-BPEL no proporciona mecanismos propios para la notificación de mensajes por e-mail. Como alternativa de solución se propone que la metaclase *«EMailExecutionClass»* sea mapeada como la invocación a un servicio de mensajería, el cual, habría que desarrollar a posteriori.
«ScriptExecutionClass»	*«bpel:empty»*		El lenguaje WS-BPEL no proporciona mecanismos propios para la ejecución de scripts. Sin embargo, algunas soluciones técnicas proporcionan su solución particular. Por ejemplo, Oracle introduce la etiqueta *«bpelx:exec»* [Oracle 2014] en su motor BPEL de ejecución de procesos. Como solución temporal, en esta tesis se proponen introducir un marcador en el código WS-BPEL en base a la etiqueta *«empty»*. Sin embargo se plantea como trabajo futuro, utiliza el mecanismo de extensión que proporciona WS-BPEL para crear un nuevo tipo de actividad genérica con la que mapear la ejecución de un script.

«HumanExecutionClass»	*«b4p:peopleActivity»*	En el caso de que se hayan definido actividades humanas en el proceso, se utilizará la extensión *«b4p:peopleActivity»* que define BPEL4People sobre WS-BPEL para realizar el mapeo.
«Constraint»	*«condition»*	Se utiliza la etiqueta *«condition»* de BPEL para mapear las precondiciones y postcondiciones de cada nodo ejecutable del proceso.
«HumanRole»	*«htd:logicalPeopleGroup»* *«b4p:humanInteractions»* *«b4p:peopleAssignments»*	La etiqueta *«htd:logicalPeopleGroup»* proviene del espacio de nombres de la especificación WS-HumanTask que como se comenta al inicio de la Sección 3.1, es utilizada por la propia especificación BPEL4People. Esta etiqueta se utiliza en combinación con la etiqueta *«b4p:humanInteractions»* del lenguaje BPEL4People y específica grupos lógicos de roles y se utilizan en actividades humanas. Además, los roles del proceso se transforma dentro de la etiqueta *«b4p:peopleAssignments»* para definir qué puede hacer cada grupo de rol sobre procesos: desde iniciarlo hasta administrarlo.
«BusinessVar»	*«bpel:variable»*	—
«WorkProduct»	*«bpel:variable»*	BPEL no proporciona ningún elemento sobre el que mapear la metaclase *«WorkProduct»*. Por este motivo se opta por emplear la etiqueta *«variable»* BPEL y utilizar su atributo *«MessageType»* para especificar el tipo de producto. Este tipo de producto debe haber sido definido en un fichero XML Schema [W3C 2005] y referenciado en el código BPEL.
«ExecutionFlow»	—	La traducción de esta metaclase hacia WS-BPEL se circunscribe del flujo de control del proceso. Respecto a la estructura del flujo de ejecución de procesos, ésta será mapeadas en estructuras análogas del lenguaje WS-BPEL (utilizando, por ejemplo, las etiquetas *«sequence»* para definir secuencias de actividades o *«flow»*, para definir actividades concurrentes).

Durante la elaboración de la tabla de mapeo anterior, ha sido posible detectar ciertos aspectos relevantes. La mayoría de ellos están relacionados con el hecho de que muchos de los elementos contemplados en el metamodelo de ejecución y orquestación – cuya semántica es adecuada y válida para el contexto de los procesos software – no tienen una traslación equivalente en WS-BPEL. Por ejemplo, la metaclase *«WorkProduct»* ha sido mapeada como una variable debido a la inexistencia de elementos equivalentes en el lenguaje WS-BPEL. Una situación similar sucede con la metaclase *«ScriptExecutionClass»*.

Además, por una parte, todos los elementos relativos a la coordinación de actividades son fácilmente mapeado en elementos WS-BPEL. Este aspecto en cierta medida proporciona soporte de aplicabilidad de nuestra propuesta ya que PLM_4BS se atisba como una propuesta para definir procesos software y su contexto de ejecución y orquestación desde una posición de alto nivel, y a la vez, su combinación con WS-BPEL proporciona la ejecutablilidad necesaria del proceso.

En cualquier caso, tal y como se ha adelantado en la Sección 1.1 de este capítulo y se representa en la Figura V.2, es importante mencionar que debido a las limitaciones de WS-

BPEL para representar toda la semántica del metamodelo de ejecución y orquestación, se hace necesaria la intervención manual del ingeniero de procesos o analista para completar todos los aspectos técnicos necesarios y obtener de esta manera la versión final del código WS-BPEL. Todo ello con el fin de desplegar y ejecutar dicho código en un motor de procesos.

Una vez planteada la tabla de correspondencia anterior, las siguientes subsecciones especifican las principales reglas MOFM2T necesarias para llevar a cabo las transformaciones *«model-to-text»* indicadas en la Tabla V.3.

4.2. Especificación de la transformación *«PLM4BS_M2TTransformation»*

Esta sección específica la última transformación – transformación estereotipada con *«PLM4BS_M2TTransformation»* en la Figura V.1 – del procedimiento sistemático de derivación propuesto en esta tesis. Esta transformación es la encargada de generar código ejecutable a partir del modelo de ejecución y orquestación del proceso software.

Esta transformación toma como entrada una instancia de la metaclase *«ProcessExecutionClass»* y a partir de ella transforma todos los elementos vistos en el metamodelo de ejecución y orquestación para generar una versión textual y ejecutable del mismo, según el lenguaje BPEL4People (y por extensión también WS-BPEL).

La Expresión V-15 especifica la estructura de esta transformación en lenguaje MOFM2T. Esta estructura está basada en la directiva *«template»* de MOFM2T con la que este lenguaje especifica plantillas de texto con marcadores clave de localización donde situar los datos que se extraen de forma directa desde los modelos. Estos marcadores son, en esencia, expresiones especificadas sobre las entidades del metamodelo a través de un mecanismo basado en consultas OCL para la selección y la extracción de modelos.

Un aspecto relevante de MOFM2T es que permite invocar *«template»* dentro de otras, permitiendo de esta manera modularizar las reglas de transformación.

Finalmente, para facilitar la comprensión de las reglas MOFM2T especificadas, es necesario comentar que las expresiones de este lenguaje se enmarcan con los delimitadores corchetes, es decir, *«[»* y *«]»*. El resto de expresiones que aparecen en el código de la regla de transformación son expresiones que aparecerán imprimidas directamente en el texto de salida, es decir, en el código WS-BPEL.

Expresión V-15. Transformación MOFM2T de *«ProcessExecutionclass»* a código BPEL4People

```
[module PLM4BS_M2TTransformation (SPExecutionMetamodel)/]
  [template public processExeClassToBPEL4People (process : ProcessExecutionClass)]
  <?xml version="1.0" encoding="UTF-8"?>
  <process name="[process.name /]" targetNamespace="www.iwt2.org" [printXMLNS() /] >

    <!-- Se añaden extensiones para tratamiento actividades humanas ->
    <extensions>
      <extension namespace="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803"
                 mustUnderstand="yes"/>
      <extension namespace="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
                 mustUnderstand="yes"/>
```

```
        </extensions>
        <documentation> [process.description /] </documentation>

        <!-- Se añaden los marcadores para la orquestación del proceso: invocación a otros procesos,
        invocación a servicios y mensajería por e-mail ->
        [ProcessExecutionClassToBPEL (process) /]
        [WSExecutionClassToBPEL      (process) /]
        [EMailExecutionClassToBPEL   (process) /]

        <!-- Se crean las variables del proceso ->
        <variables>
                [BusinessVarToBPEL (process) /]
                [WorkProductToBPEL (process) /]
        </variables>

        <!-- Se transforma los roles que intervienen en el proceso ->
        [humanRoleToBPEL4People (process) /]

        <!-- Se define la secuencia de actividades según la estructura del proceso. En esta parte del código se
        hará referencia a elementos definidos en las secciones partnerLinks,variables, y humanInteractions ->
        <sequence>
            [defineBPELStructure (process) /]
        </sequence>
    </process>
    [/template]
[/module]
```

Tal y como es posible apreciar, la transformación anterior toma como entrada una instancia de la metaclase *«ProcessExecutionClass»* del metamodelo de ejecución y orquestación del proceso software para, a partir de ella, derivar código ejecutable conforme a los lenguajes WS-BPEL y BPEL4People. Para alcanzar satisfactoriamente este objetivo y por motivos de legibilidad del código, la regla de transformación definida en la Expresión V-15 invoca las siguientes plantillas auxiliares:

- Plantilla *«printXMLNS»*. Esta plantilla no transforma ningún elemento del metamodelo de ejecución y orquestación sino que se encarga de imprimir todo el código BPEL4People asociado a los espacios de nombres[5], «*namespaces*», referenciados por el propio lenguaje BPEL4People. Entre los espacios de nombres requeridos por BPEL4People se encuentran el de los estándares WS-BPEL, WS-HumanTask, y XML Schema [W3C 2005], entre otros.
- Plantillas *«ProcessExecutionClassToBPEL»*, *«WSExecutionClassToBPEL»* y *«EMailExecutionClassToBPEL»*. Estas plantillas se encargan de imprimir diferentes marcadores para definir una interfaz con la que llevar a cabo la interacción y orquestación del proceso con otros servicios; concretamente, llamadas a otros procesos (el cual debe ser ofertado como servicio web) invocados desde el primero, invocación de servicios externos, y notificación por

[5] Un espacio de nombres – dentro del contexto de código XML – se utiliza para proporcionar atributos y nombres de elementos únicos dentro de un documento XML.

e-mail, respectivamente. El código generado de aplicar estas plantillas se enmarca dentro de la etiqueta *«partnerLinks»* de WS-BPEL.

- Plantillas *«BusinessVarToBPEL»* y *«WorkProductToBPEL»*. Se encargan de traducir, respectivamente, todas las instancias de las metaclases *«BusinessVar» «WorkProduct»* a código WS-BPEL según se indica en la Tabla V.3. El código asociado a estas instancias está englobado en la sección *«variables»* de WS-BPEL.
- Plantilla *«humanRoleToBPEL4People»*. Se encarga de imprimir todo el código BPEL4People asociado con los roles de usuario que intervienen en actividades humanas del proceso. El código asociado a estas transformaciones se enmarca dentro de la sección *«b4p:humanInteractions»* de BPEL4People, cuyo propósito es agrupar la especificación de grupos lógicos de roles.
- Plantilla *«defineBPELStructure»*. Esta plantilla se encarga de imprimir el código final con la estructura y secuencia de actividades WS-BPEL que conforman el modelo de ejecución y orquestación. En paralelo conforme se construye esta secuencia, la plantilla también transforma las instancias de la metaclase *«Constraint»*.

Una vez indicada las plantillas que se utilizan para generar el código WS-BPEL final pasamos a describir un par de ellas en los siguientes apartados. No se han definido todas y cada una de las reglas de mapeo porque no aporta valor al contenido de esta tesis ya que en su mayoría resultan análogas y algunas de ellas son incluso triviales, como es el caso de la plantilla *«printXMLNS»* antes mencionada.

4.3. Especificación del espacio de nombres

La especificación de un espacio de nombres proporciona un contexto semántico para el fichero XML (como es el caso del código WS-BPEL) en el que se define. Esta especificación es una de las recomendaciones de la W3C para proporcionar elementos y atributos con nombre único en un archivo con una estructura XML. Además, un archivo XML puede contener nombres de elementos o atributos procedentes de más de un vocabulario XML. Si a cada uno de estos vocabularios se le da un espacio de nombres referenciado a una URI donde se listen los términos que incluye, se resuelve la ambigüedad existente entre elementos o atributos que se llamen igual.

Para el tipo de código que se genera en la propuesta desarrollada en esta tesis, se requiere la indicación del espacio de nombres asociado a los estándares WS-BPEL, BPEL4People, WS-HumanTask, y XML Schema, entre otras. La Expresión V-16 recoge el código asociado a la plantilla encargada de imprimir el espacio de nombres.

Expresión V-16. Plantilla MOFM2T para generar el espacio de nombres WS-BPEL & BPEL4People

```
[template public printXMLNS ()]
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:ty="http://www.example.com/types"xmlns:b4p="http://docs.oasis-
    open.org/ns/bpel4people/bpel4people/200803"
    xmlns:htd="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803"
    xmlns:htt="http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ns/bpel4people/bpel4people/200803
    ../../xml/bpel4people.xsd http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803 ../../xml/ws-
    humantask.xsd http://docs.oasis-open.org/ns/bpel4people/ws-humantask/types/200803 ../../xml/ws-
    humantask-types.xsd">
[/template]
```

4.4. Especificación de la regla de generación de roles asociados con la metaclase *«HumanRole»*

Los roles que intervienen en actividades humanas durante el proceso – metaclase *«HumanRole»* del metamodelo de ejecución y orquestación del proceso software, Figura IV.3 – se especifican dentro de la etiqueta *«b4p:humanInteractions»* de BPEL4People tal y como se ha introducido en la Expresión V-15 y se justifica en la Tabla V.3.

La plantilla *«humanRoleToBPEL4People»* es la encargada de imprimir en el texto de salida los diferentes grupos lógicos de roles que intervienen en el proceso y para ello, cada uno debe ser especificado con la etiqueta *«htd:logicalPeopleGroup»* – etiqueta proveniente del espacio de nombres de la especificación WS-HumanTask.

La Expresión V-17 recoge el código asociado a la plantilla encargada de llevar a cabo la transformación de cada instancia de la metaclase *«HumanRole»*.

Expresión V-17. Transformación MOFM2T para transformar metaclase *«HumanRole»*

```
[template public humanRoleToBPEL4People (process : ProcessExecutionClass)]

    <!-- Se definen los grupos lógicos de roles del proceso ->

    <b4p:humanInteractions>
        [for(role : HumanRole | process.ExecutionNodeClass→
                                    select(oclIsTypeOf(HumanExecutionClass))→
                                        iterate(rol : HumanExecutionClass;
                                          set : Set={} |
                                          set→union(rol.isResponsible)→union(rol.isParticipant)))]
          <htd:logicalPeopleGroup name="[role.name /]_LPG">
              <htd:documentation xml:lang="en-US">    [role.description /]    </htd:documentation>
              <htd:parameter name="region" type="xsd:string" />
          </htd:logicalPeopleGroup>
        [/for]
        <htd:tasks>
        [for(node : HumanExecutionClass | process.ExecutionNodeClass→
                                            select(oclIsTypeOf(HumanExecutionClass)))]
          <htd:task name="[ node.name /]_TSK" >
              <htd:documentation xml:lang="en-US">    [role.description /]    </htd:documentation>
              <htd:peopleAssignments>
                  <htd:potentialOwners>
                      [for(role : HumanRole | node.isParticipant→union(node.isResponsible))]
                          <htd:from logicalPeopleGroup="[role.name /]_LPG">
                          </htd:from>
                      [/for]
                  </htd:potentialOwners>
              </htd:peopleAssignments>
          </htd:task>
        [/for]
        </htd:tasks>
    </b4p:humanInteractions>

    <b4p:peopleAssignments>

    <!-- Se define el rol que puede iniciar una instancia del proceso. Para ello, se averigua el rol responsable
    del primer nodo de ejecución del proceso ->

     [for(role : HumanRole | process.ExecutionNodeClass→
                                    select(oclIsTypeOf(HumanExecutionClass)) [0].isResponsible)]
        < b4p:processInitiator>
              <htd:from logicalPeopleGroup="[role.name /]_LPG" />
        </b4p:processInitiator>
    [/for]
    </b4p:peopleAssignments>

[/template]
```

4.5. Especificación de la regla de generación de variables asociadas a la metaclase *«BusinessVar»*

La plantilla *«BusinessVarToBPEL»* es la encargada de imprimir en el código WS-BPEL de salida las diferentes instancias de la metaclase *«BusinessVar»* (Sección 3.2.9 del Capítulo IV) que intervienen durante la ejecución del proceso.

Las variables de WS-BPEL tienen varios cometidos. Por una parte, proporcionan los medios para mantener y gestionar los mensajes en las comunicaciones – de entrada o salida – con otros procesos o incluso servicios web, constituyendo de esta forma el estado interno del proceso de negocio, los cuales a su vez pueden ser enviados o recibidos hacia o desde otros servicios. Por otra parte, las variables también constituyen un mecanismo para almacenar datos temporales del proceso.

La Expresión V-18 especifica el código asociado a la transformación de las instancias de *«BusinessVar»* del modelo de ejecución y orquestación del proceso.

Expresión V-18. Transformación MOFM2T para transformar metaclase *«BusinessVar»*

```
[template public BusinessVarToBPEL (process : ProcessExecutionClass)]
    [for(var : BusinessVar | process.BusinessVar)]
            <variable name="[var.name /]" type="xsd:string" />
    [/for]
[/template]
```

4.6. Especificación de la regla de generación de variables asociadas a la metaclase *«WorkProduct»*

La plantilla *«WorkProductToBPEL»* es la encargada de imprimir en el código WS-BPEL de salida los diferentes productos que se utilizan, se modifican o que se generan durante la ejecución del proceso.

Sin embargo, como se ha mencionado en la Sección 4.1 de este capítulo, WS-BPEL no proporciona ninguna etiqueta que soporte el mapeo de este tipo de concepto. Como solución alternativa, en esta tesis se propone utilizar la etiqueta *«variable»* de WS-BPEL y utilizar su atributo *«MessageType»* para especificar el tipo de producto.

A pesar de esta solución, sigue existiendo un handicap del propio lenguaje WS-BPEL para plasmar de manera fiel (o al menos aproximada) toda la semántica de la metaclase *«WorkProduct»* (Sección 3.2.10 del Capítulo IV). En este sentido y debido a las limitaciones existentes, se propone mapear únicamente el atributo *«completeness»* de cada producto de salida con el objetivo de conocer cuál es su grado de completitud conforme avanza la ejecución del proceso.

La Expresión V-19 especifica el código asociado a la transformación de las instancias de la metaclase *«WorkProduct»* del modelo de ejecución y orquestación del proceso.

Expresión V-19. Transformación MOFM2T para transformar metaclase *«WorkProduct»*

```
[template public WorkProductToBPEL (process : ProcessExecutionClass)]
    [for(product : WorkProduct | process.output)]
            <variable name="[ product.name /]_completenessDegree"    type="xsd:int"
                    messageType= "ty:[ product.type /]" />
    [/for]
[/template]
```

4.7. Especificación de la regla de generación los marcadores de interacción y orquestación del proceso

Cuando en un proceso de negocio intervienen actividades de orquestación tales como llamadas a otros procesos (de manera más concreta al servicio WSDL que lo implementa) o invocación a servicios web, es necesario incluir referencias de dichas actividades dentro del código WS-BPEL que implementa el servicio en cuestión. Estas referencias se especifican mediante marcadores dentro de la etiqueta *«bpel:partnerLink»*, la cual describe las interfaces usadas para la interacción del proceso con servicios web. Estos marcadores son referenciados en la estructura de ejecución del proceso WS-BPEL.

Las plantillas MOFM2T encargadas de imprimir estos marcadores son *«ProcessExecutionClassToBPEL»*, con la que mapear la llamada a otro proceso (a partir de su WSDL); *«WSExecutionClassToBPEL»*, con la que mapear la invocación a un servicio web; y *«EMailExecutionClassToBPEL»*, con la que mapear la invocación al servicio de mensajería.

Todas estas plantillas se especifican en la Expresión V-20 para generar el código WS-BPEL asociado a los marcadores de las instancias de las metaclases hacia *«ProcessExecutionClass»*, *«WSExecutionClass»* y *«EMailExecutionClass»* del metamodelo de ejecución y orquestación (Sección 3 del Capítulo IV).

Expresión V-20. Transformación MOFM2T para generar marcadores de orquestación

```
<partnerLinks>
    [template public ProcessExecutionClassToBPEL (process : ProcessExecutionClass)]
        [for (node : ExecutionNodeClass | process.ExecutionNodeClass→
                                                select (oclIsTypeOf (ProcessExecutionClass)))]
            <partnerLink name="[ node.name /]_PL" partnerLinkType=" lns:ProcessExecutionClass" />
        [/for]
    [/template]

    [template public WSExecutionClassToBPEL (process : ProcessExecutionClass)]
        [for (node : ExecutionNodeClass | process.ExecutionNodeClass→
                                                select (oclIsTypeOf (WSExecutionClass)))]
            <partnerLink name="[ node.name /]_PL" partnerLinkType=" lns:WSExecutionClass" />
        [/for]
    [/template]
```

```
[template public EMailExecutionClassToBPEL (process : ProcessExecutionClass)]
    [for (node : ExecutionNodeClass | process.ExecutionNodeClass→
                                                    select (oclIsTypeOf (EMailExecutionClass)))]
            <partnerLink name="[ node.name /]_PL" partnerLinkType=" lns:EMailExecutionClass" />
    [/for]
[/template]
</partnerLink >
```

4.8. Especificación de la regla de generación de la estructura del proceso

La última regla de transformación MOFM2T invocada en la Expresión V-15 con la que se especifica la transformación *«PLM4BS_M2TTransformation»* – representada en el proceso sistemático de derivación de la Figura V.1 –, es la plantilla *«defineBPELStructure»*.

Esta plantilla, especificada en la Expresión V-21, se encarga de imprimir el código final con la estructura y secuencia de actividades WS-BPEL que conforman el modelo de ejecución y orquestación. En esta parte del código se hace referencia a elementos ya definidos – y construidos mediante las plantillas MOFM2T especificadas en las secciones anteriores – para las secciones *«partnerLinks»*, *«variables»* y *«humanInteractions»* de WS-BPEL. En paralelo y conforme se construye esta secuencia, la plantilla también transforma las instancias de la metaclase *«Constraint»* del metamodelo de ejecución y orquestación (Sección 3.2.8 del Capítulo IV).

Como se ha comentado en la Sección 4.1 de este capítulo, el propio lenguaje WS-BPEL presenta algunas limitaciones para cubrir toda la semántica del metamodelo de ejecución y orquestación descrito en el Capítulo IV. Algunas de estas limitaciones son, por ejemplo, las que se encuentran para codificar las metaclases *«EMailExecutionClass»* y *«ScriptExecutionClass»*.

La existencia de estas limitaciones requiere la intervención del ingeniero de procesos o analista para completar de manera manual todas las características técnicas necesarias para poder desplegar y ejecutar el código WS-BPEL generado con la regla transformación de la Expresión V-21. Todas las características a concretar manualmente han sido identificadas por el literal *«ToBeSpecified»* en el código.

Expresión V-21. Transformación MOFM2T para generar estructuras del proceso

```
[template public defineBPELStructure (process : ProcessExecutionClass)]
    [for (node : ExecutionNodeClass | process.ExecutionNodeClass)]

        [if (node.oclIsTypeOf (HumanExecutionClass)]
            <extensionActivity>
                <b4p:peopleActivity name="[ node.name /]" inputVariable="ToBeSpecified"
                                    outputVariable="ToBeSpecified">
                        <b4p:localTask reference="[ node.name /]_TSK" />
                </b4p:peopleActivity>
            </extensionActivity>[elseif (node.oclIsTypeOf (WSExecutionClass)]
```

```
            <invoke name="[ node.name /]" partnerLink="[ node.name /]_PL"  portType="[ node.port/]"
                        operation="[ node.nameOperation/]" inputVariable="[ node.hasParameter.name /]"
                        outputVariable="[ node.hasResult.name /]"  createInstance="yes" />
        [elseif (node.oclIsTypeOf (EMailExecutionClass)]
            <invoke name="[ node.name /]" partnerLink="[ node.name /]_PL"  portType="ToBeSpecified"
                        operation="ToBeSpecified" inputVariable="[ node.hasParameter.name /]"
                        outputVariable="[ node.hasResult.name /]"  createInstance="yes" />
        [elseif (oclIsTypeOf (ProcessExecutionClass)]
            <invoke name="[ node.name /]" partnerLink="[ node.name /]_PL"  portType="ToBeSpecified"
                        operation="ToBeSpecified" createInstance="yes" inputVariable="ToBeSpecified"
                        outputVariable="ToBeSpecified" />
        [else] <!-- el nodo es de tipo ScriptExecutionClass -->
            <empty name="[ node.name /]" />
        [/if]
    [/for]
[/template]
```

5. Conclusiones

Durante este capítulo se han estudiado las transformaciones entre los metamodelos definidos en el capítulo anterior y se han analizado las relaciones que se puede establecer entre ellos. Estas transformaciones se especifican mediante el lenguaje QVT.

Aunque existen herramientas que permiten ejecutar código en QVT, no se aprecia que sea posible ni sencillo ejecutar estas transformaciones tal cual se muestran en este capítulo debido a las limitaciones de las herramientas existentes. No obstante, como se verá en el siguiente capítulo, esta especificación de las transformaciones es fácilmente traducible a código ejecutable en cualquier lenguaje de propósito general.

Llegados a este punto, no se puede, sin embargo, dar por concluido este capítulo sin analizar antes dos aspectos importantes.

El primero de ellos es la dependencia en la etapa final del proceso de derivación descrito en la Sección 1 de este capítulo con el lenguaje estándar de ejecución de procesos WS-BPEL (y su extensión BPEL4PEOPLE). En este sentido, conforme al propio estándar evolucione habrá que ir adaptando y manteniendo las transformaciones aquí especificadas. Para intentar desacoplar nuestra propuesta del lenguaje WS-BPEL proponemos la ejecución directa del modelo de procesos software en un motor de ejecución de procesos. Esta propuesta se plantea como trabajo futuro de esta tesis ya que actualmente no existen motores que interpreten y ejecuten modelos de procesos basados en UML. Esta es una de las líneas abiertas por alcanzar por parte de la comunidad investigadora internacional.

El segundo aspecto está relacionado con las modificaciones a posteriori. Durante el capítulo, se establece un vínculo o trazabilidad entre el metamodelo de definición de procesos software y el metamodelo de ejecución y orquestación. De esta manera, desde la

definición del proceso se deriva el modelo básico de ejecución y orquestación, y a partir de éste, es posible obtener su modelo final mediante cambios controlados. Esta idea se muestra en la Figura V.1 y se amplía en la Figura V.2 en la que se refleja la necesidad de realizar cambios en el modelo de definición del proceso después de haber pedido el modelo final de ejecución y orquestación del proceso. Este proceso de retorno, como se ha ido viendo a lo largo del capítulo, no es automático y requiere de la experiencia del grupo de ingenieros de procesos o analistas.

Además, para garantizar la trazabilidad y la vuelta atrás entre ambos modelos, se hace palpable la necesidad de desarrollar herramientas que automaticen la comprobación de ambos aspectos para garantizar la calidad de los modelos definidos. Éste es uno de los trabajos futuros planteados en el Capítulo VII.

Capítulo VI PLM4BS: HERRAMIENTA DE SOPORTE

En los capítulos anteriores se han presentado los metamodelos necesarios para modelar procesos software y definir su contexto de ejecución y orquestación. Asimismo, también se han definido el protocolo sistemático de transformación con el que es posible derivar código ejecutable desde el modelo de definición de procesos. Este protocolo está basado en dos etapas: (*i*) obtención del modelo de ejecución y orquestación a partir del modelo de definición; y (*ii*) obtención de código ejecutable a partir del modelo de ejecución y orquestación con el propósito de que el modelo del proceso pueda ser interpretado por cualquier motor de procesos de los existentes actualmente en el mercado.

Sin embargo, para hacer posible la utilización práctica de este entorno teórico para la construcción de los modelos de definición y ejecución de procesos software, es necesario desarrollar una herramienta CASE que dé soporte durante esta construcción y que permitan la automatización de las reglas de transformación definidas en el capítulo anterior. La definición de una herramienta CASE es, precisamente, el cuarto objetivo definido en la Sección 2 del Capítulo III.

Como paso previo al diseño y desarrollo de una herramienta software, es necesario abordar y decidir cuál será la sintaxis concreta con la que se va a definir cada uno de los metamodelos implicados. En este sentido, y tal y como se argumenta en la Sección 2.1 de este capítulo, se optará por una definición basada en perfiles UML.

Una vez decidido el método para definir la sintaxis concreta a utilizar, este capítulo continúa describiendo los pormenores del diseño de la herramienta CASE de soporte desarrollada en esta Tesis Doctoral. Esta herramienta, bautizada con el acrónimo de PLM4BS (*«Process Lifecycle Management for Business-Software»*), permite gestionar el ciclo de vida de procesos software. Aunque actualmente sólo proporciona soporte a las fases de definición, ejecución y orquestación de dicho ciclo, abre la puerta a soportar el resto de fases del ciclo de vida en futuros trabajos.

Finalmente, los dos últimos apartados de este capítulo describen dos casos prácticos de aplicación de la herramienta PLM4BS y una serie de conclusiones finales, respectivamente.

1. Introducción

Con el uso del paradigma MDE y su traducción directa e implícita con el uso de metamodelos y protocolos de transformación entre modelos, es posible garantizar

uniformidad, formalización a través de una terminología común y una correcta y completa definición de un determinado dominio específico.

Sin embargo, si toda la definición teórica de metamodelos y reglas de derivación no viene acompañada de herramientas software eficaces, usables y atractivas para el usuario final, tareas como el mantenimiento y la gestión de los modelos construidos en base a dichos metamodelos puede llegar a resultar demasiado compleja y costosa. Son por estos motivos por los que resulta esencial ofrecer mecanismos MDE basados en herramientas para facilitar estas tareas; sobre todo si el propósito final es transferir todo el conocimiento teórico especificado hacia entornos prácticos y empresariales.

En este sentido, este capítulo tiene como propósito abordar cómo llevar a cabo la definición de las sintaxis concreta de los metamodelos presentados en el Capítulo IV, cómo automatizar las reglas de transformación especificadas en el Capítulo V, y finalmente cómo integrar ambas soluciones en la herramienta PLM4BS que haga posible su aplicabilidad en entornos reales.

Finalmente, para demostrar la viabilidad de PLM4BS, la Sección 4 enumera una serie de casos de éxito reales en los que la propuesta ha sido utilizada de manera satisfactoria y describe en detalle uno en concreto.

2. Definiendo la sintaxis concretas de los metamodelos

Como paso previo para la definición de las sintaxis concretas de los metamodelos de definición de procesos software y, ejecución y orquestación del proceso software – ambos definidos formalmente en el Capítulo IV –, se tendrán en cuenta una serie de consideraciones iniciales descritas en la Sección 2.1 de este capítulo. Adelantando brevemente estas consideraciones, la definición de las sintaxis concretas se realizará mediante perfiles UML. Concretamente, se utilizará la versión 2.5 de UML [OMG 2012].

Siguiendo las pautas y decisiones tomadas en la Sección 2.1, las Secciones 2.2 y 2.3 describen, respectivamente, el perfil UML que define el metamodelo de definición de procesos software y el perfil UML que define el metamodelo de ejecución y orquestación de procesos.

2.1. Contexto y planteamiento previo

Para facilitar el uso de lenguajes basados en modelos dentro de entornos de explotación, el OMG (*«Object Management Group»*) propone definir sintaxis concretas utilizando una de las siguientes perspectivas: (*i*) definir un nuevo lenguaje específico; o (*ii*) extender UML con el objetivo de especializar sus conceptos para definir otros nuevos, aprovechando así tanto la semántica (restricciones, propiedades y asociaciones) de todos los conceptos de UML como su propia madurez como uno de los estándares más arraigados y con mayor trayectoria dentro del mundo empresarial.

Cada una de las alternativas anteriores tiene ventajas y desventajas. Por ejemplo, en definir un nuevo lenguaje de manera *ad-hoc* permite una mayor expresividad y correspondencia con los conceptos de un dominio de aplicación particular. Sin embargo, el

hecho de no cumplir con un estándar puede llegar a complicar su aplicación en diferentes contextos y como solución en producción dentro de una organización.

Para facilitar su aplicación en ambientes empresariales y contextos reales, en este trabajo se ha optado por extender el lenguaje UML como mecanismo para definir la sintaxis concreta de los metamodelos presentados en el Capítulo IV .

Esta elección ha sido tomada porque UML proporciona un mecanismo de extensión usable, expresivo y flexible para adaptar metamodelos definidos de forma teórica dentro de entornos y herramientas empresariales. Esto es lo que se conoce como un perfil UML. Además, como ya se ha comentado anteriormente UML es un estándar ampliamente utilizado y reconocido en la empresa. En segundo lugar, UML proporciona flexibilidad, expresividad y un mecanismo de extensión genérico para construir modelos UML dentro de dominios específicos.

El protocolo de extensión de UML está basado en tres mecanismos básicos: (*i*) estereotipos o *«stereotype»*, con los que es posible definir cada uno de los elementos de un dominio específico que deberán a su vez extender metaclases UML específicas; (*ii*) valores etiquetados o *«tagged value»*, que permiten añadir propiedades particulares sobre cualquier elemento estereotipado definido dentro del perfil; y (iii) restricciones o *«constraint»*, con las que definir las condiciones semánticas que aplican sobre los estereotipos del perfil y que condicionan la instanciación del metamodelo con el propósito de construir modelos bien definidos.

2.2. Perfil UML para el metamodelo de definición de procesos software

Durante la definición del perfil UML que implementa el metamodelo de definición de procesos software, se han seguido tres directrices básicas: (*i*) definir un estereotipo por cada uno de los elementos del metamodelo de definición de procesos y además, incluir los valores etiquetados necesarios en cada estereotipo según su elemento homólogo en el metamodelo; (*ii*) elegir y justificar qué metaclases de UML2.5 utilizar para extender cada uno de los estereotipos contemplados; y (*iii*) adaptar las restricciones semánticas del metamodelo para restringir el comportamiento de las metaclases de UML2.5 utilizadas.

Siguiendo estas pautas, la Figura VI.1 muestra el perfil UML con el que se implementa el metamodelo de definición de procesos software definido estructurada y formalmente en el Capítulo IV .

Antes de mostrar el perfil, resulta conveniente aclarar un aspecto relevante de la Figura VI.1. En ella, se utilizan dos triángulos con un grafismo muy parecido pero con una semántica completamente diferente. Las relaciones con un triángulo sombreado en uno de los extremos denotan el mecanismo de extensión UML mientras que las relaciones con un triángulo sin sombrear denotan el mecanismo de herencia UML. Sin más, la siguiente imagen muestra el perfil.

Como se puede apreciar en la figura, se ha definido un estereotipo por cada elemento del metamodelo y además será completado su definición por una serie de va,lores etiquetados según las propiedades definidas en el elemento del metamodelo.

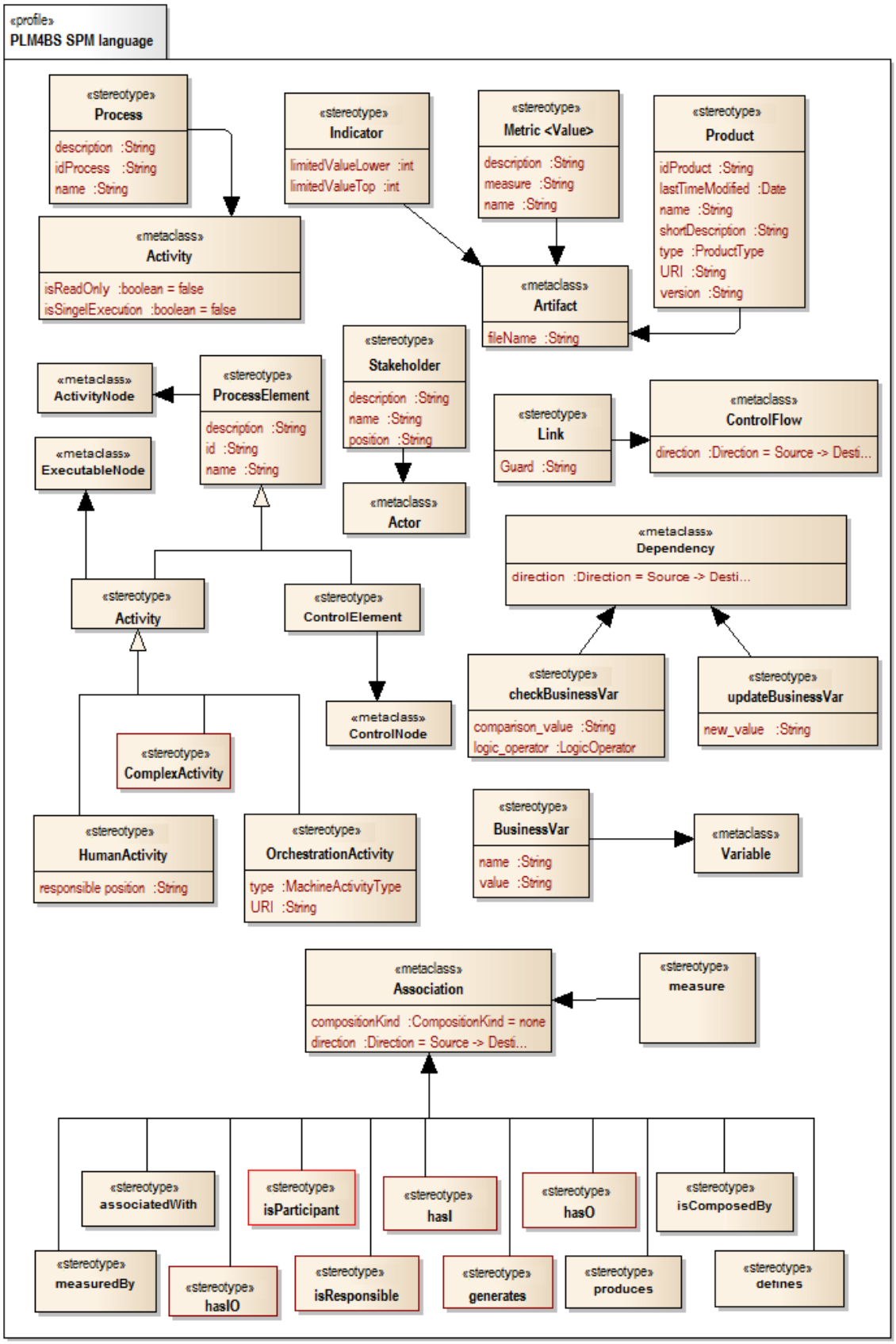


Figura VI.1. Perfil UML de definición de procesos software

La Tabla VI.1 muestra de manera resumida los estereotipos del perfil UML de definición de procesos software y cuál es la metaclase UML que se ha utilizado para extender cada uno de ellos. Asimismo, en la tabla se incluye la justificación por la que se ha optado por la metaclase UML en cuestión.

Tabla VI.1. Correspondencia de UML y metamodelo de definición de procesos

Metaclase UML	Estereotipo del perfil	Justificación de la elección
«Activity»	*«Process»*	Esta metaclase se ha utilizado para extender el estereotipo *«Process»* porque aquella se especifica como una secuencia de unidades subordinadas utilizando para ello, un modelo de flujo de control y datos.
«ActivityNodes»	*«ProcessElement»*	La metaclase *«ActivityNodes»* de UML es utilizada para modelar pasos individuales del contexto especificado por la metaclase *«Activity»*. UML especializa *«ActivityNodes»* en tres clases diferentes. En el perfil que se está definiendo en esta sección se utilizará únicamente dos de ellas: *«ControlNodes»*, que actúa como una puerta de enlace para gestionar el flujo de información; y *«ExecutableNodes»*, que lleva a cabo el comportamiento deseado de una actividad.
«ExecutableNodes»	*«Activity»*	
«ControlNodes»	*«ControlElement»*	Como se ha comentado, la metaclase *«ActivityNodes»* es especializada por la metaclase *«ControlNodes»*, que actúa como una puerta de enlace para gestionar el flujo de información.
«Actor»	*«Stakeholder»*	La metaclase *«Actor»* de UML especifica un rol jugado por un usuario o por cualquier otro sistema que interactúa con un sujeto del modelo.
«Variable»	*«BusinessVar»*	UML define la metaclase *«Variable»* como elemento para posibilitar la elección de un flujo de datos alternativo. Para ello, UML contempla la propagación del valor de la variable desde el punto en el que es asignado dicho valor hasta todos aquellos puntos del flujo en los que dicho valor es consultado.
«Association»	*«hasI», «hasO», «hasIO», «isParticipant», «produces», «isResponsible, «generates, «defines», «measuredBy, «generates», «isComposedBy», «associatedWith», «measure»*	Se ha optado por la metaclase *«Association»* de UML con la que extender los estereotipos indicados debido a que esta metaclase especifica una relación semántica que puede ocurrir entre dos instancias y declara que puede haber vínculos entre las instancias de los tipos asociados.

«ControlFlow»	*«Link»*	UML modela el flujo de ejecución con la asociación directa de dos *«ActivityNodes»* por medio de la metaclase *«ActivityEdges»*. Esta metaclase es especializada en dos metaclases, pero en el perfil definido en esta sección se utilizará sólo una de ellas: la metaclase *«ControlFlow»*, ya que es la que UML contempla para la definición explícita de la secuencia de ejecución de *«ActivityNodes».* Se ha utilizado la metaclase *«ControlFlow»* en detrimento de la metaclase *«Association»* porque aquella: (*i*) hace de puente entre dos *«ActivityNodes»*; y (*ii*) puede definir una condición que debe ser cierta antes de pasar el control a través de la *«ActivityEdges»* – esta condición se denomina guarda y es conocido por el término *«Guard»* de UML.
«Artifact»	*«Product»* *«Indicator»* *«Metric»*	UML define la metaclase *«Artifact»* como un elemento de información que representa un elemento concreto del mundo físico y que puede ser usado o producido de manera indistinta por un proceso de desarrollo software o por una operación de sistema. Según esta definición, ficheros fuente, scripts, archivos ejecutables, tablas de bases de datos, entre otros, pueden ser considerados ejemplos de artefactos.
«Dependency»	*«updateBusinessVar»,* *«checkBusinessVar»*	Para extender los estereotipos indicados se ha optado por la metaclase *«Dependency»* de UML. Esta metaclase establece una relación semántica entre los objetos enlazados de tal forma que la semántica del objeto de origen no está completa sin el objeto destino.

2.3. Perfil UML para la definición del contexto de ejecución y orquestación

La definición del perfil UML para dar soporte al metamodelo de ejecución y orquestación de proceso software ha sido realizada siguiendo las mismas directrices que las tomadas durante la definición del perfil UML descrito en la Sección 2.2 de este capítulo.

Siguiendo estas pautas, la Figura VI.2 muestra la sintaxis concreta (mediante un perfil UML) con el que se define el metamodelo de ejecución y orquestación de procesos software, definido estructurada y formalmente en el Capítulo IV .

Al igual que en el perfil anterior, resulta conveniente volver a recalcar un aspecto relevante de la Figura VI.2. En ella, se utilizan dos triángulos con un grafismo muy

parecido pero con una semántica completamente diferente. Las relaciones con un triángulo sombreado en uno de los extremos denotan el mecanismo de extensión UML mientras que las relaciones con un triángulo sin sombrear denotan el mecanismo de herencia UML. Sin más, la siguiente imagen muestra el perfil.

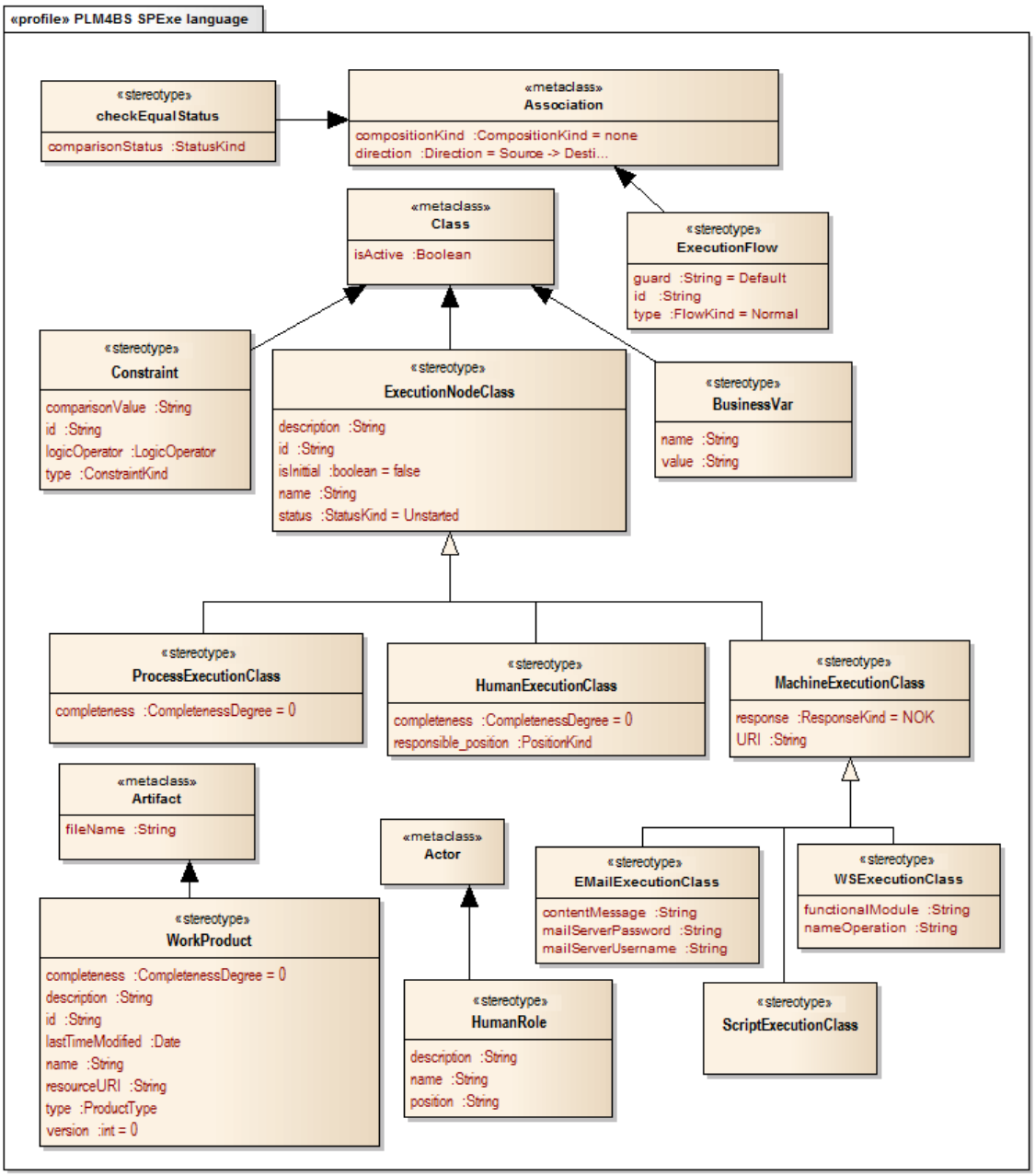


Figura VI.2. Perfil UML de ejecución y orquestación de procesos software

La Tabla VI.2 muestra de manera resumida los estereotipos del perfil UML anterior y cuál es la metaclase UML que se ha utilizado para extender cada uno de ellos. Asimismo, en esta tabla se incluye la justificación por la que se ha optado por la metaclase UML en cuestión.

Tabla VI.2. Correspondencia UML y metamodelo de ejecución y orquestación de procesos

Metaclase UML	Estereotipo del perfil	Justificación de la elección
«Class»	*«ExecutionNodeClass»*, *«Constraint»*, *«BusinessVar»*	El propósito de la metaclase *«Class»* es especificar una clasificación de objetos y especificar sus propiedades (atributos, operaciones, asociaciones, etc.) que caracterizan la estructura y el contexto de estos objetos.
«Association»	*«checkEqualStatus»*, *«ExecutionFlow»*	La justificación por la que se opta por utilizar estas metaclases UML es la misma que la proporcionada en el perfil UML descrito en la Tabla VI.1 de la Sección 2.2 para los estereotipos *«Stakeholder»*, *«Product»* y *«isComposedBy»*, entre otros.
«Artifact»	*«WorkProduct»*	
«Actor»	*«HumanRole»*	

3. PLM4BS: Herramienta CASE de soporte al marco de trabajo teórico

Como se ha trasladado a lo largo de toda esta tesis, para hacer posible la utilización práctica del marco de trabajo teórico que se ha definido para la construcción de modelos de definición y, ejecución y orquestación de procesos software, es necesario desarrollar un marco de trabajo basado en herramientas CASE que dé soporte a esta construcción y que, además, automatice las reglas de derivación definidas en el capítulo anterior.

Esta sección tiene como propósito la descripción de los fundamentos de la herramienta CASE que hará posible la aplicación práctica y efectiva de todo el marco de trabajo teórico presentado en capítulos anteriores. Esta herramienta es bautizada con el nombre de PLM4BS (*«Process Lifecycle Management for Business-Software»*) y con ella, será posible gestionar el ciclo de vida del proceso software dentro de las organizaciones del sector. Aunque actualmente sólo proporciona soporte a las fases de definición, ejecución y orquestación de dicho ciclo, abre la puerta diferentes trabajos futuros para dar soporte al resto de fases del ciclo de vida.

Para alcanzar la meta propuesta, primero se presenta el contexto y planteamiento previo en el que se enmarca la solución PLM4BS. Este contexto permite plantear, posteriormente, su arquitectura software, cómo se han implementado los perfiles UML definidos en la Sección 2 de este capítulo y cómo se han implementado las reglas de derivación descritas en las Secciones 2 y 3 del Capítulo V con el objetivo de automatizar el proceso sistemático de derivación descrito en la Sección 1.1 del Capítulo V .

3.1. Contexto y planteamiento previo

De manera general, para desarrollar la herramienta PLM4BS se podría optar por una de las dos siguientes alternativas básicas: (*i*) diseñar y desarrollar una herramienta *ad-hoc* con capacidad para modelar diagramas de procesos y ejecutar reglas de derivación; o (*ii*) utilizar como base una herramienta de modelado que proporcione mecanismos de

extensión – basados, por ejemplo, en *plugins* – y cuyo uso sea extendido en el ámbito empresarial facilitando de esta manera una posible transferencia tecnológica del conocimiento teórico desarrollado en esta tesis hacia el tejido empresarial.

En este trabajo de tesis se opta por la segunda de estas opciones y para ello, se opta por la herramienta de modelado Enterprise Architect [SparxSystems 2014] como herramienta base sobre la que desplegar y desarrollar un módulo funcional específico para aplicar PLM4BS en entornos prácticos.

Esta decisión se ha tomado después de tener en cuenta las conclusiones de un trabajo [IWT2 *et al.* 2008] realizado por el grupo de investigación Ingeniería Web y Testing Temprano de la Universidad de Sevilla y la Consejería de Educación, Cultura y Deporte y con supervisión de la Consejería de Innovación, Ciencia y Empleo de la Junta de Andalucía.

Este trabajo consistió en la realización de un estudio comparativo de nueve[6] herramientas de modelado y su propósito era identificar aquella que mejor se adecuaba a las necesidades de la Consejería. Para alcanzar este objetivo, el estudio establecía un esquema de caracterización que proporcionó un marco estándar sobre el cual fundamentar la elección tomada. Estas características fueron las siguientes: (*i*) soporte a mecanismos de extensión de UML con los que sea posible la definición de perfiles UML; (*ii*) soporte a mecanismos que posibiliten la generación sistemática de modelos y código a partir de modelos; (*iii*) soporte para la generación de documentación de forma automática, personalizada y flexible; (*iv*) soporte para la gestión del ciclo de vida software de grandes proyectos; (*v*) soporte para gestionar de manera completa el ciclo de vida software teniendo cuenta las fases de estudio de viabilidad, captura de requisitos, análisis, diseño, implementación, pruebas y mantenimiento; (*vi*) compatibilidad con UML.

La Tabla VI.3 resume los resultados del estudio comparativo. Por simplicidad y legibilidad, sólo se ha indicado qué característica – de las indicadas anteriormente – tiene cada herramienta.

Finalmente, el estudio concluyó que Enterprise Architect es la herramienta que mejor se adaptaba a las necesidades de la Consejería con una mejor relación calidad/precio.

Otra razón importante para utilizar Enterprise Architect como cimiento de la propuesta PLM4BS es que es ampliamente conocida por las empresas y organizaciones en las que, potencialmente, PLM4BS podrá ser validada y evaluada.

[6] Las herramientas evaluadas fueron: NDT-Tool [Escalona et al. 2003], herramienta software libre de la Universidad de Sevilla para dar soporte a la metodología NDT; ArgoUML y StarUML, herramientas software libre; Poseidon for UML [Gentleware 2013], versión comercial de la herramienta ArgoUML; Rational Rose [IBM 2013], herramienta de IBM para desarrollar sistemas modelando con UML; Rational Software Modeler [IBM 2013], nueva herramienta de IBM que sustituía a la antes mencionada Rational Rose; Enterprise Architect (EA); IRqA [Visure 2013] y Doors [IBM 2013], ambas, con capacidad para dar soporte a la especificación de requisitos.

Tabla VI.3. Estudio comparativo de herramientas de modelado

	NDT-Tool	ArgoUML	StarUML	Poseidon for UML	Rational Rose	Rational Software Modeler	EA	IRqA	Doors
i	×		×		×	×	×	×	
ii	×		×		×	×	×	×	
iii	×				×	×	×	×	×
iv		×	×	×	×	×	×	×	×
v		×	×	×	×	×	×		
vi	×	×		×		×	×		

De hecho, el número de empresas es relevante gracias, principalmente, a las estrechas colaboraciones que ha tenido y está teniendo actualmente el grupo de investigación Ingeniería Web y Testing Temprano – grupo en el que se gesta este trabajo de tesis – en diversos proyectos I+D con empresas, tanto públicas (por ejemplo y entre otros, la Agencia de Obra Pública, la Consejería de Educación, Cultura y Deporte, y la Consejería de Salud y Bienestar Social, todas ellas empresas y organismos de la Junta de Andalucía) como privadas (por ejemplo, Airbus Military, Tecnocom, Everis, Fujitsu y Ayesa, entre otras).

Todas las razones anteriores han propiciado la decisión de tomar Enterprise Architect como herramienta base sobre la que definir y desarrollar PLM4BS, aprovechando así toda la potencialidad, funcionalidad y arquitectura que ofrece Enterprise Architect.

Una vez definido este contexto y planteamiento previo, en los siguientes apartados de esta sección se describe con más detalle cuál es la arquitectura del marco de trabajo PLM4BS dentro del entorno que ofrece Enterprise Architect, cómo se han implementado los perfiles UML definidos en la Sección 2 de este capítulo y finalmente, cómo se han implementado las reglas de derivación descritas en el Capítulo V para hacer posible la automatización del proceso sistemático de derivación descrito en la Sección 1.1 del Capítulo V .

Asimismo, en el Anexo B de este documento, puede encontrarse un manual de usuario de la herramienta PLM4BS.

3.2. Arquitectura de PLM4BS

Como se ha argumentado y justificado a lo largo de la sección anterior, para el diseño y desarrollo de la herramienta PLM4BS se utilizará como base la arquitectura software que proporciona Enterprise Architect.

Una de las ventajas que proporciona Enterprise Architect frente a otras herramientas de modelado es su capacidad de extensión para el desarrollo de *plugins*, más conocidos por *«Add-in»* en el argot terminológico de Enterprise Architect. Para garantizar esta capacidad, Enterprise Architect se sustenta sobre tres pilares fundamentales: (*i*) una base de datos relacional; (*ii)* diferentes módulos de control con los que es posible interactuar a

nivel de datos y además, capturar eventos de la interfaz de usuario; y (*iii*) el módulo *«MDG Technologies»* con el que definir perfiles UML.

Basada en los tres pilares funcionales anteriores, la Figura VI.3 muestra una posible arquitectura del *«Add-in»* de PLM4BS.

En el nivel inferior de la arquitectura de PLM4BS se localiza un repositorio de datos en el que se almacenan las instancias de los metamodelos definidos en el Capítulo IV. Este repositorio de datos se sustenta sobre el esquema relacional de base de datos de Enterprise Architect. Este esquema puede ser implementado en diferentes entornos y sistemas de gestión de base de datos: desde entornos locales con Microsoft Access hasta entornos colaborativos utilizando MySQL, Oracle, PostgreSQL o SQL Server, entre otros.

Esta característica hace posible que cualquier modelo – modelado conforme a un metamodelo concreto como los planteados en el Capítulo IV de esta tesis – en Enterprise Architect esté almacenado en un repositorio centralizado, estructurado y colaborativo, facilitando de esta manera su almacenamiento y posterior explotación.

Por encima del nivel anterior, se sitúa toda la lógica de negocio que ofrece Enterprise Architect y que posibilita el desarrollo de *«Add-ins»*.

En el caso que nos ocupa, y siguiendo la Figura VI.3, el *«Add-in»* de PLM4BS sólo requiere acceso a determinadas funcionalidades de Enterprise Architect. Concretamente, al módulo de control de eventos de la interfaz de usuario y el módulo de control de acceso datos de Enterprise Architect.

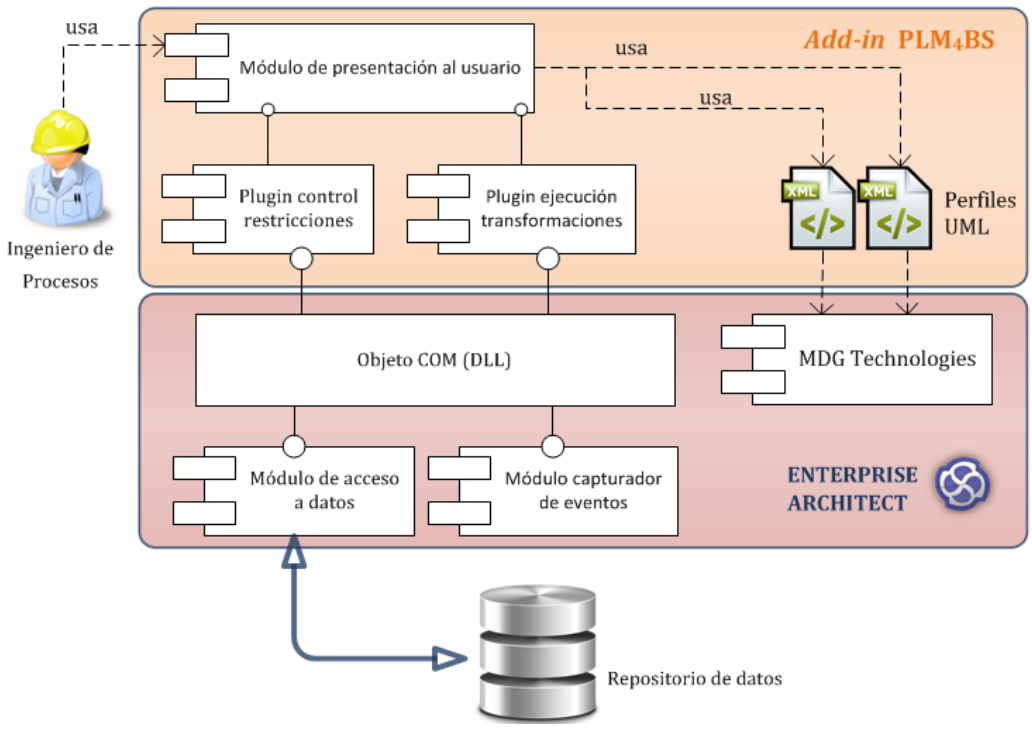


Figura VI.3. Arquitectura de PLM4BS

Sin embargo, no es posible acceder de manera directa a esta lógica de negocio. En su defecto, Enterprise Architect proporciona al desarrollador una biblioteca de vínculos dinámicos, o DLL – acrónimo inglés de *«dynamic-link library»* –, con diferentes objetos COM[7] (*«Component Object Model»*) que exponen un conjunto de métodos de uso público para el control acceso a todos y el control de eventos de la interfaz de usuario de Enterprise Architect, entre otras funcionalidades.

Por otra parte, para desarrollar el entorno de trabajo de PLM4BS es también necesario incluir dentro de la herramienta Enterprise Architect los perfiles UML definidos en la Sección 2 de este capítulo con el propósito de que el usuario final pueda modelar sus procesos software y definir sus modelos de ejecución y orquestación. En este sentido, Enterprise Architect dispone de su herramienta *«MDG Technologies»* con la que es posible describir sintaxis concretas, perfiles UML, patrones de diseño, plantillas, entre otros recursos para tareas de modelado.

Finalmente, en el nivel superior de la arquitectura presentada en la Figura VI.3 se sitúa las interfaces de usuario y los diferentes módulos funcionales que componen el *«Add-in»* de PLM4BS. Estos módulos funcionales son los encargados de implementar todas las reglas de derivación QVT definidas de manera teórica en el Capítulo V y además, implementar todas las restricciones OCL que fueron definidas en el Capítulo IV, con el propósito de asegurar la integridad de los modelos definidos conforme a los metamodelos descritos en el Capítulo IV.

A continuación, la Sección 3.3 y la Sección 3.4 describen con algo más de detalle algunas evidencias sobre cómo se han definido los perfiles UML descritos en la Sección 2 de este capítulo y cómo se han implementado las reglas de derivación QVT del Capítulo V y restricciones OCL del Capítulo IV, respectivamente.

3.3. Trasladando las sintaxis concretas definidas a Enterprise Architect

Uno de los principales aspectos más importantes que es necesario abordar antes de automatizar el proceso de derivación presentado en el capítulo anterior, es decir cómo trasladar las sintaxis concretas definidas mediante perfiles UML en la Sección 2 de este capítulo dentro del entorno de modelado elegido.

En el caso que nos ocupa y tal y como se adelanta en la Sección 3.2 de este capítulo, las sintaxis concretas son definidas dentro del entorno que ofrece Enterprise Architect.

Sin embargo, llegados a este punto es necesario enfatizar que las sintaxis concretas pueden ser implementadas en cualquier herramienta de modelado que proporcione mecanismos de extensión de UML. De hecho, la decisión de optar por Enterprise Architect no es limitante para utilizar el marco de trabajo PLM4BS en cuanto al modelado de procesos software, ya que la mayoría de las herramientas de modelado proporcionan

[7] Un componente de objeto COM es código ejecutable contenido en una biblioteca de vínculos dinámicos (.dll) o en un archivo ejecutable (.exe). Los componentes proporcionan uno o más objetos y unidades autónomas de código que realizan funciones concretas dentro del componente. Cada objeto tiene métodos, procedimientos programados y propiedades, así como atributos de comportamiento.

mecanismos para el intercambio de información – por ejemplo, con funciones de importación y exportación – por medio de formatos estructurados y estandarizados como el formato XMI [OMG 2011c].

De esta manera es posible y factible migrar cualquiera de los modelos definidos en una herramienta concreta de modelado hacia otra, sin incurrir con ello en un coste excesivo.

Volviendo al objetivo de este apartado y, tal y como se introduce en el apartado anterior, Enterprise Architect (EA) ofrece la posibilidad de definir sintaxis concretas – entre otros recursos de modelado – utilizando para ello su herramienta *«MDG Technologies»* (*«Model Driven Generation Technologies»*), una herramienta que permite extender las capacidades de modelado que ofrece por defecto EA de tal manera que es posible definir diferentes recursos para modelar – entre dichos recursos se encuentra la posibilidad de definir perfiles UML.

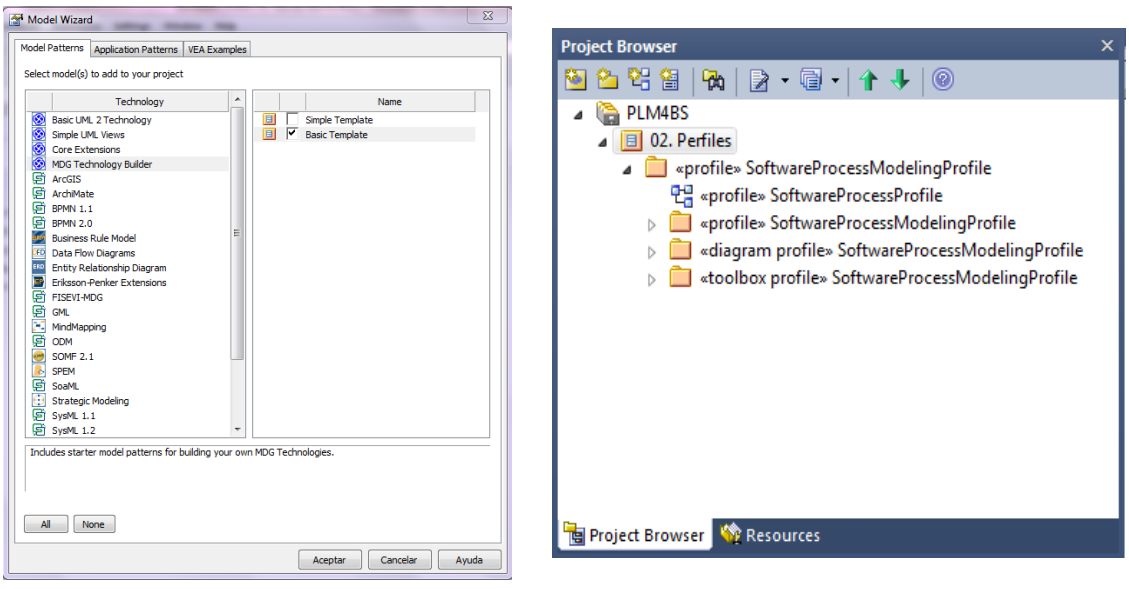
A modo de ejemplo, a continuación se enumeran y describen brevemente los pasos que se han seguido para implementar el perfil UML del metamodelo de definición de procesos software – presentado en la Sección 2.1 de este capítulo –. Estos pasos son:

II. Crear proyecto *«MDG Technology»* en Enterprise Architect.
III. Crear e instalar fichero *«MDG Technology»* desde Enterprise Architect.

Aunque este proceso no se describe en su totalidad, se presenta de la manera más rigurosa y completa posible como para hacer ver al lector el trabajo realizado.

I. Crear proyecto «MDG Technology» en Enterprise Architect

Como primer paso para la definición de un perfil UML dentro de Enterprise Architect (EA), es necesario crear un proyecto *«MDG Technology»* utilizando el asistente de Enterprise Architect (Figura VI.4.a).



(a) **(b)**

Figura VI.4. Proyecto *«MDG Technology»* en Enterprise Architect

Una vez seleccionado este tipo de proyecto e introducido el nombre del mismo, Enterprise Architect crea la estructura de paquetes mostrada en la Figura VI.4.b. El objetivo de cada uno los paquetes de esta estructura es el siguiente:

— Paquete *«profile»*. Este paquete contiene el conjunto de estereotipos – etiquetados con la palabra clave *«stereotype»* –, junto con sus valores etiquetados, que componen el perfil UML. Además, cada uno de estos estereotipos debe estar enlazado con la metaclase UML oportuna a través de una relación *«extend»*. Como directriz de buenas prácticas, cada estereotipo se ha identificado con el mismo nombre que su homólogo del perfil UML definido en la Sección 2.2.

  Por otra parte, cada uno de los estereotipos contemplados se define por medio de un conjunto de valores etiquetados – también conocidos como *«tagged value»* –. Estos valores etiquetados se corresponden con los atributos de los estereotipos definidos en el perfil UML de la Sección 2.2.

  A modo de ejemplo, la Figura VI.5 ilustra parte del perfil UML definido en Enterprise Architect (EA). Se ha optado por presentar únicamente una sección de este perfil ya que su descripción completa no aporta nada nuevo respecto al perfil UML presentado en la Sección 2.2. Respecto a la figura, ésta muestra como los estereotipos *«MachineActivity»* y *«HumanActivity»* heredan de la metaclase *«Activity»* de UML, y como los estereotipos *«updateVariable»* y *«checkVariable»* heredan de la metaclase *«Dependency»* de UML.

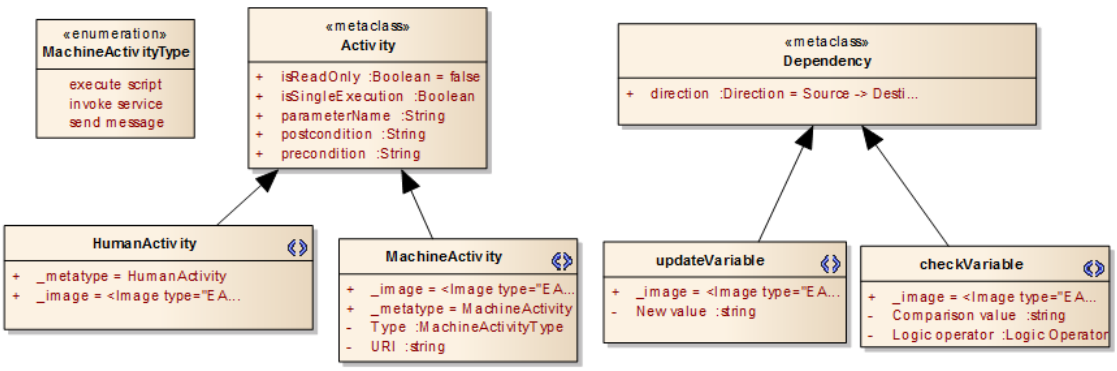


Figura VI.5. Parte del perfil UML definido en EA[8]

Como se puede apreciar en la imagen anterior, se han incluido ciertos valores etiquetados que no estaban contemplados inicialmente en el perfil definido en la Sección 2.2 de este capítulo. Concretamente, los valores etiquetados *«_image»*, *«_metatype»*, y *«size»*, entre otros.

[8] Con el propósito de no enturbiar la descripción del procedimiento seguido para la implementación en Enterprise Architect (EA) de los perfiles UML especificados en la Sección 2, sólo se muestra un fragmento del diagrama EA que implementa el perfil UML de definición de procesos software. Sin embargo, el perfil UML completo en EA ha sido desarrollado y puede ser proporcionado bajo demanda.

Estos atributos son valores etiquetados especiales que proporciona Enterprise Architect para personalizar el comportamiento de cada estereotipo cuando éste es instanciado durante la construcción de un modelo. Por ejemplo, el valor etiquetado *«_image»* permite personalizar el comportamiento visual del estereotipo sobre el que actúa, y para ello, Enterprise Architect ofrece una característica interesante denominada *«Shape Script»* que, como su nombre sugiere, es un lenguaje de *«scripting»* que permite definir la forma de elementos – e incluso conectores – de un perfil UML.

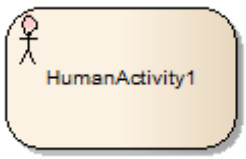
A modo de ejemplo, la Figura VI.6.a y la Figura VI.6.b muestran el código que permite personalizar el comportamiento visual asociado a los estereotipos *«HumanActivity»* y *«checkVariable»* – representados en el perfil de la Sección 2.2 de este capítulo –, respectivamente. En el caso particular del estereotipo *«checkVariable»* (Figura VI.6.b), se han establecido ciertos valores en sus valores etiquetados para ilustrar cómo es su representación visual.

```
shape main
{
    roundrect(0,0,100,100,35,35);
    addsubshape("nombrecompartment", 100, 100);
    addsubshape("img", 70, 105);
    shape nombrecompartment
    {
        h_align = "center";
        v_align = "center";
        editablefield = "Version";
        println("#Name#");
    }
    shape img
    {
        setfillcolor(255,204,204); // (R,G,B)
        ellipse(10,-90,19,-79);  // (x1,y1,x2,y2)
        startpath();
        moveto(14,-79);
        lineto(14,-68);
        moveto(14,-68);
        lineto(9,-57);
        moveto(14,-68);
        lineto(18,-57);
        moveto(9,-75);
        lineto(19,-75);
        endpath();
        fillandstrokepath();
    }
}
```

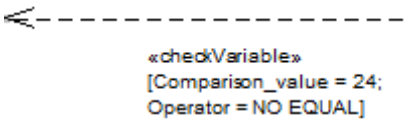
```
shape main
{
    // draw a dashed line
    noshadow=true;

    setlinestyle("DASH");
    moveto(0,0);
    lineto(100,0);
    hidelabel("middlebottomlabel");
}

shape middletoplabel{
    print ("«checkVariable» [Comparison_value = ");
    print("#TAG:Comparison value#");
    print ("; Operator = ");
    print("#TAG:Logic operator#");
    print ("]");
}
```



(a)



(b)

Figura VI.6. Personalización visual de estereotipos en EA[9]

[9] De igual manera que con la Figura VI.5, se ha optado por mostrar únicamente el código con el que se personaliza la apariencia de algunos de los estereotipos de los perfiles UML desarrollados en Enterprise Architect (EA). En cualquier caso, esta información puede ser proporcionada bajo demanda.

— Paquete *«diagram profile»*. Este paquete contiene todos aquellos artefactos de Enterprise Architect que son necesarios para definir cómo crear diagramas según determinados estereotipos de los definidos en el paquete anterior, y seleccionados previamente. A partir de este conjunto de artefactos, se le otorga al usuario la capacidad para modelar siguiendo un determinado perfil UML.

En el caso que nos ocupa, la Figura VI.7 muestra los artefactos necesarios que permitir al usuario crear diagramas o modelos en Enterprise Architect en base al perfil UML definido en el paquete anterior: el paquete *«profile»*.

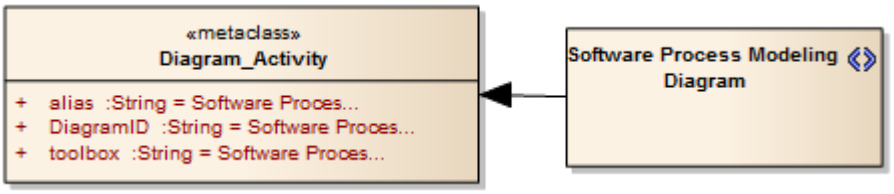


Figura VI.7. Artefactos necesarios para crear diagramas en EA en base a un perfil UML

Como se puede apreciar en la figura anterior, se ha creado un artefacto denominado *«Software Process Modeling Diagram»* y estereotipado con la etiqueta *«stereotype»*. Este artefacto puede ser denominado como un nuevo estereotipo del perfil, pero es necesario recalcar que es propio de Enterprise Architect y que su misión es única y exclusivamente la creación de diagramas a partir del cual el usuario puede comenzar a modelar.

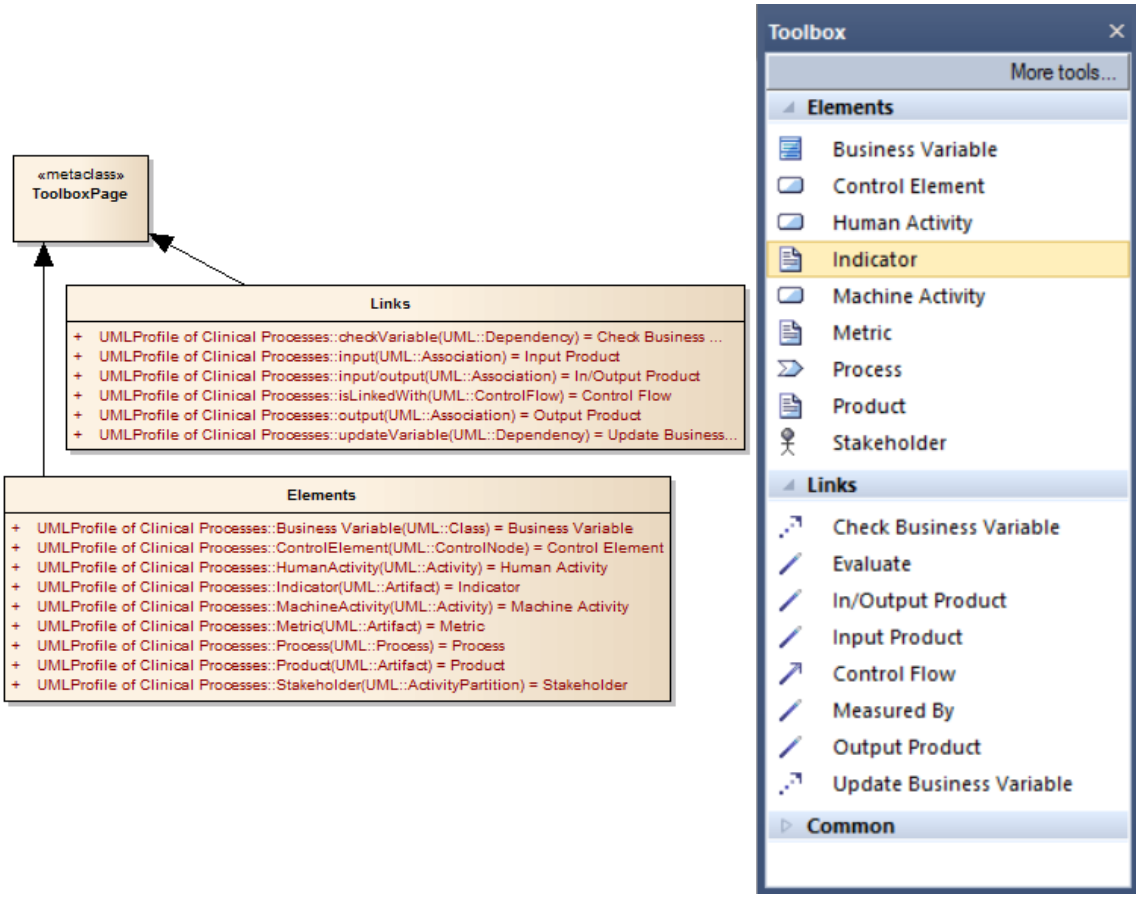
Una vez creado este estereotipo del diagrama basta con extenderlo desde una metaclase interna de Enterprise Architect para la creación de diagramas. En nuestro caso, se ha optado por extender la metaclase *«Diagram_Activity»* para disponer como base de las facilidades que proporciona Enterprise Architect para el modelado de diagramas de actividades.

Finalmente, la definición del paquete *«diagram profile»* finaliza tras establecer los parámetros de configuración del nuevo diagrama que se está definiendo. Para ello, basta con informar los siguientes atributos de la metaclase *«Diagram_Activity»*: *«alias»*, nombre que, por defecto, tomará el diagrama; *«DiagramID»*, identificador único dentro del entorno de EA; y *«toolbox»*, caja de herramientas que proporciona acceso rápido a todos los constructores de los estereotipos del perfil UML (el *«toolbox»* se define en el siguiente paquete).

— Paquete *«toolbox profile»*. Este paquete contiene los artefactos de EA necesarios para la creación de cajas de herramientas personalizadas – conocidas comúnmente como *«toolbox»* en el argot terminológico de EA –. Como se adelantó anteriormente, un *«toolbox»* proporciona acceso rápido a los constructores de los elementos contenidos en un determinado perfil UML. Además, es posible organizar visualmente estos constructores en compartimentos o páginas según la terminología EA.

En el caso que nos ocupa, para el perfil de definición de procesos software descrito en la Sección 2.2 de este capítulo, se ha definido un *«toolbox»* con dos páginas con el objetivo de diferenciar los estereotipos que tienen un comportamiento de conector o enlace, del resto de estereotipos del perfil.

La Figura VI.8.a y la Figura VI.8.b muestran el conjunto de artefactos EA que permiten definir un *«toolbox»* con dos páginas y cuál es el resultado final, respectivamente.



(a) (b)

Figura VI.8. Artefactos necesarios para crear un «toolbox» en EA

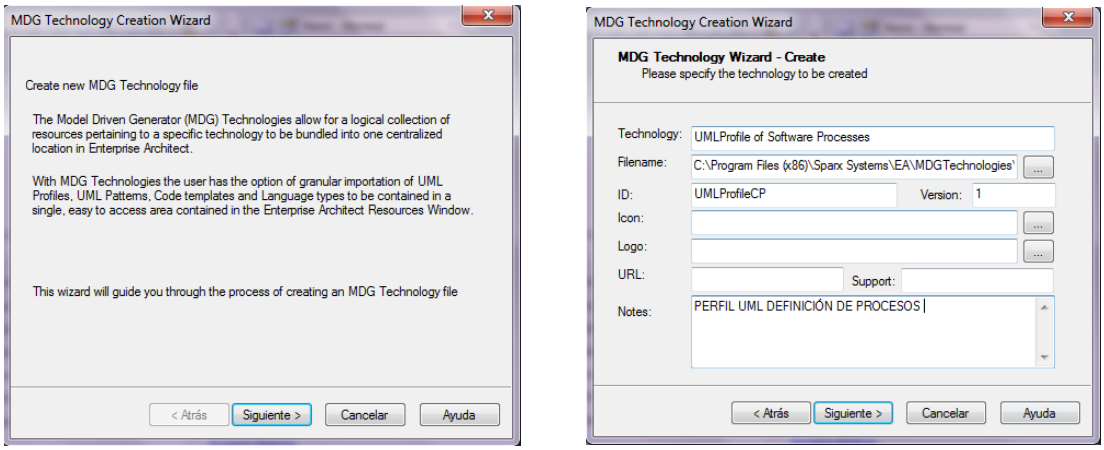
II. Crear e instalar fichero «MDG Technology» desde Enterprise Architect

Después de crear el perfil UML para los estereotipos, tipos de diagramas y cajas de herramientas (*«toolbox»*), es el momento de crear el fichero *«MDG Technology»* para la definición de procesos software. Para ello, basta con utilizar el asistente[10] que proporciona Enterprise Architect (Figura VI.9.a) e ir introduciendo ciertos parámetros de configuración (Figura VI.9.b).

Una vez finalizado el proceso marcado por el asistente, se genera un fichero XML con la configuración indicada.

[10] Este asistente está disponible en el menú principal de Enterprise Architect: Tools | Generate MDG Technology File.

Finalmente, para utilizar finalmente el perfil, basta con desplegar el fichero generado dentro de la carpeta *«MDGTechnologies»* situada en el directorio de instalación de Enterprise Architect.



(a) **(b)**

Figura VI.9. Asistente EA para la generación del fichero *«MDG Technology»*

3.4. Desarrollando «Add-in» PLM4BS para Enterprise Architect

Tal y como se ha adelantado a lo largo de esta sección, la traslación práctica de todo el entramado teórico de metamodelos y reglas de derivación definidos en los dos capítulos anteriores se traduce en el desarrollo de un nuevo módulo funcional o *plugin* para Enterprise Architect (EA). Esto es lo que se conoce como *«Add-in»* en la terminología de esta herramienta de modelado.

Los *«Add-in»* son un mecanismo para extender la funcionalidad que ofrece EA en base a la utilización de objetos Windows OLE Automation (ActiveX) que exponen métodos de uso público que responde a la interfaz de EA y a determinados eventos. Este mecanismo proporciona diferentes ventajas tales como las siguientes: (*i*) posibilidad para la definición de interfaces gráficas de usuario integradas dentro del entorno EA; (*ii*) recepción de notificaciones sobre eventos de la interfaz de usuario del propio EA, eventos de manipulación de los modelos UML y eventos asociados a la gestión del repositorio de datos, entre otros; (*iii*) proporciona una menor carga de llamadas y una mejor integración en el entorno de EA; y (*iv*) no se requiere configuración por parte del usuario ya que simplemente basta con instalar el *«Add-in»* desarrollado.

En el caso que nos ocupa, el *«Add-in»* PLM4BS incluye el perfil implementado en la Sección 3.3 y además, implementa, por una parte, todos los métodos necesarios para verificar que todas las restricciones OCL especificadas en este trabajo de tesis se satisfacen y, por otra parte, las transformaciones vistas en el capítulo anterior, permitiendo la automatización del proceso de generación del modelo de ejecución y orquestación a partir de la definición del proceso software, y la generación de código ejecutable desde el modelo de ejecución y orquestación. Todo ello, con el fin de ofrecer una respuesta sobre la

automatización del proceso de generación de una versión ejecutable de los procesos software modelados.

Aunque las transformaciones han sido definidas en lenguaje QVT Procedimental, se han implementado en la herramienta PLM4BS en lenguaje C#. La elección de C# no es determinante ya que otro lenguaje de propósito general como Java, Python, etc. podría haber sido utilizado de la misma manera.

Para garantizar la fidelidad del código C# con las transformaciones QVT especificadas en este trabajo de tesis, se han aplicado durante el desarrollo de la herramienta una serie de pasos destinados a garantizar que el código C# realiza el mismo proceso de transformación que el proceso especificado en QVT.

En un primer paso, se ha implementado en lenguaje C# una jerarquía de clases que modela de manera fiel los conceptos de los dos metamodelos presentados para que el código que implementa las transformaciones trabaje con las mismas estructuras de datos, con los mismos atributos y asociaciones que las transformaciones definidas. A modo de ejemplo, la Figura VI.10 muestra las clases C# que implementan las metaclases *«Process»*, *«BusinessVar»*, *«ProcessElements»*, *«Link»* y *«Product»* del metamodelo de definición de procesos software mostrado en el Capítulo IV (en la Figura IV.1).

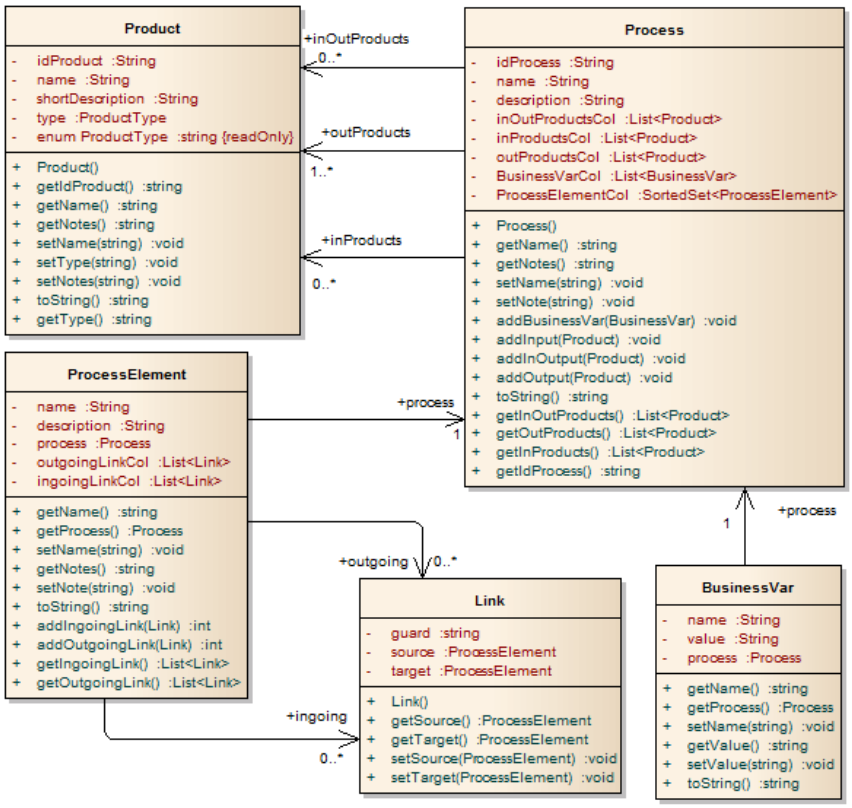


Figura VI.10. Clases C# correspondientes a un fragmento del metamodelo de definición de proceso software[11]

[11] Se ha optado por mostrar un fragmento del diagrama de clases C# para facilitar la comprensión del procedimiento seguido en la implementación del plugin para Enterprise Architect. Sin embargo,

En un segundo paso, se ha definido una estructura de clases y métodos que representan las transformaciones en QVT. Los métodos de dichas clases tienen nombres y argumentos en consonancia con los nombres y argumentos de los mapeos y funciones auxiliares definidas en las transformaciones en QVT. Esto hace que las llamadas y parámetros sean análogos en ambos códigos y, además, permite establecer cierta trazabilidad entre el código QVT y del código C# desarrollado.

A continuación, se muestra un ejemplo de una regla QVT ya vistas anteriormente y su código C# correspondiente. Como se puede apreciar, la codificación en C# es prácticamente directa.

Expresión VI-1. Ejemplo transformación QVT-C#

QVT (Expresión V-7)

```
query SPDefinitionMetamodel::ProcessElement::isInittialActivity () : Boolean
{
      return self.incoming→exist (obj | obj.oclIsTypeOf (InitialElement));
}
```

C#

```
Class Proces {
      //…
      private boolean isInittialActivity(){
          boolean result = false;
          foreach (Link link in super.getIncomingLink ()){
             if (link.getSource () is InitialElement) {      result = true;      break;   }
          }
          return result;
        }
}
```

4. Casos prácticos de aplicación

Uno de los posibles valores añadidos de esta tesis es que ha sido posible validar sus resultados a través de diferentes casos de estudio. El primero y principal es en el entorno de las empresas software, en el que nuestra propuesta ha sido incluida para mejorar el marco de trabajo NDTQ-Framework referenciado en la Sección 3.2 del Capítulo III.

Sin embargo, gracias a que durante todo el trabajo se ha buscado la máxima de facilitar su adaptación en el mayor número de contextos de negocios posible con el propósito de obtener su validación, nos hemos percatado de que la propuesta PLM4BS es lo suficientemente flexible como para ser aplicado en proyectos I+D enmarcados en diferentes contextos de negocio en diferentes organismos y empresas, consiguiendo, de esta manera, obtener experiencias positivas y satisfactorias tras su aplicación. Esto en última instancia ha propiciado la transferencia de los resultados obtenidos hacia el tejido empresarial.

este diagrama de clases ha sido desarrollado en su totalidad y puede ser proporcionado bajo demanda.

Estas experiencias han sido posibles gracias a las estrechas colaboraciones que posee el grupo de investigación Ingeniería Web y Testing Temprano – en cuyo seno se desarrolla esta Tesis Doctoral – de la Universidad Sevilla con diferentes organismos y empresas, tanto públicas como privadas.

Además de los trabajos trasladados a nivel empresarial, los contactos y relaciones establecidas durante este tiempo continúan igualmente abiertos. De manera concreta, los principales contactos – así como los proyectos acometidos – a nivel empresarial han sido:

1. La Consejería de Salud y Bienestar Social de la Junta de Andalucía a través del grupo GiT[12] (Grupo de Investigación Tecnológica) de la Fundación Pública Andaluza para la Gestión de la Investigación en Salud de Sevilla (FISEVI[13]), con el desarrollo del proyecto de adaptación de la plataforma eSalud a una arquitectura basada en procesos SOA («*Service Oriented Architecture*») para permitir una mayor modularidad, independencia, mantenibilidad y la usabilidad del desarrollo de servicios clínicos en esta plataforma.
2. La Agencia de Obra Pública de la Junta de Andalucía con el desarrollo del proyecto THOT, un proyecto enmarcado dentro de la gestión documental aplicada a expedientes de contratación de servicios y obras de infraestructura de transporte de la Junta de Andalucía.
3. La empresa Fujitsu con el que se ha colaborado en el proyecto de soporte metodológico al proyecto SICATA. Este proyecto tiene por objetivo investigar e innovar en el uso de los paradigmas más avanzados en Ingeniería del Software orientado a la web para el desarrollo efectivo de procesos de negocio enmarcados en el contexto biosanitario.

A modo de resumen, la Tabla VI.4 muestra información detallada de cada uno de los proyectos enumerados anteriormente.

[12] GiT es un grupo de investigación relacionado con la I+D+i y las nuevas tecnologías, bajo la coordinación de la Fundación Pública Andaluza para la Gestión de la Investigación en Salud de Sevilla (FISEVI), y en estrecha colaboración con las Unidades de Gestión Clínica (UGC) y las Unidades no es parte Asistenciales de los Hospitales Universitarios Virgen Macarena y Virgen del Rocío, en especial el Servicio de Tecnologías de la Información.

[13] La Fundación FISEVI surge con el objetivo de dar servicio a los investigadores de los centros del Sistema Sanitario Público de Andalucía en la provincia de Sevilla, aglutinando las actividades que anteriormente eran desarrolladas por otras entidades de gestión. Su cartera de servicios se compone de todo un conjunto de actividades de apoyo al desarrollo de la investigación en Salud en el seno del Sistema Sanitario Público de Andalucía, así como a la gestión de ayudas y contratos que permiten las actividades de investigación de los diferentes grupos de los centros de dicho Sistema Sanitario ubicados en la provincia de Sevilla.

Tabla VI.4. Casos prácticos de aplicación de la propuesta PLM4BS

Plataforma de E-Salud: Adaptación de Plataforma a una arquitectura basada en procesos y pilotaje	
Responsable:	María José Escalona Cuaresma
Referencia (Tipo):	P040-13/E09 (Contrato 68/83)
Periodo:	14/03/2013 - 30/09/2013
Coste:	14.995 €
Financiado por:	Fundación FISEVI
THOT. Proyecto de innovación de la Gestión Documental aplicada a expedientes de contratación de servicios y obras de infraestructuras de transportes	
Responsable:	María José Escalona Cuaresma
Referencia:	1561/0493 (Contrato 68/83)
Periodo:	16/04/2012 - 30/06/2014
Coste:	681.000 €
Financiado por:	Agencia de Obra Pública de la Junta de Andalucía
Proyecto de soporte metodológico al proyecto SICATA	
Responsable:	María José Escalona Cuaresma
Referencia:	P079-13/E09
Periodo:	01/10/2013 - 31/10/2014
Coste:	50.110,00€
Financiado por:	Servicio Andaluz de Salud (proveedor Fujitsu)

Como muestra de la aplicabilidad de la propuesta desarrollada en este trabajo de tesis, a continuación se describe cómo ha sido aplicado PLM4BS en el marco de uno de los proyectos mencionados anteriormente. Concretamente, dentro del proyecto de de adaptación de la plataforma eSalud a una arquitectura basada en procesos SOA. A pesar de que este proyecto es el que menor presupuesto ha tenido, su alcance ha permitido abordar todos los aspectos que aporta esta tesis: desde el modelado del proceso hasta su ejecución y despliegue en una plataforma web.

Además, otra razón importante por la que describir este proyecto es por su alto impacto en los resultados obtenidos. En este sentido, el Servicio Andaluz de Salud ha mostrado su interés en este pequeño proyecto para extrapolar los resultados a mayores ámbitos. Además, también ha mostrado interés el SERMA (Servicio Madrileño de Salud). Viendo esta proyección de futuro la empresa SERVINFORM y nuestro grupo de investigación hemos comenzado un proyecto con el que desarrollar un producto

comercializable que englobe los resultados de esta tesis. Este proyecto ha sido financiado por el CDTI (Centro para el Desarrollo Tecnológico Industrial) con un total de 700.000 euros a ejecutar durante 2015 y 2016. Dicho esto, pasamos a describir el proyecto con más detalle.

El término eSalud proviene de la expresión inglesa «*eHealth*» y hace referencia a la tramitación electrónica de procesos clínicos y el intercambio de información utilizando sistemas software de información. Actualmente no existe una definición única y globalmente aceptada del término eSalud, pero, de manera general, eSalud puede ser definido como la aplicación de tecnologías de la información y las comunicaciones al ámbito de la salud para mejorar las herramientas de los actores que intervienen en los procesos socio-sanitarios, con el fin de mejorar la calidad en la atención de los pacientes.

Haciendo suyo el término eSalud, la Junta de Andalucía desarrolló su plataforma de salud electrónica y la identificó como Plataforma eSalud, y es en el contexto de esta plataforma en el que se desarrolla el proyecto descrito en este apartado.

El proyecto [García-García *et al.* 2015b] pretende adaptar la plataforma eSalud del Hospital Universitario Virgen del Rocío de Sevilla a una arquitectura basada en procesos SOA para permitir una mayor modularidad, independencia, mantenibilidad y usabilidad durante el desarrollo de módulos funcionales que proporcionan soporte a los servicios clínicos del hospital. Para alcanzar esta meta, se han perseguido principalmente los siguientes objetivos:

1. Definir un lenguaje lo suficientemente rico semánticamente como para modelar procesos clínicos y a su vez, lo suficientemente flexible como para posibilitar su relación con estándares internacionales de referencia en el desarrollo de sistemas de información de soporte a procesos de asistencia sanitaria – concretamente, los estándares CEN/ISO EN13606 [ISO 2008c] y UNE-EN 13940 [UNE 2007].
2. Definir una sintaxis concreta para utilizar el lenguaje de definición de procesos clínicos y los estándares sanitarios antes mencionados.
3. Definir mecanismos que permitan derivar código ejecutable en la plataforma eSalud del Hospital Universitario Virgen del Rocío a partir del modelado de los procesos clínicos y el modelado de la información clínica necesaria – esta última, siguiendo los estándares antes mencionados.
4. Integrar dentro de una herramienta empresarial de modelado las sintaxis concreta definidas y los mecanismos de derivación planteados. Respecto a este punto, en el proyecto se optó por Enterprise Architect como herramienta base del proyecto.

Respecto a los estándares mencionados, a continuación se expone brevemente a modo de reseña, el propósito de los mismos.

Por una parte, el estándar UNE-EN 13940 define un conjunto de conceptos genéricos necesarios que unifican criterios y términos clínicos, propiciando así una continuidad adecuada durante la atención sanitaria. Este aspecto es relevante ya que mejora la calidad

y la seguridad en la asistencia sanitaria con interoperabilidad semántica, un requisito fundamental para la continuidad de la atención de los profesionales sanitarios hacia los pacientes.

Por otra parte, el estándar CEN/ISO EN13606 define de manera formal y rigurosa un modelo de referencia, que engloba y relaciona un conjunto de conceptos, con el propósito de definir una arquitectura de información que posibilite la transferencia – completa o parcial – de registros electrónicos clínicos (referenciados en la norma por el término EHR, «*Electronic Health Record*») de un único paciente entre diferentes sistemas EHR, o entre un sistema EHR en un repositorio central de datos EHR. Esta arquitectura permite además el intercambio de información entre sistemas con soporte a la toma de decisiones.

La Tabla VI.5 describe brevemente los principales conceptos que define la norma CEN/ISO EN13606. Asimismo, en el contexto del proyecto ha sido importante tener en cuenta el concepto de arquetipo.

Según este estándar, un arquetipo se define como la combinación de elementos de la norma con un significado clínico y con una semántica particular dentro de un contexto sanitario concreto. En otras palabras, un arquetipo es una expresión formal de un concepto de nivel de dominio que expresa restricciones sobre datos clínicos y cuyas instancias se corresponden con elementos de la norma CEN/ISO EN13606.

Tabla VI.5. Principales conceptos de la norma CEN/ISO EN13606

Metaclase	Descripción
«*EHR_EXTRACT*»	Corresponde con el contenedor superior de información dentro de la arquitectura CEN/ISO EN13606 y representa el registro electrónico clínico de un único sujeto de atención sanitaria para hacer posible la comunicación entre diferentes sistemas de gestión de registros clínicos.
«*Folder*»	Dentro de un registro electrónico EHR, esta metaclase es el elemento de alto nivel que permite organizar la información del registro clínico en función de determinadas condiciones clínicas cuya semántica viene determinada por el equipo de profesionales clínicos, institución hospitalaria o durante un período de tiempo fijo (por ejemplo durante el transcurso de una visita clínica).
«*Composition*»	Esta metaclase permite organizar información clínica creada por un agente sanitario como resultado de un encuentro clínico o de una sesión de registro de documentación.
«*Section*»	Datos del registro EHR que pertenecen a un solo encuentro clínico y que por lo general, organiza aquella información que se necesita capturar en la visita.
«*Entry*»	Refleja la información grabada en un registro EHR como resultado de una acción, una observación como una interpretación clínica.
«*Cluster*»	Este concepto permite organizar y gestionar datos clínicos estructurados.
«*Element*»	Representa el último nodo dentro de la arquitectura y permite almacenar el valor específico de un parámetro clínico concreto.

Asimismo, la Figura VI.11 muestra – utilizando para ello la notación de diagramas de clases de UML – las relaciones entre los conceptos descritos en la Tabla VI.5. En esta figura, además, se han obviado las propiedades de las diferentes metaclases con el propósito de

simplificar su representación. En cualquier caso, pueden consultarse en la especificación del estándar.

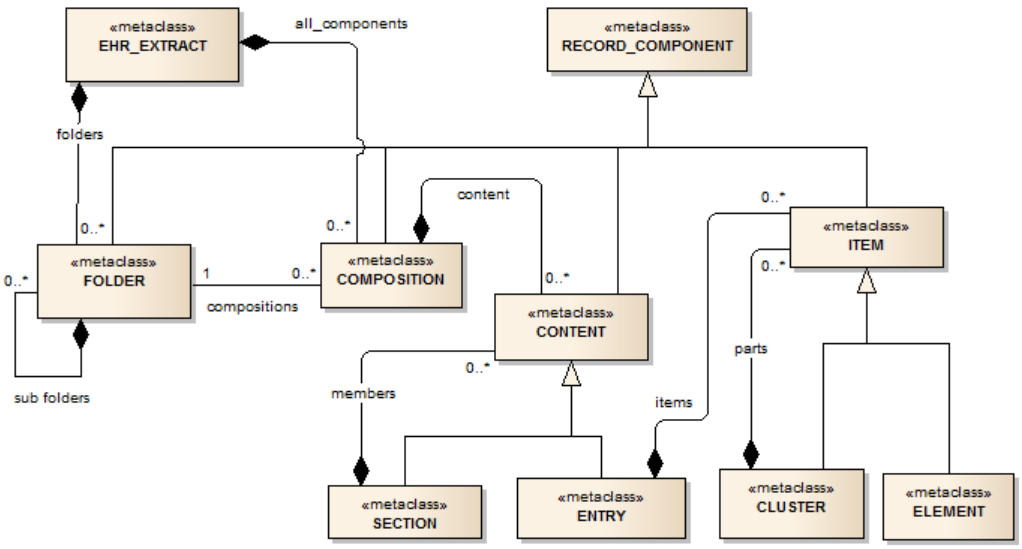


Figura VI.11. Núcleo del metamodelo de la norma CEN/ISO EN13606

Respecto al primer objetivo mencionado al inicio de esta sección, la inclusión de los estándares CEN/ISO EN13606 y UNE-EN 13940 dentro de tareas de modelado de procesos clínicos, ha sido posible gracias a la adaptación del metamodelo de definición de procesos definido en PLM4BS – Figura IV.1 del Capítulo IV de esta tesis –, lo cual a su vez ha posibilitado establecer diferentes relaciones con el metamodelo de la norma CEN/ISO EN13606 – Figura VI.11.

A modo de reseña, el grueso de la adaptación ha consistido en establecer las siguientes relaciones entre el metamodelo de definición de procesos y el modelo de referencia del estándar CEN/ISO EN13606:

— Relación entre la metaclase «*Activity*» del metamodelo de definición de procesos y la metaclase «*Composition*» del modelo de referencia de la norma CEN/ISO EN13606. Esta relación permite establecer qué información clínica debe capturar el profesional sanitario en cada paso del proceso clínico.

— Relación entre las metaclases «*BusinessVar*» y «*Conditional*» (ambas pertenecientes al metamodelo de definición de procesos) con la metaclase «*Element*» de la norma CEN/ISO EN13606. Estas relaciones hacen posible controlar el flujo de ejecución del proceso en función del valor que contenga un determinado parámetro clínico.

A modo de ejemplo, la Figura VI.12 muestra, de manera simplificada, uno de los procesos definidos en este proyecto. Este proceso concreto describe qué pasos o actividades se deben llevar a cabo durante la exploración de un paciente con lesión de médula espinal. En primer lugar, el neurólogo lleva a cabo una exploración inicial (modelada a través de la actividad *«Initial Exploration»*) y más tarde, el neurólogo realiza una exploración final (modelada por medio de la actividad *«Final Exploration»*) siempre que haya finalizado el examen inicial.

Para gestionar la condición mostrada en la Figura VI.12, se ha definido la variable de negocio *«neurological_examination»* con valor inicial *«false»*, de tal manera que cuando la primera actividad haya sido completada el valor de la variable es actualizado a *«true»* posibilitando así, la continuación del proceso. Asimismo, esta variable es comprobada en el elemento condicional nombrado – en el proceso clínico – como *«Has the neurological examination been completed?»*.

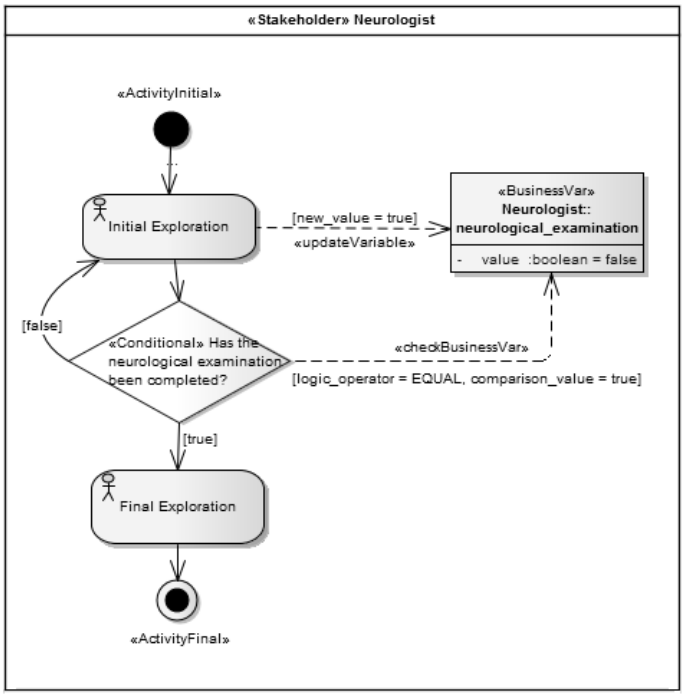


Figura VI.12. Aplicación práctica de PLM4BS dentro del contexto de los procesos clínicos

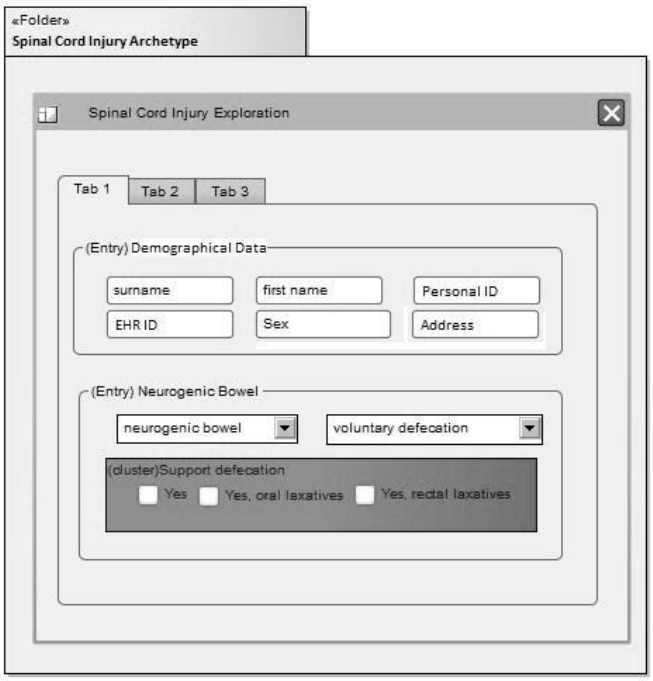


Figura VI.13. Fragmento del arquetipo de lesión medular asociado con la actividad *«Initial Exploration»*

Una vez definido el proceso clínico y antes de pasar a la generación del código ejecutable del módulo de lesión medular para la plataforma eSalud del Hospital Universitario Virgen del Rocío, es necesario el arquetipo de lesión medular y relacionarlo con el proceso definido en la Figura VI.12.

La Figura VI.13 muestra una sección del arquetipo de lesión medular modelado en el proyecto. Concretamente, esta imagen muestra el *«Composition»* – del arquetipo – asociado a la actividad *«Initial Exploration»* del proceso clínico mostrado en la Figura VI.12.

Finalmente, una vez definidas las reglas de derivación necesarias para generar código ejecutable para la plataforma eSalud a partir del modelado del arquetipo y del modelado de proceso clínico e integradas dentro de la herramienta de modelado Enterprise Architect, es posible generar una vista XHTML del módulo de lesión medular y además, alrededor del 80% de la lógica de negocio necesaria para gestionar la información clínica del proceso dentro de la plataforma eSalud.

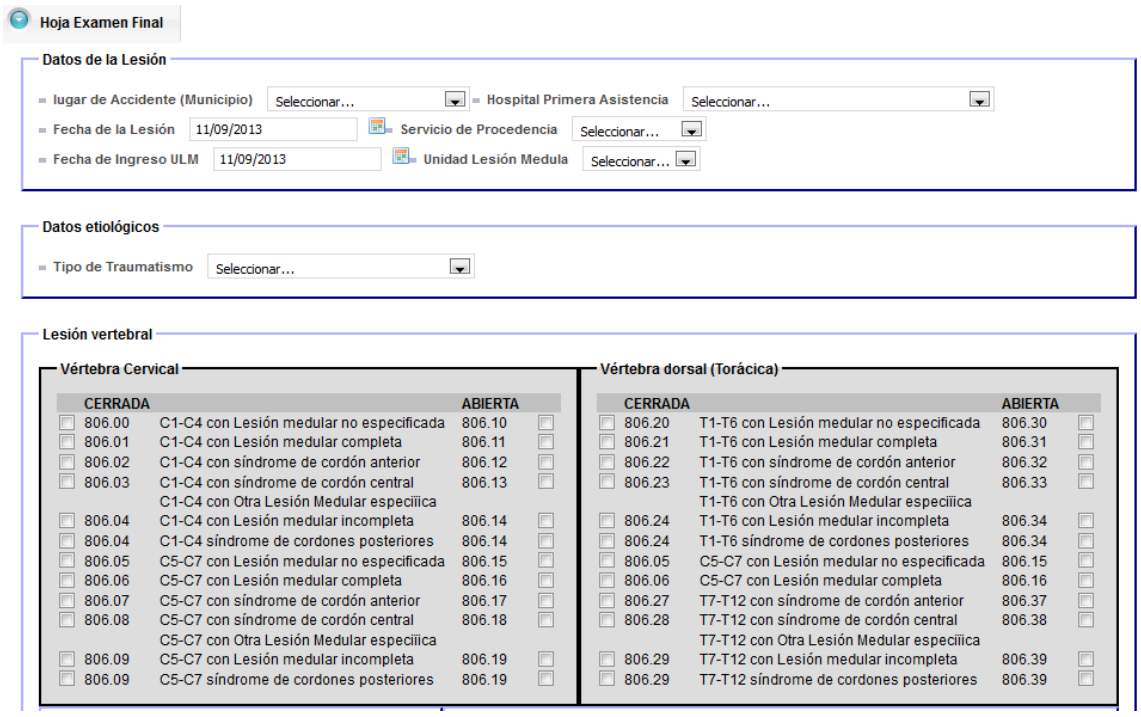


Figura VI.14. Vista XHTML final del arquetipo de lesión medular asociado con la actividad *«Initial Exploration»* dentro de la plataforma eSalud

A modo de reseña final, decir que aunque la propuesta PLM4BS desarrollada en este trabajo de tesis ha sido concebida y orientada hacia procesos software, su aplicación dentro de este contexto ha proporcionado *feedback* para su mejora y actualización en trabajos futuros.

Además, es importante señalar que la línea de trabajo que ha abierto este proyecto sigue abierta actualmente con el propósito de mejorar la aplicabilidad de la propuesta PLM4BS dentro del contexto de los procesos clínicos y mejorar la modularidad, independencia, mantenimiento y usabilidad de este tipo de procesos.

5. Conclusiones

A lo largo del Capítulo IV y del Capítulo V se ha definido de manera formal y completa todo el marco teórico en el que se sustenta la propuesta especificada en este trabajo de tesis.

Sin embargo, para garantizar la viabilidad y la aplicabilidad de este marco de trabajo teórico dentro de entornos prácticos y de un contexto de producción, resulta conveniente y necesario desarrollar una herramienta CASE que le dé soporte para hacer más sencillo el mantenimiento y mejorar la calidad de los resultados. Éste ha sido el propósito de este capítulo y para ello, se han abordado diferentes aspectos.

En primer lugar, este capítulo describe cómo se ha definido la sintaxis concreta de los metamodelos descritos en el Capítulo IV. En este sentido y tal y como se argumenta en la Sección 2.1, la definición de la sintaxis concreta se ha realizado mediante perfiles UML porque, entre otros aspectos, UML es un estándar ampliamente utilizado y conocido en el sector empresarial y además, porque UML proporciona la suficiente flexibilidad y expresividad, así como un mecanismo de extensión con el que construir nuevos lenguajes tomando como base el propio lenguaje UML.

Posteriormente, una vez definidos los perfiles UML necesarios para dar soporte al metamodelo de definición del proceso software y al metamodelo de ejecución y orquestación del proceso, este capítulo presenta la herramienta CASE que da soporte a todo el marco teórico especificado en los dos capítulos anteriores.

Esta herramienta se bautiza con el nombre de PLM4BS – acrónimo de *«Process Lifecycle Management for Business-Software»* – y ha sido diseñada y desarrollada utilizando como base una herramientas CASE de modelado de propósito general para facilitar su aplicación en entornos reales y permitir de esta forma gestionar el ciclo de vida del proceso. Todo ello con el firme propósito de propiciar una posible transferencia tecnológica de los resultados de esta tesis hacia el tejido empresarial.

Siguiendo esta idea como máxima, la Sección 3.1 justifica la elección de la herramienta Enterprise Architect (EA) como la herramienta base elegida sobre la que sustentar la funcionalidad de la herramienta PLM4BS.

En este contexto y explotando la característica de extensión – mediante *«plugins»* – que proporciona Enterprise Architect, PLM4BS se ha diseñado de tal manera que su funcionalidad está integrada dentro del propio entorno EA en base a tres *«plugins»*: (*i*) *«plugin»* de presentación, que agrupa todas las interfaces de usuario necesarias para que el ingeniero de procesos pueda utilizar los perfiles UML e invocar las funcionalidades de PLM4BS de una manera amigable y accesible; (*ii*) *«plugin»* de control de restricciones, encargado de controlar que el ingeniero construye modelos bien definidos conforme al metamodelo de definición del proceso software y al metamodelo de ejecución y

orquestación del proceso, respetando de esta manera todas las restricciones OCL definidas en el Capítulo IV; y (*iii*) un *«plugin»* de ejecución de transformaciones, con el que el usuario puede ejecutar automáticamente todas las reglas de transformación definidas de forma teórica en el Capítulo V.

Finalmente, este capítulo menciona diferentes proyectos reales en los que la propuesta PLM4BS ha sido utilizada y validada. Esta aplicación práctica se ha demostrado útil en diferentes contextos de negocio y entornos de investigación y como muestra de evidencia, la Sección 4 describe en detalle uno de los proyectos en los que se ha aplicado PLM4BS. Concretamente, el proyecto P040-13/E09 – consultar Tabla VI.4 – realizado en colaboración con el grupo de investigación GiT de la fundación FISEVI. La justificación por la que se opta por describir este caso de éxito y no otro, radica en alto impacto y proyección de futuro reconocido y además, que este caso ha permitido aplicar tanto la fase de modelado del proceso como la de ejecución y despliegue en una plataforma específica real.

En la actualidad, y como se presenta en el capítulo siguiente, PLM4BS es un trabajo abierto que ofrece posibles trabajos futuros, nuevas vías de colaboración y nuevas líneas de investigación. Los resultados que hasta ahora ha ofrecido PLM4BS incentiva a continuar con nuevos proyectos de investigación y desarrollo con otras universidades y empresas. Todo ello se detalla de manera más concreta en el capítulo siguiente.

Como conclusión final a este capítulo, decir que PLM4BS, así como los resultados obtenidos de su aplicación, demuestran que los metamodelos teóricos planteados en los capítulos anteriores son de interés real y que, llevados a la práctica, pueden dar muy buenos resultados en el contexto de la gestión de procesos de negocio (orientados o no hacia las empresas software).

Capítulo VII CONTRIBUTION, FUTURE WORK AND CONCLUSIONS

The previous chapters have presented and defined the theoretical and practical bases of a methodological framework to carry out a software process management from a model-based perspective. They have also described the motivations that have laid the foundations of this PhD thesis. In addition, we have analyzed the current situation of model-based proposals dealing with software process modeling, execution and orchestration.

Moreover, this PhD thesis describes a series of metamodels and transformations available to solve this problem from a model-driven engineering perspective. This proposal helps formalize, systematize and automate process management, as it was highlighted in the previous chapter. Accordingly, a tool named PLM$_4$BS was designed and developed to support this theoretical framework focused on metamodels and transformation rules.

Furthermore, the present work discusses and references some studies that have inspired us to achieve our goals. However, one of the aims of this chapter is to point out the specific contributions that this PhD thesis provides. It also describes the research context to frame this study, which constitutes a strategic line for IWT2 (a group referenced as PAIDI TIC021 in the Andalusian Research Plan).

In the end, this chapter advances some future work that continues our research line this work has opened and states final conclusions.

« A PhD thesis has a number of moves like a chess match, but it is necessary
to be able to predict the following moves from the beginning.
The purpose is to checkmate the opponent soon.»

*— **Umberto Eco**, writer and philosopher.*
Alessandria, Italy, 1932

1. Research framework of this PhD thesis

IWT2 has acquired, since its inception, vast experience in transferring research results to companies, due to the execution of individual projects and the collaboration with other public and private companies in numerous R&D and technology transfer projects. Thanks to this experience and relationships with other bodies, IWT2 has been able to identify some needs that may require a creative and innovative solution in the area of business process management (in particular, in the area of software process management).

The present work is conducted in terms of this research line, but it is not the only one. In fact, it is framed in the strategic research line of IWT2, which looks at how the model-driven paradigm is successfully combined with business process management to solve such needs.

This strategic line includes several aspects: on the one hand, to define a theoretical and formal framework to thoroughly justify the existence of the research line. On the other hand, to make a proposal that provides organizations with the capacity to control and «*know-how*» their business processes, by means of a serie of CASE tools that supports all the developed theoretical framework.

In this context, the objectives pursued are: to modernize and implement competitiveness in organizations, and managing their business processes effectively and efficiently in order to ensure quality continuous improvement. Setting these objectives has allowed recognizing their key needs that have being analyzed, through the MDE paradigm, in different PhD theses carried out by members of IWT2. These are further explained below:

The first one consists in providing companies with a mechanism that enhances interoperability among different business process modeling languages so that companies will be able to choose the most appropriate language to meet their current needs. This fact includes several aspects: on the one hand, a minimum flexible, extensible and adaptable business process modeling language that can be used in any business context of an organization; on the other hand, a set of protocols that allow transforming process modeling from a source modeling language to a target modeling language. This aspect is being researched by Laura García Borgoñón, an associated member of IWT2 who works for the Research, Development and Technological Services at the Aragón Institute of Technology. Her PhD thesis is entitled *«A language to define software processes integrated within model-oriented environments»* and it is directed by PhD. María José Escalona Cuaresma and PhD. Manuel Mejías Risoto, both lecturers at the University of Seville and IWT2 members.

The second need deals with business process execution and orchestration, and this is precisely the subject of the PhD thesis described in this document. It defines a model-driven framework based on a transformation protocol to generate an executable version of the business process. It has been identified as WS-BPEL in order to facilitate its deployment on process execution engines.

In the end, the last need is related to provide mechanisms to take out an overall view of business processes from the information specified in relational databases, table structures, constraints and business rules PL/SQL triggers. The goal is to automatically

provide a view of the essential business objects that are involved in all projects an organization carries out, by capturing process composition and its metrics, documents or work teams structure, among other aspects. This point is being researched by Carlos Arévalo Maldonado, another member of IWT2 and lecturer at the University of Seville, whose PhD thesis is entitled *«Extraction of business models to support software processes»* and it is directed by PhD María José Escalona Cuaresma and PhD Isabel Ramos Román, both lecturers at the University of Seville and IWT2 members.

2. Contributions

This PhD thesis is conditioned by premises taken out from other existing work. On the one hand, after studying other model-based approaches in Chapter II, in order to know the advantages and disadvantages they offer, we have realized that each one provides a particular point of view on business process modeling, execution and orchestration. On the other hand, our second source is described in Chapter III and it is firstly related to the NDT methodology (previous work carried out by IWT2), and secondly to the MDE paradigm and the different languages required to work with such a paradigm, for example UML and QVT.

Our proposal considers these influences and finishes with a set of conclusions that presents an original MDE-based solution to support software process modeling, execution and orchestration. This section summarizes these contributions.

2.1. Study of state-of-the-art

Before conceiving an original solution, we must analyze and study the existing proposals dealing with supporting software process modeling, execution and orchestration with the aim of identifying their advantages and disadvantages. This is the first object of study of this PhD thesis.

In this context, we have prepared and submitted a study of the *state-of-the-art* on model-based languages to define, execute and orchestrate software processes. This study was included in [García-Borgoñón *et al.* 2013], which was jointly written with another member of IWT2, whose PhD thesis is described in Section 1 of this chapter. This paper presents a systematic literature review of different kinds of process modeling languages. For example, this paper includes model-based languages to describe processes, such as PetriNets-based languages, rules-based languages and programming languages, among others.

In addition, we have carried out a review of existing proposals that define what phases the business process lifecycle demmands. It has been included in our paper entitled «*A model-based Approach for Software Processes Modeling in PLM$_4$BS*» [García-García *et al.* 2015a], which is today under review in the «*Computer Standards & Interfaces*» journal.

This contribution covers our first goal defined in Chapter III.

2.2. MDE-based framework to software process management

One of the most important contributions of this PhD thesis is the theoretical framework presented in previous chapters. This framework has been defined in a formal and complete manner in order to effectively manage (especially, model, execute and orchestrate) software processes. Although our proposal only covers software process modeling, execution and orchestration, it must be added that PLM_4BS is an open proposal that offers interesting future work, as Section 3 describes.

In this context, this PhD thesis proposes a model-driven framework to ultimately generate the executable version from software process model. For this purpose, our framework is characterized by two distinct theoretical parts: (*i*) conceptual and abstract definition of our problem domain, that is, software process modeling and execution; and (*ii*) definition of a systematic and automatic protocol that allows executing the conceptual definition of software process.

Conceptual definition of software process

The conceptual definition of our theoretical framework is composed of a set of metamodels that supports software process modeling, execution and orchestration. This contribution covers the second goal defined in Chapter III.

It is worth hightlighting that this PhD thesis specifically provides the formal definition of a software process definition metamodel and a software process execution and orchestration metamodel, both defined in compliance with the guidelines and recommendations of ISO/IEC TR 24744. The former is included in the paper entitled «*A model-based Approach for Software Processes Modelling in PLM4BS»* [García-García *et al.* 2015a] and the latter is included in *«A Model-Based Proposal to Define the Execution Context during the Software Process Lifecycle Management*» [García-García *et al.* 2014c], which is under review in the «*ACM Transactions on Software Engineering and Methodology»* journal.

In addition, this work also offers a concrete syntax of each metamodel based on UML Profiles [García-García *et al.* 2015a; 2014c] in order to instance each metamodel. For this purpose, we have selected UML standard modeling language and specifically, its classes and activities diagrams. This UML Profile also covers the fourth goal of this thesis.

Defining a systematic and automatic protocol

The second cornerstone of the aforementioned theoretical framework is the protocol that allows systematizing and automating the generation of an executable code, from a process model defined in terms of the conceptual models described above. This contribution also copes with the third goal outlined in Chapter III.

This systematic transformation protocol [García-García *et al.* 2014c] is derived and defined from the relationships established between our two metamodels. All these relationships and our protocol themselves are original contributions of this PhD thesis. Basically, we have specified and defined formal processes so as to obtain the software process execution and orchestration model from the software process definition model,

and WS-BPEL executable code from the software process execution and orchestration model. In consequence, we have defined *model-to-model* and *model-to-text* transformation rules through QVT and MOFM2T, respectively.

2.3. Supporting tool of our theoretical framework

One of the main motivations of this dissertation is to propose a solution that can work in real business contexts. This motivation is the third major contribution of this thesis: to design and develop a CASE tool that supports our theoretical framework. Thus, this contribution covers the fourth goal advanced in Chapter III.

Our CASE tool shares the acronym with our theoretical proposal, that is PLM$_4$BS, and as Appendix B describes, users can instance our theoretical metamodels and execute automatically our systematic protocol in a practical, user-friendly and usable environment. We have chosen to integrate our tool into EA to facilitate the applicability of PLM$_4$BS in companies. Ultimately, this environment enables raising a systematic transformation protocol to an automated protocol, thereby increasing the strength and applicability of these processes.

2.4. Validation of PLM$_4$BS and its strengths against other proposals

Our fifth and last goal described in Chapter III is to evaluate of our model-driven proposal in real environments. This goal goal has been achieved thanks to the opportunity and possibility to instantiate the results of this PhD thesis in different real projects from several business contexts. Section 4, Chapter VI, comments on these experiences.

This contribution analyzes the feasibility of PLM$_4$BS and expands the application scope of PLM$_4$BS to other non-software contexts. In addition, the instantiation of PLM$_4$BS in factual projects has uncovered several areas of improvement or new future work that will undoubtedly extends the possible trajectory and applicability of the results obtained.

Finally, once our contributions have been explained, it is interesting to compare our PLM$_4$BS proposal with the related work (see Chapter II) in order to know its advantages regarding the existing knowledge of the subject. In this sense, Table VII.1 shows the results of this comparison using the characterization scheme in Chapter II.

After carrying out this comparative study, we can now highlight a set of strengths of PLM$_4$BS compared with other proposals. They are briefly explained below again, although they have been described and justified along this thesis:

Firstly, our proposal provides mechanisms to define business rules. In this sense, we need to establish mechanisms to control the flow of a process during its execution. This control is performed through business variables and business rules understood as those that are created along the process modeling phase to define the behavior of certain elements, such as conditional elements (which establishes what path is taken depending on the condition evaluated).

Table VII.1. Summary of the characterization squeme evaluation

	Expressiveness													
	Activity	Conditional	Parallel Branches	Exception	Product	Roles	Conformity to UML	Understandability	Granularity	Executability	Measurability	Orchestrability	Tools Support	Business Rules
PLM$_4$BS	×	×	×		×	×	×	×	×	×	×	×	×	×
Chou	×		×	×	×	×	×	×	×	×				
Di Nitto	×	×	×		×	×	×	×	×	×		×	×	
UML-EWM	×	×			×	×	×	×	×				×	
UML4SPM	×	×	×	×	×	×	×	×	×	×			×	
Combemale	×	×			×	×	×	×	×					
UPME	×				×	×	×		×	×			×	
FlexUML	×	×	×		×	×	×	×					×	
Ferreira	×						×		×	×		×		
SPEM2.0	×	×	×		×	×	×	×	×		×		×	
xSPEM	×	×	×		×	×	×	×	×	×	×		×	
eSPEM	×	×	×		×	×	×	×	×		×		×	
MODAL	×	×	×		×	×	×	×	×	×	×		×	
Ellner	×	×	×	×	×	×	×	×	×	×				

Secondly, any company must achieve an effective business processes management, and also implement productivity and competitiveness in relation to other organizations of the same business area so as to continuously improve processes. In this sense, PLM$_4$BS takes into account this need and provides mechanisms to support the continuous improvement of a business process through metrics and indicators. These mechanisms provide a starting point for some of our future work (specifically, that described in Section 3.2 of this chapter), although they initially support the process monitoring phase (the third phase of our business process lifecycle described in Chapter III).

Finally, PLM$_4$BS provides mechanisms to define the process orchestration, which is very important as users frequently interact with different information systems by managing, sharing and storing information at their organizations. When these tasks are modeled in business process (and essentially in software process), the process engineer requires mechanisms and elements to identify when the process requires information from other information systems. It is also compulsory to supply these elements with the necessary dynamic semantics to their later execution.

3. Future work and new research lines

After describing our contributions, it is convenient to put forward that this work is not finished. In fact, the results open new lines of research that have been introduced throughout this document as possible short and long-term open points. Futhermore, this section describes our future work in detail.

3.1. Improving internal processes of the NDT methodology using PLM$_4$BS

As previously mentioned, Section 3 of Chapter III describes some problems related to quality assurance when methodologies, such as NDT, are used in software projects. In some cases, the execution of methodological phases is often considered as a useless formality that can ultimately means delays, changes or code patches. For this reason, inconsistencies between documentation and final system are not unusual.

NDT has evolved to provide a model-based framework named NDTQ-Framework, in order to tackle this problem and support software projects. This framework encourages new approaches, standards and paradigms for software quality development. Consequently, a model-based solution is presented. NDTQ-Framework defines formally a set of groups of processes to manage software processes, but it has a drawback, that is, it is limited to the formal definition of these processes.

It is necessary to optimize performance, even though PLM$_4$BS has been integrated into NDTQ-Framework to improve and enhance applicability. In light of this, we have obtained successful preliminary results when applying our proposal to the group of software process development (specifically, requirements engineering process and reconciling requirements). These results have been achieved thanks to the close relations with the Laboratory for Research and Advanced Training in Computer Science at the National University of La Plata (Argentina). It is described in Section 4 of this chapter.

3.2. Defining MDE mechanisms to achieve an effective process monitoring

This future line of work is seemed as one of the most important challenges presented in this PhD thesis for two reasons: it could enable the definition of mechanisms for business process monitoring and allow carrying out an effective continuous improvement of software processes at organizations. Therefore, it will be necessary to consider the definition of metrics and indicators to achieve an effective monitoring. This aspect should be studied particularly and deeply, although it has been initially included in our software processes definition metamodel described in Chapter IV.

This research line is related to the implementation of key performance indicators, both static and dynamic, in order to assess software processes performance. Our interest on this future work is mainly motivated by the growing need to design more reliable increasingly complex software systems with high level of quality with the aim of improving process maintenance, monitoring and management.

Moreover, over last decade, software organizations are aware of the need to implement well-defined processes and control the performance of these processes. The final purpose is to increase their level of maturity and their software products design with enough level of quality [Fuggetta 2000]. Thus, we must generate mechanisms to measure business processes, mainly software processes, in order to continuously better their performance.

In this context, we propose to investigate how software process monitoring can be included in PLM$_4$BS using the MDE paradigm. For this purpose, it is necessary to explore this research line in terms of a monitoring metamodel definition and a model-driven protocol that allows obtaining it from a process definition model, and an execution and

orchestration model. In addition, we must design and develop tools that support this theoretical framework and improve the applicability of our proposal. Regarding executable code, there is a need to transform the monitoring model in code interpreted by *«Business Intelligence»*[14] tools.

Figure VII.1 shows a first approximation of our solution, which is integrated into PLM$_4$BS, as it is before described in Figure I.1 of Chapter I. This initial solution supports the monitoring phase of our business process lifecycle and continuous improvement, as represented in Figure III.4 of Chapter III.

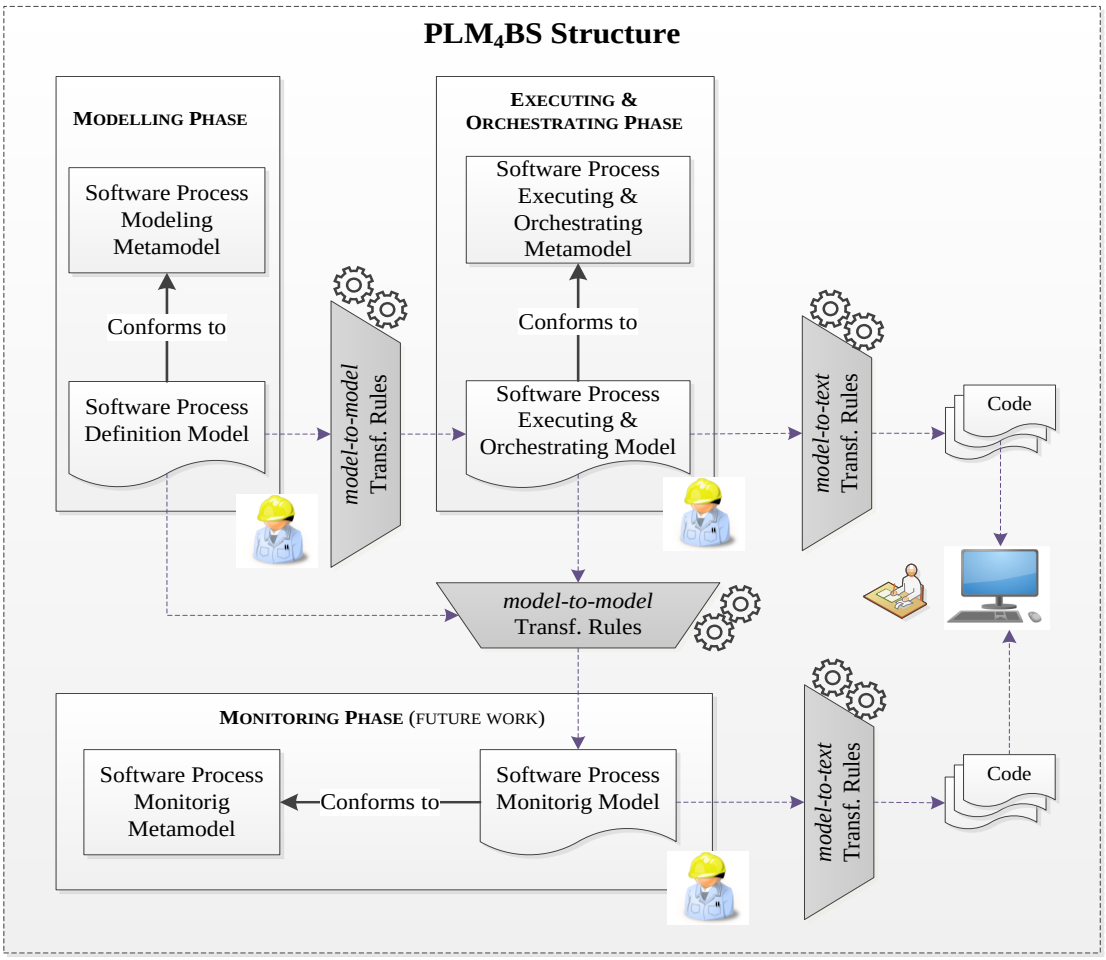


Figure VII.1. Extension of PLM$_4$BS to include process monitoring mechanisms.

3.3. Including cross-organizational rules in process management

Another interesting aspect is to look at the introduction of mechanisms to define organizational business rules in PLM$_4$BS through process management, but from a transversal point of view.

[14] *«Business Intelligence»* is the set of techniques and tools for the transformation of raw data into meaningful and useful information for business analysis purposes. Pentaho is an example of this kind of business tools. Reference: www.pentaho.com.

All organizations have universal business rules that affect the achievement of certain organizational business goals, that means, government and performance of all processes. These rules must be conceived as a set of general-application guidelines valid and mandatory on all processes. In fact, these rules should be defined and kept independently of process models. However, the existence of organizational business rules derives from the conception of the company as an economic-social entity itself. The main objective of organizational business rules is to define policies and operating modes in the organization in a granular way

At present, there are organizations dedicated to study and establish standards so as to agree on the definition of organizational business rules. In this sense, some significative specifications have emerged, such as the OMG's proposal named *«Semantics of Business Vocabulary & Business Rules»,* [OMG 2013b].

3.4. Defining MDE mechanisms to support quality assurance during process management

Chapter V advances this future work when the systematic transformation protocol among models is described. Chapter V describes a set of derivation rules that the process engineer must handle manually to obtain the final version of models according to his/her experience and knowledge.

In this sense and influenced by the NDT methodology, exactly the NDT-Quality tool, it is relevant to define a set of mechanisms, protocols and supporting tools that automate the verification of models and ensure quality, consistency and integrity at all times, particularly, when the process engineer makes manual changes in models to obtain final versions.

3.5. Defining MDE mechanisms to support software processes testing techniques

Business processes, especially software processes, constitute an important asset for companies. However, in many cases there exist differences between process model and process execution. Organizations spend time and effort to find these disagreements.

In this context, the use of software-testing techniques could be a feasible alternative to reduce costs. In fact, we have proposed an initial approach [García-Borgoñón *et al.* 2014] that shows how such techniques can be applied to assess compliance of the software process execution and the process model. This paper has been developed jointly with members of the Software Engineering Research Group at the University of Oviedo and members of the Research, Development and Technological Services of the Aragón Technology Institute. These relationships are described in Section 4 of this chapter, respectively.

Although this future line of work is conceived as a research line itself, it is closely related to our second future line of work, because if combined, process monitoring and process testing can become a research job whose results may enable the definition of mechanisms for continuous improvement. This way, organizations could be able to identify problems in advance and facilitate the decision-making process during processes execution.

3.6. Extending the application of PLM4BS to other business contexts

It seems interesting to extend PLM4BS to other non-software contexts, even though this PhD thesis is focused on providing a model-based solution for software process management, specifically process modeling phase and process execution and orchestration phase. Thus, PLM4BS can be applied to other contexts.

In fact, Chapter VI sets out some successful experiences in real projects. In any case, our proposal can be applied to other contexts such as aeronautical environments, where there is a strong coupling of the models that define manufacturing processes with PLM tools (e.g., Siemens's PLM solution[15] or Dassault Systèmes's PLM solution[16]) used to define and handle such models. If we could apply PLM4BS to these environments, aeronautical companies would not be forced to use a specific commercial PLM tool and they might be able to migrate their manufacturing processes to different PLM solutions, through MDE mechanisms. We have identified this need in CALIPSOneo project [Escalona *et al.* 2013a; Salido *et-al.* 2014], which has been jointly developed by IWT2 and Airbus Military[17] (EADS CASA).

CALIPSOneo (*«advanCed Aeronautical soLutIons using Plm proceSses & tOols»*) project ended in 2013 and it dealt with classifying PLM methodologies that help industrial engineers define, simulate, optimize and validate aeronautical assembly processes in a 3D virtual environment, before implementing these processes in a real shop floor.

Along the years, several PLM analyses carried out by Airbus Military concluded with a lot of benefits for the company [Mas 2012]. From this knowledge, CALIPSOneo work teams conducted an exhaustive requirements capture phase, requirement management plan and traceability matrix of requirements in order to satisfy Airbus' needs for PLM tools. Finally, CALIPSOneo categorized a new PLM methodology to fit a collaborative PLM solution and develop a software solution that would provide support for this collaborative concept.

3.7. Defining a process execution engine for PLM4BS

As mentioned along this PhD thesis, especially in Chapter V, UML provides comprehension and expressiveness. Both features are based on the UML graphical representation and high-level builders to extend UML itself. In addition, UML provides concepts with execution semantics (e.g., the *«Activity»* and *«Action»* metaclasses). Nevertheless, at

---

[15] *«Siemens PLM Software»* is a computer software company specialized in 3D & 2D Product Lifecycle Management software. The company is a business unit of *«Siemens Industry Automation»* division, and is headquartered in Plano, Texas, United States. Website: http://www.plm.automation.siemens.com/en_gb/

[16] *«Dassault Systèmes»* is a French software company dealing with the production of 3D design software, 3D digital mock-up and Product Lifecycle Management solutions. It also offers social, collaborative, informative and intelligent products. Website: http://www.3ds.com/

[17] *«Airbus Group»* is a European multinational aerospace and defence corporation registered in the Netherlands and headquartered in Toulouse, France. *«Airbus Group»* was originally constituted as the *«European Aeronautic Defence and Space (EADS)»* on 10th July 2000 by the merger of *«Aérospatiale-Matra»* France, *«Dornier GmbH», «DaimlerChrysler Aerospace AG (DASA)»* and *«Construcciones Aeronáuticas SA (CASA)»*. Website: http://www.eads.com/eads/int/en.html

present there are no virtual machines, interpreters or compilers to run UML models (such as models conformed to our metamodels defined in Chapter IV).

For these reasons, it was necessary to define alternative mechanisms that transfer our theoretical framework to real environments and allow performing our software process execution metamodel. Our proposal is to generate WS-BPEL/BPEL4People code from the execution model.

We plan to define and develop a process execution engine for the proposed PLM_4BS according to this future line of work, so that our process models can be directly executed.

4. Relations with other research groups and business environment

There have been many collaborations with other research groups as well as relationships established with a lot of companies, thanks to the author of the present dissertation has participated in many R&D and technological transfer projects during the development of this thesis.

Regarding other research groups, the relationships this work has encouraged are strengthening everyday, by promoting new lines of research with other groups and business environments. Some of these relationships are described below:

— **Group of Technological Innovation (GiT) of the FISEVI foundation**. Section 4 Chapter VI describes how the results of this PhD thesis have been validated in a real project [García-García *et al.* 2015b]. The goal of this project was to adapt the eHealth platform of the Virgen del Rocío University Hospital in Seville to a process-based SOA architecture in order to enable greater modularity, independence, maintainability and usability to develop functional modules that provide support to the clinical services of this hospital.

IWT2, where this PhD thesis is developed, worked closely with the GiT group in this project. GiT is an R&D and new technologies research group, coordinated by the R&D Leader and the Manager of the Andalusian Public Foundation for the Health Research Management in Seville (FISEVI). GiT is also associated with the Units of Clinical Management (UGC) and the Units of non-Relief Management of the Virgen Macarena University Hospital and the Virgen del Rocío University Hospital, both in Seville. GiT is also associated with the Department of Information Technology of these hospitals.

Over the last year, the author of the present work has collaborated with the GiT group in the following papers:

- [García-García *et al.* 2015b]: *«Improving patient care through a clinical process management based on the MDE paradigm»*. Submitted to *«The Scientific World Journal»*, impact factor 1.219 (2013). Under review.

- [Martínez-García *et al.* 2014]: *«Working with the HL7 metamodel in a Model Driven Engineering context»*. Submitted to *«Journal of Medical Systems»*, impact factor 1.372 (2013). Under review.

— **Software Engineering Research Group (GIIS) at the University of Oviedo.** Some future lines of work described in the previous section apply software-testing techniques in order to assess compliance of a software process execution in terms of the process definition, as well as the designed model. This path is currently working and has promoted closer partnerships with the GIIS group. The research lines of GIIS are focused on the definition of methodologies and tools for software quality, and particularly, those specialized in software testing.

The author of this PhD thesis has collaborated with the GIIS group in the following paper, which is directly related to the content of his work.

- [García-Borgoñón *et al.* 2014]: *«Applying testing techniques to software process assessment: A model-based perspective».* Submitted and published in *«22nd International Conference on Information Systems Development, ISD 2013».* Ranking CORE 2013: A+.

— **Research and Training in Advanced Computing Laboratory (LIFIA) of the National University of La Plata, Argentina**. As mentioned, a research to improve some methodological processes of the NDT methodology has been done, especially the requirement engineering process for the conciliation of requirements. This research was carried out jointly with the Research and Training in Advanced Computing Laboratory (LIFIA) of the National University of La Plata (Argentina) and has embraced closer collaborative ties between our groups. This relationship has led to the implementation of cooperation agreements on stays and international R&D projects.

Lately, some members of IWT2 have written the following papers in liaison with the LIFIA group:

- [García-García *et al.* 2012c] *«NDT-merge: a future tool for conciliating software requirements in MDE environments»*. Published in *«iiWAS 2012, International Conference on Information Integration and Web-based Applications & Services»*. Ranking CORE 2013: C.
- [Escalona *et al.* 2013b] *«Detecting Web requirements conflicts and inconsistencies under a model-based perspective»*. Published in *«Journal of Systems and Software»*. Impact factor: 1.135.
- [García-García *et al.* 2014b] *«Detecting Functional Requirements Inconsistencies within Multi-Teams Projects framed into a Model-based Methodology»*. In process to submit to *«IEEE Transactions on Reliability - Special Section on Software Quality Assurance»*.

— **Human and Artificial Cognition Laboratory of Paris 8 University, *«Laboratoire Cognitions Humaine et Artificielle, Universite Paris 8»*.** This research group is composed of an interdisciplinary team that studies natural and artificial cognitive systems. Accordingly, it uses different pragmatic and semantic studies in order to model cognitive processes involved in learning throughout an individual's life, from infancy to old age.

  In addition, IWT2 and this French research group jointly analyze how information technology and communication can be applied to education and neuropsychology. This collaboration agreement has allowed framing this PhD thesis in the Merimée project (see Section 4.1, Annex C), which motivates collaborations between PhD students of both groups. Personally, the author of this dissertation experienced a three-day stay in Paris in the context of this project.

Moreover, as previously put forward, we have been able to apply, test and validate our PLM$_4$BS framework and supporting CASE tool in real projects. This has also enhanced collaborative ties with a lot of software and non-software companies, both public and private. Such agreements open new lines of work and a wide range of possibilities to apply the results of this PhD thesis.

Some of the relationships established with private and public bodies are further detailed below:

— The **Andalusian Health Service** through the Group of Technological Innovation (GiT) of the FISEVI foundation. This relationship has taken place thanks to the results of the project described in Section 4 of Chapter VI. In addition, it has allowed planning new lines of work and R&D projects in order to continue our previous project (i.e., the project described in Chapter VI regarding the eHealth platform). The new projects aim to combine the MDE paradigm and clinical reference standards so as to improve the clinical and healthcare processes management.

— The **Public Works Agency of the Junta de Andalucía (AOPJA)**. It is a public body associated with the Regional Ministry of Public Works and Housing of the Junta de Andalucía, with which IWT2 has worked in liaison in the THOT project [Alba *et al.* 2014]. This project enabled, among other aspects, to apply PLM$_4$BS to administrative process modeling linked to different types of contracts, regularly issued by AOPJA. This project is mentioned in Chapter VI.

— The **Regional Ministry of Education, Culture and Sport of the Junta de Andalucía** with which IWT2 works since 2002 in numerous agreements and R&D projects. The author of this dissertation has participated over the last years in many of these projects.

— **Fujitsu**, which works in close collaboration with IWT2 on several R&D projects in different areas of business (e.g., in healthcare areas and clinical process management, among others).

— The **Department of Research, Development and Technological Services of the Technological Institute of Aragon (ITAINNOVA)**. IWT2 has collaborated with ITAINNOVA for many years, what has entailed a lot of cooperation agreements as well as numerous papers related to software process management.

All work and contributions listed above have been successfully carried out and have given very good results. Such results have promoted new vias of collaboration with research groups and business organizations.

5. Conclusions

Previous chapters of this PhD thesis hold up the work where we present and define a theoretical and practical framework. They also describe a work based on a real need in organizations (Chapter I), that later has turned into a specific problem (Chapter III) derived from the results and conclusions obtained after studying the *state-of-the-art* (Chapter II).

Once the context has been specified, the remaining PhD thesis introduces a model-driven approach to software process modeling, execution and orchestration, structured into two different theoretical parts. The first one is composed of a set of metamodels (Chapter IV) to support the process modeling, execution and orchestration, and a set of transformation mechanisms (Chapter V). The second part deals with applying the theoretical framework to real environments. To conclude, we have defined a theoretical-practical framework to software process management from a model-based perspective.

Therefore, this approach takes the form of a supporting tool named PLM_4BS – acronym of *«Process Lifecycle Management for Business-Software»* –, which allows modeling and executing software processes. Chapter VI shows how this tool has been designed and described. It also presents some case studies in which our proposal has been successfully tested and validated in authentic practical environments.

This PhD thesis offers a new and original model-driven approach to support the software process definition and execution and orchestration context. Our approach carries out a process lifecycle taking into account an overall continuous improvement process. The main goal is to improve and facilitate the applicability of business process management in software organizations in a global framework of quality and efficiency.

REFERENCIAS BIBLIOGRÁFICAS

[Acuña *et al.* 2013] Acuña S.T., Juristo N. *Software Process Modelling*. International Series in Software Engineering, Springer Verlag, vol. 10, isbn: 0-387-24261-9, University Software Engineering Institute. 2013.

[Alba *et al.* 2013] Alba, M., Ramírez, M., Pavon, I., Sanchez, N., García-García, J.A. *Comparativa de Herramientas ECM en el marco de la e-administración*. V Congreso Internacional de Computación y Telecomunicaciones, pp 24-31. ISBN: 978-612-4050-69-5. 2013.

[Ardagna *et al.* 2008] Ardagna D., Ghezzi C., R. Mirandola. *Rethinking the Use of Models in Software Architecture*. Quality of Software Architectures Models and Architectures, vol. 5281, S. Becker, F. Plasil, and R. Reussner, Eds. Springer, pp. 1-27. 2008.

[Armario *et al.* 2012] Armario, J., Gutiérrez, J., Alba, M., García-García, J. A., Vitorio, J., Escalona, M. J. *Project Estimation with NDT*. In S. Hammoudi, M. van Sinderen & J. Cordeiro (eds.), ICSOFT (p./pp. 120-126). ISBN: 978-989-8565-19-8. 2012.

[Bandara *et al.* 2005] Bandara W., Gable G., Rosemann M. *Factors and measures of business process modelling: model building through a multiple case study*. European Journal of Information Systems 14, 347–360. doi:10.1057/palgrave.ejis.3000546. 2005.

[Bendoukha *et al.* 2012] Bendoukha, H., Slimani, Y., Benyettou, A. *UML Refinement for Mapping UML Activity Diagrams into BPEL Specifications to Compose Service-Oriented Workflows*. In Networked Digital Technologies (pp. 537-548). Springer Berlin Heidelberg. 2012.

[Bendraou *et al.* 2005] Bendraou, R., Gervais, M.P., Blanc, X. *UML4SPM: A uml2.0-based metamodel for software process Modelling*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3713 LNCS, Montego Bay, Jamaica, pp. 17- 38, 2005.

[Bendraou *et al.* 2006] Bendraou, R., Gervais, M.P., Blanc, X. *UML4SPM: An Executable Software Process Modelling Language Providing High-Level Abstractions*. Enterprise Distributed Object Computing Conference, EDOC 06 10th IEEE International, vol. 6, no. 511731, pp. 297-306, 2006.

[Bendraou *et al.* 2007a] Bendraou, R., Combemale, B., Cregut, X., Gervais, M. *Definition of an executable SPEM 2.0*. Software Engineering Conference, APSEC 2007 14th Asia-Pacific, IEEE, pp. 390-397, 2007a.

[Bendraou *et al.* 2007b] Bendraou R., Sadovykh A., Gervais Marie-Pierre, Blanc X. *Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach*. Proceeding EUROMICRO '07 Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp 314-321, ISBN:0-7695-2977-1. 2007b.

[Bendraou *et al.* 2009] Bendraou, R., Jezequel, J.M., Fleurey, F. *Combining aspect and model-driven engineering approaches for software process modelling and execution*.

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5543 LNCS, Vancouver, BC, Canada, pp. 148 -160, 2009.

[Bendraou *et al.* 2010] Bendraou R., Jézéquel J.M., Gervais M.P.M.P., Blanc X. *A Comparison of Six UML-Based Languages for Software Process Modeling*, IEEE Transactions on Software Engineering 36, pp 662–675. 2010.

[Bézivin 2005] Bézivin J. *On the Unification Power of Models*. Software and Systems Modeling (SoSyM.), vol. 4(2), pp. 171–188, 2005.

[Bosch 2009] Bosch, J. *From Software Product Lines to Software Ecosystems*. SPLC '09 Proceedings of the 13th International Software Product Line Conference, pp. 111-119, 2009.

[Chou 2002] Chou, S.-C. *A Process Modeling Language Consisting of High Level UML-based Diagrams and Low Level Process Language*. Journal of Object Technology, vol. 1, no. 4, pp. 137-163. 2002.

[Chrissis *et al.* 2011] Chrissis M.B., Konrad M., Shrum S. *CMMI® for Development: Guidelines for Process Integration and Product Improvement*. Editorial Pearson Education, 2011.

[Combemale *et al.* 2006] Combemale, B., Cregut, X., Caplain, A., Coulette, B. *Towards a rigorous process modelling with SPEM*. ICEIS 2006, 8th International Conference on Enterprise Information Systems, pp. 530-533, 2006.

[Cugola *et al.* 2001] Cugola G., Di Nitto E., Fuggetta A. A *The JEDI Event based Infrastructure and its Application to the Development of the OPSS WFMS*. IEEE Transactions on Software Engineering, 2001.

[Cutilla *et al.* 2011] Cutilla C. R., García-García J. A., Alba M., Escalona M.J., Ponce J., Rodríguez L. *Aplicación del paradigma MDE para la generación de pruebas funcionales; Experiencia dentro del proyecto AQUA-WS*. Presented at the 6ª Conferência Ibérica de Sistemas e Tecnologias de Informação, ISBN: 978-989-96247-4-0. 2011.

[Debnath *et al.* 2006] Debnath, N., Riesco, D., Montejano, G., Cota, M.P., Baltasar Garcia, J., Perez-Schoeld, Romero, D., Uva, M. *Supporting the SPEM with a UML extended workflow metamodel*. IEEE International Conference on Computer Systems and Applications, vol. 2006, pp. 1151-1154, 2006.

[Di Nitto *et al.* 2002] Di Nitto E., Lavazza L., Schiavoni M., Tracanella E., Trombetta M. *Deriving executable process descriptions from UML*. Proceedings of the 24th International Conference on Software Engineering ICSE 2002, 2002, vol. Compendex, pp. 155-165. 2002.

[Ellner *et al.* 2010] Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., Philippsen, M. *eSPEM – A SPEM Extension for Enactable Behavior Modelling. Modelling Foundations and Applications*, vol. 6138, T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, Eds. Springer Berlin/Heidelberg, pp. 116-131, 2010.

[Ellner *et al.* 2011] Ellner R., Al-Hilank S., Drexler J., Jung M., Kips D., Philippsen. *A FUML-based distributed execution machine for enacting software process models*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6698 LNCS, pp 19–34. 2011.

[Escalona *et al.* 2003] Escalona MJ, Torres J, Mejías M., Reina-Quintero, AM. *NDT-TOOL: a Case Tool to Deal with Requirements in Web Information Systems*. Lecture Notes in Computer Science. Vol. 2722. Pag. 212-213. 2003.

[Escalona *et al.* 2004] Escalona MJ. *Modelos y técnicas para la especificación y el análisis de la navegación en sistemas software*. Ph. European Thesis. Departament of Computer Language and Systems. University of Seville. Seville, Spain. 2004.

[Escalona *et al.* 2008] Escalona MJ, Aragón G. *NDT. A model-driven approach for web requirements*. IEEE Transactions on software engineering, vol: 34, pp. 377-394. 2008.

[Escalona *et al.* 2013a] Escalona M.J., Garcia-Garcia J. A., Mas F., Oliva M., Del Valle C. *Applying model-driven paradigm: CALIPSOneo experience*. Proceedings of the Industrial Track of the Conference on Advanced Information Systems Engineering 2013 (CAiSE'13), vol. 1017, pp. 25-32. 2013.

[Escalona *et al.* 2013b] Escalona M.J., Urbieta, M., Rossi, G., Garcia-Garcia, J.A., Luna E.R. *Detecting Web requirements conflicts and inconsistencies under a model-based perspective*. Journal of Systems and Software, 86(12), 3024-3038. 2013.

[Faleh *et al.* 2011] Faleh, M. N. M., Bochmann, G. V. *Transforming dynamic behavior specifications from activity diagrams to BPEL*. In Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on (pp. 305-311). IEEE. 2011.

[Ferreira *et al.* 2011] Ferreira, A.L., Machado, R.J., Paulk, M.C. *An approach to software process design and implementation using transition rules*. Software Engineering and Advanced Applications (SEAA), 37th EUROMICRO Conference, pp. 330-333, 2011.

[Fondemenet *et al.* 2004] Fondemenet F. Silaghi R. *Defining Model Driven Engineering Process*. 3th Workshop in Software Model Engineering (WISME2004). October 11-15, Lisbon, Portugal. 2004.

[Fuggetta 2000] Fuggetta, A. *Software process: a roadmap*. The future of software engineering, ACM, pp. 25–34, 2000.

[Gentleware 2014] Gentleware AG. Poseidon for UML. Sitio web: http://www.gentleware.com/, último acceso octubre 2014.

[García-Borgoñón *et al.* 2013] García-Borgoñón, L., Barcelona, M.A., García-García, J.A., Alba, M., Escalona, M.J. *Software Process Modelling Languages: a Systematic Literature Review*. Information and Software Technology Journal, DOI: 10.1016/j.infsof.2013.10.001. 2013.

[García-Borgoñón *et al.* 2014] García-Borgoñón, L., Blanco R., García-García, J.A., Barcelona, M.A., *Applying testing techniques to software process assessment: A model-based perspective*. 22nd International Conference on Information Systems Development (ISD 2013). Information System Development, Improving Enterprise Communication. Eds.: José Escalona, M., Aragón, G., Linger, H., Lang, M., Barry, C., Schneider, C. ISBN 978-3-319-07214-2. 2014.

[García-García *et al.* 2011] García-García, J. A., Escalona, M. J., Cutilla, C.R., and Alba, M. *NDT-Glossary*. Proceedings of the 13th International Conference on Enterprise Information System (ICEIS 2011), no13, pp 170-175, 2011.

[García-García *et al.* 2012a] García-García J.A., Alba M., García-Borgoñon L., Escalona M. J. *NDT-Suite: A Model-Based Suite for the Application of NDT*, LNCS 7387, pp. 469–472, 2012a.

[García-García *et al.* 2012b] García-García, J.A., Cutilla, C.R., Escalona, M.J., Alba, M., Torres, J. *NDT-Driver, a Java Tool to Support QVT Transformations for NDT*. In the 20th International Conference on Information Systems Development (ISD), DOI 10.1007/978-1-4614-4951-5_8, pp. 170-176. , 2012b.

[García-García *et al.* 2012c] García-García, J. A., Escalona, M. J., Ravel, E., Rossi G., and Urbieta, M. *NDT-merge: a future tool for conciliating software requirements in MDE environments*. iiWAS, pp 177-186. ISBN/ISSN: 978-1-4503-1306-3. Bali, Indonesia. 2012c.

[García-García *et al.* 2013a] García-García J.A., Victorio J., García-Borgoñón L., Barcelona M.A., Dominguez-Mayo F.J., Escalona M.J. *A Formal Demonstration of NDT-Quality: A Tool for Measuring the Quality using NDT Methodology*. The 21st Annual Software Quality Management (SQM) conference. ISBN: 978-0-9563140-8-6. 2013a.

[García-García *et al.* 2014a] García-García J.A., Escalona, M.J.; Domínguez-Mayo, F.J.; Salido, A. *NDT-Suite: A metodological tool solution in the Model-Driven Engineering Paradigm*, Journal of Software Engineering and Applications, vol. 7. Núm. 4. Pag. 206-217. DOI: 10.4236/jsea.2014.74022. 2014a.

[García-García *et al.* 2014b] García-García J.A., Escalona, M.J., Urbieta, M., Rossi, G. *Detecting Functional Requirements Inconsi sus stencies within Multi-Teams Projects framed into a Model-based Methodology*. Revista IEEE Transactions on Reliability -- -- Special Section on "Software Quality Assurance". Bajo revisión. 2014b.

[García-García *et al.* 2014c] García-García J.A., Escalona, M.J., Mejías M. *A Model-Based Proposal to Define the Execution Context during the Software Process Lifecycle Management*. Journal of ACM Transactions on Software Engineering and Methodology (TOSEM). Bajo revisión. 2014c.

[García-García *et al.* 2015a] García-García J.A., Escalona, M.J., Mejías M., García-Borgoñon L. *A model-based Approach for Software Processes Modeling within PLM4BS*. Journal Computer Standards & Interfaces. Bajo revisión. 2015a.

[García-García et al. 2015b] García-García J.A., Escalona, M.J., Martínez-García, A., Aragón, G, Moreno-Conde A., Parra C. *Improving patient care through a clinical process management based on the MDE paradigm*. The Scientific World Journal. En revisión. 2015b.

[Genon *et al.* 2011] Genon, N, Heymans P., Amyot D. *Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation*. Software Language Engineering. Lecture Notes in Computer Science, vol. 6563, pp. 377-396. 2011.

[Goodman 1968] Goodman, N. *Languages of Art: An Approach to a Theory of Symbols*. Bobbs-Merrill Co. 1968.

[Granada *et al.* 2013] Granada, D., Vara, J.M., Brambilla, M., Bollati, V., Marcos, E. *Analysis of the Cognitive Effectiveness of WebML Visual Notations*. Technical Report. URJC. ttp://www.kybele.es/TechnicalReport/WebML_CE_TechnicalReport.pdf. 2013.

[Hlaoui *et al.* 2011] Hlaoui, Y.B. & Benayed, L.J. *A Model Transformation Approach Based on Homomorphic Mappings between UML Activity Diagrams and BPEL4WS Specifications of Grid Service Workflows*. Computer Software and Applications Conference Workshops (COMPSACW) - 2011 IEEE 35th Annual: 243-248. 2011.

[Geambaşu 2012] Geambaşu C.V. *BPMN vs. uml activity diagram for business process modelling*. Accounting and Management Information Systems, Vol. 11, No. 4, pp. 637–651, 2012.

[Giachetti *et al.* 2008] Giachetti G. Marín B. Pastor O. *Perfiles UML y Desarrollo Dirigido por Modelos: Desafíos y Soluciones para Utilizar UML como Lenguaje de Modelado Específico de Dominio*. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos. Vol. 2. No. 3, 2008.

[Havey 2005] Havey, M. *Essential Business Process Modelling*. O'Reilly Media, 1st ed., ISBN13: 978-0596008437, 2005.

[Henderson-Sellers *et al.* 2005] Henderson-Sellers B., Gonzalez-Perez C. *A comparison of four process metamodels and the creation of a new generic standard*, Information and Software Technology 47, 49–65. 2005.

[Hill *et al.* 2006] Hill, J.B., Sinur, J., Flint, D., Melenovsky, M.J. *Gartner's position on business process management, 2006*. Business Issues Gartner, Stamford, CT, 2006.

[Hill *et al.* 2007a] Hill, J.B., Kerremans, M. and Bell, T. *Cool Vendors in Business Process Management, 2007, Gartner Research*, Stamford, CT. 2007a

[Hill *et al.* 2007b] Hill, J.B., Cantara, M., Deitert, E. and Kerremans, M. *Magic quadrant for business process management suites.* Vol. ID No. G001252906, Gartner Research, Stamford, CT. 2007b

[Hill *et al.* 2008] Hill, J. B., Pezzini, M., Natis, Y. V. *Findings: Confusion Remains Regarding BPM Terminologies*. Gartner Research, ID Number: G00155817. 2008.

[IBM 2014] IBM. Sitio web: http://www-03.ibm.com/software/products/, último acceso octubre 2014.

[ISO/IEC 2004] ISO/IEC. *ISO/IEC 15504:2004 Information technology -- Process assessment*. International Organization for Standardization, 2004.

[ISO/IEC 2007a] ISO/IEC. *ISO/IEC TR 24744:2007 Software and systems engineering Lifecycle management Guidelines for process description*. International Organization for Standardization, 2007a.

[ISO/IEC 2008a] ISO/IEC. *ISO 9001:2008 Quality management systems, Requirements*. International Organization for Standardization, 2008a.

[ISO/IEC 2008b] ISO/IEC. *ISO/IEC 12207:2008 Systems and software engineering— Software lifecycle processes*. International Organization for Standardization, 2008b.

[ISO/IEC 2008c] ISO. *ISO 13606:2008, Health informatics -- Electronic health record communication*. International Organization for Standardization, 2008c.

[ISO/IEC 2009] ISO/IEC. *ISO/IEC 27000:2009 Information technology, Security techniques, Information security management systems & Overview and vocabulary*. International Organization for Standardization, 2009.

[ISO/IEC 2012] ISO/IEC. *ISO/IEC 19507:2012 Information technology - Object Management Group Object Constraint Language (OCL)*. International Organization for Standardization, formal/2012-05-09, 2012.

[ISO/IEC 2014] ISO/IEC, *ISO/IEC 29119 Software Testing. The new international software testing standard*. Sitio web: www. softwaretestingstandard.org. Último acceso octubre 2014.

[ITIL 2014] ITIL. Sitio web: Http://www.itil-officialsite.com. Último acceso, octubre 2014.

[IWT2 2013] IWT2, Consejería de Cultura de la Junta de Andalucía. *Comparativa de herramientas de modelado*. Proyecto Calidad. 2008.

[IWT2 2014] IWT2. *NDT-Suite*. Sitio web: www.iwt2.org. Último acceso, octubre 2014.

[Jouault *et al.* 2008] Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. *ATL: A model transformation tool*. Science of computer programming, Volume 72, Issues 1–2, Pages 31–39. DOI: 10.1016/j.scico.2007.08.002. 2008.

[Karner 1993] Karner, Gustav. *Resource Estimation for Objectory Projects. Objective Systems SF AB*, 1993.

[Korherr *et al.* 2006] Korherr, B., List, B. *Extending the UML 2 activity diagram with business process goals and performance measures and the mapping to BPEL* (pp. 7-18). Springer Berlin Heidelberg. 2006.

[Koudri *et al.* 2010] Koudri, A., Champeau, J. *Modal: A SPEM extension to improve co-design process models*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6195 LNCS, pp. 248-259, 2010.

[Malinova *et al.* 2013] Malinova, M., & Mendling, J. *A Qualitative Research Perspective on BPM Adoption and the Pitfalls of Business Process Modeling*. In Business Process Management Workshops (pp. 77-88). Springer Berlin Heidelberg. 2013.

[Martinho *et al.* 2008] Martinho, R., Varajao, J., Domingos, D. *A two-step approach for Modelling flexibility in software processes*. ASE 2008, 23rd IEEE/ACM International Conference on Automated Software Engineering, Proceedings, pp. 427-430, 2008.

[Martínez-García *et al.* 2014] Martínez-García, A., Escalona, M. J., García-García, J. A., Parra-Calderón C. L. *Working with the HL7 metamodel in a Model Driven Engineering context*. Journal of Medical Systems, 2014. En revision.

[Mateo *et al.* 2012] Mateo, J., Valero, V., G. Díaz. *An Operational Semantics of BPEL Orchestrations Integrating Web Services Resource Framework*. Web Services and Formal Methods, Lecture Notes in Computer Science Vol. 7176, pp 79-94. 2012.

[Más *et al.* 2012] Mas, F., Rios, J., Menendez, J.L., Gomez. A. *A process-oriented approach to modeling the conceptual design of aircraft assembly lines*, International Journal of Advanced Manufacturing Technology 2012, Vol. 62, 2012.

[Mazanek *et al.* 2011] Mazanek, S. & Hanus, M. *Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language.* Journal of Visual Languages & Computing, vol. 22, no. 1: 66–89. 2011.

[Mendling *et al.* 2010] Mendling, J., Weidlich, M. *Business Process Modeling Notation: Second International Workshop, BPMN 2010*. Potsdam, Germany, October 13-14, 2010 Proceedings (Vol. 67). Springer. 2010.

[Millas *et al.* 2013] Millas, J. L. L. *Microsoft. Net Framework 4. 5 Quickstart Cookbook*. Packt Publishing Ltd. 2013.

[Moody *et al.* 2009a] Moody, D.L., Heymans, P., Matulevicius, R. *Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of i* Visual Syntax*. Proc. of the 17th IEEE International Requirements Engineering Conference, pp. 171–180. 2009a.

[Moody 2009b] Moody, D.L., van Hillegersberg, J. *Evaluating the visual syntax of UML: an analysis of the cognitive effectiveness of the UML family of diagrams*. LNCS, vol. 5452, pp. 16–34. Springer. 2009b.

[Moody 2010] Moody, D.L. *The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering*. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 2. 2010.

[Netjes *et al.* 2006] Netjes, M., Reijers, H. A., & van der Aalst, W. M. *Supporting the BPM life-cycle with FileNet*. In Proceedings of the CAiSE (Vol. 6, pp. 497-508). 2006.

[OASIS 2007] OASIS. *Web Services Business Process Execution Language Version 2.0*. Organization for the Advancement of Structured Information Standards. URL: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. 2007.

[OASIS 2010] OASIS. *WS-BPEL Extension for People (BPEL4People) Specification Version 1.1*. Organization for the Advancement of Structured Information Standards. URL: http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html. 2010.

[OASIS 2012] OASIS. *Web Services Human Task (WS-HumanTask) Specification Version 1.1*. Organization for the Advancement of Structured Information Standards. URL: http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html. 2012.

[OMG 2002] OMG. *SPEM1.1, Software Process Engineering Metamodel*. Object Management Group, formal/2002-11-14, 2002.

[OMG 2005] OMG. *UML2.0, Unified Modelling Language*. Object Management Group, formal/2005-07-05, 2005.

[OMG 2008a] OMG. *SPEM, Software & Systems Process Engineering Metamodel specification*. Object Management Group, http://www.omg.org/spec/SPEM/. 2008a.

[OMG 2008b] OMG. *Documents Associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation*. Object Management Group. URL: http://www.omg.org/spec/QVT/1.0/. 2008b.

[OMG 2008c] OMG. 2008c. *MOF Model to Text Transformation Language (MOFM2T), 1.0*. Object Management Group. URL: http://www.omg.org/spec/MOFM2T/1.0/. 2008c.

[OMG 2011a] OMG. *BPMN, Business Process Modelling Notation, Version 2.0*. Object Management Group, http://www.omg.org/spec/BPMN/2.0/. 2011a.

[OMG 2011b] OMG. *Meta Object Facility (MOF™) Core*. Object Management Group, http://www.omg.org/spec/MOF/. 2011b.

[OMG 2011c] OMG. *MOF/XMI Mapping, Version 2.4.1*. Object Management Group, http://www.omg.org/spec/XMI/2.4.1/. 2011c.

[OMG 2012] OMG. *UML2.5, Unified Modelling Language*. Object Management Group, formal/2012-10-24, 2012.

[OMG 2013a] OMG. 2013. *Semantics of a Foundational Subset for Executable UML Models v1.1*. Object Management Group. URL: http://www.omg.org/spec/FUML/1.1. 2013.

[OMG 2013b] OMG. 2013. *Semantics Of Business Vocabulary And Rules (SBVR)*. Object Management Group. URL: http://www.omg.org/spec/SBVR/. 2013.

[Oracle 2014] Oracle. *Oracle® BPEL Process Manager Developer's Guide*. Sitio web: http://docs.oracle.com/cd/B14099_19/integrate.1012/b14448/java.htm. Último acceso, octubre 2014.

[Papazoglou *et al.* 2006] Papazoglou M., Ribbers P. *E-Business: Organizational and Technical Foundations*. Editor: John Wiley & Sons. ISBN-13: 978-0470843765. 2006.

[Pedraza *et al.* 2009] Pedraza G., CEstublier J. *Distributed Orchestration Versus Choreography: The FOCAS Approach*. Trustworthy Software Development Processes, Lecture Notes in Computer Science Volume 5543, pp 75-86. 2009.

[PMI 2008] PMI (Project Management Institute) Inc. *A Guide to the Project Management Body of Knowledge, Fourth Edition (PMBOK® Guide)*. ISBN13: 978-1933890517. 2008.

[Ponce *et al.* 2013] Ponce, J., García-Borgoñon, L., García-García, J.A., Escalona, M.J., Domínguez-Mayo, F.J., Alba, M., and Aragon, G. *A Model-Driven Approach for Business Process Management*. Covenant Journal of Engineering & Technology (CJICT) Vol. 1, No. 2, pp. 32-52. 2013.

[Richardson *et al.* 2013] Richardson C., Miers D. *The Forrester Wave™: BPM Suites, Q1 2013*. Marzo 2013.

[Riesco *et al.* 2003] Riesco, D., Uva, M., Acosta, E., Debnath N., Montejano, G. *Including workflow concurrent modelling in an extension of the UML activity diagram metamodel*. ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03), Proceedings published by IEEE Press, 2003.

[Rubinger *et al.* 2014] Rubinger, A. L., Knutsen, A. *Continuous Enterprise Development in Java*. O'Reilly Media, Inc. 2014.

[Ruiz-González *et al.* 2004] Ruiz-González F., Canfora G. *Software Process: Characteristics, Technology and Environments*. SPT - Software Process Technology, vol. 5, pp. 5-10, 2004.

[Russell *et al.* 2004a] Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P. *Workflow Resource Patterns*. BETA Working Paper Series, WP 127. 2004a.

[Russell *et al.* 2004b] Russell, N., ter Hofstede, A.H.M., Edmond, D. & van der Aalst, W.M.P. *Workflow Data Patterns*. QUT Technical report, FIT-TR-2004-01. 2004b.

[Russell *et al.* 2006a] Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N. *Workflow Control-Flow Patterns: A Revised View*. BPM Center Report BPM-06-22, BPMcenter.org. 2006a.

[Russell *et al.* 2006b] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M. *Exception Handling Patterns in Process-Aware Information Systems*. BPM Center Report BPM-06-04, BPMcenter.org. 2006b.

[Russell *et al.* 2007] Russell, N., van der Aalst. W. M. *Evaluation of the BPEL4People and WS-HumanTask extensions to WS-BPEL 2.0 using the workflow resource patterns*. Bpm center report, Department of Technology Management, Eindhoven University of Technology GPO Box, 513. 2007.

[Ryan *et al.* 2009] Ryan K.L. Ko, Stephen S.G. Lee, Eng Wah Lee. *Business process management (BPM) standards: a survey*. Business Process Management Journal, Vol. 15 Iss: 5 pp. 744 - 791. 2009.

[Sääksvuori *et al.* 2008] Sääksvuori, A., Immonen A. *Product Lifecycle Management*. Original Finnish edition published by Talentum, Third Edition. ISBN 978-3-540-78172-1. 2008.

[Saenz *et al.* 2010] Saenz, J. M., Escalona, M. J. *Generación de prototipos de usuario utilizando paradigma MDE*. Proyecto final de carrera del departamento de Lenguajes y Sistemas Informáticos de la Universidad Sevilla. 2010.

[Salido *et al.* 2014] Salido, A.; García-García J.A.; J. Gutiérrez; J. Ponce. *Tests Management in CALIPSOneo: A MDE Solution*, Journal of Software Engineering and Applications, Vol.7 No.6, PP. 506-512. DOI: 10.4236/jsea.2014.76047. 2014.

[Schmidt 2006] Schmidt, D. C. *Model-Driven Engineering*. IEEE Computer, Computer Society, vol. 39, no. 2, pp. 25-31, 2006.

[Shewhart 1986] Shewhart, W. A. *Statistical Method from the Viewpoint of Quality Control*. Department of Agriculture. Dover, page 45. 1986.

[SOB 2009] SOB (Stationery Office Books). *Managing Successful Projects with PRINCE2*. Stationery Office Books. ISBN13: 978-0113310593. 2009.

[SparxSystems 2014] SparxSystems. *Enterprise Architect*. Sitio web: www.sparxsystems.com.au, último acceso octubre 2014.

[Sutton *et al.* 1997] Sutton J., Stanley M., L. Osterweil. *The design of a next-generation process language*, in: M. Jazayeri, H. Schauer (Eds.), Software Engineering ESEC/FSE'97, volume 1301 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 142–158. 1997.

[Stark 2011] Stark J. *Product Lifecycle Management: 21st Century Paradigm for Product Realisation*. Springer Science. DOI: 10.1007/978-0-85729-546-0_1. 2011.

[Strnadl 2006] Strnadl, C. F. *Aligning business and it: The process-driven architecture model*. Information Systems Management, 23(4), 67-77. 2006.

[Thiry *et al.* 2009] Thiry L., Thirion B. *Functional Metamodels for Systems and Software*. Journal of Systems and Software, Vol. 82, Iss. 7, pp 1125–1136. DOI: http://dx.doi.org/10.1016/j.jss.2009.01.042. 2009.

[Trkman 2010] Trkman, P. *The critical success factors of business process management*. International Journal of Information Management, 30(2), 125-134. 2010.

[UNE 2007] UNE. *UNE-EN 13940-1:2007, Health informatics — System of concepts to support continuity of care*. AENOR. 2007.

[Van-der-Aalst 2004a] Van-der-Aalst, W.M.P. *Business process management demystified: a tutorial on models, systems and standards for workflow management*. Lecture Notes in Computer Science, Lectures on Concurrency and Petri Nets, vol. 3098, pp. 1-65, 2004a.

[Van-der-Aalst 2004b] Van-der-Aalst, W.M.P. *Business process management: a personal view*. Business Process Management Journal, vol. 10, no. 2, p. 5, 2004b.

[Van-Der-Straeten *et al.* 2009] Van-Der-Straeten R., Mens, T., Van-Baelen S. *Challenges in Model-Driven Software Engineering*. Models in Software Engineering, Lecture Notes in Computer Science, no. 5421, pp. 35-47. 2009.

[Visure 2014] Visure. *IRqA Requirements Definition & Management solution*. Sitio web: http://www.visuresolutions.com/. Último acceso octubre 2014.

[W3C 2005] World Wide Web Consortium (W3C), *XML Schema 2005*. Sitio web: http://www.w3.org/2001/XMLSchema. Último acceso octubre 2014.

[Wise 2006] Wise, A. *Little-JIL 1.5 Language Report*. Department of Computer Science, University of Massachusetts, Amherst, MA, UM-CS-2006-51, 2006.

[Wu *et al.* 2006] Wu, M., Li, G., Ying, J., Yan, H. *A metamodel approach to software process modelling based on UML extension*. Conference Proceedings -IEEE International Conference on Systems, Man and Cybernetics, vol. 6, pp. 4508-4512. 2006.

[Yu *et al.* 1997] Yu B., Wright D.T. *Software tools supporting business process analysis and modelling*. Business Process Management Journal, Vol. 3 Iss: 2, pp.133-150, DOI 10.1108/14637159710173096. 1997.

[Zamli 2001] Zamli K.Z. *Process Modeling Languages: A Literature Review*, Malaysian Journal of Computer Science 14, 26–37. 2001.

[Zamli *et al.* 2004] Zamli K.Z., Isa N.A.M. *A survey and analysis of process modeling languages*, Malaysian Journal of Computer Science 17–89. 2004.

[Zeng *et al.* 2004] Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H. *Qos-aware middleware for web services composition*. IEEE Transactions on Software Engineering, vol. 30, issue 5, pp 311–327. 2004.

ANEXO A. GLOSARIO DE TÉRMINOS

En este primer anexo se ofrecen, a modo de referencia y de consulta, definiciones breves para algunos de los términos y acrónimos (tanto teóricos como técnicos) más referenciados y utilizados durante el desarrollo y descripción de este trabajo de Tesis Doctoral.

BPM. Acrónimo de *«Business Process Management»* que identifica una estrategia de gestión que incluye aquellos métodos, técnicas y herramientas de soporte al ciclo de vida de los procesos de negocio, el cual a su vez incluye el diseño, aprobación, gestión y análisis de los procesos de negocio operacionales que involucran a personas, organizaciones, aplicaciones, documentos y cualquier otra fuente de información.

BPMN. Acrónimo de *«Business Process Model and Notation»* que identifica un lenguaje estándar de propósito general para el modelado de procesos de negocio.

BPMS. Acrónimo de *«Business Process Management Suite»* que puede ser definido como una nueva categoría de software empresarial que permite a las empresas modelar, implementar y ejecutar conjuntos de actividades interrelacionadas – es decir, Procesos – de cualquier naturaleza, sea dentro de un departamento o permeando la entidad en su conjunto, con extensiones para incluir los clientes, proveedores y otros agentes como participantes en las tareas de los procesos.

CASE. Acrónimo de *«Computer Aided Software Engineering»* y cuyo término engloba a aquellas herramientas informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero.

Ejecución de un Proceso de Negocio. Se refiere a la capacidad de despliegue y ejecución de un proceso de negocio dentro de un motor de ejecución BPM , el cual ejecuta instancias del proceso de negocio, delegando ciertas tareas a seres humanos y a aplicaciones automatizadas tal y como se especifica en el modelo de proceso definido.

Mapeo. Este término es la traducción al español utilizado en esta tesis del término mapping operation, definido en QVT como una operación que implementa una parte de una transformación.

MDE. Acrónimo de paradigma *«Model-Driven Engineering»*. Estas siglas denotan una filosofía de desarrollo guiada por modelos, en la cual, el principal objetivo de una fase (análisis, diseño, etc.) es desarrollar los modelos adecuados a partir del refinamiento de los modelos obtenidos en la fase anterior.

Metáfora. En el contexto de esta tesis, este término se utiliza para comunicar conceptos abstractos de una forma familiar y accesible teniendo en este sentido, un papel dominante

en el diseño de las interfaces de usuarios de cualquier tipo de software o lenguaje gráfico. El uso de metáforas ayuda a construir soluciones software que puedan ser utilizadas por comunidades de usuarios más diversas.

Metamodelo. En el contexto de este trabajo un metamodelo es la definición de una gramática (elementos y reglas de construcción) con la que poder elaborar modelos que sean conformes a dicho metamodelo.

MOF. Acrónimo de *«Meta-Object Facility»* que representa conjunto de interfaces estándares que pueden ser utilizadas para definir y manipular metamodelos interoperables y sus correspondientes modelos.

MOFM2T. Acrónimo de *«MOF Model to Text Transformation Language*» que identifica un estándar propuesto por el OMG para definir reglas de derivación para generar una versión textual de modelos.

Modelo. En el contexto de este trabajo, un modelo es una representación mediante un lenguaje concreto, con un mayor o menor grado de abstracción, de un aspecto de un sistema de información. Por ejemplo, un modelo de pruebas será la representación de un conjunto de pruebas a realizar sobre el sistema.

NDT. Acrónimo de *«Navigational Development Technique»* que identifica la propuesta metodológica que ha servido como influencia para la realización de este trabajo de tesis y que será fundamental en trabajos futuros posteriores.

OCL. Acrónimo de *«Object Constraint Language»* que identifica un lenguaje para la descripción formal de expresiones en los modelos UML. Su papel principal es el de completar los diferentes artefactos de la notación UML con requerimientos formalmente expresados.

OMG. Acrónimo de *«Object Management Group»* que identifica a un consorcio sin ánimo de lucro formado por diversas compañías y organizaciones. Se dedica al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, así como a fomentar el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas.

Orquestación de un Proceso de Negocio. Se refiere a la gestión centralizada y coordinada de eventos o componentes durante la ejecución del proceso de negocio. A menudo se trata de una variedad de componentes que forman parte de una nueva aplicación o proceso.

Perfil UML. Herramienta de extensión del lenguaje UML con el que es posible describir un problema de modelado en particular y facilitar la construcción de modelos en ese dominio.

PLM$_4$BS. Acrónimo de *«Process Lifecycle Management for Business-Software»* que representa la herramienta CASE de soporte al marco de trabajo definido en esta Tesis Doctoral. Gracias a esta herramienta, es posible utilizar en entornos reales, los metamodelos para la gestión de procesos software y las reglas de derivación definidos de manera teórica.

Proceso Software. Se define como un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos necesarios para concebir, desarrollar, desplegar y mantener un producto software.

QVT. Acrónimo de *«Query/View/Transform»* que identifica el lenguaje de transformaciones de modelo a modelo definido por la OMG y utilizado en este trabajo de tesis.

Sintaxis abstracta. En este trabajo de tesis, este término es sinónimo del termino metamodelo y se utiliza con el mismo significado.

Sintaxis concreta. En este trabajo de tesis, este término es sinónimo del termino modelo y se utiliza con el mismo significado.

Transformación. En el contexto de este trabajo una transformación es una especificación de cómo construir un modelo conforme con un metamodelo, tomando como entrada otro modelo conforme a otro metamodelo, que puede ser el mismo o distinto.

UML. Acrónimo de *«Unified Modelling Language»* y que identifica a un lenguaje estándar de modelado gráfico, utilizado en este trabajo principalmente para definir los metamodelos y modelos presentados.

ANEXO B. MANUAL DE LA HERRAMIENTA PLM$_4$BS

Este anexo describe el uso de la herramienta PLM$_4$BS. Esta herramienta se presentó en el capítulo VI, en donde se expuso que la implementación actual cuenta con una sintaxis concreta basada en diagramas de actividades (para el metamodelo de definición del proceso software) y basada en diagramas de clases (para el metamodelo de ejecución y orquestación del proceso software). Ambas sintaxis, fueron definidas sobre la herramienta UML Enterprise Architect pero no se aclaró la metodología de uso de PLM$_4$BS. Este es el propósito de este apéndice.

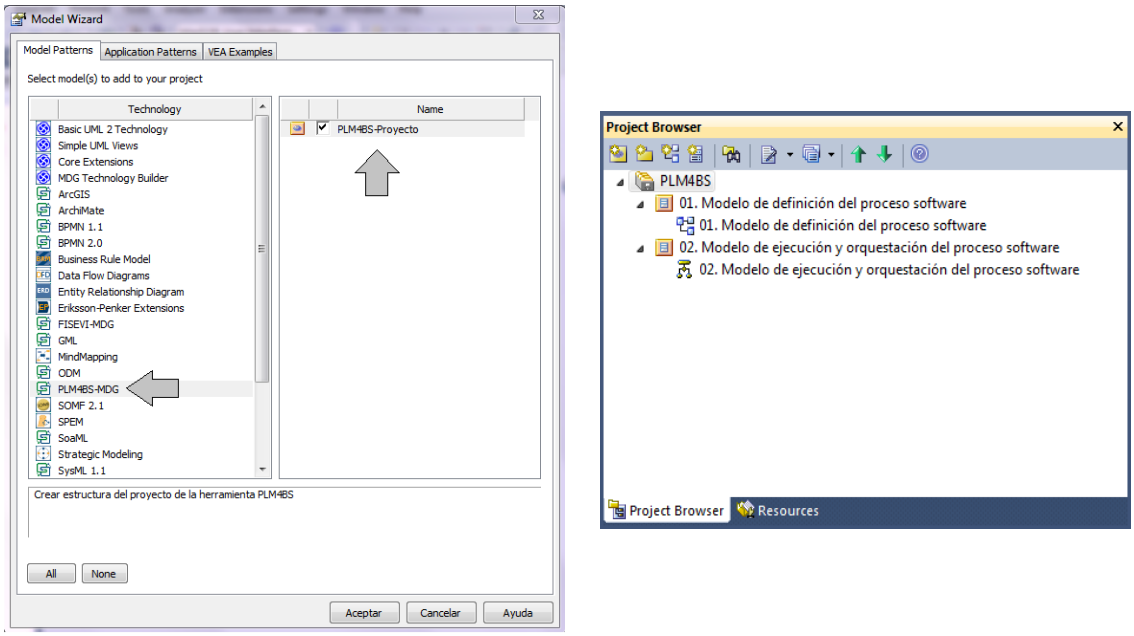
La versión actual de la herramienta PLM$_4$BS ofrece una interfaz gráfica de usuario integrada dentro de la propia interfaz de Enterprise Architect lo que proporciona, en general, un alto grado de versatilidad y flexibilidad en cuanto al uso de PLM$_4$BS. Sin embargo, a continuación se propone una metodología de trabajo para estandarizar lo máximo posible el uso de la herramienta presentada en esta tesis.

I. Crear proyecto «PLM$_4$BS» en Enterprise Architect

Como primer paso para la utilización de la herramienta PLM$_4$BS sería necesario crear dentro de Enterprise Architect un proyecto de tipo *«PLM$_4$BS»*, utilizando para ello el propio asistente de Enterprise Architect. Este proyecto ha sido desarrollado y configurado *ad-hoc* para la herramienta PLM$_4$BS de tal forma que cuando el usuario crea un proyecto de este tipo, se crea de manera automática una estructura predefinida de paquetes y diagramas.

La Figura B.1-a muestra el proyecto de tipo *«PLM$_4$BS»* dentro de la ventana del asistente de creación de proyectos de Enterprise Architect, mientras que la Figura B.1-b muestra la estructura de paquetes resultante. Respecto a este último y como se puede apreciar, la estructura contempla un paquete específico para modelar la definición del proceso software y un paquete específico para describir el modelo de ejecución y orquestación del proceso.

La ventaja principal de automatizar la creación de esta estructura de paquetes es que el usuario tiene disponibilidad y acceso inmediato a los perfiles UML especificados en el Capítulo VI a través de diferentes *«toolbox»* en Enterprise Architect.



(a) **(b)**

Figura B.1. Creación del proyecto «PLM4BS» en Enterprise Architect

II. Especificar el modelo de definición del proceso software

Una vez creada la estructura básica del proyecto *«PLM4BS»*, basta con abrir el diagrama asociado al modelo de definición del proceso para empezar a modelarlo. Una vez abierto, Enterprise Architect carga de manera automática el *«toolbox»* del perfil UML que especifica la sintaxis concreta del metamodelo de definición del proceso software. La Figura B.2 muestra la interfaz de Enterprise Architect con el mencionado *«toolbox»*.

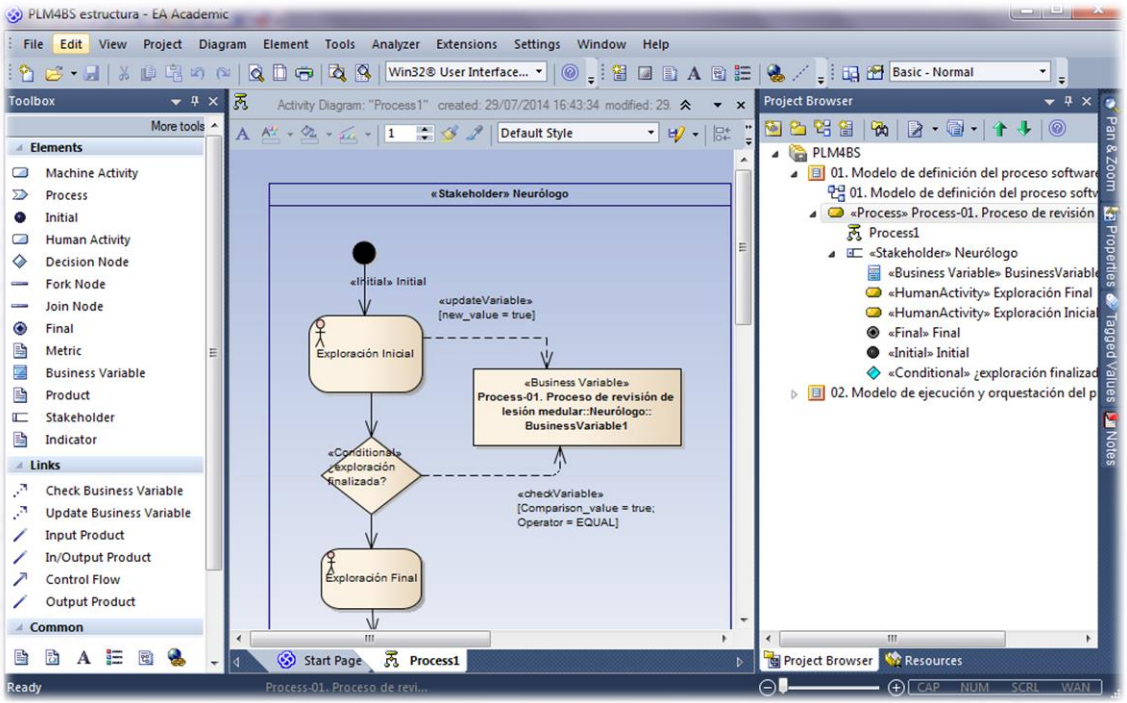


Figura B.2. Definición en «PLM4BS» del modelo de definición del proceso

III. Especificar el modelo de ejecución y orquestación del proceso software

Una vez definido el modelo con la definición del proceso, el usuario puede comenzar a definir el modelo de ejecución y orquestación asociado. Para ello, basta con generar este último modelo a partir del primero utilizando las reglas de derivación QVT definidas en el Capítulo V.

La ejecución de las reglas de derivación es totalmente transparente al usuario, el cual únicamente tiene que seguir los siguientes pasos para obtener el modelo de ejecución y orquestación del proceso:

1. Seleccionar en el *«project browser»* de Enterprise Architect, el modelo con la definición del proceso software (definido previamente en el paso anterior).
2. Invocar la generación automática del modelo de ejecución y orquestación en el menú *«Extensions | PLM4BS | Generar modelo de ejecución y orquestación»*.

La Figura B.3 muestra de manera visual el procedimiento anterior.

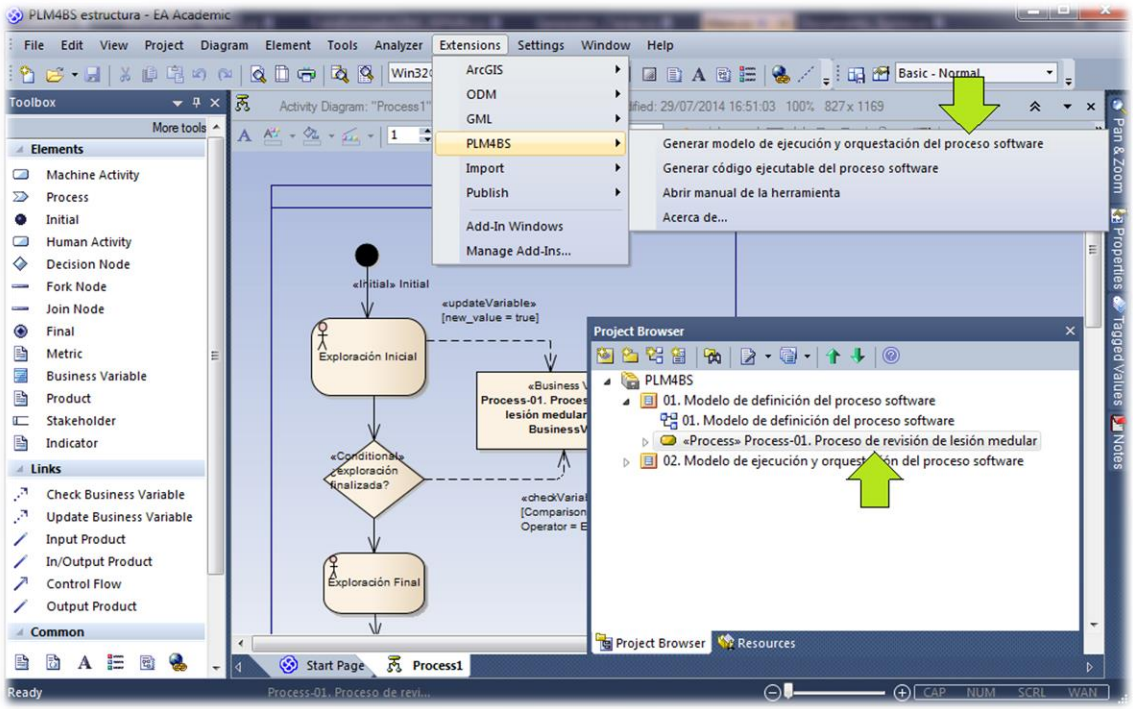


Figura B.3. Generación en «PLM4BS» del modelo de ejecución y orquestación

IV. Obtener código ejecutable del modelo de ejecución y orquestación del proceso software

Finalmente, una vez definido el modelo de ejecución y orquestación del proceso, el usuario puede obtener la versión ejecutable del mismo derivando dicho modelo hacia código WS-BPEL. Esta generación está basada en el marco teórico MOFM2T presentado en el Capítulo V.

Al igual que en el paso anterior, la ejecución de estas reglas de derivación es totalmente transparente al usuario, el cual únicamente tiene que seguir los siguientes pasos para obtener el código del proceso:

1. Seleccionar en el *«project browser»* de Enterprise Architect, el modelo de ejecución y orquestación del proceso software (definido previamente en el paso anterior).
2. Invocar la generación automática del código en el menú *«Extensions | PLM4BS | Generar código ejecutable del proceso»*.

La Figura B.4 muestra de manera visual donde se encuentra la opción de menú mencionada anteriormente.

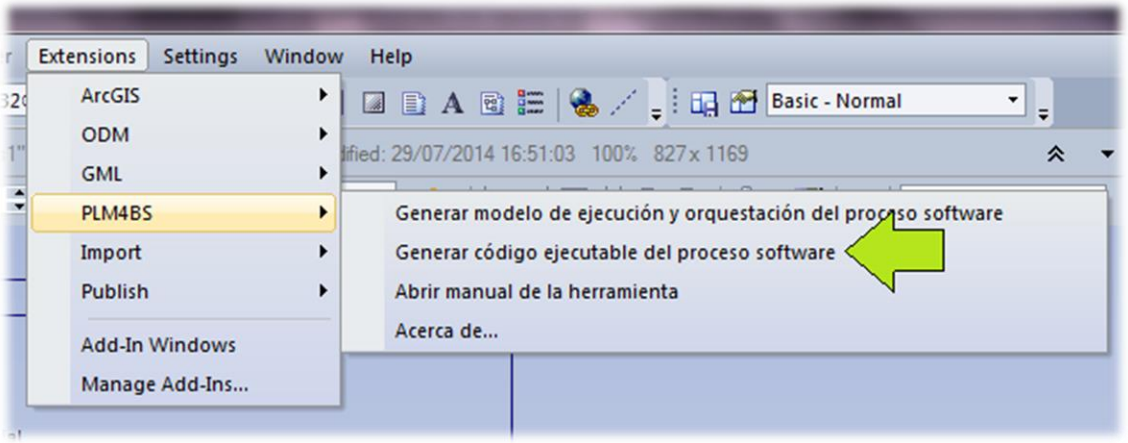


Figura B.4. Generación en «PLM4BS» del código ejecutable del proceso

ANEXO C. RECONOCIMIENTOS Y ACTIVIDAD INVESTIGADORA

En este anexo, se recoge la actividad investigadora del doctorando, catalogándola en diferentes apartados: participación en eventos de divulgación científica, publicaciones, proyectos I+D, proyectos de transferencia y redes de investigación en las que he participado.

Además, se recoge información sobre diferentes reconocimientos obtenidos durante la elaboración de este trabajo.

En cuanto a las publicaciones, éstas han sido catalogadas según su tipología: capítulo de libro, artículo de revista, artículo publicado en conferencia internacional y artículo publicado en conferencia nacional.

Finalmente, respecto a los proyectos y aunque se menciona al inicio de la Sección 3.2 de este apéndice, cabe destacar el proyecto *«Servicios de tecnologías de la información y comunicaciones a las aplicaciones de informes de los servicios centrales del servicio andaluz de salud (P023-14/E09)»* en el que participo como Investigador Principal en el mismo.

1. Reconocimientos

Durante el desarrollo de este trabajo de tesis he sido seleccionado como beneficiario de una de las tres nuevas becas anuales predoctorales que la fundación UNIVERSIA ofertó en el año 2013. Esta beca predoctoral me fue renovada un año más (curso 2014/2015) tras demostrar una dedicación investigadora con un marcado nivel de excelencia durante el curso 2013-2014. Este tipo de becas surge con el compromiso de la fundación UNIVERSIA para promover y potenciar el desarrollo profesional de personas con discapacidad en cuanto a la obtención del título de doctor.

2. Eventos de divulgación científica

Durante la elaboración de esta Tesis Doctoral he participado y colaborado en los siguientes eventos de divulgación científica:

- A lo largo del año 2014 he formado parte del comité de programa y comité científico de la conferencia ISD2014 (*«International Conference on Information Systems Development»*) que se ha llevado a cabo en Croacia durante el mes de septiembre 2014. Entre otras tareas, esta participación se ha producido en publicación de artículos, divulgación científica, revisión de artículos y soporte durante la gestión y organización del evento. Además, he sido uno de los responsables de la track sobre MDE.

- En el año 2014 he formado parte del comité científico de ATICA 2014, VI Congreso Internacional sobre Aplicación de Tecnologías de la Información y Comunicaciones Avanzadas.
- Desde abril 2014, formo parte del comité científico como investigador asesor de la revista iberoamericana GTI (Gerencia Tecnológica Informática). Además de realizar tareas de difusión, esta participación ha implicado la revisión de diferentes artículos de investigación enviados a la revista antes mencionada.
- Desde enero 2014, formo parte del comité de programa y científico de VI Congreso Internacional de Computación y Telecomunicaciones (COMTEL 2014) que se celebra en octubre 2014 en la ciudad de Lima (Perú).
- En el año 2013 formé parte del comité organizador, del comité de programa y comité científico de la conferencia ISD2013 (*«International Conference on Information Systems Development»*) que se llevó a cabo en Sevilla (España) durante el mes de septiembre 2013. Entre otras tareas, esta participación se ha traducido en publicación de artículos, divulgación científica, revisión de artículos así como gestión y organización del evento. Además, he sido uno de los responsables de la track sobre MDE.
- En el año 2013, formo parte también del comité de programa y científico de V Congreso Internacional de Computación y Telecomunicaciones (COMTEL 2013) que se celebró en octubre 2013 en la ciudad de Lima (Perú).
- En el año 2013, forme parte también del comité de programa de la conferencia *«International Symposium on Embedded Multicore/Manycore System-on-Chip»* (MCSoC-13) que se celebró en septiembre 2013 en la ciudad de Tokio (Japón).

3. Publicaciones

Los siguientes apartados catalogan la producción investigadora del doctorando según la tipología de la misma. Además, esta producción se lista de manera cronológica descendente en cada apartado.

3.1. Capítulos de libro

García-García, J.A., Cutilla, C.R., Escalona, M.J., Alba, M., Torres, J. "*NDT-Driver, a Java Tool to Support QVT Transformations for NDT*". In the 20th International Conference on Information Systems Development (ISD 2011), Reflections, Challenges and New Directions Pooley, R.J.; Coady, J.; Linger, H.; Barry, C.; Lang, M.; Schneider, C. (Eds.). DOI 10.1007/978-1-4614-4951-5_8, pp. 170-176. , 2012. Ranking CORE 2011: A+.

García-Borgoñon, L., García-García, J. A., Alba, M., Escalona, M. J. "*Software Process Management: A Model-Based Approach*". Information Systems Development: Building Sustainable Information Systems (Proceedings of the 2012 International Conference on Information Systems Development), Pp 167-178. Editors: Henry Linger, Julie Fisher, Andrew Barnden, Chris Barry, Michael Lang, Christoph Schneider. ISBN: 978-1-4614-7539-2. 2013. Ranking CORE 2013: A+.

García-Borgoñón, L., Blanco R., García-García, J.A., Barcelona, M.A. "*Applying testing techniques to software process assessment: A model-based perspective*". 22nd International Conference on Information Systems Development (ISD 2013). Information System Development, Improving Enterprise Communication. Eds.:

José Escalona, M., Aragón, G., Linger, H., Lang, M., Barry, C., Schneider, C. ISBN 978-3-319-07214-2. 2014. Ranking CORE 2014: A+.

Escalona, M.J., García-García, J.A., Domínguez-Mayo, F.J., Ramos I. "*Technical tool surveys and comparative studies A systematical approach*". 23RD International Conference On Information Systems Development (ISD2014 Croacia). Pendiente de publicación. 2014. Ranking CORE 2014: A+.

Salido, A., García-García, J.A., Escalona, M.J. "*Managing CALIPSOneo Project: Learning from trenches*". 23RD International Conference On Information Systems Development (ISD2014 Croacia). Pendiente de publicación. 2014. Ranking CORE 2014: A+.

3.2. Revistas

García-García, J. A., Rivero D., Escalona, M. J., "*NDT-Suite, una solución práctica para el uso de NDT*". Revista Procesos y Métricas, Editores: Ricardo Colomo Palacios y José Carrillo Verdún. Vol. 8, n.º 1, enero 2011, ISSN 1698-2029. 2011., 2011.

Ponce, J., García-Borgoñon, L., García-García, J.A., Escalona, M.J., Domínguez-Mayo, F.J., Alba, M., and Aragon, G. "*A Model-Driven Approach for Business Process Management*". Covenant Journal of Engineering & Technology (CJICT) Vol. 1, No. 2, pp. 32-52. 2013.

García-Borgoñón, L., Barcelona, M.A., García-García, J.A., Alba, M., Escalona, M.J. "*Software Process Modelling Languages: a Systematic Literature Review*". Information and Software TechnologyJournal, DOI: 10.1016/j.infsof.2013.10.001. 2013. JCR Índice de Impacto: 1,330.

Escalona, M. J., Urbieta, M., Rossi, G., Garcia-Garcia, J. A., Luna, E. R. "*Detecting Web requirements conflicts and inconsistencies under a model-based perspective*". Journal of Systems and Software, 86(12), 3024-3038. 2013. JCR Índice de Impacto: 1,250 (2013).

García-García J.A., Escalona, M.J.; Domínguez-Mayo, F.J.; Salido, A. "*NDT-Suite: A metodological tool solution in the Model-Driven Engineering Paradigm*". Journal of Software Engineering and Applications, vol. 7. Núm. 4. Pag. 206-217. DOI: 10.4236/jsea.2014.74022. 2014.

Salido, A.; García-García J.A.; J. Gutiérrez; J. Ponce. "*Tests Management in CALIPSOneo: A MDE Solution*", Journal of Software Engineering and Applications, Vol.7 No.6, PP. 506-512. DOI: 10.4236/jsea.2014.76047. 2014.

García-Borgoñón, L., Barcelona, M. A., Garćıa-Garćıa, J. A., & Escalona, M. J. "*Software Process Accessibility in Practice: A Case Study*". 5th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion, DSAI 2013. Procedia Computer Science, vol. 27, 292-301. DOI: 10.1016/j.procs.2014.02.032. 2014. Índice de Impacto: 0,280.

Domínguez-Mayo, F., García-García, JA., Escalona, MJ, Mejías, M., Urbieta, M., Rossi, G. A framework and tool to manage Cloud Computing service quality. Software Quality Journal. DOI: http://dx.doi.org/10.1007/s11219-014-9248-0. 2013. JCR Índice de Impacto: 0,880.

García-García J.A., Escalona, M.J., Mejías M., García-Borgoñon L. *A model-based Approach for Software Processes Modeling within PLM$_4$BS*. Journal Computer Standards & Interfaces. Bajo revisión. 2015.

García-García J.A., Escalona, M.J., Mejías M. *A Model-Based Proposal to Define the Execution Context during the Software Process Lifecycle Management*. Journal of ACM Transactions on Software Engineering and Methodology (TOSEM). En revisión. 2015.

García-García J.A., Escalona, M.J., Martínez-García, A., Aragón, G, Moreno-Conde A., Parra C. *Improving patient care through a clinical process management based on the MDE paradigm*. The Scientific World Journal. En revisión. 2015.

Domínguez-Mayo, F.J., Escalona, M.J., Mejías M, Aragón G, García-García J.A., Torres J. *A Strategic Study about Quality Characteristics in e-Health Systems based on a Systematic Literature Review*.The Scientific World Journal. En revisión. 2015.

3.3. Conferencias internacionales

Cutilla C. R., García-García J. A., Alba M., Escalona M.J., Ponce J., Rodríguez L. "*Aplicación del paradigma MDE para la generación de pruebas funcionales; Experiencia dentro del proyecto AQUA-WS*". Presented at the 6ª Conferência Ibérica de Sistemas e Tecnologias de Informação, ISBN: 978-989-96247-4-0. 2011.

García-García, J. A., Escalona, M. J., Cutilla, C.R., and Alba, M. "*NDT-Glossary: A MDE approach for glossary generation*". Proceedings of the 13th International Conference on Enterprise Information System (ICEIS 2011), no13, pp 170-175, 2011. Ranking CORE 2011: C.

Cutilla C. R., García-García J. A., Alba M., Escalona M.J., Ponce J., Rodríguez L. "*Model-driven engineering applied in functional Testing: The practical experience of the AQUA-WS project*". In S. Hammoudi, M. van Sinderen & J. Cordeiro (eds.), 7th International Conference on Software Paradigm Trends, ICSOFT 2012. ISBN: 978-989-8565-19-8. 2012. Ranking CORE 2012: B.

Armario, J., Gutiérrez, J., Alba, M., García-García, J. A., Vitorio, J., Escalona, M. J. "*Project Estimation with NDT*". In S. Hammoudi, M. van Sinderen & J. Cordeiro (eds.), 7th International Conference on Software Paradigm Trends, ICSOFT 2012 (p./pp. 120-126). ISBN: 978-989-8565-19-8. 2012. Ranking CORE 2012: B.

García-García, J.A., Alba, M., Garcia-Borgoñón, L., Escalona, M.J. "*NDT-Suite: A model-based suite for the application of NDT*". Proceedings of the 12nd International Conference on Web Engineering. 10th International Conference on Web Engineering (ICWE 2012) Lecture Notes in Computer Science. 2012. Vol. 7387. Núm. 1. Pag. 469-472. Ranking CORE 2012: C.

García-García, J. A., Escalona, M. J., Ravel, E., Rossi G., and Urbieta, M. "*NDT-merge: a future tool for conciliating software requirements in MDE environments*". iiWAS, pp 177-186. ISBN/ISSN: 978-1-4503-1306-3. Bali, Indonesia. 2012. Ranking CORE 2012: C.

Escalona M.J., Garcia-Garcia J. A., Mas F., Oliva M., Del Valle C. Applying model-driven paradigm: CALIPSOneo experience. Proceedings of the Industrial Track of the Conference on Advanced Information Systems Engineering 2013 (CAiSE'13), vol. 1017, pp. 25-32. 2013. Ranking CORE 2013: A.

Alba, M., Ramírez, M., Pavon, I., Sanchez, N., García-García, J.A. "*Comparativa de Herramientas ECM en el marco de la e-administración*". V Congreso Internacional de Computación y Telecomunicaciones, pp 24-31. ISBN: 978-612-4050-69-5. 2013.

Salido, A., García García, J.A., Ponce, J., Gutiérrez, J., Oliva, M. "*Gestión de pruebas en CalipsoNeo: Una solución MDE*". V Congreso Internacional de Computación y Telecomunicaciones, Lima - Perú. ISBN: 978-612-4050-69-5. 2013.

García-García J.A., Victorio J., García-Borgoñón L., Barcelona M.A., Dominguez-Mayo F.J., Escalona M.J. "*A Formal Demonstration of NDT-Quality: A Tool for Measuring the Quality using NDT Methodology*". The 21st Annual Software Quality Management (SQM) conference. ISBN: 978-0-9563140-8-6. 2013. Ranking CORE 2013: C.

Domínguez-Mayo, F.J., García-García, J.A., García-Borgoñón, L., Escalona, M.J., Mejías M. "*QuEF framework and quality management of software products in organizations*". The 22nd Annual Software Quality Management (SQM) conference. pp 67-78, ISBN/ISSN: 978-0-9926958-1-1, 2014. Ranking CORE 2014: C.

Martínez-García, A., Escalona, M.J., García-García J.A., Aragón, G, Moreno-Conde A., Parra C. *Using Model-Driven Engineering and Health Standards for Improving Healthcare Processes Management*, Medical Informatics Europe (MIE2015), 2015. Pendiente de publicación.

3.4. Conferencias nacionales

Carmen R. Cutilla, Julian A. García-García and Javier J. Gutiérrez. "*Hacia una propuesta de priorización de casos de pruebas a partir de NDT*". Actas de las XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD), Almeria, España. pp 345-350, ISBN: 978-84-15487-28-9, 2012.

García-Borgoñón, L., García-García, J.A., Alba, M., Domínguez-Mayo F.J. "*Gestión de Procesos en Organizaciones de Desarrollo de Software: Un Enfoque Basado en Modelos*". Actas de las XVIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp 113-126, ISBN: 978-84-695-8310-4, 2013.

García-García, J.A., Escalona, M.J. "*Demostración de NDT-Driver: una herramienta de soporte a los mecanismos de transformación de NDT*". Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 223-226, ISBN: 84-697-1152-0, 2014.

4. Proyectos

4.1. Proyectos de investigación

NDTQ-Framework (TIC-5789)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Proyectos de Excelencia de la Junta de Andalucía Referencia: TIC-5789 Fecha de comienzo: 06-07-2011 Fecha de finalización: 05-07-2013
Financiado por	Junta de Andalucía (Consejería de Innovación, Ciencia y Empresas)

Testing temprano y modelos de simulación híbrida en la producción de software (TIN2010-20057-C03-02)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Plan Nacional del 2010 Referencia: TIN2010-20057-C03-02 Fecha de comienzo: 01-01-2011 Fecha de finalización: 31-12-2013
Financiado por	Ministerio de Ciencia e Innovación

Calidad predecible y gestionada mediante simulación y técnicas de pruebas en etapas tempranas. (QSimTest) (TIN2007-67843-C06-03)	
Información del proyecto	Responsable: Isabel Ramos Román Tipo de proyecto: Plan Nacional del 2007 Referencia: TIN2007-67843-C06-03 Fecha de comienzo: 01-10-2007 Fecha de finalización: 30-09-2011
Financiado por	Ministerio de Educación y Ciencia

Oncoinves: Plataforma De Gestión De Información Para La Investigación Oncológica (PI-OL16-2012)	
Información del proyecto	Responsable: María José Escalona Cuaresma Referencia: PI-OL16-2012 Fecha de comienzo: 27/12/2012 Fecha de finalización: 27/12/2015
Financiado por	Consejería de Salud y Bienestar Social

Merimée project: Paris-Seville (ED224)	
Información del proyecto	Responsable: María José Escalona Cuaresma Referencia: ED224 Fecha de comienzo: 05/11/2012 Fecha de finalización: 31/12/2014
Financiado por	Programme Mérimée de collaboration entre Ecoles Doctorales françaises et espagnoles

Ayuda Suplementaria a grupos de Investigación (2011/00000256)	
Información del proyecto	Responsable: María José Escalona Cuaresma Referencia: 2011/00000256 Fecha de comienzo: 01/04/2011 Fecha de finalización: 12/04/2011
Financiado por	Plan propio, Universidad de Sevilla

4.2. Proyectos con empresas

Entre los proyectos con empresas mencionados a continuación cabe destacar el primero de ellos en el que participo como responsable del mismo.

Servicios de tecnologías de la información y comunicaciones a las aplicaciones de informes de los servicios centrales del servicio andaluz de salud (P023-14/E09)	
Información del proyecto	**Responsable: Julián Alberto García García, José Ponce** Tipo de proyecto: Contrato 68/83 Referencia: P023-14/E09 Fecha de comienzo: 18/08/2014 Fecha de finalización: 15/10/2014
Financiado por	Servicio Andaluz de Salud

EMPOWER (P047-14/E09)	
Información del proyecto	**Responsable: Julián Alberto García García, Francisco José Dominguez Mayo** Tipo de proyecto: Contrato 68/83 Referencia: P047-14/E09 Fecha de comienzo: 01/03/2015 Fecha de finalización: 31/11/2016
Financiado por	SERVINFORM, CDTI (Centro para el Desarrollo Tecnológico Industrial)

Plataforma WAN (P034-14/E09)	
Información del proyecto	**Responsable: Julián Alberto García García, María José Escalona Cuaresma** Tipo de proyecto: Contrato 68/83 Referencia: P034-14/E09 Fecha de comienzo: 01/01/2015 Fecha de finalización: 31/12/2016
Financiado por	WELLNES TELECOM, CDTI (Centro para el Desarrollo Tecnológico Industrial)

Apoyo en la integración de proveedores venta de entradas (Gestión dinámica de butacas) (P022-14/E09)	
Información del proyecto	Responsable: Francisco José Domínguez Mayo, Nicolás Sánchez Tipo de proyecto: Contrato 68/83 Referencia: P022-14/E09 Fecha de comienzo: 15/07/2014 Fecha de finalización: 23/10/2014
Financiado por	Fujitsu

Apoyo tecnológico y de investigación en procesos de soporte al mantenimiento de software en la oficina de gestión de proyectos de la consejería de cultura y deporte (P077-13/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P077-13/E09 Fecha de comienzo: 31/07/2013 Fecha de finalización: 01/07/2015
Financiado por	Consejería de Educación, Cultura y Deporte

Servicio de consultoría para la adecuación de la oficina de calidad de desarrollo de software al marco de mejores prácticas de ITIL (P016-10/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P016-10/E09 Fecha de comienzo: 17-05-2010 Fecha de finalización: 17-07-2010
Financiado por	Consejería de Cultura (Instituto Andaluz de las Artes y las Letras)

Desarrollo de una oficina de calidad de los proyectos TIC de la administración cultural (P050.1-07/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P050.1-07/E09 Fecha de comienzo: 15-10-2007 Fecha de finalización: 14-10-2008
Financiado por	Empresa Pública de Gestión de Programas Culturales

Apoyo en la adecuación de la arquitectura actual de la plataforma de eSalud a una arquitectura basada en SOA (P004-14/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P004-14/E09 Fecha de comienzo: 03/03/2014 Fecha de finalización: 30/06/2014
Financiado por	Consejería de Salud. Junta de Andalucía

Proyecto Sevilla en mi bolsillo (P078-13/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P078-13/E09 Fecha de comienzo: 01/07/2013 Fecha de finalización: 01/01/2015
Financiado por	Servicios de Desarrollo Orientado a Soluciones, S.L.

GEOLIA- first Generation of aErospace iDMU cOncept impLementIon And deployment (P025-13/E00)	
Información del proyecto	Responsable: Carmelo del Valle Sevillano Tipo de proyecto: Contrato 68/83 Referencia: P025-13/E00 Fecha de comienzo: 01/07/2013 Fecha de finalización: 31/12/2015
Financiado por	EADS - Construcciones Aeronáuticas, S.A. Avanade Spain

EOLO- factoriEs Of the future. industriaL develOpment (P023-13/E00)	
Información del proyecto	Responsable: Carmelo del Valle Sevillano Tipo de proyecto: Contrato 68/83 Referencia: P023-13/E00 Fecha de comienzo: 01/07/2013 Fecha de finalización: 31/12/2015
Financiado por	EADS - Construcciones Aeronáuticas, S.A. Glenser Aerospace, S.L.U.

Plataforma de E-Salud: Adaptación Plataforma a arquitectura basada procesos y pilotaje (P040-13/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P040-13/E09 Fecha de comienzo: 14/03/2013 Fecha de finalización: 30/09/2013
Financiado por	FISEVI - Fundación Pública Andaluza para Gestión de Investigación Salud de Sevilla

Asesoramiento y Desarrollo de aplicaciones informáticas ámbito clínico de Trasplantes. (P005-13/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de proyecto: Contrato 68/83 Referencia: P005-13/E09 Fecha de comienzo: 18/02/2013 Fecha de finalización: 31/10/2014
Financiado por	Fujitsu Tecnology Solutions, S.A.

Implantación de la metodología NDT en proyectos de i+d+i (P060-12/E09)	
Información del proyecto	Responsable: Francisco José Domínguez Mayo Tipo de proyecto: Contrato 68/83 Referencia: P060-12/E09 Fecha de comienzo: 29/11/2012 Fecha de finalización: 30/09/2013
Financiado por	Enxenio, S.L.

CALIPSONeo: Soluciones Aeronáuticas Avanzadas usando Procesos y Herramientas PLM (P051-12/E08)	
Información del proyecto	Responsable: Carmelo Del Valle Sevillano Tipo de Proyecto: Contrato 68/83 Referencia: P051-12/E08 Fecha de Inicio: 24-01-2012 Fecha de Finalización: 31-12-2013
Financiado por	EADS - Construcciones Aeronáuticas, S.A.

Servicio para la integración de los proyectos y herramientas de los departamentos de sistemas y explotación en el marco general de gestión de proyectos del servicio de informatica de la consejería de cultura. Lumen (P049-12/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P051-12/E08 Fecha de Inicio: 15/07/2012 Fecha de Finalización: 30/12/2012
Financiado por	Consejería de Educación, Cultura y Deporte

CALIPSO ShopFloor Technical Documentation. Desarrollo del modelado de datos e Interfaces (P059-12/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P059-12/E09 Fecha de Inicio: 09-07-2012 Fecha de Finalización: 31-12-2012
Financiado por	EADS - Construcciones Aeronáuticas, S.A.

THOT. Proyecto de innovación de la Gestión Documental aplicada a expedientes de contratación de servicios y obras de infraestructuras de transportes (1561/0493)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: 1561/0493 Fecha de Inicio: 16-04-2012 Fecha de Finalización: 30-06-2014
Financiado por	Agencia de Obra Pública de la Junta de Andalucía

Eficiencia en el Consumo Energético Mediante Dispoditivos Móviles de Cuarta Generación. (P042-12/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P042-12/E09 Fecha de Inicio: 01-01-2012 Fecha de Finalización: 30-06-2014
Financiado por	Sadiel, S.A.

ARCHIVAE: Archivo interoperable mediante datos en abierto enlazado. (P028-11/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P028-11/E09 Fecha de Inicio: 15/12/2011 Fecha de Finalización: 31/12/2013
Financiado por	Centro para el Desarrollo Tecnológico Industrial .(CDTI)

Montaje de una oficina tecnica de calidad para el proyecto AQUA (8006/5.47.2911)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: 8006/5.47.2911 Fecha de Inicio: 01/12/2008 Fecha de Finalización: 31/12/2011
Financiado por	Emasesa

Diseño de una Oficina de Proyectos Software (P019-08/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P019-08/E09 Fecha de Inicio: 21-10-2008 Fecha de Finalización: 20-11-2008
Financiado por	Empresa Pública de Gestión de Programas Culturales

Continuidad de los Procesos de Aseguramiento de la Calidad y Gestión de Proyectos TIC (P006-09/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P006-09/E09 Fecha de Inicio: 15-10-2008 Fecha de Finalización: 14-10-2009
Financiado por	Empresa Pública de Gestión de Programas Culturales

Diseño de una Oficina de Proyectos Software (P050-07/E09)	
Información del proyecto	Responsable: María José Escalona Cuaresma Tipo de Proyecto: Contrato 68/83 Referencia: P050-07/E09 Fecha de Inicio: 13-05-2008 Fecha de Finalización: 12-06-2008
Financiado por	Empresa Pública de Gestión de Programas Culturales

5. Registros de propiedad intelectual de software

Durante la carrera investigadora del doctorando, éste también ha realizado varios registros de propiedad intelectual de software en la Universidad de Sevilla.

- **NDT-GLOSSARY** SE01330-2009. Herramienta de soporte al proceso de Elicitación de Requisitos de la metodología NDT.
- **PLM$_4$BS** (en proceso administrativo de registro). Marco de trabajo y herramienta definida en esta tesis.
- **Herramienta SoaADAP** (en proceso administrativo de registro). Herramienta desarrollada en el marco del proyecto de adaptación de la plataforma eSalud a una arquitectura basada en procesos (Sección 4 del Capitulo VI).

6. Redes de investigación

Durante la carrera investigadora del doctorando, éste también ha formado parte de las siguientes redes de transferencia e investigación.

- **Redes nacionales:**
  - Red CaSA- Calidad del software Aplicada (TIN2010-12312-E).
  - TEDIS - Red Temática en Tecnologías para el Desarrollo Industrial de Software (TIN2011-15009-E).
- **Redes internacionales:**
  - AST- Red Internacional sobre arquitecturas de testing.
  - Red Temática Mexicana en Ingeniería del Software (244467) del Consejo Nacional de Ciencia y Tecnologia de México en la convocatoria Conacit Redes Tematicas 2014.
- **Redes de transferencia:**
  - INES- Grupo Calidad.