V Jornadas de Trabajo DYNAMICA

(DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures)

En Valencia, 23 y 24 de noviembre de 2006

Organización:

ISSI Ingeniería del Software y Sistemas de Información

Colaboradores:

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

ACTAS

V Jornadas de Trabajo DYNAMICA

(DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures)

En Valencia, 23 y 24 de noviembre de 2006

Editores:

Jennifer Pérez
Manuel Llavador
Cristóbal Costa
Nour Ali

Ingeniería del Software y Sistemas de Información

Financiado por el CICYT (Centro de Investigación Científica Y Tecnológica), Proyecto DYNAMICA (DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures)

Editores

Jennifer Pérez
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
E-mail: jeperez@dsic.upv.es

Manuel Llavador
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
E-mail: mllavador@dsic.upv.es

Cristóbal Costa
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
E-mail: ccosta@dsic.upv.es

Nour Ali
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
E-mail: nourali@dsic.upv.es

Distribuido en España, Grupo de Ingeniería del Software y Sistemas de Información, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia
Noviembre, 2006

Presidente de las jornadas

Isidro Ramos Salavert

Comité Organizador

Presidente: Jennifer Pérez Benedí

Vocales:
Nour Ali Irshaid
Silvia Mara Abrahao
Mª Eugenia Cabello Espinosa
Jose Hilario Canós Cerdá
Jose Ángel Carsí Cubel
Cristóbal Costa Soria
Abel Gómez Llana
Emilio Insfrán Pelozo
Patricio Letelier Torres
Rogelio Limón Cordero
Manuel Llavador Campos
Elena Mª Navarro Martínez
Mª Carmen Penadés Gramaje
Carlos Solís Pineda

Comité Técnico

Isidro Ramos Salavert, *Universidad Politécnica de Valencia*

Bárbara Álvarez Torres, *Universidad Politécnica de Cartagena*

Coral Calero Muñoz, *Universidad de Castilla-La Mancha*

Francisco José Rodríguez Urbano, *Universidad Carlos III*

Ambrosio Toval Álvarez, *Universidad de Murcia*

Subproyectos y Grupos Participantes

SUBPROYECTO: PRISMA – Plataforma OASIS para Modelos Arquitectónicos
TIC2003-07804-C05-01
Grupo de Investigación "Ingeniería de Software y Sistemas de Información" (ISSI)
Departamento de Sistemas Informáticos y Computación (DSIC)
Universidad Politécnica de Valencia (UPV)

SUBPROYECTO: ANCLA – Arquitecturas DiNámiCas para Sistemas de TeLeoperAción
TIC2003-07804-C05-02
Grupo DSIE (División de Sistemas e Ingeniería Electrónica
Departamento de Tecnologías de la Información y Comunicaciones
Universidad Politécnica de Cartagena (UPCT)

SUBPROYECTO: CALIPO – CALIdad en POrtales
TIC2003-07804-C05-03
Grupo ALARCOS
Departamento de Informática
Universidad de Castilla-La Mancha (UCLM)

SUBPROYECTO: CARATE – Controlador de Arquitectura Reconfigurable para aplicaciones de Teleoperación
TIC 2003-07804-C05-04
Grupo Laboratorio de Sistemas Inteligentes (LSI)
Departamento de Ingeniería de Sistemas y Automática (ISA)
Universidad Carlos III (UC3M)

SUBPROYECTO: PRESSURE – PREcise Software modelS and reqUirements REuse
TIC 2003-07804-C05-05
Grupo de Investigación en Ingeniería del Software (GIS)
Departamento de Informática y Sistemas
Universidad de Murcia (UMU)

Presentación

El Proyecto DYNAMICA (**DYN**amic and **A**spect-Oriented **M**odeling for **I**ntegrated **C**omponent-based **A**rchitectures) ha sido un proyecto coordinado de tres años de duración financiado por el Ministerio de Ciencia y Tecnología. DYNAMICA surgió por los intereses comunes de cinco grupos de investigación españoles: el grupo de Ingeniería del Software y Sistemas de Información (ISSI) de la Universidad Politécnica de Valencia (UPV), el grupo de División de Sistemas e Ingeniería Electrónica (DSIE) de la Universidad Politécnica de Cartagena (UPCT), el grupo ALARCOS de la Universidad de Castilla-La Mancha (UCLM), el grupo del Departamento de Ingeniería de Sistemas y Automática (ISA) de la Universidad Carlos III (UC3M) y el Grupo de Investigación en Ingeniería del Software (GIS) de la Universidad de Murcia (UMU).

Las últimas jornadas DYNAMICA nos proveen un espacio para compartir los resultados obtenidos a lo largo del proyecto. Así, el objetivo de estas jornadas es presentar los resultados en forma de comunicaciones y "demos" orientadas a enfrentar la dinámica del software desde una visión arquitectónica en dominios con problemas reales como los sistemas de teleoperación, los sistemas hidráulicos y los portales Web.

Se han recibido 16 contribuciones en forma de 11 comunicaciones y 5 "demos" desde los diferentes nodos, reflejando el espíritu e ilusión que hemos invertido en este proyecto. Además, las relaciones inter-nodos han sido intensas, mostrando que somos conscientes que es el único modo de conseguir un avance hacia un objetivo común es mediante el trabajo colectivo. La amistad y sinergia entre todos nosotros ha permitido un excelente trabajo y un gran resultado de este proyecto.

Esto nos ha situado en primera línea de investigación en Ingeniera del Software tanto nacional como internacionalmente y ha propiciado el nacimiento de META, el nuevo proyecto en el que compartiremos los esfuerzos gran parte de los miembros del consorcio.

La voluntad de estar juntos, de aceptarnos mutuamente, con las discusiones necesarias, han sido el acicate dialéctico de avance de DYNAMICA que animamos a continuar en META.

Isidro Ramos.

Presidente de las Jornadas

Índice

Artículos

Demostraciones

ARTÍCULOS

Towards a Data Quality Framework for Web Portals

Angélica Caro[1], Coral Calero[2]

[1] Department of Auditoria e Informática, University of Bio Bio
La Castilla s/n, Chillán, Chile
ancaro@inf-cr.uclm.es
[2] Head research of CALIPO project.
Alarcos Research Group. Information Systems and Technologies Department
UCLM-SOLUZIONA Research and Development Institute.
University of Castilla-La Mancha
Coral.Calero@uclm.es

Abstract. Advances in technology and the use of the Internet have favoured the appearance of a great variety of Web applications, among them Web Portals. These applications are important information sources and/or means of accessing information. Many people need to obtain information by means of these applications and they need to ensure that this information is suitable for the use they want to give it. In other words, they need to assess the quality of the data.

In recent years, several research projects were conducted on topic of Web Data Quality. However, there is still a lack of specific proposals for the data quality in Web portals. In this paper we introduce a data quality framework for Web portals. This framework is centred in the point of view of data consumers and uses a probabilistic approach for the data quality evaluation.

Keywords: Data Quality, Information Quality, Web Portal, Data Quality Framework, Bayesian Network.

1 Introduction

A Web portal is a site that aggregates information from multiple sources on the Web and organizes this material in an easy user-friendly manner [25]. Over the past decade the number of organizations which owns Web portals grows dramatically. Their have established portals to complement, substitute or widen existing services to their clients [26]. Many people use data obtained from portals to develop their work and to make decisions. These users or data consumers need to ensure that the data obtained are appropriate for the use they need. Likewise, the organizations owns of Web portals need to deliver data that meet user requirements to achieve the user's preference. So, a common interest between data consumers and portal's owners is the data quality.

In the literature, the concept of Data or Information Quality (DQ hereafter) is often defined as "fitness for use", i.e., the ability of a data collection to meet user

requirements [4, 23]. Besides, the terms “data” and “information” are often used as synonyms. In this work also we will use them as synonymous.

Research on data quality began in the context of information systems [15, 23] and it has been extended to contexts such as cooperative systems, data warehouses or e-commerce, amongst others. Due to the particular characteristics of Web applications and their differences from the traditional information systems, the research community started to deal with the subject of DQ on the Web [10].

However there are no works on DQ that address the particular context of Web portals [5], in spite of the fact that some works highlight the DQ as one of the relevant factors in the quality of a portal [18, 26]. Likewise, except for few works in the DQ area, like [3, 4, 24], most of them have looked at quality from the data producers or data custodians perspective and not from the data consumers perspective [3].

Consequently, our research aim is to create a Data Quality Model for Web portals focused on the data consumer perspective. For this, we have divided our work in two parts. The first, consisted in the definition of a theoretical model [6] named Portal Data Quality Model (PDQM). As a result of this we have identified a set of 34 DQ attributes than can be used to assess the DQ in a portal. The second, now in progress, is to convert PDQM in an operational model, i.e., that it can be used to assess DQ of Web portals. To reach this goal, DQ attributes in PDQM have to be specified in an operational way. This means that we need to define a structure where we can organize the DQ attributes and to associate measures and criteria for them.

Considering the subjectivity own of data consumer perspective and the uncertainty inherent to the quality perception, we have decided to use a probabilistic approach (based on Bayesian networks and fuzzy logic) to convert PDQM in an operational model. In this paper we show the first part of our work and the advances in the second part.

The rest of the paper is organized as follows. Section 2 presents the definition of PDQM. In Section 3 we describe briefly the approach used to convert PDQM in an operational model. Section 4 describes our advances to complete our model. Finally, section 5 shows our conclusions.

2 PDQM

PDQM is a data quality model for Web portals focused on the data consumer perspective. Their development was based on three key aspects:

- **Data consumer perspective**. When data management is conceptualized as a production process [23], we can identify three important roles in this process: (1) data producers (who generate data), (2) data custodians (who provide and manage resources for processing and storing data), and (3) data consumers (who access and use data for their tasks). The last perspective differs from the two others in two important aspects [3]: (1) data consumer has no control over the quality of available data and (2) the aim of consumers is to find data that match their personal needs, rather than provide data that meet the needs of others.

To consider the data consumer perspective in our model we have used the quality expectations of the data consumer on the Internet, proposed in [22]. These

expectations are organized into six categories: Privacy, Content, Quality of values, Presentation, Improvement, and Commitment.

- **Web data quality attributes**. Obtained from DQ frameworks proposed in the literature for different domains in the Web context. The idea was to take advantage of work already carried out in the Web context and apply it to Web portals.
- **Web portal functionalities**. Web portals present basic software functionalities to data consumer deploying their tasks. Under our perspective, the data consumer judges the quality of data by using the application functionalities. So, we used the web portal software functions that Collins proposes in [7] considering them as basics in our model. These functions are as follows: Data Points and Integration, Taxonomy, Search Capabilities, Help Features, Content Management, Process and Action, Collaboration and Communication, Personalization, Presentation, Administration, and Security.

Next, to define PDQM we developed a four-phase process showed follow.

2.1 Identification of Web DQ attributes

The first phase consisted in gathering Web DQ attributes from the literature. For this we have made a systematic review of the relevant literature [14]. Then, we selected works proposed for different domains in the Web context (Web sites [8, 13, 19], integration of data [2, 20], e-commerce [12], Web information portals [26], cooperative e-services [9], decision making [11], organizational networks [17] and DQ on the Web [10]). As result and after summarizing the collected initial set of attributes, we obtained 41 DQ attributes (see top of Table 1).

2.2 Definition of a Classification Matrix for Web DQ attributes

In the second phase, we have built a matrix for the classification of the DQ attributes obtained in previous phase. This matrix relates two basic aspects considered in our model: the data consumer perspective by means their DQ expectations on Internet [22] and the basic functionalities in a Web portal. On this matrix we carried out an analysis of what expectations were applicable in each different functionality of a Web portal, represented in Figure 1 with a "√" mark.

Category of Data Consumer Expectations \ Web Portal Functionalities	Data Points and Integration	Taxonomy	Search Capabilities	Help Features	Content Management	Process and Action	Collaboration and Communication	Personalization	Presentation	Administration	Security
Privacy					✓	✓	✓	✓		✓	✓
Content	✓	✓		✓	✓	✓			✓		
Quality of Values	✓		✓		✓	✓		✓	✓		✓
Presentation	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓
Improvement	✓	✓	✓		✓	✓			✓		
Commitment				✓	✓	✓					

Figure 1. Matrix to classify Web DQ attributes.

2.3 Classification of Web DQ attributes in the Matrix

In the third phase, we used the obtained matrix to classify the Web DQ attributes identified in phase 1. Then for each relationship between functionality and expectation, we assigned the Web DQ attributes that could be used by the data consumer to evaluate the DQ in a portal. We did it by studying the appropriateness of each attribute (based on its definition), in relation to the objective of each portal functionality and the user DQ expectation. On Table 1, we have summarized the attributes assigned for functionality.

Functionalities	Accessibility	Accuracy	Amount of data	Applicability	Attractiveness	Availability	Believability	Completeness	Concise Representation	Consistent Representation	Cost effectiveness	Customer support	Currency	Documentation	Duplicates	Ease of operation	Expiration	Flexibility	Granularity	Interactive	Internal consistency	Interpretability	Latency	Maintainable	Novelty	Objectivity	Ontology	Organization	Price	Relevancy	Reliability	Reputation	Response time	Security	Specialization	Source's information	Timeliness	Traceability	Understandability	Validity	Value-added	Total of Attributes
Data Points and Integration	✓	✓	✓				✓		✓	✓		✓	✓				✓								✓					✓	✓							✓	✓	✓		15
Taxonomy	✓		✓				✓		✓			✓				✓														✓	✓							✓	✓	✓		11
Search Capabilities	✓		✓				✓	✓	✓			✓	✓			✓									✓					✓	✓							✓	✓			13
Help Features	✓		✓						✓			✓				✓															✓								✓	✓		8
Content Management	✓		✓	✓		✓	✓	✓	✓			✓	✓	✓	✓	✓						✓			✓	✓		✓		✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	24
Process and Action	✓	✓	✓	✓		✓	✓	✓	✓			✓				✓	✓					✓				✓				✓	✓	✓		✓	✓			✓	✓	✓		21
Collaboration and Communication						✓						✓										✓								✓				✓					✓			6
Personalization		✓						✓					✓				✓																					✓	✓	✓		7
Presentation			✓		✓		✓		✓			✓	✓			✓	✓	✓				✓								✓	✓				✓			✓	✓	✓		15
Administration			✓						✓	✓						✓																	✓						✓			6
Security	✓	✓	✓					✓	✓							✓						✓											✓						✓	✓		10
Number of References	7	4	9	2	1	3	6	5	9	1	0	8	5	1	1	8	4	1	0	0	0	5	0	0	3	2	0	1	0	7	7	2	0	5	3	1	0	7	11	8	1	

Table 1. Data quality attributes assigned for functionality

As a result of this work we have a set of 34 DQ attributes that can be used for the DQ evaluation in portals, considering the data consumer perspective.

2.4 Validation

The fourth phase consists on the validation of the set of DQ attributes selected. Despite that this phase is not essential still, because later we will develop another validation with the whole model; we have decided to carry out this previous validation by means a survey. In this survey, data consumers of web portals to be asked about the importance for them of the DQ attributes considered in PDQM. The survey is run on http://alarcos.inf-cr.uclm.es/PDQM/.

3 A probabilistic approach to make PDQM an operational model

With the aim to use PDQM in an evaluation process, we need to convert it in an operational model. For this, and considering the subjectivity of data consumer perspective and the uncertainty inherent to the quality perception, we have decided to use a probabilistic approach as the proposed in [16]. This approach involves Bayesian

networks and fuzzy logic. Bayesian networks model the problems that involve uncertainty, and combine the advantages of an intuitive visual representation with a sound mathematical basis in Bayesian probability [21]. Finally, fuzzy logic provides an effective conceptual framework for dealing with the problem of knowledge representation in an environment of uncertainty and imprecision [27].

A Bayesian Network (BN) is a directed acyclic graph, whose nodes are the uncertain variables and edges are the causal or influential links between variables. A conditional probability function models the uncertain relationship between each node and its parents [21]. In our context, BNs offer an interesting framework with which it is possible to:

- Represent the interrelations among DQ attributes in an intuitive and explicit way by connecting influencing factors to influenced ones. Such a representation facilitates the comprehension of the model, its validation, its evolution and its exploitation.
- Circumvent the problems of subjectivity uncertainty.
- Actually use the obtained network to predict/estimate the data quality of a portal.
- Isolate responsible factors in the case of low data quality.

Another interesting property of the Bayesian approach is the fact that it considers the probability as being a dynamic entity that can be updated as more data arrive (self learning mechanism). New data may naturally improve the degree of belief in certain propositions [1]. Consequently, a BN model is particularly adapted to the changing domain of web portals.

4 Process to convert PDQM in an operational model

To convert PDQM in an operational model, we have developed a process with the following phases:

a. To organize hierarchically the DQ attributes of PDQM. This phase consisted on the definition of criteria to organize and classify the DQ attributes of PDQM.
b. To build a graph to represent PDQM. In this phase the DQ attributes organized in the previous phase, will be represented in the form of a BN.
c. To prepare the BN for the evaluation process. This phase is developed in sub-phases due to the size of the obtained BN. In each sub-phase we have selected a sub-network of the BN developing the next activities on it:
    - To define quantifiable variables for the DQ attributes in the last level (entry nodes) of the sub-network.
    - To define the node probability table for each intermediate node of the sub-network.
d. To develop an experiment to compare the judgements of a set of portal-user subjects with the evaluation results produced by the BN.

On the first phase we have selected the conceptual DQ framework developed in [24] as criteria of classification to organize the DQ attributes of PDQM. With the idea of considering some aspects inherent to the Web context, in our work we have renamed and redefined the Accessibility category as Operational category, emphasizing with

this the importance of the role of systems not only with respect to accessibility and security but also to aspects as personalization, collaboration, etc. So, the final DQ categories to classify the DQ attributes of PDQM are: Intrinsic, Operational, Contextual and Representational.

After this, and considering the definition of each DQ category, we have classified all the DQ attributes of PDQM into these categories.

In the second phase, we have generated new levels in the BN based on the relationships of direct influences among the DQ attributes in each category. We used the definitions of the DQ attributes to establish these relationships. Our aim was to establish which DQ attribute in a category had direct influence on other DQ attributes in the same category, and eventually on attributes in other category.

Based on these relationships we have built the graph of the BN which represents PDQM, see Figure 2. In this BN we can distinguish the following levels:

- Level 0, where the PDQ is the node that represents DQ in the whole portal.
- Level 1, where node represents the DQ in each DQ category in a portal. Obviously, the node PDQ is defined in terms of the other four nodes.
- Level 2, where nodes represent the DQ attributes with a direct influence over each one of the DQ categories.
- Level 3, where nodes represent the DQ attributes with a direct influence over each one of the DQ attributes in Level 2.
- Level 4, with only one node that represents a direct influence over each one of the DQ attributes in Level 3.

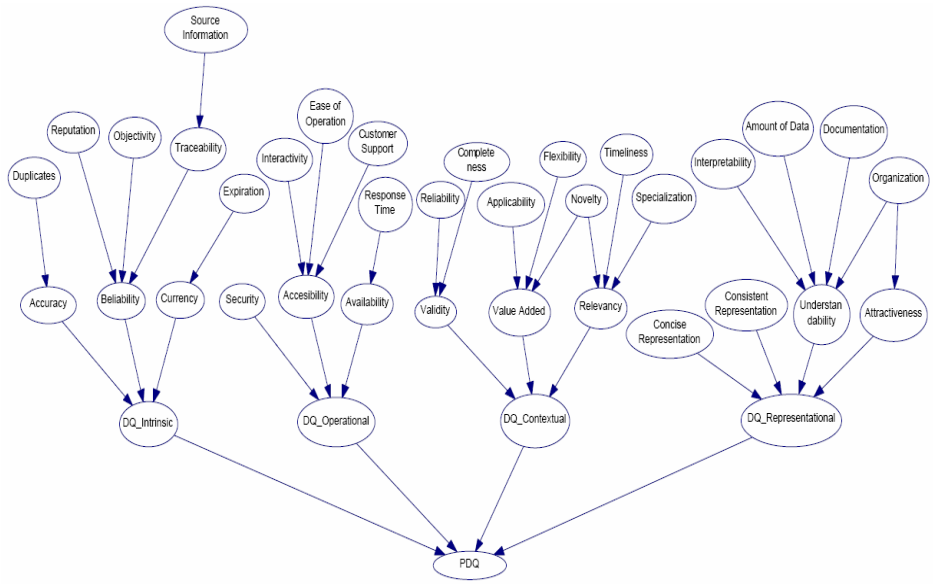


Figure 2. Graph of the BN that represents PDQM.

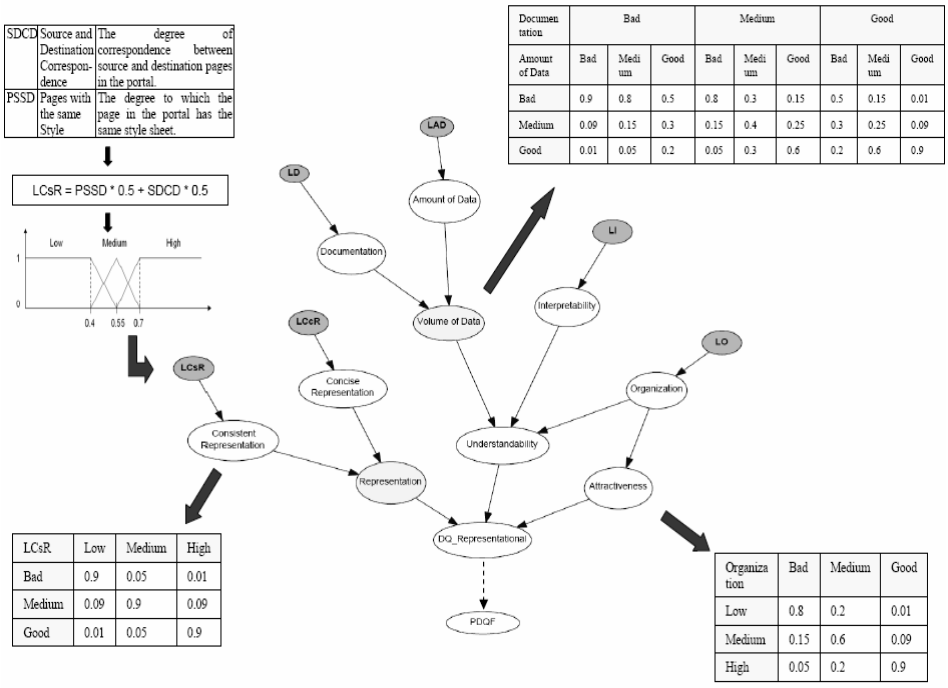
As it can be seen in the resultant graph, some relationships are more complex than others. For example: the variables Accessibility and Understandability have three and four parents respectively. Then with the aim of facilitating the use of the BN, will be

necessary to create new nodes (intermediate nodes) that permit to reduce the number of parents for each node.

In the third phase, corresponding to the preparation of the BN for the evaluation process, we will work separately with each sub-network of the BN created. For each one, we will develop two sub-phases: first, to define one quantifiable variable (indicator) for each DQ attribute in the last level of the BN, and second, to define the probability tables for each intermediate node.

As a point of start, we have selected the DQ_Representational sub-network. As we can see in Figure 2, the original sub-network had two nodes with four parents (Understandability and DQ_Representational) and was necessary to create two synthetic nodes (Representation and Volume of Data) in order to reduce the combinatory explosion in the next step during the preparation of the probability tables, obtaining the sub-network showed in Figure 3.

For this sub-network we have defined one indicator for each entry node in the BN (see on Figure 3 the nodes in the last level). In general we have selected and defined the measures according to each attribute (entry node) definition. To exemplify this in the Figure 3 we show the definition of indicator LCsR (Level of Consistent Representation). For the calculation and definition of each indicator we have used several base and derived measures. All the indicators take a numerical result between 0 and 1. Consequently, with the fuzzy approach used, we have defined for each one the labels that represent the fuzzy sets associated with it. Finally, a membership function was defined to determine the degree of membership of each indicator respect of the fuzzy labels.



SDCD	Source and Destination Correspon-dence	The degree of correspondence between source and destination pages in the portal.
PSSD	Pages with the same Style	The degree to which the page in the portal has the same style sheet.

Documen tation	Bad			Medium			Good		
Amount of Data	Bad	Medi um	Good	Bad	Medi um	Good	Bad	Medi um	Good
Bad	0.9	0.8	0.5	0.8	0.3	0.15	0.5	0.15	0.01
Medium	0.09	0.15	0.3	0.15	0.4	0.25	0.3	0.25	0.09
Good	0.01	0.05	0.2	0.05	0.3	0.6	0.2	0.6	0.9

LCsR	Low	Medium	High
Bad	0.9	0.05	0.01
Medium	0.09	0.9	0.09
Good	0.01	0.05	0.9

Organiza tion	Bad	Medium	Good
Low	0.8	0.2	0.01
Medium	0.15	0.6	0.09
High	0.05	0.2	0.9

Figure 3. Preparation of sub network DQ_Representational for evaluation process.

The intermediate nodes are nodes defined by their parents and not directly measurable. So, whereas these nodes are not measurable, their probability distribution was given by expert judgments. On the other hand, we considered that this probability distribution can differ depending on the Web portal context.
In this work as a matter of example, we have considered the educational context and we have defined the nodes probability tables considering that this BN will be applied to university portals. Figure 3 shows some nodes probability tables for sub-network.
For the last phase an experiment will be developed. This study will give us the judgments of a group of subjects about the DQ Representational in a set of Web portals in the context selected. The idea is to compare the valuations of the subjects with the results of the BN, and based on this comparison adjust the sub-network.

5 Conclusions and the future work

In this paper, we have shown the development of a DQ model for Web portals (PDQM) centered in the data consumer perspective. This work has been developed in two parts. The first consisted in the definition of the theoretical foundations of the model and in the identification of a set of 34 DQ attributes that can be used for the DQ evaluation in Web portals. The second part, now in progress, consists in the transformation of PDQM into an operational DQ model. For this, we are using a probabilistic approach. As a result of this part, we have generated a BN that represents PDQM and we have implemented a sub-network of it for a specific Web portal context.
As future work we will develop an experiment to validate the sub-network implemented and after we will work with the other sub-networks of PDQM until complete and validate the whole model.
The choice of a probabilistic approach to generate the framework is motivated by the fact that many issues in quality assessment are circumvented: threshold value definition, measure combination, and uncertainty.
One of the advantages of our framework will be its flexibility. Indeed, the idea is to develop a global framework that could be adapted for both the goal and the context of evaluation. From the goal perspective, the user can choose the sub-network that evaluates the characteristics he is interested in. From the context point of view, the parameters (probabilities) can be changed to consider the specific context of the evaluated portal. This operation can be performed using available historical data from the organization.

Acknowledgments. This research is part of the following projects: CALIPO (TIC2003-07804-C05-03), CALIPSO (TIN20005-24055-E) supported by the Ministerio de Educación y Ciencia (Spain) and COMPETISOFT (506PI0287) financed by CYTED.

References

1. Baldi, P., Frasconi, P., and Smyth, P., *Modeling the Internet and the Web; Probabilistic Methods and Algorithms.* 2003: Wiley.
2. Bouzeghoub, M. and Peralta, V. A Framework for Analysis of data Freshness. in International Workshop on Information Quality in Information Systems, (IQIS2004). 2004. Paris, France: ACM.
3. Burgess, M., Fiddian, N., and Gray, W. Quality Measures and The Information Consumer. in Proceeding of the Ninth International Conference on Information Quality. 2004.
4. Cappiello, C., Francalanci, C., and Pernici, B. Data quality assessment from the user´s perspective. in International Workshop on Information Quality in Information Systems, (IQIS2004). 2004. Paris, Francia: ACM.
5. Caro , A., Calero, C., Caballero, I., and Piattini, M. Data quality in web applications: A state of the art. in IADIS International Conference WWW/Internet 2005. 2005. Lisboa-Portugal.
6. Caro, A., Calero, C., Caballero, I., and Piattini, M. Defining a Data Quality Model for Web Portals. in WISE2006, The 7th International Conference on Web Information Systems Engineering. 2006. Wuhan, China: Springer LNCS 4255.
7. Collins, H., *Corporate Portal Definition and Features*. 2001: AMACOM.
8. Eppler, M., Algesheimer, R., and Dimpfel, M. Quality Criteria of Content-Driven Websites and Their Influence on Customer Satisfaction and Loyalty: An Empirical Test of an Information Quality Framework. in Proceeding of the Eighth International Conference on Information Quality. 2003.
9. Fugini, M., Mecella, M., Plebani, P., Pernici, B., and Scannapieco, M. (2002). Data Quality in Cooperative Web Information Systems.Personal Communication. citeseer.ist.psu.edu/fugini02data.html.
10. Gertz, M., Ozsu, T., Saake, G., and Sattler, K.-U., Report on the Dagstuhl Seminar "Data Quality on the Web". SIGMOD Record, 2004. vol. 33, Nº 1: p. 127-132.
11. Graefe, G. Incredible Information on the Internet: Biased Information Provision and a Lack of Credibility as a Cause of Insufficient Information Quality. in Proceeding of the Eighth International Conference on Information Quality. 2003.
12. Katerattanakul, P. and Siau, K., *Information quality in internet commerce desing*, in *Information and Database Quality*, M. Piattini, C. Calero, and M. Genero, Editors. 2001, Kluwer Academic Publishers.
13. Katerattanakul, P. and Siau, K. Measuring Information Quality of Web Sites: Development of an Instrument. in Proceeding of the 20th International Conference on Information System. 1999.
14. Kitchenham, B. (2004). Procedures for Performing Systematic Reviews.RN: 0400011T.1. http://www.idi.ntnu.no/emner/empse/papers/kitchenham_2004.pdf.
15. Lee, Y., AIMQ: a methodology for information quality assessment. Information and Management. Elsevier Science, 2002: p. 133-146.
16. Malak, G., Sahraoui, H., Badri, L., and Badri, M. A Proposal of a Probabilistic Framework for Web-Based Applications Quality. in 10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, (QAOOSE06). 2006.
17. Melkas, H. Analyzing Information Quality in Virtual service Networks with Qualitative Interview Data. in Proceeding of the Ninth International Conference on Information Quality. 2004.
18. Moraga, M.Á., Calero, C., and Piattini, M., Comparing different quality models for portals. Online Information Review., 2006. Vol. 30(5): p. 555-568.
19. Moustakis, V., Litos, C., Dalivigas, A., and Tsironis, L. Website Quality Assesment Criteria. in Proceeding of the Ninth International Conference on Information Quality. 2004.

20. Naumann, F. and Rolker, C. Assesment Methods for Information Quality Criteria. in Proceeding of the Fifth International Conference on Information Quality. 2000.
21. Neil, M., Fenton, N.E., and Nielsen, L., Building large-scale Bayesian Networks. The Knowledge Engineering Review, 2000. 15(3): p. 257-284.
22. Redman, T., *Data Quality: The field guide*. 2000, Boston: Digital Press.
23. Strong, D., Lee, Y., and Wang, R., Data Quality in Context. Communications of the ACM, 1997. Vol. 40, Nº 5: p. 103 -110.
24. Wang, R. and Strong, D., Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems; Armonk; Spring 1996, 1996. 12(4): p. 5-33.
25. Xiao, L. and Dasgupta, S., *User Satisfaction with Web Portals: An empirical Study*, in *In Web Systems Design and Online Consumer Behavior*, Y. Gao, Editor. 2005, Idea Group Publishing, Hershey, PA. p. 193-205.
26. Yang, Z., Cai, S., Zhou, Z., and Zhou, N., Development and validation of an instrument to measure user perceived service quality of information presenting Web portals. Information and Management. Elsevier Science, 2004. 42: p. 575-589.
27. Zadeh, L.A., Knowledge Representation in Fuzzy Logic. IEEE Transactions on Knowledge and Data Engineering, 1989. 1(1): p. 89-100.

PtSM: Modelo de Selección de Portlets

Mª Ángeles Moraga, Coral Calero, Mario Piattini

Grupo de Investigación ALARCOS.
Instituto de Desarrollo e Investigación UCLM-SOLUZIONA.
Universidad de Castilla-La Mancha
Paseo de la Universidad, 4. 13071. Ciudad Real, España
{mariaangeles.moraga, coral.calero, mario.piattini}@uclm.es

Abstract. El uso de los portales Web continúa aumentando, lo que muestra su importancia dentro de la sociedad de la información actual. Hoy en día, los usuarios pueden cambiar de forma fácil de un portal a otro, por lo que si no queremos que nuestro portal desaparezca debemos mejorar/calcular la calidad del mismo. En concreto, en este artículo nos centramos en un tipo especial de portales que son los portales construidos mediante la agregación de portlets. Los portlets son componentes web y pueden ser pensados como COTS pero en un escenario Web. Por tanto, los desarrolladores de los portales tendrán que seleccionar el portlet más adecuado para su portal de entre un conjunto de portlets que tienen un conjunto parecido de funciones para las tareas y los objetivos específicos de los usuarios. Este artículo presenta un modelo de selección de portlets que guía al desarrollador del portal a realizar dicha selección.

Keywords: modelo de calidad, medidas, portal web, portlets

1 Introducción

Los portales web son aplicaciones basadas en Internet que permiten acceso a diferentes fuentes (proveedores) a través de una única interfaz [10] que proporciona personalización, facilidades en el registro (los usuarios sólo se tienen que registrar una vez) y agregación de contenido desde diferentes fuentes [8].

Pero las características de los portales pueden diferir de unos a otros, obteniendo así diferentes tipos de portales. Actualmente, a pesar del reconocimiento por parte de los investigadores de la existencia de diferentes tipos de portales, no se ha llegado a un consenso en cuanto a la clasificación de los mismos.

De entre todas las posibles clasificaciones destacamos la realizada en [1], donde dividen a los portales en generaciones en función de la evolución que han sufrido, obteniendo así la siguiente clasificación:

- *Portales de primera generación*: estos portales solían presentar una arquitectura software monolítica que se encargaba del desarrollo y mantenimiento del portal.
- *Portales de segunda generación*: permiten que los usuarios se creen una o más páginas personales compuestas de *portlets* personalizables, es decir,

miniaplicaciones web, locales o remotas, que representan fragmentos de marcas (noticias, tiempo, deportes, etc.) que el portal puede agregar en una página.

Sin embargo, los primeros portlets presentaban una falta de interoperabilidad entre ellos, como consecuencia de la inexistencia de un modelo común. Pero dicho problema fue resuelto tras la aparición, en el año 2003, del estándar WSRP (Web Services for Remote Portlets) [12]. Esto abre la posibilidad de la existencia de un mercado de portlets similar al mercado de componentes.

Por tanto, si se quiere un "buen" portal será necesario seleccionar los portlets más adecuados para construirlos. Por ello, será necesaria la utilización de modelos de calidad que ayuden a los desarrolladores de los portales a seleccionar el portlet más adecuado.

A pesar de que existen modelos de calidad para diferentes contextos, un modelo de calidad específico para la selección de portlets no existe. Por este motivo, nuestro objetivo consiste en desarrollar un modelo de selección de portlet (PtSM) que permita determinar cuál es el mejor portlet de entre un conjunto de portlets que tienen un conjunto parecido de funciones para las tareas y los objetivos específicos de los usuarios.

Este articulo se divide como sigue: en la sección segunda se presenta el modelo de selección de portlets, el cual está formado por un modelo de calidad de portlets que se presenta en la tercera sección y un conjunto de características que no están relacionadas con la calidad. Finalmente, en la sección cuarta se presentan algunas conclusiones y el trabajo futuro a realizar.

2 Modelo de selección de portlets

El modelo de selección de portlets, PtSM, permite a los desarrolladores de los portales, por un lado, determinar el mejor portlet de entre un conjunto de portlets que tienen un conjunto parecido de funciones para las tareas y los objetivos específicos de los usuarios, de acuerdo a su calidad y a otras características que no están relacionadas con la calidad pero que influyen en la selección. Y por otro lado, detectar los puntos débiles de cada portlet que hacen que su nivel de calidad disminuya – esta información puede ser utilizada posteriormente para incrementar la calidad de los mismos.

El uso del modelo de selección (PtSM) se resume en la figura 1. En primer lugar, se estiman los valores de las diferentes características que conforman el modelo para cada uno de los portlets evaluados. A continuación, considerando las necesidades de los usuarios se determina cuál es el mejor portlet de entre el conjunto de portlets evaluados. Obteniendo como resultado final el mejor portlet en función de cada caso.

Como se observa en la figura 1 PtSM está formado por un modelo de calidad de portlets (PtQM) y un conjunto de características que no están directamente relacionadas con la calidad.

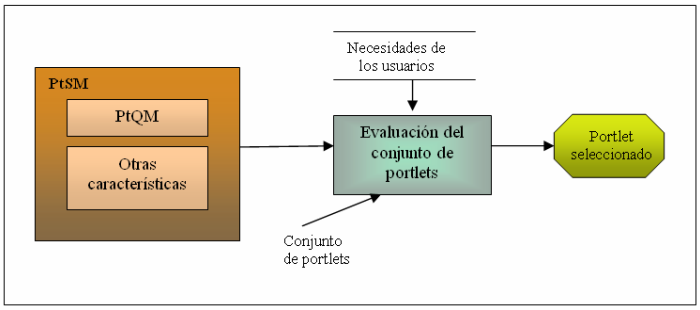


Fig. 1. Uso del modelo de selección de portlets (PtSM)

Dentro del conjunto de características que no están directamente relacionadas con la calidad se pueden diferenciar, por ejemplo, aquellas que afectan al portlet como producto que tiene que ser comprado y agregado a nuestro software o la reputación o perfil del proveedor del portlet.

En concreto, para determinar el conjunto de características y subccaracterísticas que no están directamente relacionados con la calidad, se han analizado diferentes estándares. Estos estándares se pueden dividir en dos grupos: estándares dirigidos a herramientas CASE y estándares orientados a software de computación.

Por tanto, nuestro trabajo ha consistido en adaptar las características presentadas en estos estándares al contexto de los portlets. En la tabla 1, se presentan las características junto con sus subccaracterísticas y la fuente (o estándar) a partir del cual se ha adaptado la subcaracterística.

Tabla 1. Otras características no relacionadas con calidad, para la selección de portlets.

Característica	Subcaracterística	Fuente
Adquisición	Coste	ISO/IEC 14102
	Política de licencias	ISO/IEC 14102 e IEEE 1209
	Encuestas a grupos de usuarios	ISO/IEC 90003
Indicadores de soporte	Perfil del vendedor del portlet	ISO/IEC 14102 e IEEE 1209
	Perfil del portlet	ISO/IEC 14102 e IEEE 1209
	Información relativa a la instalación	ISO/IEC 90003
	Soporte del vendedor del portlet	IEEE 1209
	Realimentación del cliente para atender sus quejas	ISO/IEC 90003
Conformidad	Certificación del vendedor del portlet	ISO/IEC 14102 e IEEE 1209

3 Modelo de calidad de portlets (PtQM)

En este apartado se presentan las características de calidad que han sido tenidas en cuenta en PtSM y que se han agrupado en un modelo de calidad denominado PtQM.

Debido a que los portlets se pueden considerar como una mezcla entre los componentes software y las aplicaciones web, para la realización de este modelo se

han tenido en cuenta trabajos relacionados con productos software [7], con aplicaciones web [4, 9, 11, 13] y con componentes software [2, 3, 5, 14]. Cabe destacar que estos trabajos no pueden ser directamente aplicados al contexto de los portlets ya que éstos presentan algunas especificidades que los hacen diferentes, por lo tanto, sólo se han utilizado como base para la realización de nuestro modelo.

En las tablas 2 y 3 se muestran, respectivamente, las definiciones de las características y subccaracterísticas que finalmente han sido identificadas en PtQM.

Tabla 2. Definición de las características de calidad para el contexto de los portlets

Característica	Definición
Funcionalidad	Capacidad del portlet para proporcionar funciones que satisfagan las necesidades explícitas e implícitas cuando el portlet es utilizado en las condiciones especificadas.
Fiabilidad	Capacidad del portlet para mantener un nivel especificado de rendimiento cuando se utiliza en las condiciones especificadas.
Usabilidad	Capacidad del portlet para ser utilizado por los desarrolladores del portal cuando construyen el portal con él.
Eficiencia	Capacidad del portlet para proporcionar un rendimiento adecuado, relativo a la cantidad de recursos utilizados, bajo condiciones indicadas.
Reusabilidad	Capacidad del portlet para ser reusado en diferentes portales por diferentes desarrolladores.

Tabla 3. Definición de las subccaracterísticas de calidad para el contexto de los portlets.

Carac-terística	Subcaracte-rística	Definición para el contexto de los portlets	
Funcionalidad	Exactitud	Capacidad del portlet para proporcionar los resultados o efectos correctos con el grado de precisión acordado.	
	Interope-rabilidad	Interoperabilidad con el portal	Capacidad del portlet para interaccionar con uno o más portales.
		Interoperabilidad con otros portlets	Capacidad del portlet para interaccionar con otros portlets.
	Seguridad	Capacidad del portlet para prevenir accesos no autorizados, tanto accidentales como deliberados, a programas o datos.	
	Autosuficiencia	Capacidad del portlet para realizar por sí mismo la función que se espera que realice	
	Cohesión funcional	Capacidad del portlet para hacer uso de todos sus elementos a la hora de llevar a cabo su funcionalidad.	
	Conformidad	Capacidad del portlet para adaptarse a los estándares, convenciones o regulaciones en leyes y prescripciones relativos a la funcionalidad.	

Fiabilidad	Madurez	Capacidad del portlet para evitar fallos provocados por errores en el software.
	Tolerancia a fallos	Capacidad del portlet para mantener un nivel de rendimiento determinado en caso de defectos en el software o incumplimiento de su interfaz.
	Recupera-bilidad	Capacidad del portlet para recuperarse de fallos inesperados
	Degradabi-lidad	Esfuerzo necesario para re-establecer la funcionalidad esencial del portlet después de un fallo.
	Evaluabilidad	Capacidad del portlet para permitir a los desarrolladores del portal evaluar su forma y/o su contenido
	Disponibilidad	Capacidad del portlet para estar operativo todos los días del año 24/7/365
	Conformidad	Capacidad del portlet para adaptarse a los estándares, convenciones o regulaciones en leyes y prescripciones relativos a la fiabilidad.
Usabilidad	Compren-sibilidad	Capacidad del portlet para permitir al usuario comprender cual es la funcionalidad del portlet.
	Facilidad de aprendizaje	Capacidad del portlet para permitir al usuario aprender como el portlet lleva a cabo sus objetivos
	Customiza-bilidad	Capacidad del portlet para ser customizado por los usuarios, consiguiendo reducir los esfuerzos necesarios para utilizarlo e incrementando la satisfacción del usuario con respecto al mismo.
	Conformidad	Capacidad del portlet para adaptarse a los estándares, convenciones o regulaciones en leyes y prescripciones relativos a la usabilidad.
Eficiencia	Comporta-miento temporal	Capacidad del portlet para proporcionar tiempos de respuesta y de procesamiento apropiados cuando realiza sus funciones bajo determinadas condiciones
	Utilización de recursos	Capacidad del portlet para utilizar cantidades y tipos de recursos apropiados cuando el portlet realiza su función bajo determinadas condiciones.
	Conformidad	Capacidad del portlet para adaptarse a los estándares, convenciones o regulaciones en leyes y prescripciones relativos a la eficiencia.
Reusa-bilidad	Comprensi-bilidad	Capacidad del portlet para permitir a los usuarios comprender cual es la funcionalidad del portlet.
	Portabilidad	Capacidad del portlet para ser transferido de un entorno a otro.

Una vez determinado el conjunto de características y subccaracterísticas que afectan a la calidad de los portlets, el siguiente paso consiste en definir atributos y medidas[1]

[1] Utilizamos la ontología de la medición del software (SMO), propuesta en [6], para presentar las medidas de una forma formal, utilizando conceptos y terminología obtenidos mediante consenso.

con el objetivo de poder determinar el nivel de calidad de cada una de las características.

Siguiendo la ontología de la medición propuesta en [6], cabe destacar que para el contexto que nos ocupa la clase de entidad es el "conjunto de portlets que proporcionan un conjunto apropiado de funciones para tareas específicas y objetivos de los usuarios". Mientras que una entidad es "un portlet concreto al que se le aplica la medición"

La necesidad de información y los conceptos medibles serán diferentes para cada una de las características, por ejemplo, la necesidad de información para la usabilidad es "evaluar la usabilidad de un conjunto de portlets que tienen un conjunto parecido de funciones para las tareas y los objetivos específicos de los usuarios", mientras que sus conceptos medibles son: comprensibilidad, facilidad de aprendizaje, customización y conformidad. Por tanto, los conceptos medibles de una determinada características se corresponden con las subccaracterísticas de la misma.

En la tabla 4 se muestran, a modo de ejemplo los atributos y medidas identificados para el concepto medible comprensibilidad de la característica usabilidad.

Tabla 4. Otras características no relacionadas con calidad, para la selección de portlets.

Subcarac-terística	***Atributo***	Medida			
		Medida Base	Escala	Tipo de escala	Unidad de medición
Compren-sibilidad	Idioma del interfaz	Número de idiomas soportados por la interfaz	Número natural	Ratio	Lenguajes
	Documen-tación	El vendedor del portlet proporciona documentación	Boolean (0/1)	Nominal	Documen-tación
		Número de lenguajes en los que la documentación se proporciona	Número natural	Ratio	Lenguajes
		El portlet especifica su funcionalidad	Boolean (0/1)	Nominal	Descripción

Una vez identificados los atributos y las medidas, el siguiente paso consiste en calcular el indicador de cada una de las características identificadas. Un indicador es una medida derivada a partir de otras medidas que utiliza un modelo de análisis como forma de medir. El objetivo del modelo de análisis es definir un algoritmo para combinar una o más medidas con los criterios de decisión asociados. Estos criterios pueden ser valores umbrales, objetivos o patrones usados para determinar la necesidad de una acción o investigación posterior o para describir el nivel de confianza de un resultado dado.

Para calcular el indicador de cada característica, en primer lugar se tendrá que calcular un indicador para cada uno de los conceptos medibles asociados a la característica. A continuación, estos indicadores se transformarán (mediante la utilización de un modelo de análisis) para obtener el indicador de la característica. En la figura 2, se muestran a modo de ejemplo los diferentes indicadores que se tienen

que definir así como las transformaciones que se han de llevar a cabo para obtener el indicador de usabilidad.

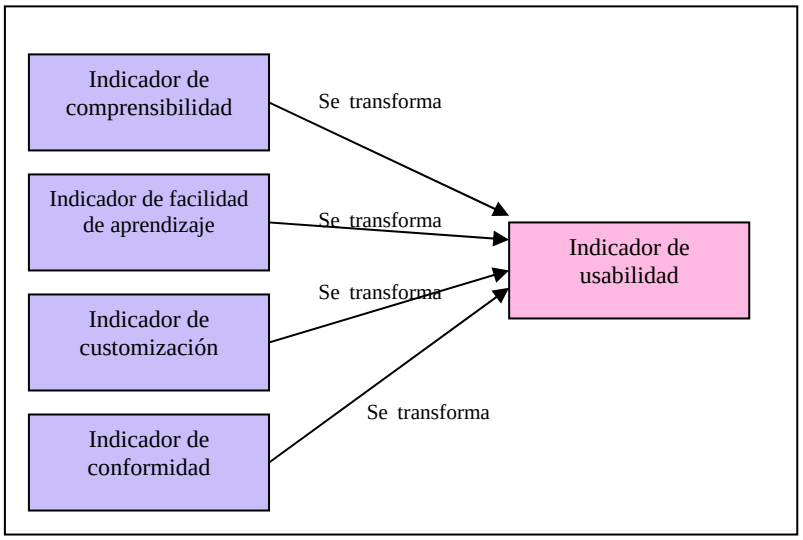


Fig. 2. Obtención del indicador de usabilidad.

Como se especificó anteriormente un indicador es una medida que es derivada de otras medidas utilizando un modelo de análisis como forma de medir. Por tanto, el primer paso a realizar consiste en definir un modelo de análisis. En este caso el modelo análisis es una función definida como sigue:

$$F(\text{Indicador}_x) = Suma\_Rango\_Medidas \quad (1)$$

Donde:

- Indicador_x= representa a un indicador.
- Suma_Rango_Medidas= representa la suma de los valores de los rangos[2] de las diferentes medidas definidas para el concepto medible que representa el indicador.

Para poder calcular el valor de la función F, en primer lugar, los valores de las medidas definidas para cada uno de los conceptos medibles se transforman en un rango de 0 a 5 y se indica cual es el valor mínimo aceptable que puede adquirir la medida y el valor máximo. Por ejemplo en la tabla 5 se muestra la transformación de los valores de la medida “número de idiomas soportados por la interfaz”, identificada para el concepto medible comprensibilidad de la característica funcionalidad, en un rango de 0 a 5.

[2] Los valores de las medidas se transforman en un rango que puede oscilar entre 0 y 5, con ello se consigue que todas las medidas adquieran el mismo conjunto de valores y por tanto, se facilita el análisis de los resultados de la función.

Tabla 5. Rango para la medida “número de idiomas soportados por la interfaz”

Rango	Valor de la medida	
0	1	1.5=>No acept.
1	1	
2	1	
3	1	
4	2	4=>Vmin
5	> 2	5=>Vmax

Esto mismo se ha de realizar para cada una de las medidas identificadas. De forma que teniendo en cuenta los rangos definidos para cada medida, la función F para la comprensibilidad adquiere un valor máximo de 17.5 y un valor mínimo de 14.5. Por tanto, estableciendo intervalos equidistantes entre ambos valores, los niveles para el indicador de comprensibilidad quedan tal y como se muestran en la tabla 6.

Tabla 6. Niveles para el indicador de comprensibilidad

Indicador de comprensibilidad	Criterios de decisión
Excelente	F=17.5
Alta	16.5 ≤ F<17.5
Media	15.5≤ F<16.5
Aceptable	14.5 ≤ F<15.5
No aceptable	F<14.5

Exactamente igual se ha de hacer para los indicadores de facilidad de aprendizaje, customización, y conformidad que se transformarán junto con el indicador de comprensibilidad en el indicador de usabilidad. Para realizar esta transformación se utiliza el modelo de análisis, es decir, la función F. Para ello los valores de los niveles que puede adquirir cada indicador son transformados en un rango de 0 a 5 de acuerdo a la tabla 7.

Tabla 7. Rango para los niveles que pueden adquirir los indicadores

Rango	Valor de la medida	
0	No aceptable	0=>No acept.
1	Aceptable	1.5=>Vmin
2	Aceptable	
3	Medio	
4	Alto	
5	Excelente	5=>Vmax

Finalmente, se calcula tanto el valor máximo como mínimo aceptable de la función F, obteniendo así el valor que la función F tiene que adquirir para que el nivel del indicador de usabilidad sea excelente (coincide con el valor máximo de la función) y no aceptable (cualquier valor inferior al valor mínimo aceptable). Los niveles intermedios se calculan estableciendo intervalos equidistantes entre el valor máximo y el valor mínimo aceptable. En la tabla 8 se muestran todos los niveles que puede adquirir el indicador de usabilidad.

Tabla 8. Niveles para el indicador de usabilidad

Indicador de usabilidad	Criterios de decisión
Excelente	F=20
Alta	$15.82 \leq F<20$
Media	$11.66 \leq F<15.82$
Aceptable	$7.5 \leq <11.66$
No aceptable	F<7.5

Este mismo proceso se ha de seguir para calcular el indicador de funcionalidad, fiabilidad, eficiencia y reusabilidad.

4 Conclusiones y trabajo futuro

En este artículo se ha presentado un modelo de selección de portlets, denominado PtSM. Este modelo puede ser utilizado para seleccionar el portlet más apropiado de entre un conjunto de portlets que tienen un conjunto parecido de funciones para las tareas y los objetivos específicos de los usuarios. PtSM utiliza un modelo de calidad (PtQM) que integra y adapta, al caso de los portlets, diferentes modelos propuestos para aplicaciones Web, componentes Software y productos Software. Además tiene en cuenta otras características que aunque no son de calidad están relacionadas con el

hecho de que un portlet es un producto comercial y por tanto, son relevantes a la hora de seleccionar un portlet u otro.

Como trabajo futuro se tendrán que validar las diferentes medidas que han sido definidas así como los umbrales que han sido establecidos. Además, está prevista la aplicación del modelo a diferentes portlets reales que tienen parecida funcionalidad, con el objetivo de determinar que portlet es más adecuado para su integración en un determinado portal.

Agradecimientos. Este trabajo es parte del proyecto CALIPO (TIC 2003-07804-C05-03) y la red CALIPSO (TIN2005-24055-E) financiados por el Ministerio de Educación y Ciencia y el proyecto DIMENSIONS (PBC-05-012-1) financiado por FEDER y la Junta de Comunidades de Castilla-La Mancha.

Referencias

1. Bellas, F., Standards for Second-Generation Portals. IEEE Internet Computing., 2004. 8(2). pp. 54-60.
2. Bertoa, M.F. and A. Vallecillo. Quality Attributes for COTS Components. 2002. 6th Internatioal Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2002). Málaga.pp. 54-66.
3. Botella, P., X. Burgués, J.P. Carvallo, X. Franch, J.A. Pastor, and C. Quer, Towards a Quality Model for the Selection of ERP Systems. Component-Based Software Quality, 2003. pp. 225-245.
4. Calero, C., J. Ruiz, and M. Piattini, Classifying web metrics using the web quality model. Online Information Review, 2005. 29(3). pp. 227-248.
5. Franch, X. and J.P. Carvallo, Using Quality Models in Software Package Selection. IEEE Software., 2003. 20(1). pp. 34-41.
6. García, F., M.F. Bertoa, C. Calero, A. Vallecillo, F. Ruíz, and M. Genero, Towards a consistent terminology for software measurement. Information and Software Technology, 2006. 48(8). pp. 631-644.
7. ISO/IEC, ISO/IEC 9126. Software Engineering-Product Quality. Partes 1 a 4. 2001. International Organization for Standardization/International Electrotechnical Commission.
8. Java Community Process. JSR 168 portlet specification. 2003.Disponible en: http://www.jcp.org/en/jsr/detail?id=168. Accedido: Enero.
9. Kalepu, S., S. Krishnaswamy, and S.W. Loke. Verity: A QoS Metric for Selecting Web Services and Providers. 2003. Fourth International Conference on Web InformationSystems Engineering Workshops (WISEW'03).pp. 131-139.
10. Mahdavi, M., J. Shepherd, and B. Benatallah. A Collaborative Approach for Caching Dynamic Data in Portal Applications. 2004. Proceedings of the fifteenth conference on Australian database Vol.27.pp. 181-188.
11. Moustakis, V., C. Litos, A. Dalivigas, and L. Tsironis. Website Quality Assessment Criteria. 2004. Ninth International Conference on Information Quality.pp. 59-73.
12. OASIS. Web Service for Remote Portals (WSRP). Version 1.0. 2003. Disponible en: http://www.oasis-open.org/commitees/wsrp/. Accedido: Mayo 2006.
13. Offutt, A.J., Quality attributes of web software applications. IEEE Software., 2002. 19(2). pp. 25-32.
14. Simão, R.P. and A. Belchior. Quality Characteristics for Software Components: Hierarchy and Quality Guides. 2003. Component-Based Software Quality.pp. 184-206.

Proyecto ANCLA: resumen de aportaciones, resultados y conclusiones

Grupo DSIE

1. Introducción

Contexto tecnológico de ANCLA: El problema.

El propósito inicial del proyecto ANCLA fue resolver la problemática ligada al desarrollo de software para los sistemas robóticos de teleoperación aprovechando en todo lo posible los últimos avances de la ingeniería del software. Para poder entender el alcance del proyecto y el enfoque propuesto como solución, es conveniente revisar el problema mencionado y las soluciones que ofrecía la Ingeniería del Software al comienzo del proyecto.

Como a continuación se detalla, la principal dificultad radica en la variabilidad inherente al dominio. Los sistemas robóticos teleoperados cubren un amplio abanico de mecanismos que llevan a cabo actividades de inspección y mantenimiento en entornos hostiles. Normalmente estos sistemas llevan a cabo un conjunto reducido de tareas altamente especializadas. Tal especialización conlleva:

- Una alta variabilidad tanto de funcionalidad como de características físicas.
- Distintas combinaciones de vehículos, manipuladores y herramientas.
- Una gran variedad de infraestructuras de ejecución que incluyen distintos procesadores, enlaces de comunicación e interfaces hombre-máquina.
- Una gran variedad de elementos de sensorización y actuación sobre el entorno.
- Diferentes tipos de algoritmos de control para llevar a cabo tareas reactivas muy simples o estrategias de navegación extremadamente complejas.
- Diferentes grados de autonomía, desde sistemas dirigidos por un operador hasta robots semi-autónomos.

A pesar de todas las diferencias expuestas, los sistemas teleoperados tienen gran cantidad requisitos en común en su definición y en consecuencia pueden compartir muchos componentes tanto lógicos como físicos en su implementación. Sería en consecuencia posible simplificar el desarrollo de estos sistemas robóticos teniendo en cuenta la definición de una arquitectura más o menos genérica que capturara la funcionalidad común entre todos los sistemas. La importancia de considerar tal arquitectura en la construcción de estos sistemas es un campo de mucho interés que ha sido muy estudiado en los últimos años [1]; sin embargo, no es posible conseguir una arquitectura que sea lo suficientemente flexible como para acomodar la variabilidad antes mencionada. Por lo tanto, un paso más es conseguir un marco

arquitectónico que sea a la vez flexible y extensible de cara a desarrollar sistemas con distintas arquitecturas, pero con muchas características comunes. En este sentido ha habido esfuerzos significativos con frameworks como en [2], [5] y [3]. Todos ellos realizan contribuciones muy destacadas que facilitan el desarrollo de sistemas. Sin embargo, a nuestro entender, la manera en la cual se adopta el enfoque de desarrollo orientado a componentes, hace que se pierdan gran parte de los beneficios esperados.

La arquitectura de un sistema involucra la descripción de elementos tanto hardware como software que interactúan entre sí de maneras muy diversas y complejas. Esta arquitectura incluye la descripción de dichos elementos, los patrones que guían su composición y las restricciones impuestas [4]. Teniendo en cuenta las diferencias entre los sistemas antes mencionados está claro que el principal objetivo de la arquitectura debe ser afrontar dicha heterogeneidad. Para conseguirlo, debe tenerse en cuenta como prioritarios los siguientes aspectos:

- El framework no debería imponer una arquitectura concreta, sino permitir la definición de varias por ejemplo permitiendo distinguir entre sistemas reactivos (con inteligencia autónoma) y deliberativos (dirigidos por el operador).
- Debería ser posible reutilizar componentes en sistemas con distintas arquitecturas. Esto implica que debe existir una distinción clara entre los componentes y sus patrones de interacción.
- Debería ser posible especificar y verificar el comportamiento temporal de los componentes dado que las restricciones de tiempo real son cruciales en este tipo de sistemas.
- La implementación de los componentes debe poder ser software o hardware.
- Debería ser posible el uso de componentes comerciales COTS.
- Debería ser posible integrar inteligencia o interoperar con sistemas que la posean.

Estos requisitos representan las características de flexibilidad y extensibilidad que necesitan tener todo framework que se precie de serlo. Puesto que el principal problema del proyecto es resolver la variabilidad en los sistemas del dominio, las soluciones tienen que venir de aquellos enfoques que permitan parametrizar esta variabilidad. En respuesta a estas demandas, en los últimos años han surgido nuevos paradigmas de desarrollo de software, promovidos no sólo desde el ámbito científico-académico, sino también fuertemente respaldados por muchas de las empresas líderes del sector informático (Sun Microsystems, Microsoft, etc). Entre estas nuevas propuestas cabe mencionar las siguientes:

- Desarrollo de Software Basado en Componentes (CBSD)
- Líneas de Productos Software (SPL)
- Generación Automática de Software (GP)
- Programación Visual (VP)
- Arquitecturas Dirigidas por Modelos (MDA)
- Desarrollo de Software Orientado a Aspectos (AOSD)
- etc.

A pesar de la revolución que estos nuevos paradigmas han supuesto para la Ingeniería del Software, la falta de consenso en cuanto a las metodologías y notaciones por emplear, así como la ausencia de herramientas que den un soporte integral al desarrollo de aplicaciones siguiendo una o más de estas nuevas propuestas, hace que algunas de ellas no resulten tan efectivas en la práctica como cabría esperar (por ejemplo, CSBD y AOSD) y otras carezcan del soporte adecuado para su consideración (por ejemplo, MDA y SPL).. Otros subproyectos del proyecto coordinado DYNAMICA han tratado de superar alguno de estos problemas para hacer más aplicables estos paradigmas. En las secciones que siguen abordaremos la aportación de cada uno de ellos a la consecución de objetivos por parte del subproyecto ANCLA y las posibilidades de colaboración a medio y largo plazo en los nuevos proyectos emplazados.

Objetivos iniciales del proyecto

Con estos precedentes los objetivos del proyecto se plantearon de la siguiente manera. Desde el punto de vista del dominio identificado, se trató de definir un marco de desarrollo de aplicaciones de teleoperación que tuviera en cuenta aspectos comunes entre sistemas, que permitiera parametrizar las diferencias, reutilizando los artefactos software generados a todos los niveles. Desde un punto de vista ingenieril, el dominio se ofrecía como un campo de estudio adecuado para la implantación de los mecanismos de la Ing. del Sw, ya que engloba una problemática amplia que: se ubica en entornos industriales, tiene en cuenta un gran abanico de necesidades, considera la interacción con el entorno, necesita de la integración de soluciones mixtas Sw/Hw, y por todo ello, se sale de los sistemas habituales de gestión (nuevos requisitos). Más específicamente, estos objetivos se concretaron en los siguientes apartados:

1. Identificación de aquellos *aspectos* tanto funcionales como no funcionales del sistema de teleoperación que puedan ser susceptibles de variación y por tanto impliquen una modificación en el modelo arquitectónico empleado.
2. Obtención de una especificación formal de dicha variabilidad (tanto la variabilidad propia de los requisitos funcionales como las diferentes necesidades de interacción entre componentes). El objetivo era hacer uso del lenguaje de definición de componentes de PRISMA y de su lenguaje de configuración para definir instancias y especificar la topología del sistema de teleoperación.
3. Rediseño arquitectónico del software para sistemas de teleoperación dedicados al mantenimiento de cascos de buques.
4. Implementación del software en un prototipo para pruebas que incluyera la dinámica de control requerida y desarrollada en el subproyecto CARATE.

Para la consecución de dichos objetivos se definieron las siguientes tareas:

1. Definición del marco de investigación.
2. Identificación de los nuevos requisitos comunes a los sistemas de teleoperación:
    a. Identificación de aspectos dinámicos internos a los sistemas teleoperados.
    b. Identificación de aspectos dinámicos entre sistemas teleoperados.

3. Representación formal mediante el enfoque PRISMA
4. Obtención del modelo arquitectónico del sistema de teleoperación
5. Aplicación de los métodos de evaluación
6. Desarrollo de un prototipo y pruebas
7. Evaluación de resultados

En la sección que sigue se presenta el detalle relativo a la realización y resultados de cada una de estas tareas.

2. Avances, pasos y aportaciones

Los objetivos del proyecto de investigación mencionados en el apartado anterior dieron lugar a una serie de actividades encaminadas a conseguirlos. Además, dependiendo del campo de estudio considerado, estas actividades se definieron y realizaron en colaboración con uno o más de los nodos participantes en el proyecto coordinado DYNAMICA.

El objetivo por tanto de esta sección es detallar las actividades realizadas en ANCLA, su relación con los objetivos iniciales, los resultados obtenidos y otras consideraciones relativas al beneficio generado.

Relativo a la definición del marco de investigación (actividad 1)

Se trataba de obtener un conocimiento del sistema real en el que se iba a aplicar el proyecto. El propósito de esta actividad era conocer las características básicas de los sistemas de teleoperación dedicados a tareas de limpieza de cascos de buques, objetivo del proyecto europeo EFTCoR. El objetivo de esta actividad era pues que los distintos participantes del presente proyecto conocieran dicho entorno de trabajo y la problemática asociada al mismo. La realización de esta actividad se materializó en los siguientes puntos:

1. Una visita a los astilleros de Izar Cartagena en enero de 2004 en la que se explicó la problemática existente en el dominio de sistemas robóticos teleoperados y se presentó el entorno de trabajo en el que debía funcionar el sistema (Fernández et al. 2005).
2. Coincidiendo con la visita anterior, la primera reunión de coordinación del proyecto se realizó en Cartagena donde por parte de ANCLA se presentaron al resto de grupos de investigación trabajos relativos al estado del arte en el desarrollo de arquitecturas software para sistemas robóticos.
3. Se elaboró un informe técnico recogiendo con detalle el conjunto de requisitos funcionales de estos sistemas (Pastor et al. 2004b).

Relativo a la Identificación de los nuevos requisitos comunes a los sistemas de teleoperación (actividad 2)

Se trataba de analizar y especificar mediante casos de uso los requisitos funcionales y no funcionales de los sistemas de teleoperación con la finalidad de identificar qué partes de dichos sistemas están sujetos a cambios estructurales o en cuanto a su dinámica de interacción con otros sistemas. Desde este punto de vista era útil contemplar las necesidades de adaptación de los sistemas de teleoperación a nuevos requisitos o condicionantes de entorno que varíen tanto la dinámica de cada sistema como los patrones de coordinación entre todos los robots de la familia. Para la realización de esta actividad, se consideró la colaboración con los grupos de investigación de la UPV, la UMU y la UC3.

La posibilidad de aprovechar el mismo sistema teleoperado para las distintas operaciones que se pueden llevar a cabo y la utilización o no de mecanismos adicionales de ajuste de la calidad del proceso representan ya por sí mismos un ejemplo de variabilidad en el comportamiento del software cuya dinámica y especificación de la misma fue objetivo de esta tarea. La posibilidad de integrar en un único sistema toda esta dinámica no está contemplado en el proyecto europeo EFTCoR en el que está previsto construir distintos sistemas para distintos requisitos.

Además, la posibilidad de añadir o suprimir robots específicos para el mantenimiento y el nivel de conocimiento requerido por el resto de robots en ejecución implicaba un grado de coordinación que variaba inevitablemente con el tiempo. Fue por tanto un propósito el precisar la dinámica del software para poder dar cobertura en la mayor medida posible a estos requisitos de cooperación entre sistemas. La realización de esta actividad se materializó en los siguientes puntos:

1. Se elaboró un informe técnico recogiendo con detalle los puntos de variabilidad analizados a nivel arquitectónico en el desarrollo de sistemas teleoperados (Pastor et al. 2005) (Álvarez et al. 2004). En dicho trabajo se identificaron y clasificaron las fuentes de variabilidad y su impacto en el nivel correspondiente del framework ACROSET (Álvarez et al. 2006). Con esto lo que se persiguió fue:
    a. Describir el espectro de posibles variaciones que pueden tener que implantarse en todos los productos de la línea.
    b. Discutir si deben ser tratadas de forma estática o dinámica (es decir, si se correspondían con diferentes productos, la evolución de un producto o a cambios en tiempo de ejecución).
    c. *Valorar* cómo el enfoque PRISMA podía ayudar a describir y a resolver estas variabilidades.
2. Se expusieron en la Universidad Politécnica Valencia en el seno del subproyecto PRISMA (enero de 2005) los resultados obtenidos del punto anterior y se contrastaron las posibilidades relativas al uso de PRISMA para solventar las necesidades identificadas.
3. Se elaboró un informe técnico recogiendo requisitos que tenían que ver con la seguridad (*safety*) de sistemas teleoperados (Ortiz et al. 2006) (Álvarez et al. 2004). Este conjunto de requisitos supuso para el nodo de la Universidad de Murcia un caso de estudio muy rico para la aplicación de su metodología para la caracterización de requisitos comunes entre sistemas y para la Universidad

Politécnica de Valencia un marco adecuado para sistematizar la especificación de requisitos (Navarro et al. 2005) (Navarro et al. 2006) (Navarro et al. 2004).

Relativo a representación formal mediante el enfoque PRISMA, obtención del modelo arquitectónico del sistema de teleoperación y realización de pruebas de validación (actividades 3, 4 y 5)

El objetivo que se persiguió fue hacer uso del lenguaje de definición de componentes y configuración PRISMA para la representación de los requisitos identificados en las actividades precedentes relativos a la variabilidad entre sistemas en todo lo que tenía que ver con distribución, coordinación y evolución de aspectos. La especificación de los requisitos dinámicos mediante un lenguaje formal no sólo exigió un alto grado de conocimiento de los mismos sino que también permitió hacer uso de las herramientas desarrolladas alrededor de dicho formalismo de cara a la validación y verificación de la especificación resultante. En este sentido, esta actividad permitió descubrir incompatibilidades o conflictos entre los requisitos anteriores así como la realización de pruebas que permitieran deducir el grado de cumplimiento de los mismos. La validación mediante animación a partir de prototipos generados de manera semiautomática se consideró desde el principio como una herramienta clave en esta actividad aunque por motivos de disponibilidad no fue posible su consecución. La realización de esta actividad se materializó en los siguientes puntos:

1. Un objetivo inicial fue poder probar los resultados anteriores en el entorno real escogido (el robot escalador Lázaro) y usando un lenguaje de programación de alto nivel como Ada lo que dio como resultado la publicación (Alonso et al. 2006).
2. Definición de un marco de desarrollo basado en componentes y un framework arquitectónico para el diseño de sistemas robóticos teleoperados (Álvarez et al. 2006) (Vicente et al. 2004) (Pastor et al. 2004) (Vicente et al. 2006) (Vicente et al. 2004b).
3. Elaboración de un informe que recogiera la instancia ACROSET de la arquitectura para el caso del robot escalador Lázaro y una primera versión de conjunto de elementos identificados en PRISMA (Sánchez et al. 2005b).
4. Realización de una especificación en PRISMA del robot didáctico 4U4 (Pérez et al. 2003) (Pérez et al. 2004) que puso de manifiesto la viabilidad del enfoque desde distintos puntos de vista y se realizaron grabaciones de su funcionamiento por parte de la UPV.

Relativo al desarrollo de un prototipo y pruebas (actividad 6)

El tipo de pruebas que se pretendían llevar a cabo contemplaban todos aquellos aspectos dinámicos generados en las tareas precedentes. Para ello se planificaron cambios en las herramientas del robot, en cuestiones relativas a la supervisión y seguimiento de la calidad, coexistencia con otro robot, etc. No se trataba de probar la funcionalidad del sistema sino la capacidad del modelo arquitectónico para adaptarse

a los cambios de requisitos entre los sistemas del dominio. Inicialmente se pensó en la realización de pruebas mediante un el prototipo desarrollado por el grupo de investigación de UC3. En los apartados que siguen se comenta las dificultades que surgieron con dicho intento. Una vez realizadas la pruebas y aprovechando la infraestructura tecnológica hardware y software proporcionada en el seno del proyecto europeo EFTCoR se trasladaron las pruebas al sistema real del robot escalador Lázaro (Ortiz et al. 2005). Inicialmente se pensó en la realización de las pruebas considerando el robot tipo Grúa Ártabro pero no fue viable y se optó por el escalador Lázaro por las condiciones del entorno de pruebas y las pocas pruebas que se habían hecho de la arquitectura para este sistema que había crecido mucho en complejidad por las interacciones con otros sistemas con los que había sido conectado (subsistema externo de visión artificial y subsistema de monitorización de datos en tiempo real).

Relativo a la evaluación de los resultados obtenidos

El objetivo planteado (fundamental dentro del proyecto) fue analizar la conveniencia de las técnicas empleadas en lo que al desarrollo de familias de robots para teleoperación de operaciones de limpieza de cascos de buque se refiere. Para ello se determinaron los beneficios obtenidos por la aplicación de métodos formales en la construcción de este tipo de sistemas. Como resultado de dicha evaluación se tomó en consideración la no construcción de un único sistema muy flexible y con la capacidad de adaptarse a necesidades cambiantes (dentro de un límite razonable) sino que la solución se decantó por distintas soluciones para distintos requisitos dado que la complejidad asociada a la incorporación de dicha dinámica aparecía como una desventaja mayor al beneficio que proporcionaba. En este sentido, y tal y como se apunta en la última sección de este trabajo, se planteó la necesidad de ir un marco de gestión de modelos, lenguajes específicos de dominio y mecanismos de compilación automática a distintas plataformas de ejecución que recogiera de los aspectos comunes y diferenciadores que un enfoque de líneas de producto puede ofrecer.

3. Lecciones aprendidas

El objetivo de esta sección es presentar un resumen de conclusiones extraídas de los tres años de trabajo en el proyecto ANCLA. Hablar de resultados conlleva desde una perspectiva amplia considerar desde distintos puntos de vista qué aspectos son destacables y de alguna forma pueden contribuir a mejorar la realización de futuros trabajos. Esta perspectiva puede en primera instancia enfocarse desde dos ópticas distintas: (1) desde un punto de vista del proceso mismo considerando las interacciones con otros grupos y con la comunidad científica a modo de publicaciones en revistas y congresos internacionales, y (2) desde un punto de vista científico en cuanto a la aplicabilidad real de los resultados obtenidos y las dificultades encontradas desde un punto de vista metodológico y tecnológico. Los resultados que se recogen son cuestionables desde distintos contextos y como tal están sujetos a

crítica, pero consideramos que en cualquier caso suponen una reflexión importante sobre el desarrollo del proyecto.

3.1 Posibilidades de difusión de resultados.

En los últimos tres años se han publicado varios trabajos sobre los resultados de las investigaciones relacionadas con la validación de los métodos estudiados en DYNAMICA. En concreto, se han publicado las experiencias en el desarrollo de un marco arquitectónico basado en componentes abstractos (ACROSET) y su aplicación para el desarrollo de unidades de control para robots de servicio teleoperados (EFTCoR).

En el ámbito de las publicaciones específicamente orientadas a la robótica, se han publicado varios trabajos en revistas y congresos internacionales. De entre ellas, cabe destacar por su especial relevancia las siguientes:

- Arquitectura de Control para robot de servicios teleoperados. Revista Iberoamericana de Automática e Informática Industrial (RIAI) April 2006.
- An Architectural Framework for Modeling Teleoperated Service Robots. Robótica, May 2006.

De estas dos, la segunda es especialmente importante dado el índice de impacto de la publicación. A pesar de estos logros, en el último año se ha rechazado en varias revistas con índice de impacto trabajos que describían ACROSET y su posibilidad de aplicación en los robots de la familia de productos EFTCoR. Esto ha ocurrido a pesar de recoger las recomendaciones de los revisores en cada rechazo. No se ha conseguido "convencer" a los revisores sobre la conveniencia de las aproximaciones seguidas. A modo ilustrativo se citan en la Tabla 1 algunas de las publicaciones rechazadas.

IEEE-SE	IEEE Trans. Software Engineering	An architectural framework to manage variability in a service robot product line
IJ-ARS	Int. Journal Advanced Robotic Systems	ACROSET: A Component based Architectural Framework for Teleoperated Service Robots
JSS	Journal of Systems and Software	Experiences on the Definition and Implementation of a Component-Oriented Architectural Framework for Teleoperated Robots

Tabla 1. Revistas listadas en ISI-JCR donde se han RECHAZADO trabajos

En cuanto a los congresos, se han publicado trabajos en cuatro foros de reconocido prestigio, dos relacionados con el control y la robótica (IFAC e ICINCO) y otros dos relacionados con la programación en Ada95 (Ada-Europe). Sin embargo, algunos de los trabajos han sido rechazados en otros dos congresos internacionales (ICRA y EUROS) y en uno nacional (JISBD). En los dos primeros se puede justificar el rechazo por la cantidad de artículos recibidos, aunque ha de reconocerse que los comentarios de los revisores han estado en la misma línea que los de las revistas. Llama la atención que tanto en ICRA como en JISBD y en la revista IEEE-SE, se han

presentado trabajos relacionados con la variabilidad en líneas de producto software y en ningún caso la revisión ha resultado favorable, incluyéndose comentarios negativos en cuanto a la manera de abordar esta materia. Es decir, según la opinión de los revisores, no se ha sabido transmitir uno de los principales objetivos de ANCLA: la obtención de una especificación formal de la variabilidad en los sistemas de teleoperación. Los comentarios de los revisores relativos a los trabajos rechazados se pueden resumir en la siguiente tabla:

Comentarios	**IEEE-SE**	**IJ-ARS**	**JSS**	**ICRA**	**JISBD**	**#**
Faltan comparativas con el estado del arte. No se especifican claramente las mejoras desde un punto de vista del dominio.			X	X		2
Idem desde el punto de vista de la ingeniería del software.			X	X		2
Falta describir más experiencias exitosas en la reutilización de la arquitectura.	X	X	X	X	X	5
¡Yet another software modular architecture for robots!				X		1
La arquitectura parece demasiado compleja o demasiado abstracta.		X	X			2
No se describen suficientes implementaciones de componentes concretos.	X					1
Se echan en falta, estadísticas, pruebas, resultados, comparativas, etc.	X		X			2
Críticas parciales al tratamiento de la variabilidad .	X				X	2

Tabla 2. Resumen de los comentarios de los revisores.

La principal conclusión que se puede extraer de los comentarios de los revisores es que el carácter fronterizo de los trabajos ha perjudicado al intentar publicarlos, por varios motivos. Respecto de los foros próximos a la Ingeniería del Software explicando la aplicación de técnicas de Ingeniería del Software al desarrollo de robots, se ha criticado la falta de aportaciones a dicha Ingeniería. Los revisores preferían aportaciones de esta índole frente a la descripción de uso de técnicas ya conocidas en los desarrollos de los robots. Desde nuestro punto de vista, se ha echado en falta en los últimos años casos de estudio industriales en los que se demostrara la viabilidad de las últimas técnicas y metodologías desarrolladas, especialmente CDB, Líneas de Producto, etc. Por ello se consideró por parte de los autores una aportación

significativa la descripción de experiencias reales fuera del dominio de la gestión. Sin embargo, queda patente la necesidad de adoptar un nuevo enfoque de presentación de resultados, probablemente incluyendo datos empíricos que pongan de manifiesto la ventajas de la aproximación seguida.

Respecto de los foros próximos a la Robótica es importante reseñar que apenas han valorado los conceptos de Ingeniería del Software frente a lo que cabría esperar. Entendemos que esta infravaloración viene dada por la falta de datos y de herramientas que permitan extraer resultados que justifiquen los beneficios de la arquitectura, es decir, la conveniencia de la Ingeniería del Software en el desarrollo de sistemas para el dominio de la Robótica. Ha perjudicado mucho la juventud de ACROSET frente a otros *frameworks* de componentes que ya disponen de soluciones ejecutables de implementación. En general, se ha detectado una falta de conciencia de desarrollo metodológico desde los requisitos a la arquitectura e implementación. Les interesan más los datos empíricos que demuestren los beneficios de la propuesta que su justificación teórica o metodológica.

En consecuencia, interpretamos que cuando se acuda a un foro de ingeniería del software con una aplicación real de conceptos no basta con describir el trabajo, sino que es necesario realizar el esfuerzo de realimentar a la propia ingeniería a partir de las experiencias extraídas en el desarrollo de dicha aplicación. Esta realimentación puede venir de las necesidades detectadas y no cubiertas por el actual estado de la técnica (herramientas, lenguajes más adecuados, notaciones y modelado necesarios, etc.). Por otro lado, es necesario demostrar el beneficio obtenido de una forma más cuantitativa (mejoras en el tiempo de desarrollo, en la gestión de los recursos, en la reutilización del código, etc.) para lo cual es posible que haya que esperar a tener desarrollos más concluyentes antes de publicar los resultados. Este último punto es, como se ha visto, especialmente importante cuando se acuda a foros interesados en los resultados finales en forma de sistema (publicaciones sobre robótica).

3.2 Interacción con otros grupos

El hecho de que los otros nodos del proyecto coordinado DYNAMICA trabajaran para dar soporte al empleo de nuevos paradigmas de desarrollo de software ha dado lugar a numerosas aportaciones en la consecución de los objetivos planteados en el proyecto ANCLA. Para ello se han llevado a cabo distintas actividades de colaboración obteniéndose unos beneficios más que aceptables y permitiendo la validación de muchos trabajos realizados. En el apartado anterior se mencionan las principales actividades de colaboración con la UPV, la UMU y la UC3M. En las fases iniciales del proyecto dichas actividades fueron fundamentalmente relativas a un intercambio de conocimiento ontológico, siendo en fases más avanzadas donde se ha demostrado que es factible la coordinación entre nodos, obteniéndose resultados que muestran avances significativos respecto de los objetivos planteados.

Por otro lado, cabe destacar el interés por parte del resto de nodos en la problemática planteada por el proyecto ANCLA. En este sentido, podría concluirse que los beneficios han sido mutuos: ANCLA ha podido hacer uso de nuevas herramientas, metodologías y notaciones para alcanzar los objetivos planteados, y los otros nodos han dispuesto de un caso de estudio que resultó ser un marco adecuado para la aplicación y validación de las mismas.

Por tanto, a la vista de los resultados obtenidos del proyecto coordinado DYNAMICA y en relación a la interacción con otros grupos se puede decir que el trabajo ha sido satisfactorio y enriquecedor, y si bien pudiera pensarse en un principio que la coordinación podría estar limitada por la no presencia física de los investigadores en un mismo centro, la comunicación ha sido fluida y la colaboración estrecha como se deduce de numerosas publicaciones (ver divulgación de resultados).

Por último, resaltar que el trabajo ha permitido, y quizás sea lo más positivo, abrir nuevos horizontes de colaboración para la resolución de problemas aún sin resolver con las técnicas empleadas, pero con grandes expectativas ante el empleo de nuevos enfoques que actualmente aporta la Ingeniería del Software.

3.3 Aplicabilidad real

En el marco del proyecto ANCLA la posibilidad de construir una versión del sistema de control de robots tele-operados utilizando una especificación formal en el lenguaje PRISMA siempre ha tenido gran interés. Los beneficios que ofrece este planteamientos son: demostrar propiedades; tener en cuenta la variabilidad inherente a este tipo de sistemas de manera que quedara reflejada en el diseño del mismo usando un enfoque aspectual; generar código para los componentes del sistema utilizando alguno de los trasformadores a código ejecutable a partir de las herramientas provistas por la UPV; reconfigurar dinámicamente los sistemas a partir de la variabilidad capturada en la especificación y la capacidad reflexiva de PRISMA, etc.

Teniendo en cuenta todos estos beneficios, se realizaron por parte de ANCLA y de la UPV esfuerzos considerables para conseguir dicha especificación en el lenguaje PRISMA. Buena cuenta de ello fue el caso demostrativo desarrollado para un robot didáctico, especificado completamente en PRISMA y traducido con las herramientas desarrolladas por la UPV a C# de .NET (Pérez et al. 2004). A nuestro juicio este trabajo demuestra la viabilidad del enfoque planteado en el proyecto. El siguiente paso fue entonces trasladar los resultados obtenidos al entorno real de los dispositivos robóticos de la familia EFTCoR. No obstante, aquí aparecen nuevas dificultades impuestas por la infraestructura de desarrollo utilizada (el entorno STEP-7 de SIEMENS) y por las características especialmente exigentes de este tipo de sistemas (rendimiento y fiabilidad). Estas dificultades, aunque no hacen imposible los objetivos, sí suponen una carga de trabajo difícilmente abordable en el marco del proyecto ANCLA (no dotado de los recursos humanos necesarios: becarios, personal contratado, etc.). Otra dificultad añadida a lo anterior es el empleo muy arraigado en la industria de técnicas de muy bajo nivel para el diseño y programación de estos sistemas, y por tanto la resistencia a la utilización de nuevos métodos y técnicas. Si bien hasta la fecha este planteamiento ha venido dando buenos resultados, creemos que es insostenible a medio plazo por la complejidad de las nuevas aplicaciones demandadas por la industria. En este contexto, hay una oportunidad muy clara que ya está dando buenos resultados en dominios específicos y que pensamos puede trasladarse a la robótica de servicios: adoptar un enfoque dirigido por modelos que integre lenguajes específicos del dominio además de los mecanismos de transformación semi-automática a las infraestructuras actualmente utilizadas. Es en esta línea incipiente donde se van a concentrar gran parte de los esfuerzos

planificados para el proyecto CICYT MEDWSA, que cuenta también con la colaboración de gran parte de los nodos de DYNAMICA.

3.4 Complejidad tecnológica y metodológica

Como se ha comentado en las secciones anteriores, dentro del proyecto ANCLA se ha planteado desde el principio adoptar PRISMA como el enfoque en el que capturar las especificaciones de sistemas robóticos, pensando en una orientación pura a componentes y en una separación clara de aspectos que permitan capturar entre otras facetas la variabilidad presente en estos sistemas. Por los resultados obtenidos, desde nuestra óptica el enfoque PRISMA podría enriquecerse si se contemplaran las siguientes posibilidades:

1. Definición de un conjunto mínimo PRISMA como núcleo básico con el que poder trabajar. Este núcleo mínimo debería recoger las posibilidades necesarias para poder especificar cualquier aplicación con el lenguaje y permitiría a los usuarios del lenguaje familiarizarse más rápidamente con él además de posibilitar la compilación a más entornos de ejecución distintos.
2. Establecimiento de un marco metodológico que recoja los pasos que se deberían seguir para llegar a la especificación a partir de un conjunto de requisitos concreto. Estos pasos podrían venir acompañados de heurísticos y reglas que guiaran al diseñador del sistema en el particionamiento en componentes, puertos, conectores y aspectos.
3. Definir patrones arquitectónicos genéricos habituales en el desarrollo de sistemas y ofrecer de manera automática su representación en PRISMA aunque fuera a nivel esquemático para su posterior compleción. Por ejemplo, el patrón observador definido por Erich Gamma está presente en muchas soluciones en el ámbito de sistemas reactivos que integran sensores y actuadores. Poder considerar a nivel arquitectónico dicha solución de diseño y partir de un esqueleto en PRISMA para el mismo podría beneficiar de manera considerable el uso del enfoque.
4. Crear documentos con hipervínculos relativos a la sintaxis y significado de cada uno de los bloques de definición del lenguaje PRISMA. Al menos la definición de una BNF sería adecuada para poder conocer exactamente las posibilidades sintácticas disponibles.

4. Conclusiones y prospección de futuro

La ejecución del proyecto ANCLA en el marco del proyecto coordinado DYNAMICA ha resultado claramente enriquecedor tanto desde el punto de vista de las interacciones entre grupos como desde el aspecto formativo resultante. La consideración de un caso de estudio como ha sido la familia de sistemas robóticos teleoperados EFTCoR ha contribuido con datos reales a todos los niveles en el desarrollo de sistemas: requisitos, diseño arquitectónico, diseño detallado de componentes y realización final en la infraestructura de ejecución. De los resultados

obtenidos y reflejados en las secciones anteriores, queda patente la necesidad de considerar un enfoque orientado a componentes pero, a su vez, ha venido a demostrar las deficiencias existentes al usar este único paradigma. Todos los problemas y dificultades encontradas en la realización del proyecto parecen confluir en la necesidad de un enfoque en el que se suba el nivel de abstracción, pasándose de la perspectiva de que "todo es un objeto o un componente" a "todo es un modelo". Desde este enfoque, surge la necesidad de adoptar distintas soluciones que tienen que ver tanto con la definición de nuevos lenguajes específicos del dominio considerado, con el desarrollo de traductores automáticos a la infraestructua de ejecución, con la obtención de mecanismos rigurosos de verificación de propiedades y validación en las etapas tempranas de modelado, con el uso de metamodelos para la caracterización precisa de los modelos desarrollados, con la definición de métricas para poder cuantificar la conveniencia de los modelos obtenidos, con el desarrollo de herramientas que den soporte al proceso, etc.

Consideramos además muy importante no limitar las posibilidades de obtención de resultados únicamente a los sistemas robóticos sino que abrimos la puerta al estudio de otras familias de sistemas reactivos con los que el grupo DSIE de la UPCT ya ha venido teniendo experiencia en los últimos años. A saber, sistemas domóticos, sistemas de visión y sistemas basados en redes de sensores inalámbricas. Un estudio detallado de cada uno de estos dominios usando un enfoque de líneas de producto junto con los propósitos reseñados en el párrafo anterior van a constituir el propósito fundamental planteado en el proyecto MEDWSA como continuación del proyecto ANCLA en la UPCT y por parte del DSIE.

5. Difusión de resultados

Esta sección recoge los trabajos publicados que tienen que ver con aspectos relevantes del proyecto y que han sido citados en las secciones anteriores.

(Alonso et al. 2006) Diego Alonso, Pedro Sánchez, Bárbara Álvarez, Juan A. Pastor, A systematic approach to developing safe teleoperated robots. Lecture Notes in Computer Science, Reliable sw Techologies Ada Europe 2006. Vol. 4006, pp. 119-130, June 2006. Springer Verlag. Porto, Portugal.

(Alvarez et al. 2004) B. Alvarez, P. Sánchez, J.A. Pastor, Experiencias, Estrategias y Retos en la incorporación de requisitos de seguridad en el sistema EFTCoR. pág. 523-530, IX Jornadas Nacionales de Ingeniería del Software y Bases de Datos, ISBN 84-688-8983-0, Málaga, Noviembre 2004.

(Alvarez et al. 2004b) Bárbara Álvarez, Juan A. Pastor, Pedro Sánchez, Francisco Ortiz, An architectural framework to manage the variability in a service robots product line, II Jornadas DYNAMICA, 2004.

(Alvarez et al. 2006) Bárbara Álvarez, Francisco Ortiz, Juan A. Pastor, Pedro Sánchez, F. Losilla, N. Ortega, Arquitectura de Control para robot de servicios teleoperados. Revista Iberoamericana de Automática e Informática Industrial (RIAI). ISSN: 1697-7912, Abril 2006. Vol. 3(2), pp. 79-89.

(Alvarez et al. 2006b) Bárbara Álvarez, Pedro Sánchez, Juan A. Pastor, Francisco Ortiz, An Architectural Framework for Modeling Teleoperated Service Robots, ISSN: 0263-5747, Robotica Journal, Vol.4(4), pp. 411-418, Cambridge University Press, July 2006.

(Fernandez et al. 2005) C. Fernández, J.A. Pastor, P. Sánchez, B. Álvarez, A. Iborra, Ship Shape in Europe: Co-operative Robots in the Ship Repair Industry, Robotics and Automation Magazine (RAM), Special issue on Industrial Robotics Applications & Industry-Academia Cooperation in Europe. New Trends and Perspectives, Vol. 12(2), pp.65-77, september 2005, IEEE Robotics And Automation Society, USA.

(Navarro et al. 2004) Elena Navarro, Patricio Letelier, Isidro Ramos, Bárbara Álvarez, Especificación de requisitos software basada en características de calidad, separación de concerns y orientación a objetivos, IX Jornadas en Ingeniería del Software y Bases de Datos (JISBD'04).

(Navarro et al. 2005) Elena Navarro, Pedro Sánchez, Patricio Letelier, Juan A. Pastor, Isidro Ramos, Sistematizando la Especificación de Requisitos Safety: un Caso de Estudio sobre aplicaciones teleoperadas, IV Jornadas de enseñanza vía Internet/Web de la Ingeniería de Sistemas y Automática (EIWISA 2005), ISBN: 84-9732-451-X, Granada, España.

(Navarro et al. 2006) E. Navarro, P. Sánchez,P. Letelier, Juan A. Pastor, I. Ramos, A Goal-Oriented approach for Safety Requirement Specifications, Proc. of the 13th Annual IEEE International Symposium on Engineering of Computer based systems, Postdam, Germany, pp.319-326, IEEE Computer Society, USA 2006.

(Ortiz et al. 2005) Francisco Ortiz, Bárbara Álvarez, Fernando Losilla, David Rodriguez, Noelia Ortega. An implementation of a teleoperated robot control architecture on a PLC and Field-Bus based platform reference, XVI IFAC World Congress, Praga 2005.

(Ortiz et al. 2006) Ortiz F, Sánchez P, Alonso D, Álvarez B, Pastor JA, Iborra A, Aplicación de Estándares y Procesos de Seguridad en el Desarrollo de Robots de Servicio Teleoperados. XVII Jornadas de Automática. Almería, España, 6-9 febrero 2006, pag. 771-779, 2006.

(Pastor et al. 2004) J.A. Pastor, B. Alvarez, P. Sánchez, F. Ortiz, A Layered Architectural Component Model for Service Teleoperated Robots, pág. 435-446, IX Jornadas Nacionales de Ingeniería del Software y Bases de Datos, ISBN 84-688-8983-0, Málaga, Noviembre 2004.

(Pastor et al. 2004b) Juan A. Pastor, Pedro Sánchez, Bárbara Álvarez, Requisitos funcionales de los sistemas robóticos teleoperados. Informe técnico Proyecto CICYT ANCLA, Ref. ANCLA-02-2004., 2004.

(Pastor et al. 2005) Juan A. Pastor, Pedro Sánchez, Bárbara Álvarez, Francisco Ortiz, Escenarios de Variabilidad de los Sistemas Robóticos Teleoperados. Informe técnico Proyecto CICYT ANCLA, Ref. ANCLA-08-2005. Enero de 2005.

(Pérez et al. 2003) J. Pérez, N. Hussein, I. Ramos, J.A. Pastor, Pedro Sánchez, B. Alvarez, Desarrollo de un Sistema de Teleoperación utilizando el enfoque PRISMA. VIII Jornadas de Ingeniería del Software y Bases de Datos, páginas 411-420, ISBN 84-688-3836-5, Alicante, Noviembre 2003.

(Pérez et al. 2004) Pérez J., Cabedo R., Sánchez P., Carsí J.A., Ángel J., Ramos I., Álvarez B. Arquitectura PRISMA para el Caso de Estudio: Brazo Robot Actas del II workshop DYNAMICA – DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures, pags. 119-127, junto a Jornadas de Ingeniería del Software y Bases de Datos (JISBD), Málaga, noviembre 2004.

(Sánchez et al. 2005) Pedro Sánchez, Juan A. Pastor, Bárbara Álvarez, Francisco Ortiz, Requisitos Safety para el Sistema EFTCoR, Informe técnico Proyecto CICYT ANCLA, Ref. ANCLA-06-2005, 2005.

(Sánchez et al. 2005b) Pedro Sánchez, Francisco Ortiz, Juan A. Pastor, Bárbara Álvarez, Arquitectura de Componentes para el robot escalador Lázaro y su representación en el lenguaje PRISMA, Informe técnico Proyecto CICYT ANCLA, Ref. ANCLA-10-2005, 2005.

(Vicente et al. 2004) Cristina Vicente, Carlos Fernández y Pedro Sánchez, Automated Visual Inspection Systems Development from a Generic Architectural Pattern Description, Novática, Volumen 171, pp.63-65, Octubre 2004.

(Vicente et al. 2004b) Cristina Vicente, Carlos Fernández, Pedro Sánchez, Sistemas de inspección Visual Automatizada: de la arquitectura sw genérica a la generación de prototipos ejecutables. pag. 515-522. IX Jornadas Nacionales de Ingeniería del Software y Bases de Datos, ISBN 84-688-8983-0, Málaga, Noviembre 2004.

(Vicente et al. 2006) Cristina Vicente, Ana Toledo, Carlos Fernández, Pedro Sánchez, Generación Automática de Aplicaciones Mixtas Sw/Hw mediante la Integración de Componentes COTS. IEEE América Latina, pp. 25-31, vol.4(2), Abril, 2006.

6. Referencias

1 Coste-Manière, E. & Simmons, R., 2000. Architecture, the Backbone of Robotic System, Proc. of the 2000 IEEE International Conference on Robotics & Automation, pp. 505-513, April 2000, San Francisco, USA.

2 Bruyninckx, H., Konincks, B. & Soetens, P.,2002. A Software Framework for Advanced Motion Control, Dpt. of Mechanical Engineering, K.U. Leuven. OROCOS project inside EURON. Belgium.

3 Scholl, K.U. Alb iez, J. & Gassmann, B.,2001. MCA: An Expandable Modular Controller Architecture, Karlsruhe University, 3rd Real-Time Linux Workshop, Milano, Italy.

4.. Shaw, M. & Garlan B.,1996. Software Architecture : Perspective on a emerging discipline. Prentice Hall, ISBN 0131829572, 0New Jersey, USA.

5..Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; and Das, H.,2001. The CLARAty architecture for robotic autonomy. In IEEE Proceedings., ed., Aerospace Conference, vol. 1, pp 121-132, 2001 Montana, USA.

Perspectivas de futuro del desarrollo basado en modelos*

Diego Alonso Cáceres, Bárbara Álvarez Torres y Pedro Sánchez Palma

División de Sistemas e Ingeniería Electrónica (DSIE)
Universidad Politécnica de Cartagena, Campus Muralla del Mar s/n, 30202 Cartagena
{diego.alonso | balvarez | pedro.sanchez}@upct.es

Abstract. Durante el desarrollo del proyecto ANCLA aparecieron una serie de problemas relacionados con la complejidad de los sistemas por diseñar y con las limitaciones de las técnicas existentes en aquel entonces. En estos años, la ingeniería del software se ha fijado en la forma que tienen otras ramas del conocimiento para realizar sus diseños y está evolucionando hacia el llamado *desarrollo basado en modelos*. Este artículo presenta las bases de este nuevo enfoque software así como las posibles aplicaciones, a la luz de las lecciones aprendidas en ANCLA, al nuevo proyecto en que se embarca en el grupo de investigación DSIE: MEDWSA.

Keywords: Diseño basado en modelos, Kermeta, Modelado, Metamodelo.

1 Introducción

Según el diccionario de la Real Academia de la Lengua Española, un modelo es un «esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento». Hasta hace relativamente poco tiempo la ingeniería del software no se había fijado en esta forma de desarrollo, que durante tanto tiempo y con tan buenos resultados lleva aplicándose a otras ramas de la ciencia, como la física o la química.

El uso de modelos para desarrollar programas se denomina genéricamente *Model Driven Engineering* (MDE) y según Schmidt [1] representa «una aproximación esperanzadora para solucionar la incapacidad de los lenguajes de tercera generación de reducir la complejidad de las plataformas de implementación y de expresar conceptos del dominio de forma efectiva».

Uno de los primeros detalles que llaman la atención del desarrollador, y que le llevan a pensar que se encuentra ante un enfoque totalmente revolucionario, es el nombre elegido. Hasta ahora, todos los enfoques y paradigmas utilizados para desarrollar software contenían siempre la palabra *«programación»* (orientada a

* Este trabajo ha sido parcialmente financiado por el programa PMPDI 2006 de la Universidad Politécnica de Cartagena y por los proyectos CICYT ANCLA (TIC2003-07804-C05-02) y MEDWSA (TIN 2006-15175-C05-02).

objetos y últimamente orientada a aspectos) en su título[3] , haciendo referencia a la forma de implementar, en código, el programa que se encontraba en la mente del programador (y con suerte documentado en algún diagrama más o menos estándar).

El enfoque que promueve MDE hace referencia explícitamente a los conceptos importantes del diseño de la aplicación, utilizando la palabra *«modelo»*. Puesto que MDE utiliza modelos, y no un lenguaje de programación, como elementos atómicos del diseño, MDE es independiente del lenguaje y el paradigma de desarrollo utilizado. Según comenta Bézivin en [2], el cambio se resume en un giro del enfoque: de *«todo es un objeto»* a *«todo es un modelo»*.

La evolución es clara: se ha pasado de la programación estructurada (en la que lo importante eran las funciones) a la orientación a objetos (en la que lo importante son los objetos) al desarrollo basado en modelos (en el que lo importante son los conceptos presentes en los modelos). Se ha aumentado considerablemente el nivel de abstracción, centrándose en los conceptos y relegando los detalles de la implementación a una tercera o cuarta posición. Aquí es donde reside el gran potencial de esta nueva forma de desarrollo: se diseñan y manipulan los conceptos importantes para la aplicación que se pretende desarrollar, en vez de líneas de código en un lenguaje de programación. La sección 2 presenta una descripción general de este enfoque de desarrollo basado en modelos.

Para que pueda desarrollarse software siguiendo el enfoque MDE es imprescindible que existan una serie de herramientas que proporcionen el soporte necesario al diseñador, de forma que se concentre únicamente en el sistema que se va a modelar. El apartado 3 recoge un conjunto de herramientas diseñadas e integradas con Eclipse que proporcionan ese soporte, mientras que el apartado 4 presenta una herramienta experimental del entorno académico para aumentar la potencia expresiva del modelado: Kermeta.

Por último, la sección 5 resume las conclusiones que ha extraído el grupo DSIE del proyecto que ahora finaliza *«Arquitecturas dinámicas para sistemas de teleoperación»* (ANCLA), y cómo van a afectar al desarrollo del proyecto *«Marco conceptual y tecnológico para el desarrollo de software de sistemas reactivos»* (MEDWSA). Finalmente, se presenta un apartado de conclusiones del artículo.

2 Desarrollo Software Dirigido por Modelos (MDE)

Si bien es cierto que, como se ha comentado en la introducción, el desarrollo basado en modelos no es algo nuevo, su aplicación no ha sido posible hasta que se han desarrollado las primeras herramientas que proporcionan el soporte necesario para su aplicación. En este punto ha desempeñado un papel destacado el *Object Management Group* (OMG), desarrollando un amplio conjunto de herramientas y definiendo la *Model Driven Architecture* (MDA) [3].

MDA es la visión particular del OMG del desarrollo basado en modelos (MDE) que utiliza especificaciones desarrolladas por el propio OMG, como son el *Unified Modeling Language* (UML) [4], *Meta-Object Facility* (MOF) [5], *XML Metadata*

3 Salvo el no del todo fructífero *Desarrollo Basado en Componentes* (CBD)

Interchange (XMI), *Software Process Engineering Metamodel* (SPEM) y *Common Warehouse Metamodel* (CWM).

El desarrollo basado en modelos utiliza una abstracción del sistema que va a ser modelado, en la que están presentes todos los conceptos importantes del sistema y las relaciones entre ellos. Esta abstracción se denomina *meta-modelo* y restringe completamente los elementos presentes en el modelo de un sistema en particular. Por tanto, un modelo siempre es conforme a su meta-modelo (por definición). Siguiendo este enfoque, este meta-modelo debe tener también su propio meta meta-modelo, al que por supuesto tiene que ser conforme. Esta recurrencia tiene que romperse en algún momento para que el enfoque MDE pueda ser aplicable. La forma de romper esta recursividad en la definición de meta-modelos es diseñando un meta-modelo que sea conforme consigo mismo a nivel de modelo. Así es como está definido MOF.

MOF, como meta-meta-modelo (o meta-lenguaje) de la OMG, es la base que sustenta el enfoque MDA. De acuerdo con su página web, MOF proporciona el entorno en el que los modelos pueden ser exportados e importados entre herramientas, almacenados y extraídos de repositorios, transformados entre distintos meta-modelos y usados para generar código. Esta aplicación no está restringida a modelos basados en UML, sino que es extensible a modelos cuyo meta-modelo esté basado en MOF.

Esto último ha permitido el crecimiento de MDE, ya que no todos los desarrolladores comulgan con la filosofía particular de MDA. Sin embargo, la especificación de MOF les proporciona la herramienta necesaria para poder aplicar el enfoque MDE: un meta-modelo que es su propia definición. Las diferencias entre el enfoque genérico MDE y la visión particular (MDA) de la OMG se pueden resumir en los siguientes puntos:

- MDA contempla la separación de meta-modelos en dos capas principales: la capa independiente de los detalles de implementación (PIM) y aquélla en la que se van detallando (PSM).
- El lenguaje para especificar estos meta-modelos es UML.
- Las transformaciones entre modelos van siempre disminuyendo el nivel de abstracción, detallando cada vez más aspectos de la plataforma final.

El desarrollo software siguiendo el enfoque genérico MDE necesita herramientas que proporcionen soporte a cada una de las diferentes tareas que deben realizarse a lo largo del proceso de aplicación de MDE a un problema. Entre estas tareas destacamos las siguientes:

- Definición de lenguajes específicos del dominio (DSL) [6].
- Definición de meta-modelos y modelos.
- Definición de transformaciones entre meta-modelos y entre modelos.
- Adición de semántica y comportamiento a los meta-modelos.
- Ejecución y simulación de modelos.
- Definición de entretejido de modelos (*model weaving*).
- Definición de restricciones en los meta-modelos.
- Comprobación de que el modelo cumple las restricciones del meta-modelo, tanto en tiempo de creación como de ejecución.

MOF es una especificación orientada a desarrolladores de herramientas CASE que pretendan seguir un enfoque de desarrollo MDE. Para ello, la nueva especificación de MOF v2.0[4] define dos posibles niveles de compatibilidad de la herramienta con MOF, dependiendo de la funcionalidad que se haya implementado:

Essential MOF (EMOF): se corresponde con las características de los lenguajes orientados a objetos y XML. Proporciona un framework para mapear un modelo MOF a modelos de implementación, como JMI o XMI. Implementa un conjunto mínimo de paquetes de MOF y el paquete `Core::Basic` de la Infraestructura de UML v2.0 [7].

Complete MOF (CMOF): contiene el paquete EMOF, el resto de paquetes de MOF y el paquete `Core::Contructs` de la Infraestructura de UML v2.0, con el que se incluyen el resto de paquetes de la infraestructura de UML.

3 Herramientas de Soporte al Desarrollo MDE

A pesar de todo el esfuerzo que ha requerido desarrollar la especificación de MOF, a día de hoy sólo existe una implementación de amplia difusión: *Eclipse Modeling Framework* (EMF). EMF está basado en EMOF y se distribuye como plug-in para el entorno de desarrollo Eclipse. EMF proporciona no sólo las facilidades necesarias para crear y manipular la definición de modelos y meta-modelos, sino que, además, añade la gestión de la persistencia de éstos (usando XML) y la generación automática y configurable de código Java a partir del modelo. [8] es la fuente principal de información sobre EMF.

Alrededor del desarrollo de EMF han aparecido otros proyectos que extienden su funcionalidad pero que, al no estar contemplados en la especificación de MOF, han tenido que sacarse fuera del propio EMF. Todos estos proyectos están agrupados en un proyecto genérico denominado *Eclipse Modeling Framework Technology* (EMFT).

Por último, se está desarrollando un proyecto que proporciona soporte gráfico para la edición de modelos, de forma que se facilite a los usuarios el desarrollo de la aplicación que quieren diseñar. Este proyecto se denomina *Graphical Modeling Framework* (GMF) y su objetivo es servir de puente entre la especificación del meta-modelo, realizada gracias a EMF, y la creación de herramientas gráficas que faciliten la edición de modelos, basadas en el proyecto de edición gráfica genérica *Graphical Editing Framework* (GEF).

4 El Lenguaje de Programación Kermeta

MOF, en su definición actual, es un meta-lenguaje que permite describir la estructura de un meta-modelo, pero no tiene soporte para especificar completamente su semántica. El lenguaje Kernel Metamodeling [9] (Kermeta[5]) ha sido desarrollado por

4 Actualmente la versiyn oficial de MOF es la v1.4

5 http://www.kermeta.org

el grupo Triskell[6] como un lenguaje experimental para especificar no sólo la estructura sino también el comportamiento de un meta-modelo. Las principales características de Kermeta son las siguientes:

Imperativo: incluye las estructuras de control tradicionales, bucles y sentencias de bifurcación.

Orientado a objetos: soporta herencia múltiple y enlace dinámico (*late binding*).

Orientado a modelos: conceptos como asociación y composición son entidades de primer nivel.

Funcional: soporta la definición de funciones e incluye un subconjunto de expresiones del *lambda-calculus*.

Fuertemente tipado: en la definición de operaciones y en el uso de genéricos.

Reflexividad: el modelo completo del lenguaje es accesible en ejecución.

Para ser capaz de expresar comportamiento, Kermeta extiende el meta-modelo de EMOF añadiéndole un paquete extra. Las figuras **1** y **2** muestran, respectivamente, los dos paquetes principales del meta-modelo de Kermeta.

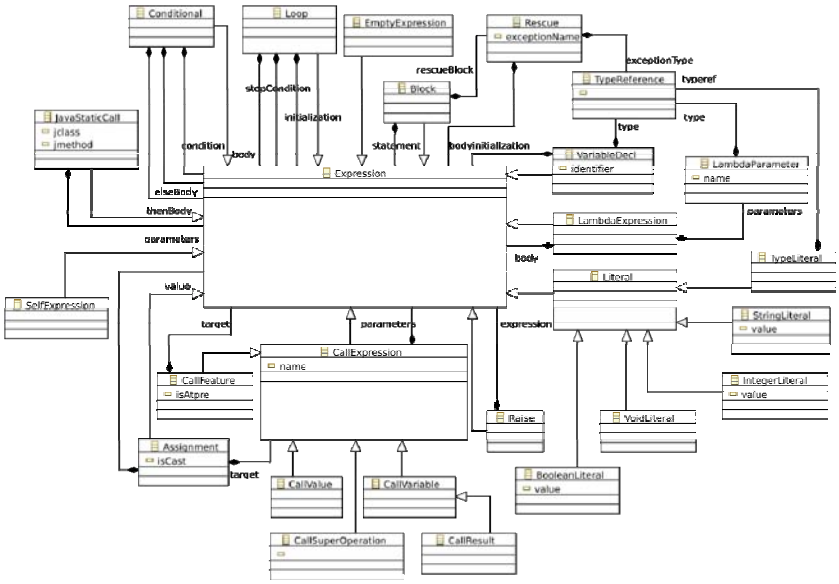


Fig. 1. El paquete `kermeta::structure`

La primera de ellas, figura 1, muestra el paquete `kermeta::structure`, con el que se puede especificar la estructura de un modelo. Puesto que Kermeta extiende EMOF, ambos meta-modelos son prácticamente iguales, siendo la adición de la clase `Expression` la única diferencia sustancial en este punto. Esta clase se utiliza como nexo de unión con el paquete desarrollado por Triskell para especificar comportamiento, `kermeta::behavior` (ver figura 2).

6 Triskell se encuentra en el centro de investigación francés IRISA, http://www.irisa.fr/triskell

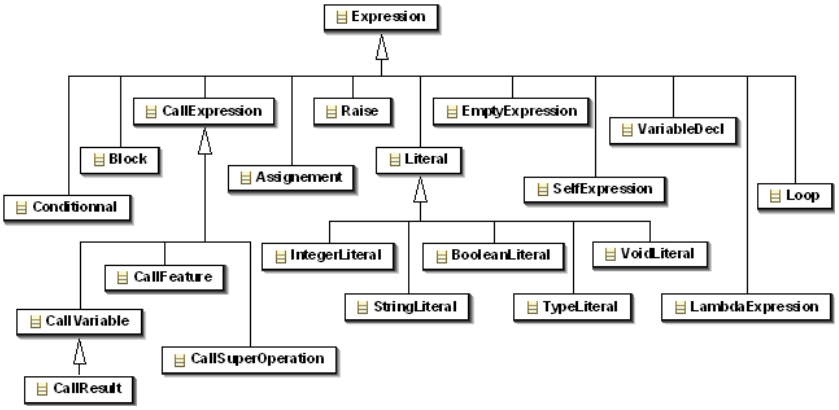


Fig. 2. El paquete `kermeta::behavior`

Kermeta ha sido desarrollado, además, como una solución integrada que proporciona un entorno uniforme al desarrollo software basado en MDE, de forma que no se tengan que manejar diversos programas y lenguajes diferentes para realizar cada una de las tareas involucradas en el desarrollo MDE, según se describen en el apartadosec:mde. Para ello ha sido integrado como un plug-in del entorno de programación Eclipse, utilizando como base EMF. Kermeta se distribuye bajo la licencia libre EPL (Eclipse Public License) y el plug-in incluye varias herramientas, como un intérprete, un depurador, un editor y varios filtros para importar y exportar programas Kermeta. Kermeta puede, por tanto, ser utilizado como un lenguaje capaz de realizar las siguientes tareas:

- Especificar de forma precisa el comportamiento de un meta-modelo.
- Simular modelos.
- Definir transformaciones de modelos.
- Definir restricciones en los modelos.

5 Perpectivas de aplicación del enfoque MDE

Las posibilidades de aplicación de esta nueva forma de desarrollar software utilizando el enfoque MDE son, potencialmente, enormes. Aunque actualmente se encuentra en fase de desarrollo y asentamiento de las bases teóricas y de las herramientas de soporte, MDE se encuentra en un estado «más que utilizable» para labores de investigación en este campo.

En el año 2003 se concibió el proyecto ANCLA para «especificar y diseñar una arquitectura dinámica de software para el desarrollo de sistemas de teleoperación, que pueda ser configurada en función de las necesidades particulares de cada sistema, reutilizando de esta forma no sólo componentes sino también patrones de interacción entre los mismos». Como es lógico, este planteamiento contempla tanto los intereses

del grupo DSIE (sistemas de teleoperación) como las técnicas e inquietudes (desarrollo basado en componentes, arquitecturas dinámicas) existentes en aquel instante de tiempo.

Tres años después, los intereses del grupo y las técnicas e inquietudes de la ingeniería del software han evolucionado. Sin embargo, ANCLA, tal y como muestra [10] detalladamente, ha servido para que afloren una serie de problemas nuevos, que no fueron contemplados en su momento, y que gracias al avance de la técnica pueden ser, esperamos, solucionados. ANCLA ha servido, además, para ampliar el campo de mira del grupo DSIE, para aumentar sus intereses. De toda esta experiencia nace el proyecto MEDWSA, con un ambicioso objetivo: «definir un marco conceptual y tecnológico para el desarrollo de sistemas reactivos, basado en líneas de producto, que aproveche las ventajas de las tendencias actuales del desarrollo dirigido por modelos». La redacción del objetivo de MEDWSA resume parte de las lecciones aprendidas en ANCLA y de los avances de la técnica:

- Se pretende adoptar un enfoque de desarrollo basado en modelos. El proyecto ANCLA demostró que el desarrollo de sistemas robóticos teleoperados es muy complicado, debido a la variabilidad inherente al dominio, a pesar de que todos estos sistemas comparten un conjunto de requisitos [11]. Esperamos que adoptando un enfoque de desarrollo utilizando modelos como pieza central del mismo ayude a aliviar la complejidad del proceso de desarrollo software.
- Se ha aumentado el área de interés del grupo. El centro de atención se amplía, pasando de contemplar únicamente sistemas teleoperados para abarcar *sistemas reactivos*: sistemas que interaccionan continuamente con el entorno, recibiendo estímulos de él y produciendo salidas en respuesta a los mismos. Dentro de esta familia se contemplan las siguientes áreas de atención: sistemas de inspección visual (*VIPS*), sistemas teleoperados, redes de sensores (*Wireless Sensor Area Networks, WSAN*[7]) y redes domóticas. En todos estos dominios, distintos componentes del grupo DSIE han venido trabajando en los últimos años. Por tanto, el grupo dispone del conocimiento necesario del dominio para adoptar el enfoque MDE.
- La adopción del enfoque MDE requiere de un conocimiento profundo del dominio de aplicación, ya que es la primera entrada de datos al proceso. El enfoque *Software Product Lines* (SPL) [12] se presenta como un complemento del proceso MDE, ya que ayuda estudiar el dominio al que se va a aplicar este último.

Por tanto, parte del trabajo que se desarrollará en el contexto de MEDWSA consistirá en la aplicación de estas técnicas a los diversos dominios de interés del grupo DSIE. Se pretende utilizar la herramienta Kermeta[8] para definir meta-modelos y modelos, transformaciones entre ellos y lenguajes de dominio, de cara a una posible evaluación de la misma. Se pretende además comparar Kermeta con otros enfoques propuestos en el seno del proyecto coordinado *Models Environments Transformations and Applications* (META).

---

7 Trabajo financiado por la fundación Séneca, proyecto 02998-PI-05

8 Descubierta gracias a la estancia de Cristina Vicente Chicote y Diego Alonso Cáceres en el grupo Triskell durante el verano de 2.006

6 Conclusiones

En este artículo se ha presentado el nuevo enfoque de desarrollo software basado en modelos, denominado genéricamente MDE. MDE supone un salto en el nivel de abstracción utilizado hasta ahora para desarrollar software: de los detalles concretos de implementación (*código*) a los conceptos importantes del dominio (*modelo*). Este enfoque sería imposible de utilizar si la OMG no hubiera realizado la especificación de MOF v2.0 y sin el desarrollo de EMF.

Se ha descrito también el lenguaje experimental a nivel académico Kermeta, que pretende completar la especificación de MOF añadiendo la posibilidad de definir el comportamiento de un meta-modelo, de forma que pueda ser ejecutado. Kermeta proporciona, además, un entorno homogéneo para realizar gran parte de las actividades involucradas en el ciclo MDE, como son la definición de restricciones en el meta-modelo y de transformaciones entre modelos.

Por último, se han mostrado algunos detalles de cómo el proyecto ANCLA, y principalmente los nuevos problemas que surgieron durante su desarrollo, y de cómo la maduración de los nuevos enfoques basados en modelos (principalmente MDE) han influido en los objetivos del proyecto MEDWSA y cómo se pretenden aplicar al desarrollo del mismo durante estos tres siguientes años.

Bibliografía

1 Schmidt, Douglas C.: «Model-Driven Engineering». *Computer*, 2006, **39(2)**, IEEE Computer Society. ISSN 0018-9162. 10.1109/MC.2006.58.

2 Bézivin, Jean: The unification power of models». *Software and Systems Modeling*, 2005, **4(2)**, pp. 171–188. 10.1007/s10270-005-0079-0.

3 *Model Driven Architecture Guide Version v1.0.1, omg/2003-06-01*. Object Management Group (OMG), 2003. Disponible en http://www.omg.org/docs/omg/03-06-01.pdf

4 *Unified Modeling Language (UML) Superstructure Specification v2.0, formal/05-07-04*. Object Management Group (OMG), 2005. Disponible en http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

5 *Meta-Object Facility (MOF) v2.0, ptc/04-10-15*. Object Management Group (OMG), 2004. Disponible en http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF

6 van Deursen, Arie; Klint, Paul y Visser, Joost: «Domain-specific languages: an annotated bibliography». *SIGPLAN Not.*, 2000, **35(6)**, pp. 26–36. ISSN 0362-1340. 10.1145/352029.352035.

7 *Unified Modeling Language (UML) Infrastructure Specification v2.0, formal/05-07-05*. Object Management Group (OMG), 2006. Disponible en http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

8 Budinsky, Frank; Steinberg, David; Merks, Ed; Ellersick, Raymond y Grose, Timothy: *Eclipse Modeling Framework*. Eclipse series. Addison-Wesley Professional, 1ª edición, 2003. ISBN 0131425420.

9 Muller, Pierre-Alain; Fleurey, Franck y Jézéquel, Jean-Marc: «Weaving executability into object-oriented meta-languages». **En:** *Proceedings of MODELS/UML'2005*, volumen 3713

de *Lecture Notes on Computer Science*, pp. 264–278. Springer-Verlag. ISBN 3-540-29010-9. ISSN 0302-9743, 2005. 10.1007/11557432\s\do5(1)9.

10 Grupo DSIE: «ANCLA: resumen de aportaciones, resultados y conclusiones del trabajo realizado», 2006.

11 Álvarez, Bárbara; Sánchez-Palma, Pedro; Pastor, Juan y Ortiz, Francisco:006. «An architectural framework for modeling teleoperated service robots». *Robotica*, 2006, **24(4)**, Cambridge University Press. ISSN 0263-5747. 10.1017/S0263574705002407.

12 Clements, Paul y Northrop, Linda: *Software Product Lines : Practices and Patterns*. Software Engineering (SEI). Addison-Wesley Professional, 3ª edición, 2001. ISBN 0201703327.

Verificación de Propiedades de Dominio mediante Modelado Preciso♦

Francisco Javier Lucas Martínez[1], Ambrosio Toval Álvarez[1], and Francisco J. Ruiz[2]

[1] Grupo de Ingeniería del Software
Departamento de Informática y Sistemas
Universidad de Murcia (España)
fjlucas@um.es, atoval@um.es
[2] Grupo de Investigación DSIE. .
Universidad Politécnica de Cartagena
30202 Cartagena (España)
francisco.ortiz@upct.es

Resumen. Los sistemas teleoperados (STO) constituyen un dominio donde la seguridad, y la calidad, de los modelos que se utilizan es de especial importancia debido a los costes que un error puede tener, no sólo económicos sino incluso en la seguridad física de las personas que operan el sistema. Este artículo presenta un enfoque riguroso para mejorar características específicas de este tipo de sistemas, que pueden ser detectadas durante las fases de modelado del mismo, a través de los diagramas de UML utilizados en su desarrollo.

Keywords: Propiedades de Dominio, Sistemas Teleoperados, UML, Maude

1 Introducción

UML [1] es un lenguaje de modelado que surge como unificación de diversas notaciones que ha sido promovido por el OMG. Esta notación ha sido adoptada como la notación estándar para la descripción de sistemas OO y es utilizada en multitud de dominios. Pero el modelado con UML tiene ciertas imprecisiones que, junto con posibles errores cometidos por los propios modeladores, provocan que los modelos del sistema creados puedan no ser correctos.

Durante el desarrollo de un sistema con UML, un modelo puede contener errores y ambigüedades desde el punto de vista de la semántica dinámica del modelo, con arreglo a las reglas de buena formación (*well-formedness rules*) de su metamodelo, o bien del propio dominio al que pertenece. Esquemáticamente, las propiedades pueden estar relacionadas con:

♦ Financiado por el Ministerio de Ciencia y Tecnología (Spain), proyecto DYNAMICA/PRESSURE TIC 2003-07804-C05-05, y por Consejería de Educación y Ciencia, en el marco del Plan Regional de Investigación Científica, Desarrollo Tecnológico e Innovación de Castilla-La Mancha, proyecto DESERT PBC-05-012-3.

1. Semántica estática de los diagramas de acuerdo con las reglas de la notación empleada. Esta semántica define como una instancia de un constructor debería ser conectado a otras instancias.
2. Semántica dinámica de los diagramas. Que define el significado de los constructores bien formados.
3. Semántica del dominio concreto al que pertenezcan los diagramas. Este tipo de propiedades son las más interesantes ya que las herramientas CASE convencionales no les suelen dar soporte.

La propuesta que se desarrolla en este artículo, que está todavía en un estado inicial, aborda el tercer grupo de propiedades. En trabajos anteriores [2,3,4,5,6] abordamos propiedades relativas a los dos primeros puntos. Este enfoque se basa en la formalización del metamodelo de diagramas de UML en Maude [7] (que es un lenguaje de especificación formal, basado en lógica ecuacional y lógica de reescritura).

El dominio elegido para la identificación de estas propiedades fue el de los sistemas teleoperados (STO) y en concreto se trabajó con el sistema desarrollado en el proyecto EFTCOR [8]. Estos sistemas constituyen un dominio donde la seguridad y la calidad de los modelos que se utilizan es de especial importancia debido a los costes que un error puede tener, no sólo económicos sino incluso en la seguridad física de las personas que operan el sistema.

Para la definición de las propiedades relacionadas con este dominio, el equipo de trabajo revisó diversa documentación relacionada con el proyecto EFTCOR [9,10,11] y se identificaron un conjunto de propiedades relacionadas con este dominio, y que se podrían clasificar en dos tipos:

- Propiedades relacionadas directa y específicamente con el dominio de los sistemas teleoperados.
- Propiedades relacionadas con la semántica un diagrama concreto de modelado, pero que aplicadas en este dominio se muestran especialmente útiles.

El resto del trabajo se estructura de la siguiente forma: la sección 2 analiza las principales características del sistema que se está desarrollando dentro del proyecto EFTCOR. En la sección 3 aparecen algunas propiedades identificadas que están relacionadas directa y específicamente con el dominio de los sistemas teleoperados. La sección 4 muestra otro conjunto de propiedades relacionadas con diagramas UML pero que aplicadas en este dominio se muestran especialmente útiles. Por último, se muestran las conclusiones y trabajos futuros.

2 Proyecto EFTCOR

El proyecto EFTCOR desarrollado, entre otros grupos, por la Universidad Politécnica de Cartagena, tiene como finalidad el desarrollo de un sistema hardware (robot) para la realización de operaciones de limpieza de cascos de buques.

El proyecto consta de varios subsistemas bien diferenciados, que abarcan desde el control de los dispositivos hardware utilizados, pasando por monitorización del

sistema y de visión del casco del barco. Para la realización de este artículo nos centraremos en los modelos software, utilizados para el desarrollo de este sistema así como de la documentación antes mencionada.

En este dominio parecía especialmente útil el identificar un conjunto de propiedades que pudieran ser utilizadas como ejemplo de este tipo de aplicaciones, y extendidas en futuros trabajos a otras propiedades similares de interés en el dominio, de forma que ayudaran en la construcción más rápida y segura de sistemas dentro de este ámbito. En particular, nos centraremos en el estudio de propiedades relacionadas con el dominio que pueden ser comprobadas sobre los diagramas de estados de UML y serán especificadas sobre las especificaciones creadas en [12].

3 Identificación de Propiedades de Dominio Específicas

En esta sección, se analizará una serie de propiedades que se han identificadas y que están relacionadas directa y específicamente con el dominio de los sistemas teleoperados. En la sección 4 veremos otro tipo de propiedades que, si bien son generales del lenguaje de modelado utilizado, son de especial utilidad aplicadas dentro de este dominio.

Analizando el sistema y los modelos utilizados durante el desarrollo del mismo, se identificaron una serie de propiedades relacionadas con la capacidad o no de tratar eventos por parte de los estados que los reciben. Un ejemplo para el diagrama mostrado en la Figura 1 sería el evento *'se disparó alarma'* que sólo puede tratarse por el estado *'Parada Segura'*, cualquier transición con este evento, que conduzca a otro estado provocaría errores y posibles daños en el sistema.

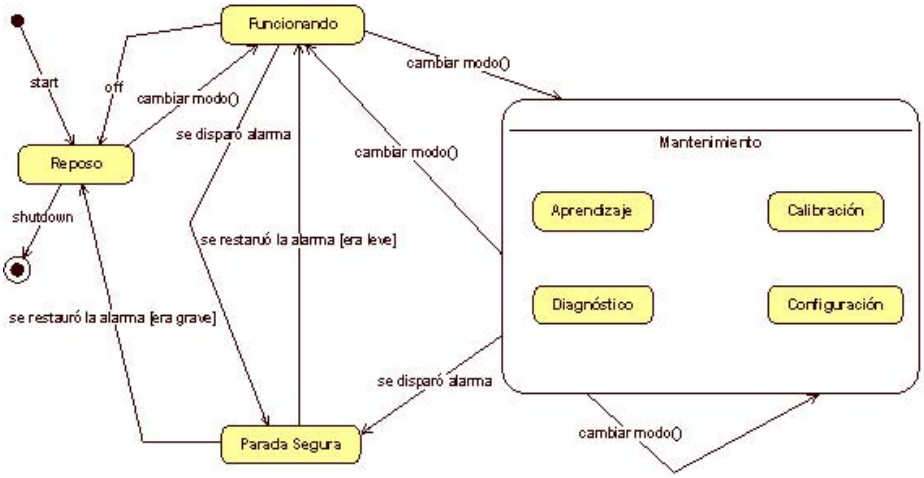


Figura 1. Ejemplo de diagrama de estados utilizado para modelar los posibles estados del sistema

Para intentar solucionar este problema introducimos el concepto de *Código de Seguridad* asociado a un estado o a una transición. Este código representa:

- En un estado el valor máximo de seguridad que es capaz de aceptar, es decir, indica el valor máximo que dicho estado 'sabe' tratar.

- En una transición el código de seguridad asociado al evento que dispara la transición y que debe de poder ser tratado en el estado destino.

La inclusión de estos códigos en un diagrama de estados hace necesaria la extensión del metamodelo de UML para estos diagramas y, por tanto, la modificación de las formalizaciones realizadas en [12]. La extensión del metamodelo que da soporte a estos códigos se muestran de forma resumida en la Figura 2 en forma de perfil.

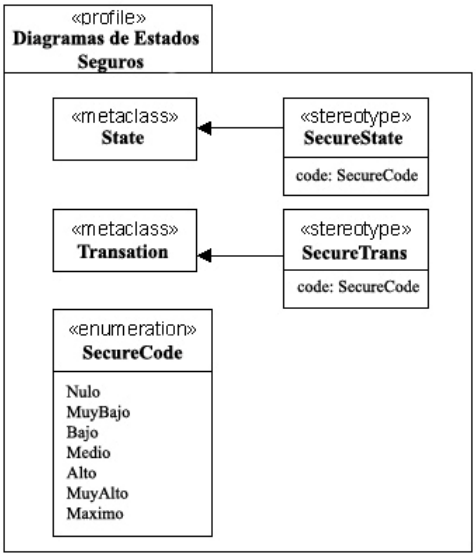


Figura 2. Perfil UML para dar soporte a los códigos de seguridad en Diagramas de Estados

En base a estos dos códigos (en transición y en estado) las propiedades que se han definido tienen como finalidad asegurar que el estado al que se llega tras una transición sepa tratar ese evento. Y como veremos, estos códigos nos ayudarán a resolver el tipo de problemas detectados. Las propiedades se analizarán en detalle a continuación.

3.1 P1. Máximo Código Tratable

Una de las aplicaciones de la inclusión de este código, sería saber si el máximo código existente en las transiciones de un diagrama puede ser tratado por algún estado del mismo. De esta forma podremos comprobar si los códigos generados pueden ser tratados por el diagrama.

Un fallo en la comprobación de esta propiedad implicaría una de las siguientes acciones:

- Revisar el código de la transición que provoca el fallo.

- Revisar los códigos de los estados del diagrama, por si fuera necesario aumentar su nivel.
- Insertar un nuevo estado si la transición que lo provoca es correcta, que pueda tratar ese código de seguridad.

La Figura 3 muestra la especificación de esta propiedad.

```
(fmod PROPERTY1 is
    ...
    op testProp : StatechartDiagram -> String .
    op testProp1 : StateVertex TransitionList -> String .
    op testProp : NEStateVertexList TransitionList -> String .
    op testProp : Transition StateVertexList -> String .

    var estado : StateVertex . var fin : StateVertex .
    var inicio : PseudoState . var lista : TransitionList .
    var transicion : Transition . var trans : Transition .
    var etiqueta : TransitionLabel . var listaEstados : NEStateVertexList .
    var nombreEstado : VertexName . var QID : Qid .

    eq testProp(statechartDiagram(estado,lista)) =testProp1(estado,lista) .
    eq testProp1 (sequentialState(nombreEstado,noInternalAction,
            inicio listaEstados), lista) = testProp(listaEstados,lista) .

    eq testProp (listaEstados, noTransition) = "" .
    eq testProp (listaEstados, transicion lista) =
        testProp(transicion,listaEstados) + testProp(listaEstados,lista) .

    eq testProp (transicion, finalState (QID, noInternalAction, 100)) =
        "ERROR: Transicion no tolerada en diagrama por ningun estado.
        Codigo de error: " + ... .

    eq testProp (transicion, estado listaEstados) =
        if getCodigoSeguridad(estado) <
            getCodigoTransaccion(getTransitionLabel(transicion)) then
                testProp(transicion, listaEstados)
        else "" fi .
endfm)
```

Figura 3. Propiedad 1. Máximo Código Tratable

3.2 P2. Correspondencia estado-transición

Ésta es una propiedad que está relacionada con la anterior y que es un poco más específica. En este caso, se quiere asegurar que todas las asociaciones transición-estado destino son compatibles. Es decir, que el estado que debe tratar un evento es realmente capaz de trabajar con ese nivel de seguridad.

Con la propiedad anterior nos decía si sería necesario incluir un nuevo estado o si una transición tenía asociado un código incorrecto. Esta otra propiedad, sería

complementaria a la anterior, y partiríamos de una situación en la que sabemos que al menos existe un estado que puede tratar cualquier transición del diagrama, ya que su código de seguridad es más alto.

```
(fmod PROPERTY2 is ...
    op testProp2 : StatechartDiagram -> String .
    op testProp21 : StateVertex TransitionList -> String .
    op testProp2 : NEStateVertexList TransitionList -> String .
    op getEstado : NEStateVertexList Transition -> StateVertex .
    ...
    eq testProp2(statechartDiagram(estado,lista)) =
                                                    testProp21(estado,lista) .
    eq testProp21(sequentialState(nombreEstado,
                                  noInternalAction,listaEstados), lista) =
                        testProp2(listaEstados,lista) .

    eq testProp2(listaEstados, noTransition) = "" .
    eq testProp2(listaEstados, transicion lista) =
        if getCodigoSeguridad(getEstado(listaEstados, transicion)) <
            getCodigoTransaccion(getTransitionLabel(transicion)) then
                "ERROR: Estado [objetivo] "+string(getReceptor(transicion))+...
                + testProp2(listaEstados, lista)
        else
            testProp2(listaEstados,lista)
        fi .

    eq getEstado(finalState (QID, noInternalAction,100),transicion) =
        finalState (QID, noInternalAction, 100) .
    eq getEstado(estado listaEstados, transicion) =
        if getReceptor(transicion) == getNombre(estado) then estado
        else getEstado(listaEstados, transicion) fi .
    ...
endfm)
```

Figura 4. Propiedad 2. Correspondencia estado-transición

Un error en la comprobación de esta propiedad implicaría una de las siguientes acciones:

- Revisar los códigos asociados al transición-estado que provocaron el error.
- Revisar el estado destino de la transición y cambiarlo por otro que pueda tratar dicho evento.

Si no se hubiera comprobado la anterior propiedad, también podría ser necesaria la inclusión de un nuevo estado. La especificación de esta propiedad se muestra en la Figura 4.

4 Identificación de Propiedades Generales Aplicadas al Dominio

En esta sección, como ya se ha comentado, analizaremos propiedades que deben cumplirse dentro de un diagrama concreto de modelado, pero que aplicadas en este dominio se muestran especialmente útiles. Estas propiedades están definidas sobre el diagrama de estados y están relacionadas con la ortogonalidad y la alcanzabilidad.

Dentro de este dominio, el modelado a distintos niveles de diagramas de estados y su posterior composición puede hacer aparecer problemas típicos de este tipo de diagramas: estados inalcanzables, ciclos,... Propiedades como las que veremos a continuación son propias del diagrama de estados, tienen especial interés en su uso dentro del modelado de estos sistemas.

4.1 P3. Ortogonalidad

Esta propiedad ya se encuentra resuelta y formalizada en [6], por lo que en esta sección sólo veremos una aplicación de la misma dentro del dominio de los sistemas teleoperados.

Intuitivamente, la ortogonalidad requiere que un sistema que se encuentra en un estado compuesto concurrente, tenga un subestado activo en cada región concurrente que lo compone [12]. Esto implica que un evento producido en el diagrama, no puede provocar la entrada en sólo uno de los estados concurrentes o la salida de sólo uno de los estados concurrentes que forman el estado compuesto. Esta propiedad puede ser especialmente útil en el modelado de sistemas grandes, donde se reutilicen sub-máquinas de estados dentro de otras más grandes, en las que la comprobación de esta propiedad sería más costosa. La Figura 5 muestra un ejemplo que verificaría dicha propiedad ya que ninguna transición provoca un fallo (En [6] puede verse un ejemplo de diagramas que no la cumple).

4.2 P4. Alcanzabilidad de un estado

Dentro de este dominio, esta propiedad nos informaría sobre si es posible que el sistema pueda hacer que el robot alcance o no un determinado estado. Por ejemplo, en la Figura 1 se podría garantizar que se puede alcanzar el estado *'Parada Segura'* desde cualquier otro estado. Al igual que la propiedad anterior, esta propiedad puede ser especialmente útil cuando estemos modelando sistemas grandes en los que se reutilicen diagramas de estados más pequeños.

Un error en la comprobación de esta propiedad implicaría revisar el modelo para corregir está inaccesibilidad. La especificación de esta propiedad se muestra en la Figura 6.

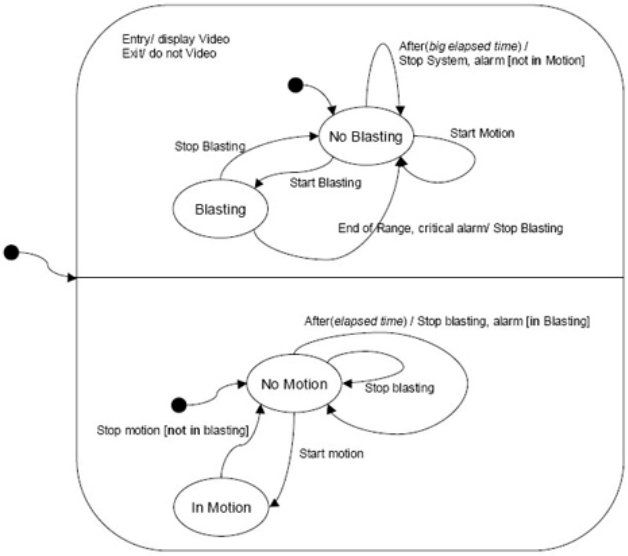


Figura 5. Ejemplo de diagrama de estados

4.3 P5. Simulación de Diagramas de Estados mediante Diagramas de Colaboraciones

Otra de las características propias del modelado con UML es la consistencia entre un diagrama de colaboraciones y los diagramas de estados asociados a las clases que participan en la colaboración con el objetivo de validar si la ejecución de la interacción provoca que algún objeto de los que participan queda en un estado incorrecto o comprobar si se aplican métodos que no están disponibles en estado actual de cada objeto.

Esta propiedad puede resultar útil directamente sobre este dominio para garantizar que la ejecución del sistema no lo va a llevar a estados no deseados. Pero además, si tenemos en cuenta el perfil definido en la Figura 2, esta simulación puede ser utilizada para validar el correcto uso de métodos en el diagrama de colaboraciones según los códigos de seguridad que se hayan definido. La especificación de esta propiedad está todavía en curso y será mostrada en trabajos futuros.

5 Conclusiones y Trabajos Futuros

En este trabajo, se muestra la viabilidad del uso de métodos formales para la realización de tareas de verificación para dominios específicos, en concreto el trabajo se centra en el dominio de los STO, que son sistemas especialmente críticos. Además, se han mostrado tanto propiedades generales de un lenguaje de modelado que pueden

ser aplicados en este dominio como propiedades específicas que pueden ser comprobadas y demostradas cuando se está todavía modelando el mismo.

```
(fmod PROPERTY4 is
    ...
    op reachable : VertexName VertexNameList VertexNameList
            TransitionList StateVertex -> Bool .

    eq reachable (VN1, Q + VNL1, VNL2, TL, initialState(Q)) = true .
    eq reachable (VN1,Q + noVertexName,VNL2,TL,initialState(Q)) = true .
    eq reachable (VN1,VN2 +noVertexName,VNL2,TL,initialState(Q)) = false .
    eq reachable (VN1,noVertexName,VNL2,TL,initialState(Q)) = false .
    eq reachable (Q, VNL1, VNL2, TL, initialState(Q)) = true .
    eq reachable (VN1, VN2 + VNL1, VNL2, TL, initialState(Q)) =
        reachable (VN1, VNL1, VNL2, TL, initialState(Q)) OR
        canAdd (VN1, VN2 + VNL1,VN1 + VNL2,TL,initialState(Q),VN1 + VNL2).

    eq canAdd(VN1, VN2 + VNL1, VN3 + VNL2, TL, initialState(Q),VNL3) =
        if VN2 == VN3
        then false
        else canAdd(VN1, VN2 + VNL1, VNL2, TL, initialState(Q),VNL3)
        fi .
    eq canAdd(VN1,VN2 + VNL1,VN3 + noVertexName,TL,initialState(Q),VNL3) =
        if VN2 == VN3
        then false else
            reachable(VN2,sourceStateList(VN2,TL),VNL3+VN2,TL,initialState(Q))
        fi .
    eq canAdd(VN1, VN2 + VNL1, noVertexName, TL, initialState(Q),VNL3) =
        reachable(VN2,sourceStateList(VN2,TL),VNL3+VN2,TL,initialState(Q)).
    eq sourceStateList (VN, transition (TN, SOUNEVNL, TARNEVNL, TLA) TL) =
        getSVNL (VN, SOUNEVNL, TARNEVNL) + sourceStateList (VN, TL).
    eq sourceStateList (VN,transition(TN,SOUNEVNL,TARNEVNL,TLA)
                noTransition) = getSVNL (VN, SOUNEVNL, TARNEVNL).
    eq sourceStateList (VN, noTransition) = (noVertexName) .

    eq getSVNL (VN, SOUNEVNL, TARNEVNL) =
        if (is VN in TARNEVNL)
        then SOUNEVNL
        else noVertexName fi .
    ...
endfm)
```

Figura 6. Propiedad 4. Alcanzabilidad

Esta aproximación ofrece ventajas como la fácil incorporación de nuevas propiedades, la integración de toda la propuesta dentro del mismo marco formal que facilita la comprobación de propiedades sin tener que acudir a distintos formalismos o herramientas para cada tipo de propiedad. En su contra, aparecen aspectos como la necesidad de tener un conocimiento alto de las formalizaciones así como de las técnicas utilizadas para las mismas al menos mientras no se disponga de una herramienta que oculte todos estos aspectos a los profesionales próximos al dominio.

Además, en determinados dominios la identificación de propiedades útiles y no triviales que puedan comprobarse durante las fases de modelado del sistema puede ser difícil.

Como continuación directa de este trabajo, se está trabajando en la modificación del perfil definido en la Figura 2 para eliminar el código de seguridad de las transiciones e incluirlo en los métodos que se definen en el diagrama de clases asociado al sistema, de forma que el nivel de seguridad estará asociado al método que se ejecuta en la clase cuando se produce un evento un evento y no a la transición que se produce. De este modo, se facilitará enormemente el modelado, ya que no obligaremos al usuario a introducir este código en cada una de las transiciones; y se podrán comprobar propiedades más complejas relacionadas la explicada en la sección 4.3. Además, también se seguirá estudiando el dominio de los STO, para identificar más propiedades complejas y útiles dentro de este dominio.

Agradecimientos

A los alumnos de 5º de Ingeniería en Informática de la Universidad de Murcia Pedro Delgado Yarza y Antonio Alejando López Lorca por su colaboración en la realización de este trabajo, en el marco de la asignatura de Técnicas Formales en Ingeniería del Software.

Referencias

1. OMG: UML Specification 1.5, http://www.omg.org/uml. (2001)
2. Lucas, F., Molina, F., Toval, A., De Castro, M., Cáceres, P., Marcos, E.: Precise WIS Development. International Conference on Web Engineering, ICWE ’06. Menlo Park, California (USA). ACM Proceedings. ISBN: 1-59593-352-2 (2006)
3. Lucas Martínez, F.J., Toval Álvarez, A.: A Precise Approach for the Analysis of the UML Models Consistency. BP-UML’05: 1st International Workshop on Best Practices of UML, dentro del 24th Int. Conf. on Conceptual Modeling (ER 2005), Klagenfurt (Austria). 24-28 octubre, 2005. ISBN: 3-540-29395-7. LNCS 3770 Springer (2005)
4. Fernández Alemán, J.L., Toval Álvarez, A.: Improving System Reliability via Rigorous Software Modeling: The UML Case. Proceedings of the 2001 IEEE Aerospace Conference (track 10: Software and Computing), Montana, USA IEEE Computer Society (2001)
5. Lucas Martínez, F.J., Toval Álvarez, A.: Formal Verification of Properties in the UML Collaboration Diagram. ICSSEA 2004: 3rd Workshop on SYSTEM TESTING AND VALIDATION. Paris (2004)
6. Fernández Alemán, J.L., Toval Álvarez, A.: Can Intuition Become Rigorous? Foundations for UML Model Verification Tools. International Symposium on Software Reliability Engineering, Published by IEEE Press (2000)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcote, C.: Maude 2.0 Manual. Versión 1.0, http://maude.csl.sri.com/. (2003)

8. EFTCOR: Environmental Friendly and cost-effective Technology for Coating Removal. Master's thesis, European Project within the 5th Marco de trabajo Program (GROWTH G3RD-CT-00794) (2003)
9. Ortiz, F.J.: Arquitectura de Referencia para Unidades de Control de Robots de Servicio Teleoperados. Master's thesis, ETSII Universidad Politécnica de Cartagena. (2005)
10. Pastor, J.A.: Sistemas Teleoperados. Master's thesis, ETSII Universidad Politécnica de Cartagena. (2002)
11. Pastor, J., Álvarez, B., Sánchez, P., Ortiz, F.: An Architectural Framework for Service Robot Control Applications. Actas de las II Jornadas DYNAMICA (2004)
12. Fernández Alemán, J.L.: Una Propuesta de Formalización de la Arquitectura en Cuatro Capas de UML. Master's thesis, Departamento de Informática y Sistemas. Facultad de Informatica (2002)

Selección de componentes basada en requisitos, en el marco de SIREN♦

Miguel A. Martínez[1], Manuel F. Bertoa[2], Antonio Vallecillo[2], Ambrosio Toval[1]

[1] Grupo de Investigación de Ingeniería del Software
Departamento de Informática y Sistemas
Universidad de Murcia. Campus de Espinardo. 30071. Murcia
{mmart, atoval}@um.es
[2] Grupo de Investigación de Ingeniería del Software de la Universidad de Málaga
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga. Campus de Teatinos. 29071. Málaga
{bertoa, av}@lcc.uma.es

Resumen. La Ingeniería de Requisitos (IR) es un campo muy activo dentro de la Informática, y en particular dentro de la Ingeniería del Software (IS), y se dirige a unas actividades esenciales en el trabajo diario de las organizaciones de desarrollo de software. En este artículo realizamos un estudio previo de los principales enfoques de selección de componentes que prestan especial atención a los requisitos del sistema. Además, presentamos las ideas preliminares de una propuesta de trazabilidad, en el marco del método de IR SIREN, entre requisitos y componentes software que los implementan. El objetivo final de nuestra propuesta es conseguir un método de IR que guíe la selección, el desarrollo y la composición de un conjunto de componentes software independientes. Este enfoque puede ser usado en cualquier Sistema de Información (SI) en general, presentando en este artículo ejemplos muy sencillos relacionados con la seguridad en los SI.

Keywords: *Ingeniería de Requisitos, Componentes COTS, Selección de Componentes, Reutilización, Trazabilidad*

1 Introducción

Se ha demostrado mediante varios estudios experimentales [1,2,3] que los principales problemas en las áreas de desarrollo y producción de software se encuentran en la especificación y la gestión de los requisitos del cliente y del entorno de funcionamiento.

El método SIREN (SImple REuse of software requiremeNts) [4,5], propuesto por el Grupo de Investigación de Ingeniería del Software de la Universidad de Murcia, aporta un proceso sistemático para obtener y especificar los requisitos de un sistema

♦ Parcialmente subvencionado por el proyecto de la CICYT DYNAMICA/PRESSURE TIC 2003-07804-C05-05, y por el proyecto DESERT (DEveloping Secure systEms through Requirements and Tools), PBC-05-012-3 de la Consejería de Educación y Ciencia, Junta de Castilla-La Mancha (España).

software, basado en estándares de Ingeniería del Software, y sobre todo, en reutilización.

Existe una amplia aceptación [6,7] en que los beneficios de la reutilización aumentan cuando el nivel de abstracción es incrementado, y no se reutiliza sólo código sino también especificaciones y diseños. Las principales ventajas que aporta la reutilización son [8]:

1. Reducción de tiempos y costes, aunque inicialmente puede ser costoso crear un producto reutilizable, su uso posterior en varios proyectos reducirá considerablemente el coste y tiempo de estos nuevos proyectos.
2. Incremento de la productividad.
3. Incremento de la fiabilidad, ya que se utilizan productos que han sido probados con anterioridad.
4. Incremento en la facilidad de mantenimiento.
5. Mejora en la documentación.

SIREN incluye un modelo de proceso en espiral, unas plantillas de documentos de requisitos, un repositorio de requisitos reutilizables y una herramienta (SirenTool) que da soporte automatizado al método. Las plantillas de requisitos propuestas están jerarquizadas y se basan en estándares de IEEE. El repositorio contiene requisitos organizados en distintos dominios de aplicación o perfiles, que son llamados catálogos.

Los catálogos y el DRP (Documento de Requisitos del Producto) en uso pueden compartir una jerarquía común de sub-documentos o bien utilizar distintas estructuras. Inicialmente, se establece una jerarquía de plantillas de especificación vacías. Esas plantillas se rellenan con requisitos genéricos (más fáciles de reutilizar), procediendo, en el caso general, de diferentes dominios de aplicación o perfiles determinados. Por ejemplo, actualmente disponemos en SIREN de los siguientes perfiles:

- *Perfil de Protección de Datos Personales (PDP)* [5,9].
- *Perfil de Seguridad de Sistemas de Información* [4].
- *Catálogo de Historias Clínicas*, que se encuentra actualmente en desarrollo.

Los requisitos de los catálogos, además del propio texto, contienen metainformación asociada (atributos con información sobre cada requisito) que enriquecen al requisito. Actualmente se encuentran definidos 18 atributos, entre los que destacan los siguientes: fuente, nivel de seguridad, motivación, prioridad, cumplimiento, etc.

Por otro lado, el Desarrollo de Software Basado en Componentes (DSBC) permite reutilizar piezas de código elaborado previamente (Componentes Software) de manera que se satisfagan los requisitos del sistema. Los beneficios del DSBC, son principalmente:

1. Se alcanza un mayor nivel de reutilización de software.
2. Simplifica las pruebas. Permite que las pruebas se ejecuten probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.

3. Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
4. Mayor calidad. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

En ambas disciplinas, un problema común es la necesidad de poder seleccionar tanto requisitos como componentes con diferentes propósitos.

El objetivo de este trabajo preliminar es el de identificar las principales propuestas existentes en la literatura, de técnicas de selección de componentes que prestan de una manera o de otra, especial atención a los requisitos del sistema. El objetivo final, más allá del ámbito de este trabajo, es hacer una propuesta de método de selección, así como identificar la tecnología y mecanismos necesarios para abordar el problema de la selección de componentes para un sistema en construcción, a partir de los requisitos correspondientes y teniendo en cuenta todas las relaciones, interacciones y posibles contradicciones con otros requisitos del sistema. Para ello, necesitamos identificar cuáles son las principales técnicas para la selección de requisitos, cuáles son aquellas para la selección de componentes, estudiar cómo las técnicas de selección de requisitos pueden beneficiarse del trabajo análogo realizado para la selección de componentes, con el objetivo final de encontrar una técnica adecuada para los fines mencionados.

En la sección 2 mostramos los principales trabajos identificados hasta el momento, la sección 3 muestra, a modo de ejemplo, un posible método de selección, muy básico, en el marco de SIREN, para ilustrar el tipo de procedimientos buscados y, finalmente, la sección 4 muestra las conclusiones y el trabajo inmediato a realizar.

2 Trabajos Relacionados

En los últimos años, se han propuesto varios métodos que tratan con la selección de componentes COTS, como por ejemplo, los métodos SCARLET [10] (que deriva de PORE [11]), OTSO [12], CRE [13] y CARE/SA [14]. En todos estos métodos, un punto clave es la comparación de los requisitos de los usuarios que dirigen el proceso de selección con las capacidades de los componentes COTS evaluados.

Así por ejemplo, en [11] se presenta el método Procurement-Oriented Requirements Engineering (PORE), cuya característica principal es que utiliza un proceso iterativo de captura de requisitos y de selección y evaluación de componentes. PORE tiene un proceso iterativo de selección de componentes basado en el rechazo, es decir, los productos que no satisfagan el núcleo principal de los requisitos del cliente, serán seleccionados para ser rechazados y eliminados de la lista de candidatos en las sucesivas iteraciones. A la misma vez que los productos son selectivamente rechazados, como resultado de la disminución del número de productos candidatos, el número y detalle de los requisitos del cliente se verá incrementado. El resultado es un proceso iterativo, a través del cual el proceso de captura de los requisitos permite la selección de los productos, y el proceso de selección de los productos informa de la captura de requisitos.

Las desventajas de la propuesta de Maiden, son debidas a que el proceso iterativo de obtención de requisitos y evaluación/selección de componentes es muy complejo, ya que por ejemplo los stakeholders pueden definir un elevado número de requisitos, los cuales pueden dar lugar a un número elevado de componentes, y así sucesivamente. Otras limitaciones que tiene este método es que se centra solamente en el proceso de evaluación de los componentes, pero sin llegar a detallar los atributos concretos que se han de medir, o las métricas a ser utilizadas, y que no permite la selección múltiple de componentes, es decir, el caso de que una organización pueda necesitar al mismo tiempo diferentes componentes que cubran diferentes necesidades, componentes que tendrán que ser seleccionados más o menos simultáneamente y que deberán ser integrados adecuadamente.

El método SCARLET [10] (inicialmente conocido como BANCKSEC) adapta a su predecesor PORE al dominio de la banca. Este método define un proceso para la selección de componentes guiado por los requisitos, el cual soporta la obtención de los requisitos y la selección de los componentes de manera simultánea. SCARLET, amplia a su predecesor PORE permitiendo la selección múltiple de componentes y aporta una herramienta de soporte al método.

El método Off-The-Shelf Option (OTSO) desarrollado por Kontio [12] establece un proceso de selección de "paquetes" reutilizables de software, que son denominados por los autores como OTS (Off-The-Shelf), y que incluyen tanto componentes comerciales (COTS) como componentes desarrollados internamente por las propias organizaciones.

El método OTSO proporciona técnicas específicas para la definición de criterios de evaluación, comparando los beneficios y costes de cada una de las alternativas posibles. Estos criterios de evaluación se irán refinando conforme avanza el proceso de selección.

Una aportación interesante que realiza este método es la clasificación en cinco grupos de los factores de influencia en la selección de COTS:

- Requisitos de Usuarios.
- Arquitectura de la aplicación.
- Restricciones y objetivos del proyecto.
- Disponibilidad de productos.
- Infraestructura de la organización.

OTSO, puede ser considerado como el primer método de selección de componentes de amplia aceptación, sin embargo, presenta la limitación de que aunque el método resalta que el problema clave en la selección de componentes COTS es la falta de atención a los requisitos de calidad, el método no proporciona ni sugiere una solución efectiva.

El método COTS-Based Requirements Engineering (CRE) propuesto por Alves [13] tiene como propósito facilitar el proceso de selección y evaluación de los componentes basado en los requisitos, prestando una especial atención al análisis de los requisitos extra-funcionales. La selección de componentes se realiza, al igual que en el método PORE por rechazo, es decir, los componentes candidatos que no cumplen alguno de los requisitos del cliente van siendo rechazados y retirados de la lista de candidatos. Conforme esta lista va decreciendo, es necesario aumentar el número y el detalle de los requisitos. El resultado es un proceso iterativo mediante el

cual el proceso de adquisición de los requisitos permite la selección de productos y, adicionalmente, el propio proceso de selección genera información para la obtención de nuevos requisitos de calidad.

CRE es un método orientado a objetivos, es decir, cada fase se orienta hacia conseguir unos objetivos predefinidos. Para ello, cada fase tiene unas plantillas que incluyen guías y técnicas para la adquisición y/o modelado de requisitos y para la evaluación de productos. El método tiene cuatro fases iterativas: identificación, descripción, evaluación y aceptación.

Esta propuesta, tiene limitaciones similares a las identificadas para el método PORE, donde en aquellos casos en los que haya un número elevado de alternativas COTS y de criterios de evaluación, el proceso de toma de decisiones puede ser muy complejo al haber múltiples posibilidades. Además no se trata en detalle la priorización y negociación de los requisitos, así como la selección múltiple de componentes.

Por último, en cuanto a métodos de selección de componentes, el método COTS-Aware Requirements Engineering and Software Architecting (CARE/SA) propuesto por Chung [14] soporta un "*matching*", clasificación y selección iterativa de componentes COTS, usando una representación de estos componentes como un conjunto de sus requisitos y arquitectura. CARE/SA, puede ser visto como una extensión de los métodos actuales presentados anteriormente, con un enfoque sistemático para emparejar, clasificar y seleccionar componentes COTS. La complejidad de esta propuesta es elevada, lo cual puede afectar la simplicidad de su aplicación práctica.

Por otro lado, un problema bien conocido en IR, es el de la existencia de posibles conflictos entre los requisitos del DRP. Este problema debe ser descubierto, negociado y solucionado antes del proceso de desarrollo del software, ya sea éste basado en componentes o no. De lo contrario, si nosotros usamos por ejemplo, un DSBC los componentes a ensamblar podrían resultar erróneos en la aplicación final. Existen una variedad de técnicas que tratan este problema, como por ejemplo el enfoque WinWin propuesto por Boehm [15], o el trabajo propuesto por Robinson y Volkov [16]. Grünbacher, en su artículo [17], presenta un método que proporciona una manera sistemática para mediar entre los requisitos y la arquitectura del sistema, usando modelos intermedios. Este método, recibe el nombre de CBSP y pretende dar solución al problema que aparece cuando un simple requisito puede relacionarse con múltiples aspectos de la arquitectura, o cuando un simple elemento de ésta (por ejemplo, un componente) tiene numerosas relaciones (no triviales, algunas veces) con varios requisitos. CBSP es aplicado en el contexto de EasyWinWin [18], una propuesta para la negociación de requisitos que soporta la elicitación de requisitos por múltiples stakeholders y que captura los requisitos informalmente, pero de un modo estructurado.

3 SIREN y el Desarrollo de Software Basado en Componentes

Un Componente Software es según [19] un "*paquete coherente de una implementación software que (a) tiene interfaces explicitas y bien definidas para los servicios que proporciona; (b) tiene interfaces explicitas y bien definidas para los*

servicios que espera de otros; y (c) puede ser compuesto con otros componentes, quizá adaptando alguna de sus propiedades, sin modificar los componentes en sí mismos. Como consecuencia de esas propiedades, un componente puede ser desarrollado independientemente, entregado y desplegado como una unidad."

Optar por comprar componentes de terceros en lugar de desarrollarlos, posee ciertas ventajas adicionales, como una reducción del ciclo de desarrollo y un mayor retorno sobre la inversión, por lo que el uso de componentes comerciales comienza a extenderse, refiriéndose en este caso a componentes COTS (Comercial-Off-The-Self), que son vendidos al público en general, mantenidos y actualizados por el propio vendedor, y cuyo código no puede ser modificado por el usuario [20].

En la construcción de aplicaciones siguiendo un DSBC, se siguen las siguientes tareas específicas usando componentes COTS [21]:

1. Búsqueda de componentes que satisfagan los requisitos impuestos por el cliente o por la arquitectura de la aplicación.
2. Evaluación de los componentes candidatos para seleccionar aquellos más apropiados.
3. Adaptación y/o extensión de los componentes seleccionados para que se ajusten a los requisitos anteriores.
4. Integración de dichos componentes para la construcción de la aplicación final.

En el proceso de desarrollo de una Ingeniería del Software Basada en Componentes (ISBC), los requisitos son la parte más importante para cualquier adquisición efectiva de COTS. Los requisitos informan sobre los criterios para evaluar y seleccionar los componentes candidatos, proporcionando incluso criterios de aceptación que usará el cliente para comprobar si el producto entregado, satisface sus necesidades. Otro factor imprescindible, es la documentación de los componentes, siendo preciso contar con especificaciones precisas y completas para poder realizar una selección adecuada. Esta fuente de información sobre el componente (*datasheet*), suele venir dada por el fabricante o vendedor del mismo, aunque según un estudio presentado en [22], esta información suele ser escasa e insuficiente para poder evaluar la calidad del componente.

3.1 Procedimiento de Selección de Requisitos-Componentes

Uno de los procesos críticos del DSBC es la selección de los componentes que formarán parte del producto final y que deben cumplir con los requisitos funcionales definidos por el usuario [23], por lo que nuestro objetivo final será el de diseñar un proceso que nos permita dado un requisito genérico de nuestro catálogo de requisitos SIREN [5], obtener un componente software que satisfaga dicho requisito. La idea es facilitar al equipo de desarrollo la búsqueda y diseño de los componentes que verifiquen los requisitos establecidos en el documento de especificación de requisitos.

Para facilitar esta tarea, los requisitos del catálogo, tendrán un atributo que indique la fuente donde se puede encontrar un componente software (o varios) que, incorporado al sistema que se está desarrollando, haga que dicho requisito se valide. Los valores introducidos en este atributo, dependerán de donde se encuentren los componentes (dirección web, repertorio de componentes, etc.).

Conforme se vayan desarrollando proyectos utilizando los catálogos de requisitos SIREN, éstos se irán viendo enriquecidos no sólo con nuevos requisitos genéricos, sino también con enlaces a las fuentes donde poder encontrar los componentes. En la Figura 1, se puede ver de forma gráfica el proceso de incorporación de nuevas fuentes de componentes al catálogo de requisitos.

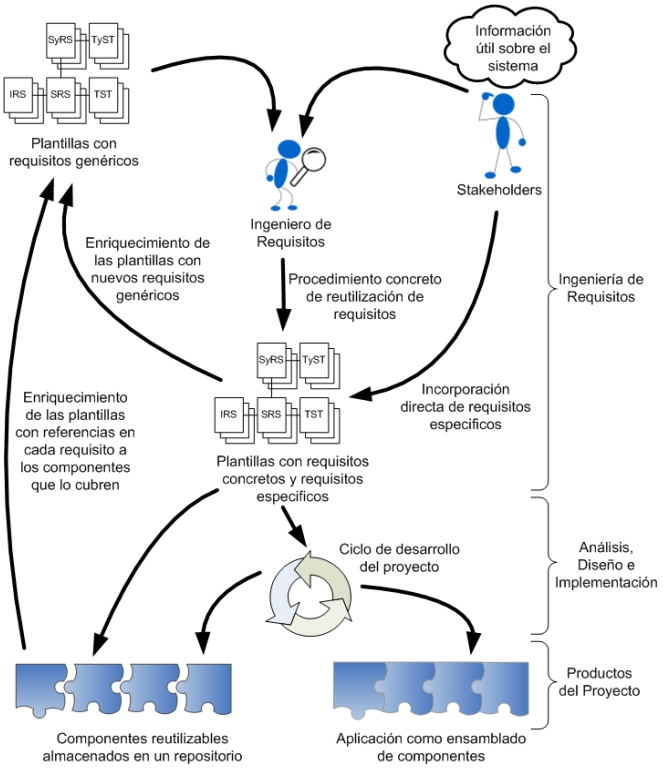


Fig. 1. Proceso iterativo de incorporación de referencias al catálogo de requisitos.

Cuando un requisito es incorporado al documento final de especificación de requisitos, puede ocurrir que contenga referencias a componentes que lo validen o que no contenga tales referencias.

En el caso de que no haya registrado ningún componente, el equipo de desarrollo sólo tiene una opción, comenzar a desarrollar el proyecto teniendo en cuenta que uno de los componentes desarrollados deberá validar ese requisito.

Por otro lado, también puede ocurrir que el requisito no se corresponda exactamente con las necesidades del proyecto y sea necesaria su adaptación para que tal cosa ocurra. En este caso los componentes que lo validaban es posible que ya no lo hagan, o que lo hagan parcialmente y por tanto también sea necesario un rediseño de los mismos.

Por tanto, la casuística que nos podemos encontrar es la que se muestra en la Tabla 1, la cual nos lleva a tres situaciones diferentes.

SITUACION	SOLUCION
No se registran referencias a componentes que validen requisitos.	El equipo de desarrollo parte de cero y debe completar todo el proceso de desarrollo.
Existen referencias a componentes que validan el requisito. El requisito se incorpora tal cual.	El equipo de desarrollo sólo tiene que ensamblar el componente en su aplicación.
Existen referencias a componentes que validan el requisito. El requisito ha sido adaptado.	El equipo de desarrollo debe hacer reingeniería para poder reutilizar el componente.

Tabla 1. Incorporación de un requisito al Documento de Requisitos del Producto.

Por ejemplo, supongamos que nos encontramos en la situación 2, donde encontramos uno o varios componentes software que cumplen con cada uno de los requisitos (o al menos, con los más críticos) incluidos en el Documento de Requisitos del Producto (DRP), que han sido extraídos de las plantillas de requisitos genéricos del catálogo de PDP. Cada uno de estos requisitos, tienen una lista inicial de enlaces a componentes, los cuales no implican que sean todos válidos para el actual proyecto, pues sólo podemos afirmar que con estos componentes iniciales asociados al requisito la funcionalidad de éste, puede ser implementada.

Para seleccionar el componente adecuado, tenemos que tener en cuenta el resto de requisitos relacionados, en particular, aquellos requisitos no funcionales que impongan algún tipo de restricción en la aplicación final (lenguaje de programación, sistema operativo, memoria, etc.).

Para ayudarnos en esta tarea, construimos una matriz de selección (Cuadro 2) para cada requisito, la cual contendrá una columna para cada uno de los requisitos relacionados con éste, y una fila con cada uno de los componentes identificados. El criterio para decidir qué requisitos se tienen en cuenta y cuáles no, se deja a elección del analista, aunque podrían aplicarse técnicas más sofisticadas como métricas aplicadas a los atributos de los requisitos [24] o generación automática de casos de prueba para la evaluación del producto [25].

Consideramos en nuestra propuesta inicial, que una matriz de selección será una herramienta conceptual muy útil e intuitiva para realizar la selección de los componentes. Sin embargo, esta matriz está en su versión más simple, por lo que se deben conseguir métodos más realistas. Inicialmente, proponemos las siguientes mejoras:

- Un requisito puede ser cumplido por un componente en un cierto grado. Dependiendo del requisito y de las características del componente, es posible puntuar el cumplimiento entre requisito y componente. En este caso, las entradas de la Matriz de Selección tendrán una puntuación en vez de un simple valor binario. Esta puntuación se puede calcular usando la Técnica de Análisis de Características propuesta por DESMET[9] [26], o cualquier otra técnica.
- A un requisito se le pueden asignar pesos usando alguna de las Técnicas de Decisión Multicriterio existentes [27], de esta manera se sabe como un requisito particular afecta a la puntuación final de un componente.

[9] DESMET es un método para la evaluación de herramientas y métodos de IS.

Con estas mejoras, proponemos inicialmente usar la siguiente técnica: sean s_1, s_2,…, s_n las puntuaciones de los requisitos r_1, r_2,…, r_n obtenidas usando la Técnica de Análisis de Características para el componente c_1; Sean w_1, w_2,…, w_n los pesos para cada requisito, obtenidos usando, por ejemplo, la Técnica de Decisión Multicriterio conocida como Proceso Analítico Jerárquico [28]. La puntuación final para c_1 será:

$$\text{score}(c_1) = w_1s_1 + w_2s_2 + \ldots + w_ns_n \quad (2)$$

El resultado del proceso de selección será un único componente, o un conjunto de componentes finalistas, los cuales están en la misma condición con respecto a los requisitos indicados en el DRP. En este caso, la decisión final dependerá de las preferencias del analista o del diseñador.

Otro aspecto a tener en cuenta en un proceso de selección de componentes basado en requisitos es que otros requisitos en el DRP final, no sólo los técnicos, pueden imponer más limitaciones (en precio, licencias disponibles, cualidades del equipo de desarrollo) hasta el punto de que el procedimiento de selección termine en una lista vacía de componentes. Para hacer una selección de calidad, es de gran ayuda tener para cada componente una especificación precisa de sus características (*datasheet*).

A continuación, mostramos un ejemplo sencillo de nuestra primera aproximación del método de selección propuesto, utilizando los requisitos contenidos en el Perfil SIREN de Protección de Datos Personales (PDP) [5,9].

Supongamos que un equipo de desarrolladores está iniciando un proyecto en el que la aplicación a desarrollar tiene que tratar con datos de carácter personal, por lo que decide utilizar el Perfil SIREN PDP. Suponemos también, que nos encontramos ante la segunda de las situaciones posibles de las presentadas en la Tabla 1.

El Requisito A, tiene la siguiente descripción textual: *La aplicación usará un [algoritmo de cifrado] para encriptar los datos que vayan a ser transmitidos a través de redes de telecomunicaciones.*

Es evidente, que pueden haber muchos componentes que nos ofrezcan capacidades de cifrado de datos. El Requisito A, tendrá en su atributo "*componentes*" una lista con todos aquellos componentes que validan este requisito.

Además de cumplir con este requisito, el equipo de desarrolladores también tiene que cumplir varios requisitos relacionados con la tecnología a utilizar, el lenguaje de programación, sistema operativo, etc. De esta manera, tenemos un Requisito B que dice: *La aplicación deberá ser implementada utilizando el entorno C++Builder o Delphi.* Por otro lado, el Requisito C tiene la siguiente descripción textual: *La aplicación se debe poder ejecutar en cualquiera de los siguientes sistemas operativos: Windows98 y WindowsNT.*

Estos requisitos nos limitan bastante la tecnología de componentes que podemos utilizar para cubrir las necesidades de cifrado nuestra aplicación, quedándose en este caso la lista de componentes reducida a dos candidatos, como podemos ver en la Tabla 2.

Componente	**Requisito B**	**Requisito C**	**...**
ABCEncrypt	X	X	...
AspEncrypt		X	...
Crytocx	X	X	...
EDSSimpleEncrypt/Decryp		X	...

EnergyEncryptionCom		X	...
NCRYPT		X	...
PowertTCP SecureTool	X		...
Seal-It! V 2.0		X	...
Visual Soft Crypt		X	...

Tabla 2. Ejemplo de Matriz de Selección para el requisito A.

Este enfoque resulta bastante simple en la práctica, teniendo, entre otras, las siguientes limitaciones:

1. La existencia de un repositorio estable de componentes, no suele ser una práctica habitual. Este hecho, todavía se acentúa más en los componentes COTS, pues un componente comercial suele tener entre uno y cinco años de vida útil, con numerosas versiones intermedias. Esto nos obliga, a pensar en un enfoque de búsqueda dinámica de componentes.
2. La existencia de un componente que valide o implemente totalmente un requisito dado no suele ser lo habitual. Normalmente encontraremos componentes que implementen parcialmente a un requisito o también que un conjunto de componentes (no uno sólo) validen a un requisito.
3. Se debería hacer distinción, de alguna manera, entre requisitos funcionales y no funcionales, pues el tratamiento que deberían recibir será distinto. Dentro de los requisitos no funcionales, creemos también que se deberían establecer varias categorías de tratamiento (por ejemplo, los que pueden medirse de forma objetiva, los que pueden medirse de forma subjetiva, etc.).

4 Conclusiones y Trabajo Futuro

Las conclusiones que hemos obtenido en este trabajo preliminar, son:

1. Existen diversas propuestas en la literatura de métodos de selección de componentes basados en el rechazo, según los requisitos identificados para el sistema.
2. Las técnicas de selección y negociación de requisitos, no están contempladas directamente en ninguna de las propuestas encontradas.
3. Nuestra propuesta inicial, al estar basada en un método de IR, como es SIREN, permitiría solucionar de una manera más eficiente el problema de la selección de componentes basada en requisitos, ya que desde el primer momento se garantiza la existencia de unos documentos de requisitos de calidad, en teoría carentes de inconsistencias o conflictos entre los requisitos del DRP.

Como trabajo futuro, y además de intentar resolver las limitaciones planteadas al final de la sección anterior, nos planteamos estudiar las técnicas basadas en el uso de redes bayesianas, que están apareciendo recientemente para la creación e implementación de modelos e indicadores de calidad, para analizar si son adecuadas para el proceso de selección de componentes guiado por los requisitos que queremos

obtener. En [29] se presentan las redes bayesianas como instrumento útil para tratar determinados aspectos de la medición de la calidad, y en [30] se muestra una aplicación de las redes bayesianas para medir la calidad de ciertos aspectos de las aplicaciones web.

Referencias

1. Glass, R.: Software Runaways: Monumental Disasters. (1998)
2. Smith, J.: Troubled IT Projects prevention and turnaround. (2001)
3. Glass, R.: Software Engineering: Facts and Fallacies. (2002)
4. Toval, A., Nicolás, J., Moros, B., Garcia, F.: Requirements Reuse for Improving Information Systems Security: A Practitioner's Approach. RE J. 6 (2002) 205-219
5. Toval, A., Olmos, A., Piattini, M.: Legal Requirements Reuse: A Critical Success Factor for Requirements Quality and Personal Data Protection. In: RE, IEEE Computer Society (2002) 95-103
6. Cybulski, J.L., Reed, K.: Requirements Classification and Reuse: Crossing Domain Boundaries. In Frakes, W.B., ed.: ICSR. Volume 1844 of Lecture Notes in Computer Science., Springer (2000) 190-210
7. Glass, R.: Software Engineering (6th edition). (2001)
8. Toval, A., Nicolás, J., Moros, B.: Un proceso de Ingeniería de Requisitos Basado en Reutilización. Actas de I Jornadas de Ingeniería de Requisitos Aplicada (JIRA). (2001)
9. Martínez, M.A., Lasheras, J., Toval, A., Piattini, M.: An Audit Method of Personal Data Based on Requirements Engineering. In: Proceedings of the 4th International Workshop on Security in Information Systems, (WOSIS'06), In conjunction with ICEIS'06, Paphos, Cyprus. (2006) 217-231
10. Maiden, N.A.M., Croce, V., Kim, H., Sajeva, G., Topuzidou, S.: SCARLET: Integrated process and tool support for selecting software components. In Cechich, A., Piattini, M., Vallecillo, A., eds.: Component-Based Software Quality – Methods and Techniques. Volume 2693 of LNCS., Springer (2003) 85-98
11. Ncube, C., Maiden, N.: PORE: Procurement-oriented requirements engineering method for the component-based systems engineering development paradigm. In: Proceedings of the 2nd International Workshop on Component Based Software Engineering (CBSE) (in conjunction with ICSE'99), Los Angeles, USA. (1999)
12. Kontio, J.: A case study in applying a systematic method for COTS selection. In: 18th International Conference on Software Engineering, Springer (1996) 201-209
13. Alves, C., Castro, J.: Um método baseado em requisitos para seleçao de cots. In: Proceeding of the 4th Iberoamerican Workshop on Software Engineering and Software Environment (IDEAS'01) San Jose, Costa Rica. (2001)
14. Chung, L., Cooper, K.: Cots-aware requirements engineering and software architecting. In: Software Engineering Research and Practice. (2004) 57-63
15. Boehm, B.W., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R.J.: Using the winwin spiral model: A case study. IEEE Computer 31 (1998) 33-44
16. Robinson, W.N., Volkov, V.: Supporting the negotiation life cycle. Commun. ACM 41 (1998) 95-102
17. Grünbacher, P., Egyed, A., Medvidovic, N.: Reconciling software requirements and architectures with intermediate models. SoSyM 3 (2004) 235-253
18. Briggs, R.O., Grünbacher, P.: Easywinwin: Managing complexity in requirements negotiation with GSS. In: HICSS. (2002) 21
19. Francis, D., Cameron, A.: Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach. (1999)

20. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. (1999)
21. Iribarne, L., Vallecillo, A.: Construcción de aplicaciones software a partir de componentes COTS. Actas de VII Jornadas de Ingeniería del Software y Bases de Datos (JISBD). (2002)
22. Bertoa, M.F., Troya, J.M., Vallecillo, A.: A survey on the quality information provided by software component vendors. In: Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'03). Darmstadt, Germany. (2003) 25-30
23. Bertoa, M.F., Troya, J.M., Vallecillo, A.: Measuring the usability of software components. Journal of Systems and Software 79 (2006) 427-439
24. Piattini, M., Cechich, A., Vallecillo, A., eds.: Component-Based Software Quality - Methods and Techniques. Volume 2693 of LNCS., Springer (2003)
25. Maiden, N.A., Ncube, C.: Acquiring COTS software selection requirements. IEEE Software 15 (1998) 46-56
26. Kitchenham, B.A., Jones, L.: Evaluating software engineering methods and tools. The influence of human factors. ACM SIGSOFT Software Engineering Notes 22 (1997) 13-15
27. Triantaphyllou, E.: Multi-Criteria Decision Making Methods. (2000)
28. Vargas, L.G., Saaty, T.L.: Models, Methods, Concepts and Applications of the Analytic Hierarchy Process. (2001)
29. Fenton, N.E., Neil, M.: Software metrics: roadmap. In: ICSE - Future of SE Track. (2000) 357-370
30. Malak, G., Sahraoui, H., Badri, L., Badri, M.: A proposal of a probabilistic framework for web-based application quality. In: Proceed. of the 10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'06). (2006) 91-100

Esquemas de control PID para una maqueta hidráulica

Francisco Rodríguez[1], Felipe Zottola[1], Sandra Alonso[2], Lucía Pintos[2]

[1] Universidad Carlos III de Madrid, Escuela Politécnica Superior, Avda. Universidad 30, 28911 Leganés, Madrid
{urbano, fzdiz}@ing.uc3m.es
[2] EADS-CASA, Escuela Politécnica Superior, Avda. de John Lennon s/n, 28906 Getafe, Madrid
{Sandra.Alonso.External, Lucia.Pintos.External}@casa.eads.net

Abstract. En el marco del proyecto CARATE (Controlador de arquitectura reconfigurable para aplicaciones de teleoperación) se están desarrollando componentes de software para diversas estrategias de control de sistemas con accionamientos hidráulicos. En este trabajo se describe la arquitectura más sencilla basada en control PID y se muestran los resultados de los experimentos realizados sobre la maqueta real *HYMATIC* desarrollada en la Universidad Carlos III de Madrid.

Keywords: Arquitecturas software, arquitecturas de control, hidráulica, control PID.

1 Introducción

El control de un sistema complejo requiere disponer de una amplia gama de controladores de manera que pueda seleccionarse el más adecuado en cada caso. En este artículo se exponen los controladores PID básicos para el control independiente de un solo eje (el segundo) de la maqueta hidráulica HYMATIC, que puede verse en la figura 1.

Este tipo de controladores son utilizados hoy en día en más del 95% de los procesos industriales [1]. La implantación de un sistema de control PID en la maqueta real, tal y como se muestra en la figura 2, requiere por un lado el ajuste de los parámetros, que en este trabajo se ha realizado mediante técnicas de ajuste experimentales, y por otro lado la selección de un tipo de PID concreto, ya que existe varias estructuras distintas que conducen a comportamientos diferentes [2][3].

En las siguientes secciones se comenzará describiendo un PID básico, para después ir analizando sus posibles variantes y comparando los resultados obtenidos con cada una de ellas.

Fig. 1. Maqueta hidráulica HYMATIC.

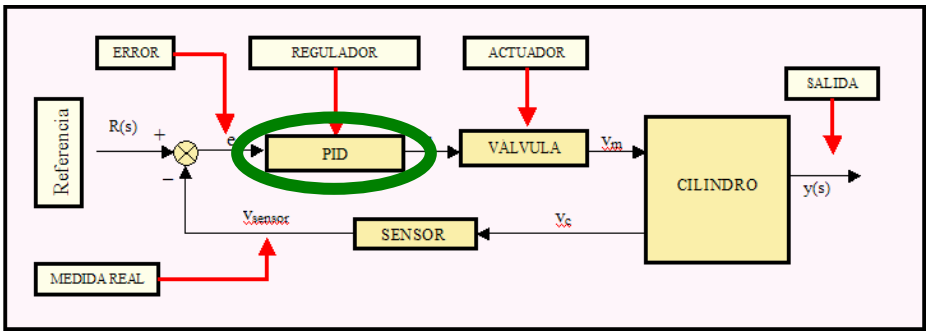


Fig. 2. Esquema de control de un solo cilindro.

2 Controlador PID estándar

Desde una perspectiva general, un controlador PID [4] recibe como entrada una señal de error *e(t)*, que se obtiene previamente comparando una señal de referencia deseada por el usuario (en este caso la posición del robot) con la posición real obtenida utilizando los sensores. Cómo resultado proporciona una señal de actuación *u(t)* que en este caso estará relacionada con la velocidad de movimiento del robot. En general los algoritmos de control PID tienen tres partes:

- El término básico del PID es el **P**, que origina una actuación de control correctiva proporcional al error.
- El término **I** proporciona la corrección proporcional a la integral del error, lo que asegura que, en última instancia, se aplicará suficiente acción de control para reducir el error a cero.

- El término **D** capacita de predictiva a la actuación, generando una acción de control proporcional a la velocidad del error. Tiende a dar más estabilidad al sistema pero suele generar grandes valores en la señal de control.

El controlador PID proporciona una señal de control según la ecuación 1, y su función de transferencia viene dada por la ecuación 2.

$$u(t) = K_P \cdot e(t) + K_D \cdot \frac{de(t)}{dt} + K_I \cdot \int_0^t e(t)dt \tag{1}$$

$$Gc(s) = K_P + \frac{K_I}{s} + K_D \cdot s \tag{2}$$

Este tipo de controlador también se conoce como PID no interactivo o PID estándar. Los resultados de algunos experimentos realizados con este controlador se muestran a continuación en la figura 3, donde pueden verse las posiciones en centímetros alcanzadas en función del tiempo.

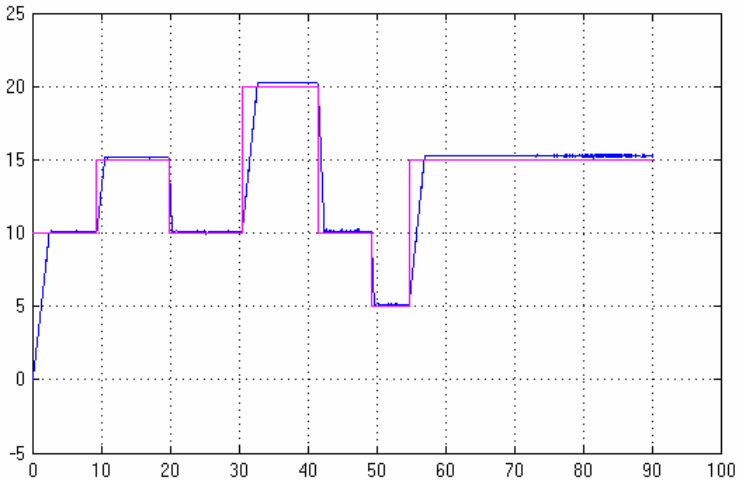


Fig. 3. Resultados del controlador PID estándar.

3 PID interactivo

El PID non-interactive, también denominado ***Algoritmo ideal***, raramente se usa en la práctica. Se trata más de un PID "de libro". Esto se debe a que se obtienen mejores resultados con los PID modificados y su uso es más frecuente en la industria.

El PID interactivo, recibe este nombre por la interacción que existe entre sus partes integral y derivativa, es decir, $\mathbf{T_D}$ **'** influye en la parte integral y $\mathbf{T_I}$**'** lo hace en la derivativa. En los libros puede encontrarse como ***PID Serie***.

El diagrama de bloques de este controlador se muestra en la figura 4, y su función de transferencia en ecuación 3:

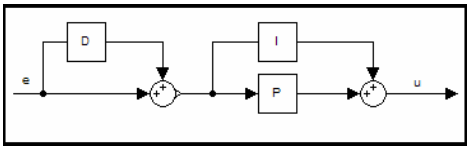


Fig. 4. Diagrama de bloques del PID interactivo.

$$Gc'(s) = K_P' \cdot \left(1 + \frac{1}{s \cdot T'i}\right) \cdot \left(1 + s \cdot T'd\right) \tag{3}$$

La sintonización manual de los parámetros de este controlador es más sencilla que en el caso no interactivo, utilizándose los siguientes parámetros obtenidos de manera experimental:

$$\begin{aligned} K_P' &= 1.999941 \\ T_I' &= 0.0216993 \\ T_D' &= 0.005409999 \end{aligned} \tag{4}$$

Antes de presentar el esquema, hay que hacer constar que estos dos controladores, el *no interactivo y el interactivo*, son exactamente iguales si no contienen las tres acciones de control simultáneamente. Suprimiendo la acción integral o la diferencial, ambos serán equivalentes.

Los resultados obtenidos al aplicar al sistema escalones de diferentes magnitudes, y durante un tiempo total de ejecución de 100 s se expone en la figura 5.

Respecto a los resultados obtenidos con el PID estándar, mostrado en las figura 3, se observa cierta mejoría en el control. El tiempo en alcanzar el estado estable y el error son menores.

Se sigue viendo que el escalón positivo conlleva más tiempo a la hora de alcanzar la posición indicada. Sin embargo, también proporciona un error en estado estable menor que si se aplica a la planta un escalón negativo. Esto es debido a que la acción de la gravedad hace que el sistema sea más rápido cuando se mueve en sentido descendente. Por otro lado, en el movimiento ascendente el sistema se comporta de una manera mucho más amortiguada, favoreciendo que no se produzcan oscilaciones en torno al valor de equilibrio.

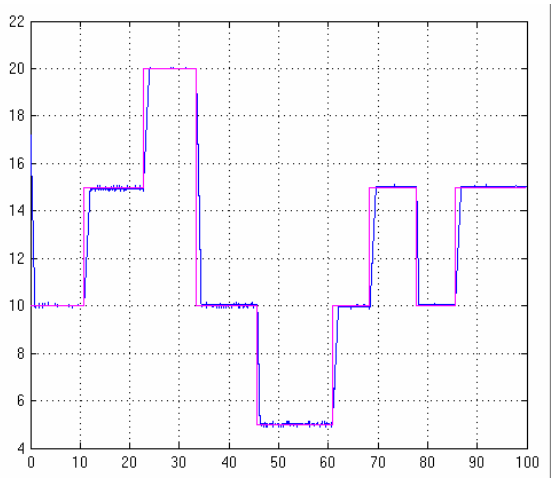


Fig. 5. Resultados del controlador PID interactivo.

4 PID con limitación de la ganancia derivativa

La acción derivativa puede ocasionar problemas si existen medidas de ruido de alta frecuencia, y puede ocasionar grandes amplitudes de la señal de control. Para entradas escalón, como es el caso, por la presencia del término derivativo en la acción de control, la variable de control '**u**' contendrá una función impulso (una delta). Para evitar este efecto en los PID llevados a la práctica, se limita la ganancia del término derivativo. Así, la acción derivativa se implementa de la siguiente forma:

$$D = -\frac{T_D}{N} \cdot \frac{dD}{dt} - K_P \cdot T_D \cdot \frac{dy}{dt} \tag{5}$$

En el diagrama de bloques, este término debe aparecer linealizado. Entonces, la función de transferencia resulta:

$$D = \frac{s \cdot T_D}{1 + s \cdot T_D \cdot \gamma} \tag{6}$$

Esta modificación sobre el término derivativo puede ser interpretada como un derivador ideal filtrado con un sistema de primer orden que posee una constante de tiempo 'γ', denominada constante de tiempo derivativa. Gracias a la inclusión de un polo se evita el usar acciones de control grandes en respuesta a errores de control de alta frecuencia, tales como errores inducidos por cambios del punto de consigna o mediciones de ruido. Esta constante se define:

$$\gamma = \frac{T_D}{N} \tag{7}$$

El parámetro **γ**, normalmente es elegido tal que:

$$0.1 \leq \gamma \leq 0.2 \tag{8}$$

Cuanto más pequeña es **γ**, mejor es la aproximación entre el término '*derivativo filtrado*' de la ecuación (6) y el '*derivativo normal*', $\mathbf{T_D \cdot s}$.

Con todo esto, aunque la entrada de referencia sea un escalón la variable de control no contendrá una función impulso, sino una función en forma de pulso estrecho. Tal fenómeno se denomina ***patada en el punto de consigna***.

Hay que tener en cuenta, sin embargo, que aunque esta aproximación actúa como un derivador para componentes de bajas frecuencias, la ganancia está limitada por $\boldsymbol{K_P \cdot N}$, lo que va a amplificar las señales de ruido a altas frecuencias en esa proporción. Es por esta razón, que tendrá que elegirse con cuidado el parámetro **N**. Típicamente, su valor será **10.** Este esquema de limitación de la ganancia derivativa se ha incluido en el esquema de control del siguiente apartado, por lo que los resultados experimentales de la sección 5 incluyen esta mejora.

5 PID con ponderación del punto de consigna

Se obtiene una estructura más flexible del controlador tratando el punto de consigna (setpoint) o referencia, y la salida resultante separadamente. Un PID de esta forma viene dado por la siguiente ecuación:

$$u(t) = K_P \cdot \left(e_P + \frac{1}{T_I} \int_0^t e(\tau) \cdot d\tau + T_D \cdot \frac{de_d}{dt} \right) \tag{9}$$

donde el error en la parte proporcional, en la parte integral y en la derivativa se definen del siguiente modo:

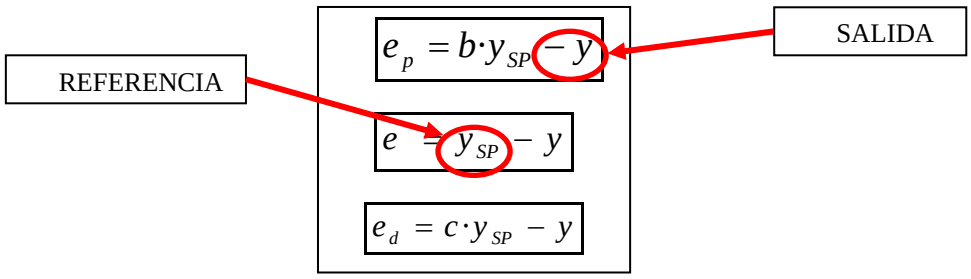


Fig. 6. Cálculo de los errores con ponderación del punto de consigna.

Los controladores obtenidos para diferentes valores de ‘b’ y ‘c’ responderán de la misma manera ante perturbaciones de carga y la presencia de ruido. La respuesta para cambios en el punto de consigna dependerá, por tanto, de los valores que tomen estos coeficientes.

Normalmente se toma ‘c’ nulo, para evitar grandes transitorios en la señal de control debidos a cambios repentinos en la referencia.

Según ‘b’ sea cero, ‘c’, o ambos, se originan dos configuraciones de PID y que se describen a continuación:

5.1 Controlador I-PD

Para formar este controlador se anulan ambos coeficientes. Considerando que la entrada de referencia es un escalón, el control PID estándar implica una función escalón en la señal manipulada. En muchas ocasiones, tal cambio escalón puede no resultar conveniente. El comportamiento puede mejorarse trasladando las acciones proporcional y derivativa al lazo de realimentación con el fin de que sólo afecten a esta señal de realimentación. Esto es lo que hace el control ***I-PD***. La señal manipulada viene regida por la siguiente ecuación:

$$U(s) = K_P \cdot \frac{1}{T_I s} \cdot Y_{SP}(s) - K_P \left(1 + \frac{1}{T_I s} + T_D s \right) \cdot Y(s) \qquad (10)$$

Obsérvese que la entrada de referencia, ***‘$y_{SP}(s)$’***, sólo aparece en la parte de control integral, luego para realizar este control, es necesario tener esta acción de control integral para una operación adecuada del sistema de control.

Los resultados obtenidos al aplicarle diversas entradas escalón a la maqueta, se muestran a continuación:

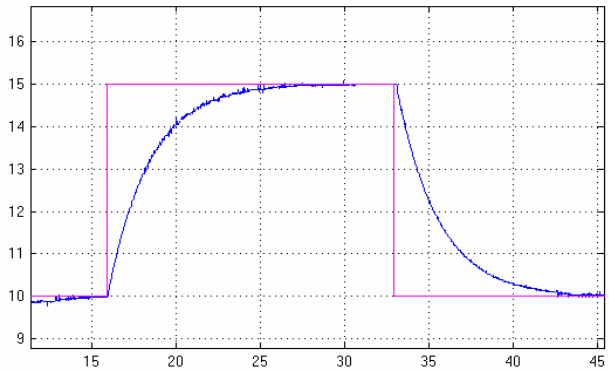


Fig. 7. Resultados del controlador I-PD.

5.2 Controlador PI-D

Antes se hizo mención al fenómeno de la patada de consigna. Para evitar dicha manifestación, puede operarse la acción derivativa sólo en el camino de realimentación a fin de que la derivación ocurra exclusivamente en la señal de realimentación y no en la señal de referencia. El esquema así dispuesto es el control PI-D.

Este controlador puede formarse a partir de la ecuación (9) anulando el coeficiente 'c' y dándole a 'b' el valor de la unidad. De este modo, la ecuación por la que se rige este controlador viene dado por la ecuación 11.

$$U(s) = K_P\left(1+\frac{1}{sT_I}\right)\cdot Y_{SP}(s) - K_P\left(1+\frac{1}{sT_I}+T_D s\right)\cdot Y(s) \qquad (11)$$

Se comenzará con el esquema simple de este controlador, sin ningún tipo de mejora ni limitación. Los resultados se exponen en la figura 8.

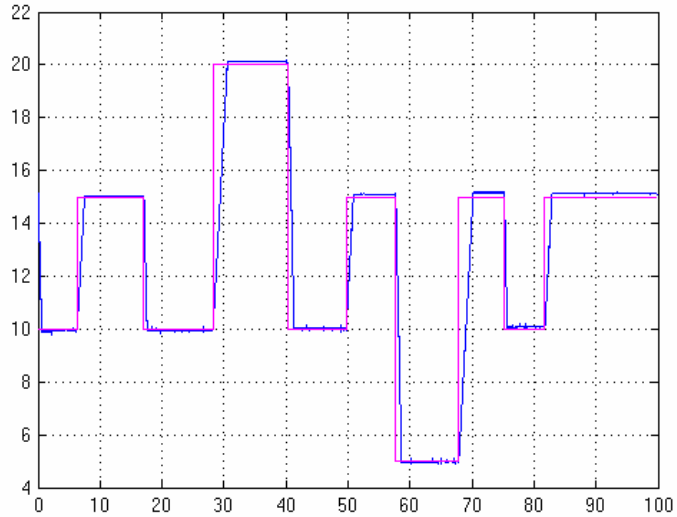


Fig. 8. Resultados del controlador PI-D.

Como se venía anunciando, aquí sí puede verse la mala influencia de la gravedad. Puede verse que para los escalones negativos, el control es muy bueno. El tiempo de bajada y establecimiento es adecuado y el error es mínimo. Sin embargo, cuando se trata de los positivos la cosa cambia. El que el tiempo en que se establece la posición estable aumente es normal, por la fuerza que se opone y que es la gravedad como ya se vio. Cuando se trata del error, se ve que en los casos donde la subida es mayor, esto es que el escalón es de amplitud más grande, este es más grande también y crece más aun cuando aparece un escalón positivo nuevo. Este efecto incremental también se debe a la gravedad [5][6][7] y a la fricción [8] presente en el sistema.

Es asimismo esta fricción la causante de estas desviaciones que sufre la variable de salida en diferentes puntos de control. Para evitar estos errores en las subidas habría que ajustar los parámetros del regulador en cada uno de estos puntos.

6 Conclusiones

En este artículo se ha realizado una comparación de distintos algoritmos de control de un solo cilindro de la maqueta HYMATIC. Se puede ver que incluso para este caso tan simple las alternativas de control son varias, y que debe seleccionarse la más adecuada en cada caso según la experiencia del operador. El uso de componentes de software reutilizables permite la rápida implantación y pruebas de las diversas alternativas de una manera flexible. También se observa que los resultados obtenidos son satisfactorios, obteniéndose precisiones aceptables para las tareas a desarrollar por el sistema.

Aunque todos los algoritmos de control descritos en este artículo se han ajustado de manera experimental, también se han desarrollado en el marco del proyecto CARATE otros métodos más complejos basados en modelos matemáticos [9][10] de los componentes y en la identificación del sistema real [11][12].

Agradecimientos. Estos trabajos han sido financiados por el Ministerio de Educación y Ciencia español en el marco del proyecto TIC2003-07804-C05-04.

Bibliografía

1. Astrom, K. and Hagglund, T., “PID Controllers: Theory, Design and Tuning”, 1995, Instrument Society of America.
2. Jelali, Mohieddine and Kroll, Andreas, “Hydraulic Servo-systems: Modelling, Identification and Control”, 2003, Springer Verlag.
3. Merrit, Herbert E., “Hydraulic control systems”, 1967, John Wiley and Son
4. Ogata, Katsuhiko, “Ingeniería de Control Moderna”, 2003, Prentice Hall
5. Spong, Mark W., Lewis, F.L., Abdallah, C.T., “Robot Control: Dynamics, Motion Planning and Analysis”, 1993, IEEE Press.
6. Lewis, F.L., Abdallah, C.T., Dawson, D.M., “Control of Robot Manipulators”, 1993, Macmillan Publishing Company
7. Sciavicco, Lorenzo and Siciliano, Bruno, “Modeling and control of robot manipulators”, 1996, McGraw-Hill
8. Slotine, Jean-Jacques E. ; Li, Weiping, “Applied Nonlinear control”, 1991, Prentice Hall
9. Albertos, P. and Sala, A., “Multivariable Control systems”, 2004, Springer-Verlag
10. An, Chae H., Atkeson, Christopher G. and Hollerbach, John M., “Model-Based Control of a Robot Manipulator”, 1988, The MIT Press
11. Aguado Behar, Alberto; Martínez Iranzo, Miguel, “Identificación y Control Adaptativo”, 2003, Prentice Hall
12. Ljung, Lennart, “System Identification: Theory for the User”, 1987, Prentice Hall

Ambient-PRISMA: Ambients in Distributed and Mobile Aspect-Oriented Software Architectures

Nour Ali, Isidro Ramos

Department of Information Systems and Computation
Polytechnic University of Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
{nourali, iramos}@dsic.upv.es

Abstract. Nowadays, distributed and mobile systems are acquiring greater importance and becoming more widely used to support ubiquitous computing. However, developing systems of this kind is a difficult task. Instead of concentrating on how problems should be solved developers must worry about implementation details. Ambient Calculus is a formalism that provides primitives to describe mobile systems in an abstract way. Aspect-oriented software development and software architectures promise to achieve reusability, maintenance and adaptability, which are all essential for the development of distributed systems. In this paper, we present how a platform-independent model called Ambient-PRISMA combines both Ambient Calculus and Aspect-Oriented Software Architecture for the description of distributed and mobile systems. An implementation of Ambient-PRISMA has been performed in .Net for supporting Ambient-PRISMA code generation.

Keywords: Software Architecture, distributed and mobile systems, AOSD

1. Introduction

Nowadays, software systems have to exploit the distributed nature of systems. Distributed software systems have to take into account particularties; for example complex structures, new non-functional requirements such as fault tolerance and security, and dynamic adaptation. As a result, software development processes must be able to support the distributed, mobile and ubiquitous nature of software systems.

The development of these software systems is a difficult task. Currently, decisions about these characteristics are usually postponed to late stages of the software life cycle (design and implementation). As a result, there is a loss of traceability, and the system is subject to a specific technological platform. Developers of distributed systems also have to spend more time programming than solving and considering distribution problems. This can be solved by considering distribution and mobility characteristics at a high abstract level, specifying them in a technology independent way, and at early stages of the software life-cycle.

Component Based Software Development (CBSD) [1] and Aspect-Oriented Software Development (AOSD) [2] are two techniques that promise to achieve

reusability, flexibility, and maintainability through the software development process. CBSD proposes to construct the system by connecting entities that provide and require services. AOSD allows the separation of concerns by modularizing crosscutting concerns in a separate entity called an aspect.

In this paper, we present a metamodel called Ambient-PRISMA [3] [4] [5] that allows the definition of distribution and mobility characteristics in a technology-independent way. Ambient-PRISMA allows the description of these characteristics by using AOSD and CBSD techniques as it extends the PRISMA approach [6] with Ambient Calculus (AC) [7] concepts. PRISMA integrates CBSD and AOSD to specify software architectures. AC is a formalism that describes distribution and mobility properties in an abstract way. Also, Ambient-PRISMA has been implemented in .Net in order to support the generation of code from models of the Ambient-PRISMA metamodel.

The paper is structured as follows: Section 2 presents Ambient-PRISMA and how a mobile agent case study can be described in Ambient-PRISMA. Section 3, presents how concepts of Ambient-PRISMA are implemented in .Net technology. Finally, Section 4 presents conclusions and future works.

2. Ambient-PRISMA

The Ambient-PRISMA metamodel extends the PRISMA metamodel [5] by introducing concepts of the ambient calculus. The result of the Ambient-PRISMA metamodel is to allow the description of distributed and mobile system software architectures by using both the aspect-oriented and the component-based software development techniques.

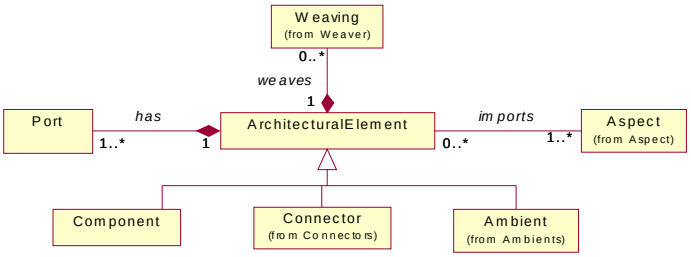


Fig. 1. Ambient as an aspect-oriented, component-based architectural element

The Ambient-PRISMA metamodel introduces the ambient concept as an architectural element (see Figure 1), which is responsible for providing mobility features to distributed architectural elements. As a result, an ambient inherits the CBSD and AOSD characteristics of PRISMA (as do other architectural elements of PRISMA such as components and connectors). The CBSD characteristics describe an ambient as a black box where it communicates with others by using ports that send and receive invocations of services. To enable the communication among architectural elements, channels called attachments are defined. Each attachment is defined by attaching two ports of different architectural elements.

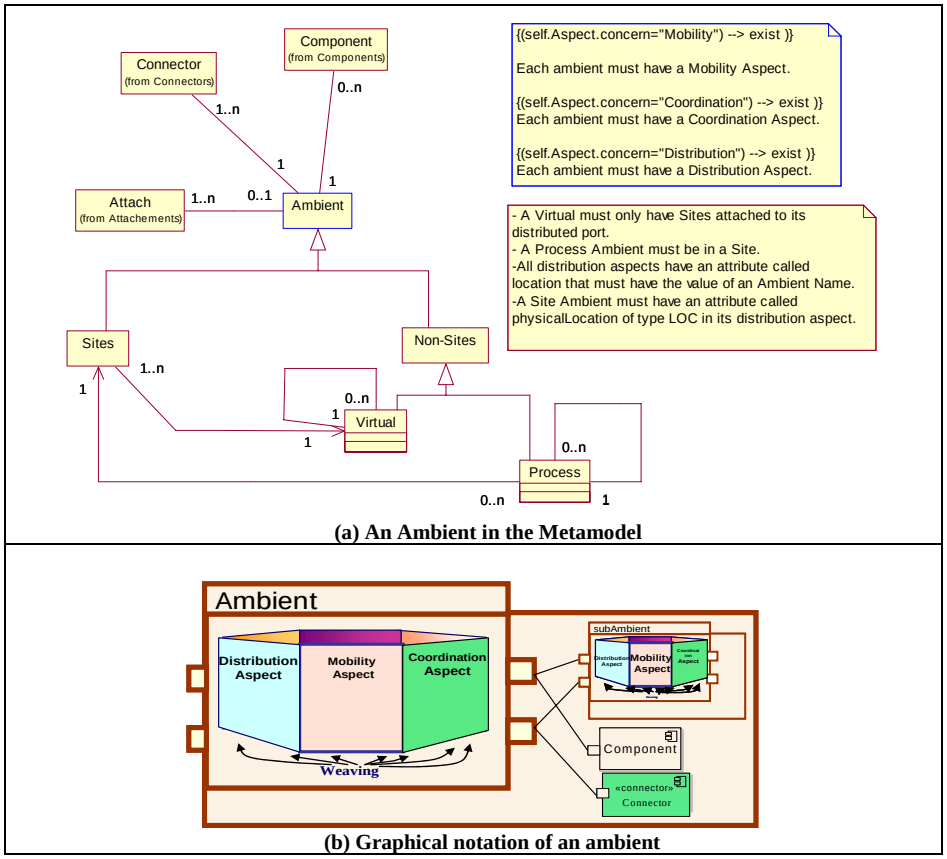


Fig. 2. The Ambient package in the Metamodel

In Ambient-PRISMA, ambients are architectural elements that represent the places where components, connectors and other ambients are located. Therefore, in the metamodel, the ambient is associated with the other architectural elements (see Figure 2(a)). For example, in Figure 2(b) the *Component* and *Connector* are located in the same ambient. Also, an ambient can have other subambients. This allows the hierarchy of distributed and mobile systems to be modelled in Ambient-PRISMA. An ambient offers two type of services to architectural elements located in it: mobility services and distributed architectural element services. Architectural elements that need these services are connected to the ambient through attachments (lines in Figure 2(b)).

The AOSD view describes the PRISMA ambient with a set of aspects that can be weaved. The ambient uses different aspects to specify the services it offers and requests. As Figure 2(a) shows, each ambient must have a Mobility, a Coordination, and a Distribution Aspect.

As ambients are responsible for the mobility concern, all ambients must have the Mobility Aspect to provide mobility services to their local architectural elements.

These services are the ones that provide the ambient calculus capabilities. The following are the Mobility Aspect functionalities:

- It allows an ambient to offer the exit service to the subambients that need to exit from it. (The specification of the AC exit capability).
- It allows an ambient to offer the enter service to the subambients that need to enter other subambients. (The specification of the AC enter capability).
- It allows an ambient to accept a new ambient from external ambients.

This is a generic aspect that must be reused by all PRISMA ambients. As a result, ambients are defined by importing the generic mobility aspect and adapting it to the needs of the software system through weavings. For example, a LAN ambient may need some security policies that are different from a PC ambient inside of the LAN. Therefore, both the LAN and PC ambient import the same Mobility Aspect, but the Mobility Aspect is weaved with different security aspects.

The coordination aspect of an ambient is needed to coordinate its architectural elements with the exterior. This coordination aspect receives calls from external architectural elements of the ambient (distributed calls) and redirects them to corresponding architectural elements of the ambient. It also receives calls from its architectural elements and redirects them to the exterior. This aspect, like the Mobility Aspect, is also a generic one that all ambients must reuse.

The Distribution Aspect of an ambient stores the name of its parent ambient. This aspect also specifies whether or not an ambient is mobile. A mobile ambient is an ambient that requests mobility services from its parent ambient. When an ambient is mobile, its parent ambient can be changed.

In Ambient-PRISMA, we have defined three types of ambients: Site, Virtual, and Process ambients (see Figure 4(a)). This separation is necessary since each one of them has its own semantics and constraints. Sites are ambients that represent devices; that is, they have a physical address. A Process is an ambient that represents a collection of software elements. A Process can only have subambients of type Process. An example of one use of a Process ambient is to move a group of architectural elements together. A Virtual ambient groups either a collection of ambient Sites or other gateways. A Virtual ambient that consists of a collection of Sites represents the borders of a network (e.g. a Local Area Network). The modelling of a Virtual ambient hierarchy can represent the hierarchy of LANs to form a Wide Area Network (WAN). In Ambient-PRISMA there is always a default Virtual ambient that represents the Root.

2.1.Case Study: Mobile Agents in an Auction Site

Mobile Agents [8] can be applied in different applications. In this paper we consider an electronic commerce application: an auction site where products are auctioned. A customer of this auction site is interested in buying a specific product with certain specifications at a maximum price. To keep track of the new auctions being performed on the auction site the customer designs two mobile agents to take charge of the purchase. One of these agents is responsible for collecting information and the other agent is in charge of managing the purchase. The two agents have to collaborate with each other. The customer sends the agents to the auction site. The

agents act for the customer. When the purchase is performed the agents return to their original location.

2.2. The Case Study Described in Ambient-PRISMA

In the case study, the customer designs the agents: the *purchaser* and the *collector*. As the *purchaser* and the *Collector* need to collaborate with each other, a connector called *AgentCnctr* coordinates them. The customer locates these architectural elements in a Process ambient called *Agents*. The *Customer* software component is connected with the *Agents* by a connector called *AgentCustCnct*. Initially, the *Customer* and the *Agents* are located in the *ClientSite*. Figure 3 shows the configuration of the software architecture before the *Agents* ambient is moved to the *AuctionSite*. In the *AuctionSite*, there is a component that represents the *Auction*. The *AuctionCnct* is a connector that coordinates the *Auction* with other architectural elements. As, the *Auction* component can communicate with distributed architectural elements, the *AuctionCnct* has an attachment with *AuctionSite* ambient.

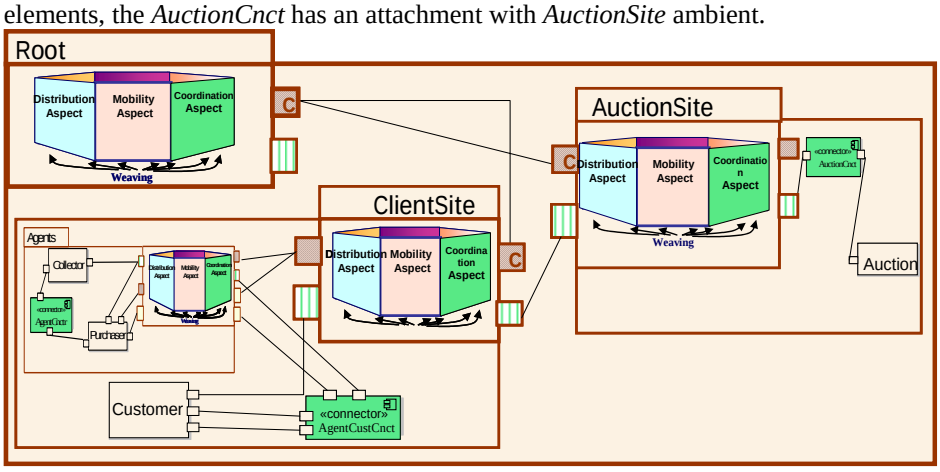


Fig. 3. The Initial Configuration of the Software architecture of the Mobile Agent Case Study

The mobility of the *Agents* is initiated when the *Customer* component requests it to move to the *AuctionSite*. The *Agents* receives the mobility request through its port. This causes the execution of *move(NewAmbient:loc)* of its Distribution Aspect that the it imports. The *move(NewAmbient:loc)* specifies that the *Agents* has to *exit* its parent ambient and then *enter* another ambient.

The *Agents* then requests the *exit(AgentNonSite, ClientSite)* from the *ClientSite*. The *ClientSite* receives this petition through its port *InCapabilitiesPort* (the doted port of ClientSite connected to Agents in Figure 3). As a consequence, the Mobility Aspect of the ClientSite is executed. The Mobility Aspect checks if the requested element (*Agents*) is one of its children (is one of its local elements). Since the *Agents* is its child, the *ClientSite* executes the *getParent* service in order to know who its parent ambient is. This service has a ***Weaving*** with the Distribution Aspect, i.e. it provokes the execution of a service in the Distribution Aspect of the ClientSite. The Distribution Aspect is executed to provide the value of *location*. Figure 3 shows the *ClientSite* has the *Root* ambient as its parent. This is a default parent ambient for all

sites and is of type *Virtual*. When the parent value is achieved, the *Mobility Aspect* removes the *Agents* from the *ClientSite*, modifying its local attachments and ensuring that the receptor (the *Root*) receives and adds it as its child. The result of the configuration after the execution of the exit is shown in Figure 4.

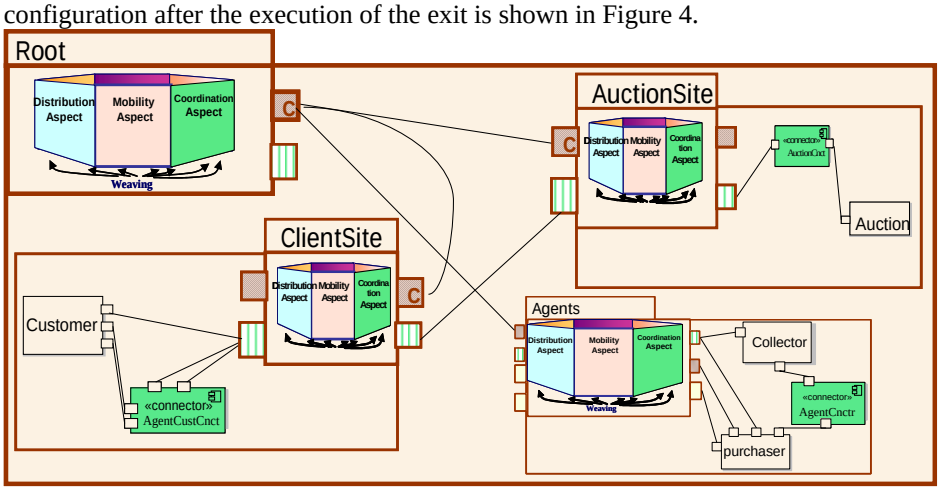


Fig. 4. The Software Architecture when the AgentNonSite executes the exit

The *Agents* ambient then requests the *enter(Agents, AuctionSite)* to the *Root* ambient. The *Root* ambient also has the Mobility Aspect. The Root ambient checks if the *Agents* and the *AuctionSite* are both its children. If they are, the *Agents* ambient becomes a child of *AuctionSite* as shown in Figure 5.

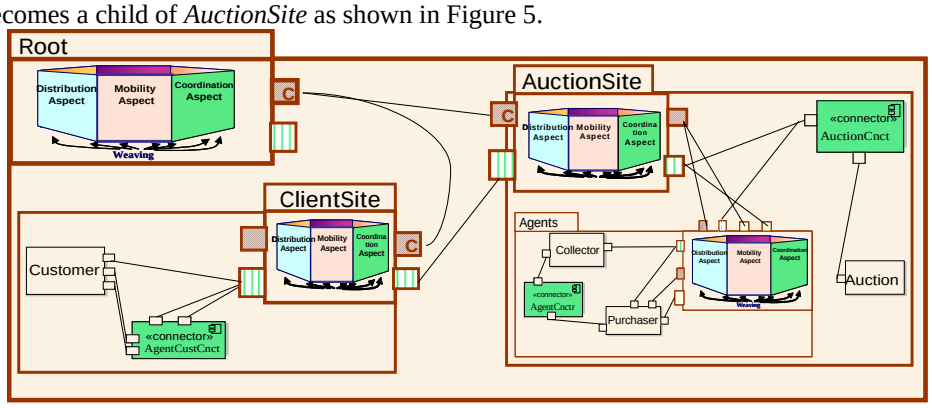


Fig. 5. The AgentNonSite becomes a child of AuctionSite

As Figure 5 shows, the *Customer* component can still send commands to the *Agents* in the *AuctionSite*. This is achieved thanks to the fact that all the attachments between the *AgentCustCnct* and the *Agents* (see Figure 3) have been replaced by attachments between these elements and their ambients (see Figure 5). Thus, distributed communication is performed through ambients.

When the *purchase* agent in Figure 5 performs the purchase that the customer desires, it invokes the *move* of the *Agents*. The entire process described above is

repeated until the *Agents* returns to *ClientSite* (the architecture configuration returns to the configuration shown in Figure 3).

The explanation described above allows locating architectural elements in ambients proving a way of distributing the software architecture. Also, the services that ambients provide through their aspects allow the mobility of the architectural elements.

3 Implementation of Ambient-PRISMA in .Net

The PRISMA metamodel is supported in .Net by the PRISMANET middleware [19]. The PRISMANET middleware provides an execution platform for PRISMA aspect-oriented software architecture configurations. For example, it contains classes that map the concepts of aspects, ports and attachments. Each PRISMANET middleware of a host can collaborate with PRISMANET middlewares on other hosts in order to provide a distributed runtime environment (see Figure 6). The runtime environment has a distributed Domain Name Server (DNS) where all middlewares on hosts can access. PRISMANET must be extended in order to provide the mobility and distributed communication of architectural elements that Ambient-PRISMA provides. Therefore, in this section Ambient-PRISMANET is explained.

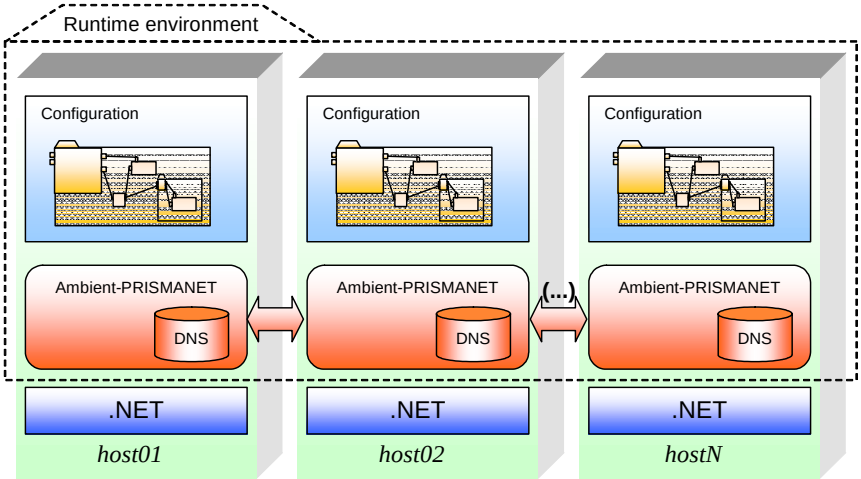


Fig. 6. Distributed Runtime Environment of PRISMANET

Ambient-PRISMANET extends the PRISMANET middleware in order to include the ambient architectural elements. Figure 7 shows the class *ArchitecturalElement* that includes the common functionality of all architectural elements. All architectural elements are composed of aspects, weavings, and ports. The classes *Component*, *Connector* and *Ambient* inherit from *ArchitecturalElement*. The Collector, the Purchaser, the Customer, and the Auction are represented in C# as classes that inherit from the class *Component*. The AgentCustomerCnct, the AgentCnctr and the AuctionCnctr are classes that inherit from the class *Connector*.

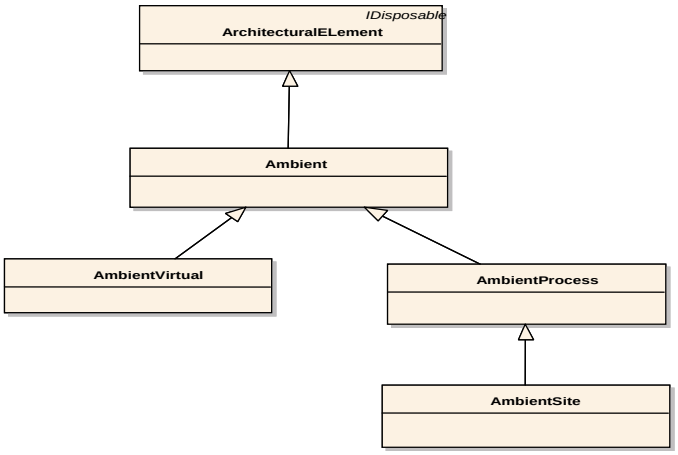


Fig. 7. The Ambient inherits from ArchitecturalElement class of PRISMANET

The class *Ambient* provides services for managing the information related to the architectural elements of an ambient. These services can change the state of an ambient by adding or removing either architectural elements or attachments. For example in the case study, when the Agents ambient requests to exit from the ClientSite ambient, the ClientSite has to replace the attachments between the Agents and the AgentCustCnctr with attachments between the AgentCustCnctr and the port of the ClientSite that provides distributed communication. Therefore, an ambient needs services that facilitate the dynamic generation of attachments depending on its state.

Moreover, the different types of Ambient-PRISMA ambients have been introduced by the classes *AmbientSite* and *AmbientProcess* and *AmbientVirtual* which inherit from class Ambient (see Figure 7). Each of these classes expresses the restrictions of the type of ambient they represent. For example, Agents ambient must inherit from the class AmbientProcess to be implemented. In the same way, ClientSite and AuctionSite ambients inherit from AmbientSite to be implemented and the Root ambient inherits from AmbientVirtual.

In addition, all ambients must have a Mobility Aspect, a Coordination Aspect and a Distribution Aspect. The Mobility and Coordination Aspects of an ambient are predefined ones that all ambients must reuse and that designers do not modify. As these aspects have a predetermined definition, they are included as specific types of aspects that inherit from the class *AspectBase* (the class that represents a PRISMA aspect). Figure 8 shows the class *MobileAspect* that maps the Mobile Aspect and *ACoordinationAspect* that maps the Coordination Aspect. They are marked with the C# sealed modifier in order to represent that they cannot be extended by the user.

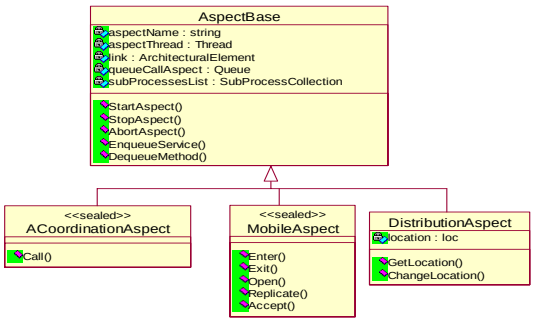


Fig. 8. The Aspects of an ambient

On the other hand, the distribution aspect of an ambient can be extended by the user. As a result, a class *DistributionAspect* that also inherits from *AspectBase* has been created. The class *DistributionAspect* implements the minimum functionality of a distribution aspect. That is, it stores the reference of the ambient location and the services needed for managing it.

The implementation described above is the basis for supporting the code generation of distributed and mobile applications from Ambient-PRISMA models. To automatically generate the code, the compiler will have to use the implementation of the Ambient-PRISMANET and extend it.

4 Conclusions and Further Work

In this paper, we have shown how ambient calculus can be used in software engineering to develop distributed and mobile applications. These applications, which are based on ambient calculus, can be developed using AOSD and CBSD techniques. The concepts of ambient calculus have been introduced into an aspect-oriented and component based metamodel (PRISMA) in order to achieve Ambient-PRISMA. Ambient-PRISMA provides the construction of distributed and mobile software systems by reusing and adapting specifications.

Ambient-PRISMA has been able to specify the distributed and mobile characteristics of the software architecture of a mobile case study. This has been done in a platform-independent way. In order to execute distributed and mobile applications based on Ambient-PRISMA, an implementation of the concepts of Ambient-PRISMA in .Net technology has been performed.

In the near future, we are going to take advantage of the ambient calculus primitives concerned with security. These primitives will be specified using our AOADL so that the case study presented in this paper could be extended to incorporate security properties. In addition, we are going to introduce the concepts presented in this paper into the PRISMA tool to be able to model and execute mobile distributed software architectures in an intuitive way. This will be done in by introducing the ambient graphical metaphor and code templates in the modelling framework in order to automatically generate code from the graphical metaphor.

Acknowledgements

This work has been funded by the Department of Science and Technology (Spain) under the National Program for Research, Development and Innovation, DYNAMICA project TIC2003-07776-C02-02. We would like to thank Carlos Millán for implementing the Ambient-PRISMANET middleware.

References

[1] Szyperski, C., *Component Software: Beyond Object Oriented programming*, ACM Press and Addison Wesley, New York, USA, 2002.

[2] Aspect-Oriented Software Development, http://aosd.net

[3] Ali, N., Millán, C., Ramos, I. *Developing Mobile Ambientes using an Aspect-Oriented-Software Architectural Model*, In Proc of the Distributed Objects and Applications (DOA 2006), LNCS, October 30-November , 2006, Montpellier, France.

[4] Ali, N., Pérez, J., Costa, C., Ramos, I., Carsí, J.A. Mobile Ambients in Aspect-Oriented Software Architectures. In Proc. of the IFIP Working Conference on Software Engineering Techniques, LNCS ISSN: 1571-5736, October 18-20, 2006,Warsaw, Poland

[5] Ali, N., Pérez, J., Costa, C., Ramos, I., Carsí, J.A. Replicación Distribuida en Arquitecturas Software Orientadas A Aspectos Utilizando Ambientes, XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD), Octubre 2006, Sitges, Barcelona, España.

[6]Perez, J., Ali, N., Carsí, J.A., Ramos, I. "Dynamic Evolution in Aspect-Oriented Architectural Models", European Workshop on Software Architecture, Pisa, June 2005 © Springer LNCS vol n.3527.

[7] Cardelli, L. "Abstractions for Mobile Computation." In Vitek, J. and (Eds.), C. J., editors, Secure Internet Programming: Security Issues for Distributed and Mobile Objects, volume 1603 of LNCS, Springer Verlag, pp. 51-94.

[8] Chess, D., Harrison, C., Kershenbaum, A. "Mobile Agents: Are They a Good Idea?" IBM Research Report RC.

[9]Perez, J., Ali, N., Costa, C., Carsí, J.A., Ramos, I. "Executing Aspect-Oriented Component-Based Software Architectures on .NET Technology",3rd International Conference on .NET Technologies, Pilsen, Czech Republic, May-June 2005 , 2005

Coordinación y Acceso a Información en Gestión de Emergencias

José H. Canós[1], Manuel Llavador[1], Carlos Solís[1], Patricio Letelier[1],
Mª. Carmen Penadés[1], Marcos R. S. Borges[2]
[1]Grupo SIA@ISSI
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia Camino de Vera s/n
E-46071 Valencia – España
{mllavador | jhcanos | letelier | mpenades | csolis} @ dsic.upv.es
[2]Núcleo de Computação Eletrônica and Instituto de Matemática
Universidade Federal do Rio de Janeiro,
Brasil
mborges@nce.ufrj.br

Resumen. En este artículo se describe el progreso realizado dentro del proyecto DYNAMICA por el grupo SIA en el campo de la gestión de emergencias. En particular, se explica el trabajo desarrollado en aspectos como la definición y ejecución de procesos flexibles, y la gestión y recuperación de información multimedia. Se presentan además algunos resultados obtenidos en aspectos relacionados con la gestión de modelos mediante herramientas METACASE.

Palabras Clave: Gestión de Emergencias, Procesos Flexibles, Bibliotecas Digitales, Ingeniería de modelos

1 Hacia los nuevos sistemas de gestión de emergencias

La gestión eficiente y eficaz de emergencias se está convirtiendo en uno de los aspectos prioritarios de organizaciones y gobiernos en todo el mundo, especialmente tras las respuestas, en ocasiones poco satisfactorias, que se han dado a desastres recientes generados tanto por fenómenos naturales adversos como por acción del hombre. Los analistas de tales respuestas han llegado a la conclusión de que en muchos casos se han producido fallos de coordinación entre los diferentes participantes en la respuesta, en ocasiones motivado por la falta de información acerca del desarrollo de los incidentes [1, 2].

El grupo de Sistemas de Información Avanzados (SIA), integrado en el grupo de Ingeniería del Software y Sistemas de Información (ISSI), ha trabajado en el dominio de la gestión emergencias desde finales de los años 90. En particular, fue el responsable del Plan de Emergencia Hipermedia de MetroValencia, sistema pionero que permitió mejorar la respuesta a emergencias mediante el uso de tecnología hipermedia [3]. A partir de su desarrollo, el grupo ha dedicado su esfuerzo a entender la problemática de la gestión global de emergencias –incluyendo no sólo respuesta, sino también prevención y análisis de la respuesta— con énfasis en el soporte tecnológico a la misma.

La experiencia previa del grupo nos ha permitido identificar un número de dimensiones dentro de la gestión de emergencias [4]:

- **Coordinación.** La solución de una emergencia consiste en la ejecución de un proceso cuya especificación incluye los participantes, la secuencia ordenada de acciones que deben realizar, la información utilizada por cada actividad, etc.
- **Presentación.** Cada uno de los actores implicados en la resolución de una emergencia percibe el proceso de forma diferente, entre otras cosas porque manejan diferentes piezas de información para realizar sus tareas. En consecuencia, los sistemas para la gestión de emergencias deben permitir diferentes modos de presentación que destaquen en cada momento la información más relevante para cada uno de los actores.
- **Gestión y recuperación de información.** La validez y relevancia de la información de la que se dispone durante la gestión de una emergencia puede cambiar. En consecuencia, los sistemas para la gestión de emergencias deben soportar la gestión de información multimedia con esas características.
- **Comunicaciones.** Los mecanismos de comunicación deben soportar la interacción por voz de los participantes en la resolución de la emergencia. Al mismo tiempo, gran cantidad de información multimedia debe ser trasmitida entre los subsistemas implicados.
- **Colaboración.** Una vez activada una alarma por sensores o por humanos, expertos en diferentes campos deben colaborar para analizar la situación, calcular los daños, identificar riesgos potenciales y ayudar a los gestores de las emergencias en la toma de decisiones.
- **Inteligencia.** La inteligencia es la habilidad que debe tener el sistema para generar información valiosa a partir de datos almacenados en los distintos repositorios. Otro factor importante a tener en cuenta es el tratamiento de la información contextual que se debe transmitir para complementar aquélla de la que se dispone de antemano.

Un sistema de gestión de emergencias debe dar soporte a todas y cada una de las dimensiones mencionadas. El desarrollo de tal sistema será complejo, puesto que hay numerosos factores a tener en cuenta: un elevado número de requisitos funcionales y no funcionales, diferentes estilos arquitectónicos, heterogeneidad de usuarios, etc. Dado el tamaño del problema, el grupo se ha centrado en dos aspectos básicos de la gestión de emergencias: la coordinación y el acceso a la información. En las secciones siguientes se describen los progresos realizados hasta el momento.

2 Coordinación basada en Procesos Flexibles

En el proceso de respuesta a una emergencia participan actores, que representan a cada una de las entidades, ya sean humanos o sistemas de información, que interactúan durante la gestión de una emergencia de forma coordinada. Los procesos de gestión de emergencias son eminentemente cooperativos, por lo cual los sistemas que dan soporte a la gestión de emergencias deben garantizar la correcta coordinación de los actores para conseguir gestionar una emergencia de forma satisfactoria. En los

sistemas de soporte a procesos (como por ejemplo los sistemas de gestión de flujos de trabajo o *workflow*), el módulo de especificación permite a los administradores y analistas definir las actividades y asignarlas a las personas para su ejecución. Sin embargo, las aproximaciones tradicionales no son válidas en el caso de la gestión de emergencias, ya que algunas construcciones típicas, como los saltos en función del estado del sistema para un conjunto de actividades, resultan complicadas de expresar. Por otra parte, las aproximaciones basadas en reglas tampoco son útiles, ya que son poco apropiadas para representar procesos que no se basan en el estado sino en el orden de ejecución, y además resultan difíciles de interpretar para los humanos.

Por ello, a lo largo del proyecto se ha trabajado en una aproximación mixta, es decir, que combina procesos en el sentido clásico y reglas, que proporciona toda la expresividad requerida. La propuesta, descrita en [5, 6], permite describir el proceso de resolución de una emergencia mediante un conjunto de actores, un conjunto de variables de estado y un conjunto de fórmulas de lógica dinámica que representan el flujo de ejecución de las actividades. Las fórmulas dinámicas son el resultado de transformar a un formalismo común los flujos de trabajo y las reglas empleadas para la definición de los procesos flexibles. Actualmente se está trabajando en el desarrollo de editores visuales que nos permitan especificar este tipo de procesos.

El hecho de estar utilizando formalismos nos permite disponer de mecanismos para validación y simulación de los procedimientos de resolución de emergencia que nos permiten mejorar los procesos. Actualmente se está trabajando en el desarrollo de herramientas de verificación formal por animación de los procesos definidos [7].

3 Acceso a Información

En cada etapa del proceso de resolución de una emergencia se han de tomar una serie de decisiones de las que depende en gran medida su finalización con éxito. Es por ello muy importante disponer de información de la mayor calidad posible, lo cual no siempre significa tener acceso a una avalancha de datos que en ocasiones pueden ser irrelevantes y ralentizar la toma de decisiones.

Las últimas generaciones de sistemas de gestión de emergencias han incorporado fuentes de información georreferenciada –vía Sistemas de Información Geográfica (SIG)—que permite ubicar de manera automática los distintos equipos que participan en la respuesta a un incidente dado. Más allá de este tipo de información, cada vez más se cuenta con información audiovisual procedente de cámaras situadas en diferentes ubicaciones –y posiblemente registradas en el correspondiente SIG—al igual que otros tipos de datos que se suelen mostrar en una interfaz al estilo “hoja de cálculo”, no siempre intuitiva ni efectiva.

A pesar de la variedad mencionada, toda esa información, en diferentes formatos y procedente de diferentes fuentes, no termina de ser efectiva debido fundamentalmente a dos razones fundamentales:

1. a cada tipo de información se accede mediante aplicaciones diferentes, incluso en ordenadores distintos, lo que hace difícil dar una visión única de la situación; y

2. la información está desacoplada del proceso. Es decir, no existe un proceso de selección de información basada en el estado del proceso, lo cual hace difícil seleccionar de manera automática la información relevante para cada etapa del mismo.

La tecnología hipermedia aparece como la más indicada para subsanar ambas deficiencias. En primer lugar, un hiperdocumento puede ser el concepto unificador de acceso a toda la información multimedia que puede manejarse en la resolución de un incidente. Y, en segundo lugar, la navegación subyacente en el hiperdocumento puede definirse a partir de la navegación implícita en los modelos de proceso definidos en los planes de emergencia. Los trabajos del grupo en esta línea han apuntado a dos direcciones fundamentales, que se resumen a continuación.

Generación de modelos navegacionales de aplicaciones hipermedia a partir de modelos de proceso. Se ha desarrollado un método de desarrollo de hipermedia dirigido por modelos llamado MDHDM (*Model Driven Hypermedia Development Method*) que tiene como particularidad principal –que le distingue de los demás métodos propuestos hasta el momento—la definición de heurísticas y transformaciones para obtener el modelo navegacional a partir del modelo de proceso y, complementariamente, el modelo conceptual [8, 9]. En estos momentos se está trabajando en el desarrollo de una herramienta de soporte al mismo.

Uso de Bibliotecas Digitales para la gestión y acceso a la información. Toda la información multimedia a mostrar durante la resolución de una emergencia debe estar adecuadamente catalogada, organizada y preparada para su efectiva recuperación. Por ello, el grupo ha trabajado en el uso de las bibliotecas digitales como tecnología de soporte a estos aspectos. En particular, se está explorando el empleo de software de repositorios como uno de los componentes básicos de cualquier sistema de gestión de emergencias. Algunos resultados preliminares se muestran en [10]

4 Herramientas de soporte a la transformación de modelos

En el grupo SIA estamos definiendo un entorno que permitirá construir sistemas de soporte a la resolución de emergencias considerando las dimensiones presentadas en la sección 1. Los desafíos son múltiples, debido, en primer lugar, a que los sistemas a generar son muy complejos, principalmente por la complejidad de los requisitos funcionales y no funcionales mencionados en la sección 1; y, en segundo lugar, que dichos sistemas deben ser capaces de manipular información muy heterogénea –tanto en cuanto a los formatos de representación como respecto al tipo de contenido multimedia—y por tanto deben dar soporte a la integración semántica de los datos.

Para resolver ambos problemas vamos a utilizar técnicas basadas en modelos. Como paso inicial, hemos desarrollado un conjunto de herramientas [11,12] que integran las distintas técnicas que permiten desarrollar software a partir de modelos: desarrollo de software guiado por modelos, gestión de modelos y factorías de software. Estas herramientas se han concretado en un entorno METACASE para la generación de herramientas de desarrollo automático de software basado en tres pilares:

- Definición de lenguajes específicos de dominio para la descripción formal de los modelos que describirán las aplicaciones a generar
- Definición de plantillas para las transformaciones de los distintos modelos [
- Definición de compiladores de modelos específicos de dominio para la generación del código fuente de la aplicación final [

Herramientas como las mencionadas pueden ser la vía para, entre otras cosas, la federación de repositorios heterogéneos, interoperabilidad basada en metadatos, y cualesquiera otras funciones basadas en la manipulación y/o transformación de modelos, ya sean de datos, de proceso, o incluso de interfaz de usuario.

5 Conclusiones y trabajos futuros

La gestión de emergencias requiere la participación de expertos en diferentes áreas, que han de trabajar codo con codo y bajo fuertes restricciones de tiempo en encontrar la mejor solución para problemas que amenazan vidas y propiedades. En la búsqueda de la eficiencia en las respuestas, una correcta coordinación y el manejo de la información que se precisa en cada momento son dos de los requisitos fundamentales. En este documento hemos resumido el trabajo realizado por el grupo SIA durante el proyecto DYNAMICA. En nuestro plan de trabajo futuro aparece como prioridad esencial el diseño de una arquitectura de referencia para los sistemas de gestión de emergencias, así como el desarrollo del entorno para su especificación y desarrollo.

Referencias

1. The 9-11 Commission Report. Final Report of the National Commission on Terrorist Attacks Upon the United States, Official Government Edition. http://www.gpoaccess.gov/911/index.html
2. A Failure of Initiative. The Final Report of the Select Bipartisan Committee to Investigate the Preparation for and Response to Hurricane Katrina. http://katrina.house.gov/full_katrina_report.htm
3. Canós, J.H. and Zulueta, F. “Using Hypermedia to improve Safety in Underground Metropolitan Transportation”. *Multimedia Tools and Applications*. vol. 22, no. 1, January 2004.
4. Canós, J.H., Alonso, G., and Jaén, J. “A Multimedia Approach to the Efficient Implementation and Utilization of Emergency Plans”, *IEEE Multimedia* Vol. 11, No. 3, July/September 2004, pp. 106-110
5. Llavador, M., Letelier, L., Borges, M., Canós, J.H., Penadés, M.C., Solís, C. "Un enfoque Orientado a Procesos para la Especificación de Planes de Emergencia", X Jornadas de Ingeniería del Software y Bases de Datos JISBD'2005, ISBN. 84-9732-434-X, Septiembre 2005
6. Llavador, M., Letelier, P., Penadés, M.C., Canós, J.H., Borges, M., Solís, C. "Precise yet Flexible Specification of Emergency Resolution Procedures", 3rd International

Conference on Information Systems for Crisis Response and Management ISCRAM'2006, ISBN. 90-9020601-9, Mayo 2006 Mike Meleshkin Award '2006

7. Roche, A., Letelier, P., Navarro, E., Llavador, M. "Validación incremental de modelos usando escenarios y prototipado automático", XI Jornadas de Ingeniería del Software y Bases de Datos JISBD'2006, ISBN. 84-95999-99-4, Octubre 2006
8. Solís, C., Canós, J.H., Llavador, M., Penadés, M.C. "De modelos de proceso a modelos navegacionales", XI Jornadas de Ingeniería del Software y Bases de Datos JISBD'2006, ISBN. 84-95999-99-4, Octubre 2006
9. Solís, C., Canós, J.H., Penadés, M.C., Llavador, M. "Model Driven Hypermedia Development Method", Internacional Conference WWW/Internet IADIS, ISBN. 972-8924-19-4, Octubre 2006
10. Canós, J.H., Borges, M., Llavador M., "Expanding the use of Digital Libraries: the case of Emergency Management Systems", International Workshop DL-CUBA'2006 (celebrado junto a JCDL'2006)
11. Llavador, M., Canós, J.H. "XWebMapper: A Web-based Tool for Transforming XML Documents", 10th European Conference on Research and Advanced Technology for Digital Libraries ECDL'2006, LNCS 4172 Springer, ISBN. 3-540-44636-2, Septiembre 2006
12. Llavador, M., Canós, J.H., Letelier P., Solís, C. "McGen: un entorno para la generación automática de compiladores de modelos específicos de dominio", XI Jornadas de Ingeniería del Software y Bases de Datos JISBD'2006, ISBN. 84-95999-99-4, Octubre 2006

Hacia la Construcción de Arquitecturas Software Dinámicas

Cristóbal Costa, Jennifer Pérez, José Á. Carsí

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera, s/n
{ccosta, jeperez, pcarsi}@dsic.upv.es, web: http://issi.dsic.upv.es

Abstract. Muchos sistemas software han de adaptarse a los cambios que sufre su entorno sin detener su ejecución, ya que su parada puede ser crítica para su funcionamiento o recuperación. En este trabajo se propone una solución para dar soporte a esta necesidad. Dicha solución, consiste en la reconfiguración dinámica en arquitecturas software. Concretamente, la propuesta se ha aplicado a aquellos sistemas software que han sido desarrollados siguiendo el enfoque PRISMA. Por lo tanto, el trabajo no sólo propone mecanismos de reconfiguración dinámica para arquitecturas software, también para arquitecturas software orientadas a aspectos. Este trabajo extiende el modelo PRISMA incorporando un nuevo *concern* (asunto) de *configuración* que proporciona reflexión y la posibilidad de modificar dinámicamente la configuración de un modelo arquitectónico. Dicha reconfiguración puede ser *programada*, si es el propio sistema en ejecución el que la desencadena debido a cambios en su entorno, como *ad-hoc*, si es el arquitecto software el que invoca los servicios de reconfiguración.

Keywords: arquitecturas software, reconfiguración dinámica, reflexión, CBSD, AOSD

1. Introducción

En la actualidad los sistemas software son cada vez más complejos. Éstos se construyen mediante la integración de diferentes subsistemas que deben coordinarse entre sí de forma adecuada para ofrecer el comportamiento esperado por el arquitecto software. La configuración e implantación de dichos sistemas consumen muchos recursos temporales y económicos. Del mismo modo, frecuentemente surgen errores que son detectados a posteriori y deben ser reparados cuando el sistema ya se encuentra en funcionamiento. Además, en la actualidad cada vez es más difícil que el arquitecto software sea capaz de anticipar todas las características que un sistema software va a requerir en un futuro, dada la gran complejidad del software y a la gran cantidad de requisitos no funcionales que se han de tener en cuenta a la hora de desarrollar un producto software. Tradicionalmente, el mantenimiento del software se ha realizado mediante la modificación del código fuente afectado, su compilación y posterior reinicio del sistema (completo) con los cambios incorporados. Sin embargo,

este proceso es tan costoso que las organizaciones tienden a evitarlo, con lo que aumenta el riesgo de que el sistema se degrade y se vea rápidamente desfasado. Como alternativa, la tendencia es realizar el mantenimiento en periodos de baja actividad. Sin embargo, esto no es aplicable para aquellos sistemas software en los que la adaptación dinámica es un requisito fundamental, ya que por el tipo de actividad que desarrollan no pueden parar su ejecución para ser adaptados. Un claro ejemplo de este tipo de sistemas son los sistemas en tiempo real.

Es por esto que surge la necesidad de incorporar el soporte a la adaptabilidad en tiempo de ejecución. El objetivo es que los sistemas software sean capaces de soportar cambios en su arquitectura sin necesidad de detenerse completamente. Para ello, los subsistemas afectados se modificarán en tiempo de ejecución para soportar los cambios necesarios, mientras el resto de subsistemas son ajenos a dichos cambios y pueden trabajar normalmente. Como un primer paso, el área de las arquitecturas software lleva tiempo trabajando en proponer estrategias que faciliten la evolución y el mantenimiento del software, gestionando adecuadamente la complejidad a distintos niveles de abstracción y facilitando la reutilización de los elementos que las componen. Sin embargo, los enfoques tradicionales no han considerado el dinamismo como una característica prioritaria para la construcción de arquitecturas software. Tan sólo han considerado la evolución y el mantenimiento a nivel de especificación y diseño, de forma similar al enfoque MDD (Model Driven Development) y sus dos tendencias: Model Driven Architecture (MDA) [12] y Software Factories [8]. En ellas sólo se contemplan los cambios en el modelo arquitectónico y mediante generación automática de código se vuelve a reconstruir el sistema "adaptado".

Por esta razón, el siguiente paso hacia la adaptabilidad dinámica del software es la incorporación de mecanismos que soporten la reconfiguración dinámica de la arquitectura software [5, 6]. La reconfiguración dinámica, o dinamismo estructural, hace referencia a los cambios producidos en tiempo de ejecución que afectan a la topología de una arquitectura software (es decir, cambios que afectan al número de instancias de los elementos arquitectónicos en ejecución y a sus interconexiones).

En este artículo se propone una aproximación para proporcionar capacidades de reconfiguración dinámica al modelo PRISMA [14], el cual se basa en la integración del Diseño de Software Basado en Componentes (DSBC) [7, 16] y en el Desarrollo de Software Orientado a Aspectos (DSOA)[2]. El objetivo es dar soporte a la reconfiguración dinámica orientada a aspectos. Así mismo, se pretende permitir el modelado de las características de reconfiguración evitando, en la medida de lo posible, añadir nuevas construcciones sintácticas que hagan más complejo el meta-modelo PRISMA y con ello dificulten la comprensión del mismo.

La estructura del artículo es la siguiente: en primer lugar se presenta brevemente el modelo PRISMA. En el apartado 3 se describe la propuesta para modelar las capacidades de reconfiguración, mientras que en el apartado 4 se describen los trabajos relacionados. Finalmente, en el apartado 5 se presentan las conclusiones y los trabajos futuros.

2. El modelo PRISMA

El modelo PRISMA permite definir arquitecturas de sistemas software complejas, distribuidas y reutilizables. Principalmente, se caracteriza por la integración que realiza del DSOA y del DSBC. Los elementos arquitectónicos se identifican mediante descomposición funcional, mientras que las características comunes (*crosscutting-concerns*: distribución, coordinación, seguridad, etc.) de la arquitectura son las que identifican los aspectos. Un aspecto PRISMA es un *concern* común y compartido por el conjunto de elementos arquitectónicos de la arquitectura. Los aspectos son definidos de forma independiente a los elementos arquitectónicos. El tipo de aspectos que existan en una arquitectura software varía dependiendo de las características del sistema software y de su dominio. La integración de los enfoques DSOA y DSBC que realiza PRISMA se materializa mediante la importación de aspectos que realizan los elementos arquitectónicos para definir su comportamiento. Los elementos arquitectónicos importan los aspectos mediante una referencia, permitiendo de este modo la reutilización de un mismo aspecto por todos los elementos de la arquitectura que necesiten incorporar sus características. De este modo, el elemento arquitectónico no contiene código mezclado (*tangled code)* de distintos concerns, sino que dichos concerns están modularizados mediante aspectos y sincronizados dentro del aspecto mediante *weavings.*.

Por lo tanto, un elemento arquitectónico PRISMA se caracteriza por estar formado simétricamente [9] por un conjunto de aspectos de distinto tipo, por las relaciones de sincronización entre estos aspectos (*weavings*), y por uno o más puertos que exportan la funcionalidad ofrecida por el elemento arquitectónico. Los elementos arquitectónicos pueden ser elementos arquitectónicos simples (componentes y conectores), o elementos arquitectónicos complejos (sistemas) [14].

Los sistemas son componentes complejos que, además de las características propias de un elemento arquitectónico PRISMA, permiten encapsular otros elementos arquitectónicos, añadiendo capas adicionales de abstracción y con ello disminuyendo la complejidad del sistema. Los componentes y conectores a través de sus puertos se interconectan entre sí mediante canales de comunicación denominados *attachments*. Además, existen otros canales denominados *bindings*, los cuales se definen dentro de un sistema y permiten exportar servicios de los elementos arquitectónicos internos al exterior del sistema (ver Figura 1).

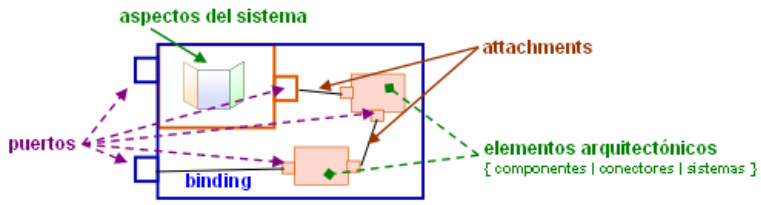


Figura 1. Sistema PRISMA y elementos que lo constituyen

PRISMA proporciona un Lenguaje de Descripción de Arquitecturas Orientado a Aspectos (LDAOA) para especificar arquitecturas software siguiendo el modelo PRISMA. Dicho LDAOA divide su lenguaje en dos niveles de abstracción: lenguaje

de tipos y lenguaje de configuración [13]. En el primero se definen elementos arquitectónicos, aspectos y tipos de enlace, y en el segundo se definen las instancias y se interconectan entre sí con el objetivo de lanzar a ejecución una arquitectura específica. Dicha arquitectura se conoce por el nombre de configuración inicial.

3. Reconfiguración dinámica de sistemas PRISMA

Un sistema PRISMA encapsula una serie de elementos arquitectónicos interconectados entre sí, que se pueden ver como un patrón arquitectónico. Dicho patrón es instanciado para formar la configuración inicial del sistema. Tras la instanciación y ejecución de dicha configuración inicial, puede surgir la necesidad de ser modificada durante su ejecución. Por lo tanto, debe introducirse un mecanismo que permita modelar y llevar a cabo dicha reconfiguración en tiempo de ejecución *(reconfiguración dinámica*). Dicho mecanismo es la principal aportación de este trabajo, pero además, otro de sus propósitos es que PRISMA proporcione este mecanismo sin introducir artefactos que compliquen su meta-modelo. Además, con el objetivo de proporcionar mayor versatilidad, dicho mecanismo debe permitir realizar la reconfiguración dinámica de dos formas complementarias: (i) *Ad-Hoc*, a través de la invocación directa de los servicios correspondientes o, (ii) *programada*, de tal forma que el sistema pueda adaptarse al nuevo entorno cuando detecte determinadas condiciones preestablecidas (por ejemplo, romper una conexión con un elemento arquitectónico al detectar su caída; o instanciar otra copia de un determinado elemento arquitectónico y conectarlo adecuadamente para equilibrar la carga).La solución que tiene en cuenta todas estas consideraciones y necesidades consiste en materializar y hacer accesibles las capacidades reflexivas necesarias para (i) obtener la configuración actual de la arquitectura y (ii) modificar dicha configuración.

Para ello, se ha definido un *concern* llamado *configuration*, el cual se ha materializado en un único aspecto genérico llamado *configuration* (ver figura 2). Dicho aspecto debe ser importado por todo sistema que quiera dar soporte a la reconfiguración dinámica de los elementos arquitectónicos y conexiones que incluye. El aspecto *configuration* mantiene el estado del sistema al que pertenece mediante tres listas dinámicas: (i) las instancias de los elementos arquitectónicos en ejecución (tanto componentes, conectores como sistemas), (ii) las referencias a los *attachments*¸ que junto a (iii) los *bindings* definen el conjunto de conexiones entre los elementos del sistema (la topología). También proporciona los servicios correspondientes para la inserción, eliminación y consulta de dicha información. Por ejemplo, la creación de un nuevo *attachment* se llevaría a cabo invocando el servicio *AddAttachment* indicándole los extremos a conectar *(puerto1, componente1, puerto2, conector2)*. Del mismo modo, invocando el servicio *GetAttachment* y proporcionando el ID del *attachment* que se necesita, podría consultarse la información de dicho *attachment*, como son los elementos arquitectónicos que conecta y a través de qué puertos.

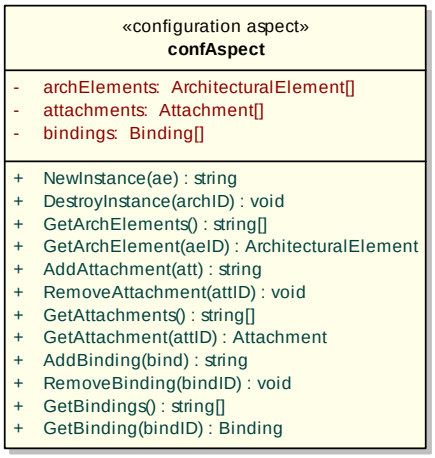


Figura 2. Aspecto de configuración

La implementación de dichos servicios es proporcionada por el middleware PRISMANET [4, 15]. PRISMANET, además de ejecutar arquitecturas PRISMA, ofrece los servicios básicos para llevar a cabo la reconfiguración dinámica, garantizando que la creación y destrucción de instancias de *attachments*, *bindings* o elementos arquitectónicos no afecte al resto de instancias en ejecución. Por ejemplo, se garantiza que cuando un canal de comunicación entre dos elementos arquitectónicos es destruido, previamente se hayan finalizado las comunicaciones que pudieran estar manteniendo dichos elementos.

El aspecto de configuración proporciona únicamente la funcionalidad para consultar o modificar la configuración de la instancia en ejecución. La semántica que solicita y consume dichos servicios debe implementarse en cualquiera de los aspectos que constituyen el sistema, dependiendo del contexto que origine el cambio. La sincronización entre los distintos servicios de los aspectos será realizada mediante la definición de *weavings* entre ellos.

Por ejemplo (ver Figura 3), dado un sistema PRISMA que modela una entidad bancaria (*BankSystem*) en la cual cada cuenta se modela como un componente (*Account*), la creación de nuevas cuentas sólo se llevará a cabo cuando el servicio *CreateAccount* del aspecto funcional valide que puede realizarse.

Una vez este servicio se ejecute con éxito, (i) se disparará el *weaving* relacionado con el servicio *NewInstance()* del aspecto de configuración y (ii) la ejecución de dicho servicio instanciará un nuevo componente de tipo *Account*.

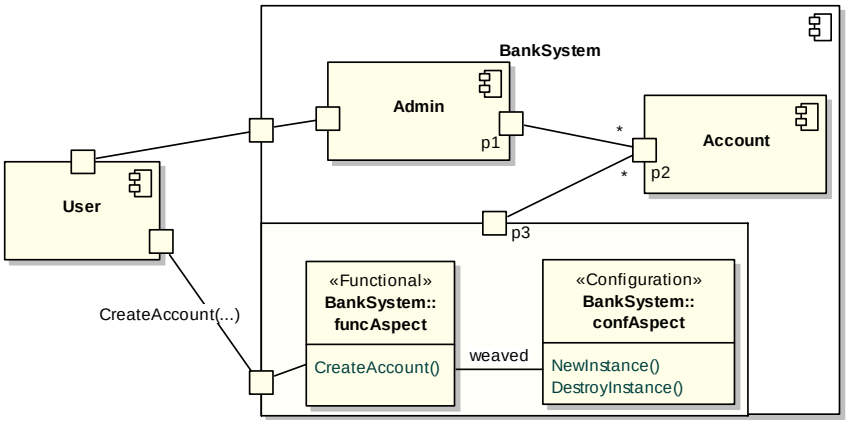


Figura 3. Ejemplo BankSystem

```
System BankSystem
[...]
Weavings
    confAspect.newInstance("Account", out AccountID, [Name,...])
    before
    funcAspect.createAccount(Name,...)

    funcAspect.accountCreated("Account", AccountID)
    after
    confAspect.newInstance("Account", out AccountID, [Name,...])

    confAspect.addAttachment(out attID, P1, "admin1", P2, AccountID)
    after
    confAspect.newInstance("Account", out AccountID, [Name,...])
[...]
```

Figura 4. Ejemplo de la especificación de los weavings entre el aspecto *configuration* y el aspecto *functional* del sistema bancario *BankSystem*

Por otro lado, como puede verse en el fragmento de la especificación PRISMA de la Figura 4, mediante el uso de los *weavings* puede definirse cómo debe ir configurándose la topología de forma dinámica a medida que los elementos arquitectónicos se van instanciando. En el ejemplo, cuando se crea una nueva instancia de tipo *Account*, se activa el *weaving* que crea un nuevo *attachment* entre dicha instancia y la instancia del componente cuyo ID es *"admin1"*.

Con este sencillo ejemplo se ha ilustrado el funcionamiento de la reconfiguración *programada*: cada sistema PRISMA posee la capacidad de reconfigurarse a sí mismo, a través del uso de los weavings entre el aspecto *configuration* y el resto de aspectos del sistema. En el ejemplo, las nuevas instancias de *Account* sólo serán creadas cuando el sistema determine que es adecuado (se ejecuta correctamente el servicio *CreateAccount* del aspecto funcional, por lo que se activa el *weaving* correspondiente). De esta forma, se evita tener el código entremezclado, ya que las

"reglas" de reconfiguración no se entremezclan con el código funcional (*concern* funcional) ni con el propio aspecto que proporciona los servicios de reconfiguración.

Respecto a la reconfiguración *Ad-Hoc*, una primera aproximación es suponer que cualquier servicio de reconfiguración es accesible directamente por el arquitecto software. Sin embargo, esto viola el principio de encapsulación: componentes de otros proveedores (COTS) no tienen por qué permitir su reconfiguración por terceros. Por otro lado, en sistemas distribuidos no es adecuado que el arquitecto software pueda acceder a los servicios de reconfiguración de componentes remotos sino es a través de los canales establecidos por la arquitectura. Es por estas razones que se considera más apropiado limitar el acceso a los servicios de reconfiguración por parte de entidades externas (incluido el arquitecto software), a menos que se hagan accesibles mediante un puerto específico que los exporte. De esta forma, (i) puede limitarse a un pequeño subconjunto los servicios de reconfiguración accesibles (por ejemplo, permitir la creación de elementos arquitectónicos, pero impedir la modificación de conexiones); (ii) el arquitecto software podrá conectarse remotamente e invocar dichos servicios convirtiéndose en usuario de la arquitectura, es decir, utilizando las conexiones creadas al efecto que enlazan las distintas ubicaciones; y además, (iii) cualquier otro elemento arquitectónico podrá actuar como *configurador* de otros elementos arquitectónicos, añadiendo expresividad al modelo.

Por lo tanto, la diferencia entre ambas es que la reconfiguración *programada* se produce a consecuencia de la sincronización entre aspectos, a nivel interno y por políticas del propio sistema, mientras que la reconfiguración *ad-hoc* es iniciada por entidades externas mediante la invocación de los servicios de reconfiguración exportados por los puertos.

4. Trabajos relacionados

El problema de la reconfiguración dinámica, proveniente del área de los sistemas distribuidos, ha sido tratado previamente en otros enfoques arquitectónicos. Una revisión muy completa puede encontrarse en [5]. Básicamente, la mayor parte de las propuestas ofrecen al menos las cuatro operaciones básicas de reconfiguración, establecidas por Conic [10]: *create*, *destroy*, *link* y *unlink*.

Cabe destacar las propuestas basadas en álgebras de procesos como Darwin [11], LEDA [3], ambas basadas en π-Calculus, y PiLar [6], basada en CCS y reflexión. Tanto en Darwin como en Leda, la instanciación de componentes es implícita. Por su parte, PiLar sí que soporta la instanciación explícita, mediante el uso del operador *New*. Por lo tanto, tan sólo PiLar implementa las cuatro operaciones básicas, ya que Darwin y Leda no ofrecen directamente mecanismos para invocar la destrucción de instancias. Leda la ofrece de forma implícita al salir los nombres del ámbito del proceso. Estas propuestas están claramente enfocadas hacia el soporte de la reconfiguración *programada*. Sin embargo, tan sólo soportan parcialmente la reconfiguración *ad-hoc*, en el sentido de que se permite manipular la arquitectura a través de alguna herramienta (orientada hacia la modificación manual), pero no se contempla la posibilidad de que otros elementos arquitectónicos puedan actuar como configuradores de la arquitectura.

Sólo algunas propuestas como PiLar incorporan mecanismos de reflexión para consultar la topología de la arquitectura y actuar en consecuencia. En el resto de trabajos no se indica ni *cómo* almacenar el estado ni *cómo* los elementos pueden acceder claramente a dichas operaciones. En [6] se presenta una taxonomía de cómo las distintas propuestas modelan el dinamismo arquitectónico. No obstante, para el caso que nos ocupa, dicha taxonomía puede reducirse a dos grandes grupos:

1. *Entidad Externa/Supervisor*. Se define un elemento especial, global, que es el encargado de realizar las operaciones de reconfiguración (como Wright dinámico [1] o Conic [10]) de forma explícita o implícita. Sin embargo, el principal inconveniente es que las acciones de reconfiguración se encuentran centralizadas, definidas a nivel global, con lo que no se permite definir reglas de reconfiguración a menores niveles de granularidad; esto es, que los diferentes subsistemas (componentes complejos) especifiquen a su vez operaciones de reconfiguración que sólo afecten a su ámbito de actuación (los elementos que lo constituyen). Por esta razón, nuestra propuesta permite que cada elemento arquitectónico complejo (sistemas PRISMA) pueda configurar en tiempo de ejecución su propia topología, sin requerir la intervención de la arquitectura global en la que se encuentra ubicado.

   También se incluyen en este grupo aquellas propuestas en las cuales es posible la reconfiguración *ad-hoc* mediante el uso de *Frameworks* específicos para configurar la arquitectura, enfocadas hacia la intervención humana (como Darwin). No obstante, al no definir una API accesible por otros elementos arquitectónicos, no pueden llevarse a cabo reconfiguraciones *programadas* que alteren a otros elementos. En nuestra propuesta un elemento externo puede reconfigurar un sistema si éste ha exportado, mediante el uso de los puertos, los servicios de reconfiguración ofrecidos por sus puertos.

2. *Componente/Conector mixto*. En otras propuestas, el ADL ofrece a todos los elementos arquitectónicos una serie de primitivas que les capacitan para modificar -total o parcialmente- la topología de la arquitectura, como en Darwin, Leda y PiLar. En estos casos, se supone que una "entidad" (el intérprete, compilador, framework, etc.) ofrecerá dichos servicios, pero no es modelada como un elemento físico. En nuestra propuesta, dichos servicios son ofrecidos por un elemento del modelo PRISMA, el aspecto *configuration*. Frente a incorporar dichos servicios como parte del ADL, esta alternativa presenta dos ventajas:
   - Los servicios de reconfiguración pueden extenderse, de acuerdo a como se extienden tipos de aspectos con nueva funcionalidad. Por ejemplo, el servicio *NewInstance*() podría extenderse para que comprobase que no se crean más de *n* instancias de un determinado tipo de componente, o para que se comprueben precondiciones adicionales.
   - Los servicios del nivel *meta* están perfectamente delimitados en el aspecto, y por tanto, se conoce perfectamente su ámbito de actuación. Cuando se solicite la creación de una nueva instancia o enlace, se creará en el ámbito del sistema al cual pertenezca el aspecto de configuración. Además, también pueden restringirse los servicios de reconfiguración accesibles.

5. Conclusiones y trabajos futuros

En este trabajo se ha presentado una propuesta para introducir capacidades de reconfiguración dinámica en sistemas PRISMA mediante la definición de un nuevo tipo de aspecto: el aspecto *configuration*. Este aspecto mantiene la información de la configuración del sistema y permite manipularla en tiempo de ejecución. Al definir toda la semántica de reconfiguración en un aspecto, se evita tener que introducir nuevas estructuras sintácticas en el LDAOA de PRISMA original, beneficiándose de la nueva funcionalidad.

Por otro lado, se ha dado una solución a la reconfiguración *ad-hoc* de sistemas mediante la exportación de los servicios de evolución a través de un puerto, y a la reconfiguración *programada* mediante la sincronización entre el aspecto *configuration* y otros aspectos del sistema, haciendo uso de los *weavings*.

Este trabajo es la idea inicial para dar soporte a la reconfiguración dinámica de arquitecturas PRISMA, el cuál va a dar lugar a una gran cantidad de trabajos futuros que realizar. Uno de ellos podría ser el soporte seguro de las reconfiguraciónes *ad-hoc*. Esto es debido a que cualquier elemento arquitectónico diseñado maliciosamente podría conectarse al puerto que proporciona los servicios de evolución del sistema y solicitar servicios de configuración que le permitiesen modificar la configuración interna del sistema de forma incorrecta, peligrosa y perjudicial para el sistema. Por lo tanto, un trabajo futuro que se ha de desarrollar es la incorporación de protocolos de seguridad para garantizar qué sólo solicitan servicios de evolución aquellos elementos que están capacitados.

Otro trabajo futuro a realizar, una vez se validen y afiancen los mecanismos de reconfiguración, es incorporar el dinamismo arquitectónico, o evolución de tipos, mediante el cual los sistemas estarán capacitados para modificarse a sí mismos: los elementos arquitectónicos podrán ver modificado su tipo y comportamiento en tiempo de ejecución. Esta etapa es muy compleja, pues intervienen muchos factores: (i) la consistencia de los cambios, (ii) el instante de aplicación, (iii) las dependencias entre elementos arquitectónicos, (iv) la posible degradación del sistema, y (v) la coexistencia o no de versiones. En el modelo PRISMA, el dinamismo arquitectónico deberá aplicarse a nivel de componentes (componentes simples), de tal forma que se pueda consultar y modificar la estructura de un componente, y de forma similar a nivel de puertos, aspectos y *weavings*. Sin embargo, para mantener la consistencia será necesario modelar de algún modo la población de instancias del elemento arquitectónico modificado, con el objetivo de propagar los cambios. En este punto también deberá decidirse cuándo deben aplicarse los cambios y si se deben mantener versiones.

Finalmente, como un trabajo futuro a abordar de forma progresiva durante el desarrollo de este trabajo será la validación y comprobación de las ideas y mecanismos propuesto con un caso de estudio real. Dicho caso de estudio deberá pertenecer a un dominio de aplicación que necesite los requisitos de evolución enunciados en este trabajo para su correcto funcionamiento.

Agradecimientos. Este trabajo ha sido financiado por el CICYT (Centro de Investigación Científica Y Tecnológica), Proyecto DYNAMICA (DYNamic and Aspect-Oriented Modeling for Integrated Component-based Architectures), TIC 2003-07804-C05-01.

Referencias

1. Allen, R., Douence, R., Garlan, D.: Specifying and analyzing dynamic software architectures. In proc. of First International Conference on Fundamental Approaches to Software Engineering (FASE'98), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS'98). Lisbon, Portugal (March 28 – April 4 1998) 21-37
2. AOSD. Aspect-Oriented Software Development, http://aosd.net (2005)
3. Canal, C., Pimentel, E., Troya, J.M.: Specification and refinement of dynamic software architectures. In proc. of First Working IFIP Conference on Software Architecture (WICSA'99). San Antonio, Texas, USA (February 22-24 1999) 107-126
4. Costa, C., Pérez, J., Ali, N., Carsí, J.Á, Ramos, I.: PRISMANET middleware: Soporte a la evolución dinámica de arquitecturas software orientadas a aspectos. In proc. of X Jornadas De Ingeniería Del Software y Bases De Datos (JISBD'05). I Congreso Español de Informática (CEDI'05). Granada, Spain (september 2005) 27-34
5. Cuesta, C.E.: Arquitectura de Software Dinámica basada en Reflexión. Capítulo 3. Tesis Doctoral. Biblioteca Virtual Miguel de Cervantes, (2002)
6. Cuesta, C.E., Fuente, P.d.l., Barrio-Solárzano, M.: Dynamic coordination architecture through the use of reflection. In proc. of 2001 ACM Symposium on Applied Computing. Las Vegas, Nevada, United States 2001) 134-140
7. D'Souza, D., Wills, A.: Objects, Components and Frameworks with UML: The Catalysis approach. Addison-Wesley. (1999)
8. Greenfield J., Short K, Cook S., and Kent S. Software Factories. Wiley Publising Inc., 2004.
9. Harrison, W.H., Ossher, H.L., Tarr, P.L.: Asymmetrically Vs. Symmetrically Organized Paradigms for Software Composition. IBM Research Report, RC22685 (W0212-147) Thomas J. Watson Research Center, IBM (December 2002)
10. Kramer, J., & Magee, J.: Dynamic Configuration for Distributed Systems. Software Engineering, IEEE Transactions on, SE-11 (1985) 424-436
11. Magee, J., & Kramer, J.: Dynamic structure in software architectures. In: Foundations of Software Engineering - Proceedings of 4th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE-4). Lecture Notes in Computer Science, Vol. 21. ACM Press, (1996) 3-14
12. Object Management Group. Model Driven Architecture Guide, 2003 http://www.omg.org/docs/omg/03-06-01.pdf
13. Pérez, J., Ali, N., Carsí, J.Á, Ramos, I.: Designing Software Architectures with an Aspect-Oriented Architecture Description Language. In proc. of 9th International Symposium on Component-Based Software Engineering (CBSE06). Lecture Notes in Computer Science, Vol. 4063. Springer, Västerås, Sweden (June 29 - July 1 2006) 123-138
14. Pérez, J., Ali, N., Carsí, J.Á, Ramos, I.: Dynamic evolution in aspect-oriented architectural models. In proc. of 2nd European Workshop on Software Architecture

(EWSA 2005). Lecture Notes in Computer Science, Vol. 3527. Springer, Pisa, Italy (June 13-14 2005) 59-76

15. Pérez, J., Ali, N., Costa, C., Carsí, J.Á, Ramos, I.: Executing Aspect-Oriented Component-Based Software Architectures on .NET Technology. In proc. of 3rd International Conference on .NET Technologies. Pilsen, Czech Republic (June 2005) 97-108
16. Szyperski C.: Component software: beyond object-oriented programming, ACM Press and Addison Wesley, New York, USA. (1998)

Introducción al Cálculo-π con Prioridad *

Carlos E. Cuesta[1], Jenifer Pérez Benedí [2], and M. Pilar Romay[3]

[1] Kybele, Departamento de Lenguajes y Sistemas Informáticos
Universidad Rey Juan Carlos, Madrid
carlos.cuesta@urjc.es
[2] Issi, Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
jeperez@dsic.upv.es
[3] Departamento de Sistemas Informáticos
Universidad Europea de Madrid
pilar.romay@uem.es

Resumen En este trabajo se presenta πP, un álgebra de procesos con prioridad basado en el cálculo-π, diseñado para ser utilizado como la base formal del Lenguaje de Descripción de Arquitecturas Orientado a Aspectos de la plataforma Prisma. Se comienza definiendo el modelo de prioridad, para desarrollar a continuación la sintaxis del cálculo y su semántica de reducción, dada por una relación de congruencia estructural y una semántica operacional estructurada de Plotkin. Tras comentar brevemente un ejemplo formal completo, el trabajo termina indicando el uso del cálculo en Prisma.

Palabras clave: Cálculo-π, Prioridad Estática, Precesión Global, πP, Prisma.

1. Introducción

La expresividad y precisión de los formalismos de concurrencia, y en particular de las álgebras de procesos, ha hecho de éstas, en los últimos treinta años, algunas de las herramientas más útiles a la hora de manejar la complejidad y el indeterminismo de los sistemas concurrentes y distribuidos. En los últimos años, la definición de cálculos móviles, en particular el cálculo-π, han supuesto un espaldarazo definitivo a la aplicación real de estos cálculos, más allá de sus bondades como formalismo teórico para la descripción y verificación de sistemas. Más aún, el uso habitual de álgebras de procesos como soporte formal básico de diversos Lenguajes de Descripción de Arquitectura, dentro del campo de la Arquitectura de Software, ha acercado a estos modelos a su aplicabilidad real, dentro de un proceso de Ingeniería de Software.

Fue precisamente durante el desarrollo de un lenguaje de este tipo, el LdaOa del modelo arquitectónico Prisma, en el que se hizo evidente que el soporte formal utilizado, basado en el cálculo-π, aunque adecuado en otros aspectos, necesitaba de una extensión que le hiciera capaz de describir adecuadamente ciertas situaciones.

* Este trabajo ha sido parciamente financiado por el Ministerio de Educación y Ciencia dentro del marco del Proyecto MCYT-TIC2003-07804-C05-01 (Dynamica, subproyecto Prisma).

En particular, se necesitaba incorporar una noción de *prioridad*, que permitiera definir sin ambigüedad la emisión de señales de *emergencia* y la gestión de las mismas en el análogo arquitectónico de una *interrupción* o una *excepción*. Cuando en un sistema real, como por ejemplo un sistema robótico, se produce una señal de emergencia, toda tarea ha de ser dejada de lado para atender a la interrupción en primer lugar. Sin embargo, ninguno de los LDA existentes, hasta la fecha, soportaba la definición de un modelo de precedencia en el que se priorizase la ejecución de una tarea privilegiada.

Para dotar a PRISMA de este soporte, se ha hecho preciso definir una extensión del cálculo-π básico que incluya mecanismos de prioridad análogos a los existentes en otras álgebras de procesos. El resultado de esta extensión es el formalismo denominado πP, a cuya definición y explicación se dedica este artículo.

Aunque se pretende proporcionar en este trabajo una descripción suficientemente compleja del cálculo-π con prioridad, no se pretende entrar en excesivos detalles formales. Se presentarán, por tanto, los aspectos imprescindibles para la definición del cálculo, pero no se tratarán en profundidad otros aspectos, especialmente en lo relativo a la parte semántica, que caen fuera de los objetivos de este trabajo, y que precisan todavía de un análisis más detallado.

2. Algebras de Procesos con Prioridad

Desde su aparición, las álgebras de procesos han sido uno de los formalismos más populares para la descripción de sistemas concurrentes. Los sistemas “clásicos”, como CCS, CSP o ACP, muestran una serie de características comunes que pueden considerarse como el modelo básico. Sin embargo, a menudo es necesario considerar aspectos adicionales, tales como rendimiento o probabilidad, prioridades, localización y distribución o el paso del tiempo. Añadir estos aspectos suele implicar importantes modificaciones en la semántica, hasta el punto de definir álgebras extendidas que suelen tratarse como dialectos totalmente independientes.

Lo mismo ha ocurrido con las álgebras de procesos móviles, y en particular con el cálculo-π. Aunque existen diversas variantes e intensificaciones del cálculo básico, tal como se describe por ejemplo en [11], las extensiones que incluyen estos aspectos son consideradas como dialectos diferenciados. El caso más conocido es probablemente el de $S\pi$, el Cálculo-π Estocástico [10], de amplia difusión en Bioinformática.

Debido a la naturaleza intrínsecamente móvil del cálculo-π, ampliar su modelo se considera en general más difícil. Por ello, muchas de las extensiones de otras álgebras no han sido aplicadas en los cálculos nominales; la única excepción notable son las relativas a localización, ya que la distribución se plantea como un corolario necesario de la movilidad. En este contexto, destaca en particular el Cálculo de Ambientes [3].

Entre las extensiones que nunca han sido aplicadas a un cálculo nominal están las relativas a *prioridad*. Existe toda una larga serie de precedentes en el diseño de álgebras de procesos con prioridad, entre las que destacan los trabajos de Cleaveland y Lüttgen, por una parte, y los de Camilleri y Winskel, por otra. Se dispone de un resumen muy completo de todos estos enfoques y sus diferencias [4], que se utilizará como referencia básica en el resto de este trabajo. Pero, hasta donde alcanza nuestro conocimiento, ésta

es la primera ocasión en la que se describe un cálculo de procesos móviles con prioridad. En concreto, se define una variante del cálculo-π, que recibirá el nombre de πP.

En [4] se enumeran los distintos modelos de prioridad existentes, que se pueden agrupar en las combinaciones de dos propiedades esenciales. Éstas son el *tipo* de esquema de prioridad, y el *alcance* de la precesión[4] causada por el mecanismo de prioridad. Concretamente, el esquema de prioridad puede ser *estático* o *dinámico*, y la precesión puede ser *global* o *local*. En la literatura existen precedentes para casi todas las combinaciones posibles [4]; pero sin duda, el modelo más extendido y estudiado es el que tiene *prioridad estática* y *precesión global*, como será también el caso de πP.

Brevemente, se dice que un modelo de prioridad es *estático* si todo canal definido tiene siempre la misma prioridad, durante toda su existencia, y es *dinámico* en el caso contrario. Por su parte, se dice que la precesión tiene un alcance *global* si la prioridad de las acciones de un proceso puede hacer que precedan a las acciones de *otro* proceso; y es *local* cuando los límites del ámbito de un proceso han de ser respetados.

El modelo de prioridad estática, en el caso de $\mathrm{CCS}^{\mathrm{sg}}$ [4], se basa en la idea de que los niveles de prioridad se asignan a los puertos, esto es, a los canales que comunican entre sí los procesos. En el caso sintáctica y semánticamente más simple del cálculo-π, se puede decir que los niveles de prioridad se asignan a los *nombres*. Dada la naturaleza nominal del cálculo-π, las consecuencias de esta afirmación son mayores que en el caso de cualquier variante de CCS. Esto permite obtener un resultado similar, aunque más expresivo, pero con una formulación sintáctica y semántica más sencilla.

3. Sintaxis de πP

La sintaxis del *cálculo-π poliádico con prioridad*, πP, se define en la Tabla 1. En términos generales, puede decirse que la sintaxis es casi idéntica a la del cálculo-π estándar; tan sólo hay una diferencia con la formulación básica, referida como es lógico a la definición del nivel de prioridad de un canal.

Existen múltiples presentaciones más o menos equivalentes del cálculo-π; ésta se ha basado, con algunas modificaciones, en la introducción de Parrow [7], a la que nos remitimos para mayores detalles. Puede observarse que se definen los prefijos de comunicación básicos y los operadores de composición y restricción habituales, además de los operadores de comparación propios de la variante poliádica.

En algunos casos, se muestran dos notaciones diferentes para una misma operación, separadas por el símbolo $\rightsquigarrow$. Tan sólo se pretende indicar que cualquiera de ambas podría ser utilizada, indistintamente; no obstante, a lo largo de este trabajo tan sólo se empleará, en cada caso, la primera notación indicada. Dado que ya se define el la invocación de proceso, el operador de replicación no es necesario, pero se incluye en la Tabla 1, por afán de exhaustividad.

El único operador afectado por la inclusión del modelo de prioridad es el operador de *restricción*, dado que es el único capaz de *crear* a un nombre. En πP, además de

[4] Se utiliza *precesión* como traducción del complejo término inglés *pre-emption*. En general, se dice que una acción se realiza de manera *preemptive* (*precedente*) si, debido a su mayor prioridad, *precede a* –se ejecuta antes que– otras acciones menos prioritarias.

Prefijos	$\alpha ::=$	$c!(\widetilde{x}) \rightsquigarrow \overline{c}\langle\widetilde{x}\rangle$	Emisión (salida de datos)
		$c?(\widetilde{x}) \rightsquigarrow c(\widetilde{x})$	Recepción (entrada de datos)
		τ	Silencio (acción interna)
Agentes	$P ::=$	$\mathbf{0}$	Nulo
		$\alpha.\, P$	Prefijo (Acción)
		$P + P$	Suma (Elección)
		$P \mid P$	Composición Paralela
		$(\nu\, x{:}k)\, P$	Restricción (Nombre Nuevo)
		$[x = y]P$	Comparación
		$[x \neq y]P$	Comparación Negativa
		$A\langle y_1, y_2, \ldots, y_n\rangle$	Invocación de Proceso
		$!P \rightsquigarrow *P$	Replicación
Definiciones	$A(x_1, x_2, \ldots, x_n) \triangleq P$		para $i \neq j \Rightarrow x_i \neq x_j$

Tabla 1. Sintaxis del Cálculo-π con Prioridad, πP

crearlo le asigna una prioridad k determinada. En adelante se utilizará la notación postfija $: k$ para indicar que un canal o acción tiene este nivel de prioridad. En el apartado siguiente se discuten estos aspectos en mayor detalle.

Se introduce también, como es habitual, la noción de *congruencia estructural*, que permite definir sintácticamente algunas de las propiedades semánticas del modelo. Se define la congruencia estructural ($\equiv$) como la mínima relación de congruencia que satisface todos los axiomas definidos en la Tabla 2.

La presentación es análoga a otras ya existentes, y manifiesta propiedades similares; por ejemplo, al igual que en otras variantes del cálculo-π, tanto la suma como el producto (la composición paralela) definen algebraicamente un grupo abeliano. Puede parecer que esta presentación es muy extensa en comparación con otras; por ejemplo, la presentación de Milner en [6] tan sólo enumera los tres primeros axiomas de la restricción. En realidad, la mayoría de los axiomas adicionales se refieren, a su vez, a operaciones propias de la variante poliádica, como la comparación o la replicación.

Puede comprobarse que, como es habitual en las presentaciones del cálculo-π, la notación $\mathtt{fn}(P)$ representa al conjunto de los nombres libres (no ligados) del proceso P, y que por tanto podrían ser *capturados* por el contexto del mismo. De manera complementaria, la notación $\mathtt{bn}(P)$ representa al conjunto de los nombres ligados del proceso P, que se encuentran en el ámbito de una recepción o una restricción, y que pueden ser libremente modificados por α-conversión (como se indica en la Tabla 2).

Finalmente, señalar que la propiedad de extrusión del ámbito, tan característica del cálculo-π, no resulta en absoluto afectada por la definición de prioridad. Aparece en la congruencia explícitamente, como tercer axioma de la restricción. Esto significa, en definitiva, que la movilidad del cálculo se mantiene intacta.

α-Conversión

$[x]P$	$\equiv$	$[y]P\{y/x\}$	$\forall x \in \mathtt{bn}(P) \wedge y \notin (\mathtt{bn}(P) \cup \mathtt{fn}(P))$

Axiomas para Composición Paralela y Suma

$P \mid Q$	$\equiv$	$Q \mid P$	Conmutatividad
$(P \mid Q) \mid R$	$\equiv$	$P \mid (Q \mid R)$	Asociatividad
$P \mid \mathbf{0}$	$\equiv$	P	Elemento Neutro
$P + Q$	$\equiv$	$Q + P$	Conmutatividad
$(P + Q) + R$	$\equiv$	$P + (Q + R)$	Asociatividad
$P + \mathbf{0}$	$\equiv$	P	Elemento Neutro

Axioma de Sustitución *(unfolding)*

$A(\widetilde{y})$	$\equiv$	$P\{\widetilde{y}/\widetilde{x}\}$	para $A\langle\widetilde{x}\rangle \triangleq P$

Axioma de Replicación

$$!P \equiv P \mid !P$$

Axiomas de Restricción (Extensión del Ámbito)

$(\nu\, x)\, \mathbf{0}$	$\equiv$	$\mathbf{0}$	
$(\nu\, x)(\nu\, y)\, P$	$\equiv$	$(\nu\, y)(\nu\, x)\, P$	
$(\nu\, x)\, (P \mid Q)$	$\equiv$	$P \mid (\nu\, x)\, Q$	si $x \notin \mathtt{fn}(P)$
$(\nu\, x)\, (P + Q)$	$\equiv$	$P + (\nu\, x)\, Q$	si $x \notin \mathtt{fn}(P)$
$(\nu\, z)\, [x = y]P$	$\equiv$	$[x = y](\nu\, x)\, P$	si $z \neq x \wedge z \neq y$
$(\nu\, z)\, [x \neq y]P$	$\equiv$	$[x \neq y](\nu\, x)\, P$	si $z \neq x \wedge z \neq y$

Tabla 2. Definición de la Congruencia Estructural para πP

3.1. Interpretación del Modelo de Prioridad de πP

De entre las alternativas señaladas en la sección 2, se ha elegido para πP un modelo de *prioridad estática* con *precesión global*. Esta elección se debe a varios motivos: en primer lugar, es el modelo más simple e intuitivo. En segundo lugar, es el modelo más extendido en otras álgebras de procesos, por lo que sus propiedades e implicaciones son mejor conocidas. En tercer lugar, las aportaciones de otros modelos no compensan su mayor complejidad; ni la prioridad dinámica ni la precesión local son especialmente útiles de cara a la aplicación real de πP.

Por tanto, como ya se ha indicado, en πP la asignación de prioridades es estática, y una acción prioritaria de cualquier proceso toma precedencia sobre cualquier otra.

Cada una de estas propiedades tiene una serie de consecuencias sobre la semántica del cálculo, similares pero *no* idénticas a las de un álgebra de procesos de naturaleza no nominal. A continuación se examinan las consecuencias de ambas elecciones.

En primer lugar, respecto al modelo de *prioridad estática*. Como ya se ha indicado, esto indica, en resumen, que un canal (un *nombre*) de πP tiene siempre el mismo nivel del prioridad. Como se detalla en el apartado siguiente, esto no tiene por qué significar que la asignación de prioridades es rígida. Al contrario, se adapta perfectamente a la naturaleza dinámica del cálculo. Incluso puede afirmarse que el hecho de que el esquema de prioridades no fluctúe libremente facilita su uso en un contexto tan cambiante como el definido por el cálculo-π, y de hecho podría servir de referencia.

En las diversas extensiones con prioridad de CCS [4], se dice que para que la comunicación entre dos procesos sea posible, los puertos involucrados –esto es, un nombre y su conombre– han de tener la misma prioridad. Un puerto con prioridad k no puede interactuar con otro cuya prioridad sea $(k + 1)$ o $(k - 1)$. En el cálculo-π, la interpretación es mucho más simple: no se habla de una comunicación que utiliza dos puertos, sino de dos procesos que comparten un canal. No se trata de dos puertos con la misma prioridad, sino de un único *nombre*, con una prioridad asignada.

Se puede inferir inmediatamente, también, que si un canal tiene una prioridad determinada, toda *acción* en la que este canal esté involucrado –es decir, todo prefijo basado en ese nombre– tiene exactamente el mismo nivel de prioridad. Se puede hablar, por tanto, de la prioridad de una acción, a pesar de que inicialmente la asignación se refiere sólo a los nombres. Más formalmente:

$$\alpha : k \in \mathcal{A}_k(P) \iff x : k \in (\mathtt{fn}(P) \cup \mathtt{bn}(P)) \wedge x \in \alpha \tag{1}$$

donde $\mathcal{A}_k(P)$ es el conjunto de las acciones que puede realizar el proceso P.

Del mismo modo, y de manera análoga a lo que ocurre en otras álgebras, cuando se produce una comunicación sobre un canal c de prioridad k, se consumen un prefijo de emisión y uno de recepción en dos procesos, dando lugar a una acción interna (τ) en la composición que los contiene. Como se ha visto, ambos prefijos son acciones de prioridad k: por esto mismo, se dice que la acción interna resultante produce un *silencio* con la misma prioridad ($\tau : k$). Esto será esencial para la definición semántica del cálculo, ya que permite expresar la precesión entre acciones. Véase al respecto la regla $\textsc{Comm}_k$ de la semántica operacional, tal como se desarrolla en la Tabla 5.

Dado que la prioridad de los prefijos se deduce de la propia del canal, y dado que la mayoría de los agentes sólo definen formas de composición, se muestra que, en realidad, sólo *una* operación de la sintaxis necesita hacer referencia a la prioridad, tal como indica de hecho en la definición de la Tabla 1. Se trata de aquélla que asigna al canal la prioridad que tiene durante toda su existencia; en definitiva, la propia operación de *creación* del canal, esto es, la restricción de ámbito (νx).

En resumen, un nombre es creado con una prioridad determinada, y a partir de este momento puede inferirse la misma prioridad en todas las acciones en las que está involucrado. No es necesario, por tanto, más que indicarlo en este momento, aunque se admitirá el uso de la notación postfija ($: k$) cuando se desee *anotar* de manera específica la prioridad que tiene una acción dada.

En la versión de este cálculo que se usa en Prisma, en lugar de indicar la prioridad de un canal durante su creación, dispone de un operador simiar que específicamente especifica la asignación de prioridad a una acción, tal como se expone en la sección 6. Se hace de este modo por considerar que esta presentación es más intuitiva; pero se hace en todo momento equivalente a la que aquí se ha desarrollado. La semántica de Prisma no permitiría, en ningún caso, la asignación de dos prioridades diferentes al mismo servicio, ya que esto violaría el modelo de prioridad estática definido.

Respecto a la propiedad de *precesión global*, es evidente que simplifica notablemente el uso del cálculo. Por otra parte, limitar la difusión de las prioridades según el ámbito de un proceso, en un modelo estático como el de CCS, puede tener sentido; pero en un cálculo móvil con extrusión de ámbito, los limítes no están tan definidos. Es claro que implementar un modelo de precesión local sólo tendría sentido si se combina con un modelo de localización adecuado. De nuevo, en este caso, lo que sería razonable es lantear la combinación del modelo de prioridad no con el cálculo básico, sino con una de sus variantes distribuidas; en particular, con el ya citado Cálculo de Ambientes [3]. En cualquier caso, πP se concibe como un modelo básico de prioridad, y en este contexto, sólo la precesión global tiene sentido.

Finalmente, se ha de hablar de la naturaleza de los niveles de prioridad. Los niveles de prioridad pueden definirse a partir de cualquier dominio que defina un orden total; por simplicidad se asume siempre que éste es el de los números naturales, $\mathbb{N}$.

Por tanto, se define el conjunto de los niveles de prioridad, $\mathcal{N}$, como un intervalo *finito*, aunque sin límite definido, de $\mathbb{N}$. Se define el conjunto de acciones $\{\Lambda_k \mid k \in \mathcal{N}\}$ como una familia $\mathcal{N}$-indexada, compuesta por un conjunto infinito contable de nombres (o pares de puertos, según se prefiera). Todo nombre (canal) de πP es creado con una prioridad y pertenece al conjunto Λ_k correspondiente.

En el caso convencional se asume que $\mathcal{N} = \{ 0, p \}$, esto es, que el número 0 es uno de los extremos del intervalo, siendo p un valor cualquiera, de tamaño indefinido. Se adopta la convención de que *"a menor número, mayor prioridad"*. Por tanto, 0 es el valor con la prioridad más alta. Siguiendo [4], se puede reducir el caso general de un sistema con $\mathcal{N}$ niveles de prioridad al caso particular de *dos* niveles (por tanto, $p = 1$). Aunque desde un punto de vista algebraico no supone una verdadera diferencia, en πP se ha decidido que no es necesario limitar este aspecto.

Por tanto, una especificación en πP tiene el número de niveles de prioridad dado por el intervalo $\mathcal{N}$ definido. En ausencia de otros detalles, se asume que 0 es la prioridad máxima y que todo nombre se crea con la prioridad *mínima* (p), siendo ésta el nivel "convencional" de trabajo del cálculo. De este modo, siempre es posible incorporar canales de mayor prioridad con posterioridad, si se hacen realmente necesarios.

3.2. Extensión de la Sintaxis

Podría parecer que la elección de un modelo de prioridad estática limita las posibilidades del formalismo, especialmente considerando que un cálculo móvil se considera apto para la descripción de sistemas especialmente dinámicos. Pero, como se ha dicho, únicamente la *asignación* de prioridades es estática, lo que sólo significa que un canal tiene siempre la misma prioridad.

Priorización

$\lceil(\alpha:k) \equiv \alpha:(k+1)$ para $(\alpha:k) \not\equiv (\tau:k)$

De-priorización

$\lfloor(\alpha:k) \equiv \alpha:(k-1)$ para $(k > 0) \wedge (\alpha:k) \not\equiv (\tau:k)$

Tabla 3. Operadores de Cambio de Prioridad

Como se ha dicho, esto resulta especialmente conveniente en un cálculo móvil, en el que la estructura del sistema se altera de manera continua, ya que la asignación de prioridades puede servir de referencia. Sin embargo, podría parecer que tener un esquema de prioridad estático, cuando la propia estructura es dinámica, dota al sistema de cierta rigidez, y supone una coerción innecesaria.

Sin embargo, esto no es cierto. Que la asignación de prioridades a los canales sea estática no significa que la asignación a las *acciones* tenga que serlo. Es cierto que, como se ha indicado en el apartado anterior, la prioridad de una acción se infiere a partir de la prioridad del canal involucrado; sin embargo, se puede alterar el modo en que se realiza esta inferencia, sin tener que modificar el modelo.

En ocasiones puede surgir la necesidad de que un canal que normalmente tiene prioridad k se vea involucrado en una actividad con una prioridad distinta. Por ejemplo, en un momento concreto puede ser necesario utilizar un canal convencional para realizar una transmisión de alta prioridad. No obstante, éste no es motivo suficiente para optar por un modelo dinámico. De hecho, una vez que la necesidad puntual ha pasado, suele ser preferible que el canal utilice de nuevo su prioridad habitual.

Por tanto, lo que realmente se necesita es disponer de algún método para modificar temporalmente la prioridad de una *acción*. Con este fin se pueden definir operadores de *priorización* ($\lceil\alpha$) y *depriorización* ($\lfloor\alpha$), definidos tal como se indica en la Tabla 3. Estos operadores engloban a una acción y modifican su nivel de prioridad, subiéndolo o bajándolo en una unidad, y afectando únicamente dentro de su ámbito de actuación.

Incorporar estos operadores al cálculo no implica realizar ningún tipo de cambio ni en su semántica operacional básica, ni en el modelo de prioridad elegido. Los canales siguen teniendo una asignación de prioridad totalmente estática; lo único que se altera es el modo en que se selecciona la acción de mayor prioridad, que ahora habría de tener en cuenta la existencia de estos operadores.

En cualquier caso, su definición no es esencial para la definición de πP, por lo que se ha preferido no incluirlos como parte de la sintaxis básica. Por otra parte, no se plantea su utilización efectiva a nivel arquitectónico, al menos por el momento; por este motivo, no han sido incorporados en Prisma.

Así pues, los operadores de priorización no se mencionarán en el resto de este trabajo, ni en la definición semántica del cálculo. Se han descrito únicamente para mostrar que la elección de un modelo de prioridad estática no compromete en absoluto el dinamismo del cálculo. No obstante, se ha de señalar para incorporarlos a esta definición

$\mathcal{I}(\alpha.\,P) = \{\alpha\}$	$\mathcal{I}(!P) = \mathcal{I}(P)$
$\mathcal{I}((\nu\, x)\, P) = \mathcal{I}(P)$	$\mathcal{I}([x = y]P) = \mathcal{I}(P)$ si $(x = y)$, $\emptyset$ si $(x \neq y)$
$\mathcal{I}(P + Q) = \mathcal{I}(P) \cup \mathcal{I}(Q) = \mathcal{I}(P,\, Q)$	$\mathcal{I}([x \neq y]P) = \mathcal{I}(P)$ si $(x \neq y)$, $\emptyset$ si $(x = y)$
$\mathcal{I}(P \mid Q) = \mathcal{I}(P) \cup \mathcal{I}(Q) \cup \{\tau{:}k \mid \bigcap_{\mathcal{N}}^{k} (\mathcal{I}(P_k), \mathcal{I}(\overline{Q_k})) \neq \emptyset\}$	

Tabla 4. Axiomas para un Conjunto Inicial de Acciones en πP

bastaría con añadir dos esquemas de regla, uno para cada operador. El resto del modelo no sufriría ningún tipo de alteración.

4. Sobre la Semántica de πP

Para una presentación formal de la semántica operacional de un álgebra con un modelo de prioridad resulta necesario definir el concepto de *conjunto inicial de acciones* de un proceso, $\mathcal{I}(P)$. Formalmente, se define como el conjunto mínimo que satisface los axiomas de la Tabla 4. Intuitivamente, es el conjunto de todas las acciones que pueden ser la primera en una invocación de P, esto es, todas las candidatas a ser la próxima acción en realizarse. Este conjunto se utiliza para describir el ámbito de precesión en las reglas de la semántica definida en la Tabla 5, de modo que sólo las acciones de máxima prioridad puedan ser seleccionadas.

Por simple compacidad, se utilizará la notación abreviada $\mathcal{I}(P, Q)$ como equivalente a $(\mathcal{I}(P) \cup \mathcal{I}(Q))$, tal como ya se ha indicado en la Tabla 4.

La semántica de πP se presenta en la Tabla 5 como una Semántica Operacional Estructurada en el estilo de Plotkin. La formulación concreta, al igual que la de la sintaxis, se basa en la presentación de Parrow [7] para el cálculo-π básico, adecuadamente extendida con la noción de prioridad. El planteamiento es conceptualmente similar al dialecto CCS^{sg} de Cleaveland y Lüttgen, pero semánticamente más simple.

Las reglas con subíndice, es decir, Prefix, Sum, Par, Comm y Open no son realmente reglas de inferencia, sino *esquemas de reglas*. Cada una representa, en realidad, a tantas reglas como niveles define la cardinalidad de $\mathcal{N}$. Así, por ejemplo, en el caso particular en el que $\mathcal{N} = \{0, 1\}$, cada una de estos esquemas representa a *dos* reglas de inferencia, una para la prioridad máxima ($k = 0$) y otra para la prioridad mínima ($k = 1$). En [4] se presenta una semántica operacional estructurada para CCS^{sg} en el que se sigue esta formulación expandida; pero la notación abreviada que aquí se ha empleado permite referirse al caso general en el que existe un número indeterminado, aunque finito, de niveles de prioridad. Se hace posible, además, mantener sin cambios aquellas reglas de inferencia –Struct, Match, Mismatch y, tal vez de manera más sorprendente, Res– que en realidad no resultan afectadas por la adición de un modelo de prioridad.

No se entrará en detalle a explicar cada una de estas reglas, ya que en definitiva son muy similares a las del cálculo-π básico, y siguen una formulación bastante estándar, excepto en lo relativo a las condiciones de los esquemas de reglas. Así, Struct muestra la aplicación de la congruencia estructural definida en la Tabla 2. Sum y Par, respectivamente, la elección priorizada de un subproceso y su desarrollo en paralelo. Prefix y

Struct	$\dfrac{P' \equiv P,\ P \xrightarrow{\alpha} Q,\ Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$	
Prefix$_k$	$\dfrac{-}{\alpha{:}k.P \xrightarrow{\alpha:k} P}$	$\forall k \in \mathcal{N}$
Sum$_k$	$\dfrac{P \xrightarrow{\alpha:k} P'}{P + Q \xrightarrow{\alpha:k} P'}$	$\forall k \in \mathcal{N},\ \nexists j \in \mathcal{N} : (j < k) \wedge \tau{:}j \in \mathcal{I}(P, Q)$
Par$_k$	$\dfrac{P \xrightarrow{\alpha:k} P',\ \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha:k} P' \mid Q}$	$\forall k \in \mathcal{N},\ \nexists j \in \mathcal{N} : (j < k) \wedge \tau{:}j \in \mathcal{I}(P, Q)$
Comm$_k$	$\dfrac{P \xrightarrow{c:k?(x)} P',\ Q \xrightarrow{c:k!(u)} Q'}{P \mid Q \xrightarrow{\tau:k} P'\{u/x\} \mid Q'}$	$\forall k \in \mathcal{N},\ \nexists j \in \mathcal{N} : (j < k) \wedge \tau{:}j \in \mathcal{I}(P, Q)$
Match	$\dfrac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$	
Mismatch	$\dfrac{P \xrightarrow{\alpha} P',\ x \neq y}{[x \neq y]P \xrightarrow{\alpha} P'}$	
Res	$\dfrac{P \xrightarrow{\alpha} P',\ x \notin \alpha}{(\nu\, x)P \xrightarrow{\alpha} (\nu\, x)P'}$	
Open$_k$	$\dfrac{P \xrightarrow{a:k!(x)} P',\ a \neq x}{(\nu\, x)P \xrightarrow{a:k!(\nu x)} P'}$	$\forall k \in \mathcal{N},\ \nexists j \in \mathcal{N} : (j < k) \wedge \tau{:}j \in \mathcal{I}(P)$

Tabla 5. Semántica Operacional Estructurada para πP

Comm, la activación de una acción y el modo en que dos procesos se sincronizan en una comunicación. Match y Mismatch, las dos ramas de una estructura alternativa. Finalmente, Res y Open muestran la independencia de una restricción y la posibilidad de una extrusión de ámbito. Para una explicación más detallada, al margen de las prioridades, este trabajo se remite a la presentación de Parrow [7].

En lo que respecta a los cinco esquemas de reglas incluidos en la Tabla 5, se indica, en primer lugar, que todos han de instanciarse como una regla específica para cada nivel de prioridad ($\forall k \in \mathcal{N}$). En segundo lugar, y sólo en cuatro de ellos (Sum$_k$, Par$_k$, Comm$_k$ y Open$_k$) se especifica una condición más compleja. En ésta se especifica que la acción a la que se refiere la regla sólo se realiza si no puede desarrollarse ninguna otra acción con prioridad mayor. Es decir, literalmente, que si la regla tiene prioridad k, no existe

en el conjunto inicial de acciones $\mathcal{I}(P, Q)$ de los procesos involucrados ningún *silencio* con una prioridad j numéricamente *menor*, y por tanto con mayor precedencia.

Como se deduce de la regla $\textsc{Comm}_k$, y tal como se dijo en la sección 3.1, dos procesos no pueden comunicarse si sus acciones no tienen la misma prioridad. Esto es, a menos que compartan el mismo canal con la misma prioridad. Dada la naturaleza móvil del cálculo, la precesión puede desarrollarse en momentos incialmente no previstos. En cualquier caso, la ejecución de esta regla implica la realización de una acción interna, un silencio con el mismo nivel de prioridad que el canal involucrado. Por tanto, una sincronización de alta prioridad *precede* a cualquier otra acción, y sólo una acción de este tipo podrá inhibir a composiciones, elecciones y extrusiones de menor prioridad.

Esta formulación de la Semántica Operacional Estructurada no es la más compacta que se puede presentar para este dialecto del cálculo-π, pero en cualquier caso es mucho más breve que la de las variantes análogas de CCS descritas por Cleaveland *et al.* [4]. Esto se debe, principalmente, a que la semántica del cálculo-π es más simple que la de CCS, al tiempo que más expresiva; y debido a esto, la extensión con prioridad del álgebra resulta mucho más directa. Así, CCS^{sg} necesita tres reglas –duplicadas– para definir la comunicación entre procesos en un modeloq con prioridad, mientras que en πP, como en el cálculo-π, basta con un único esquema de regla ($\textsc{Comm}_k$). Del mismo modo, allí era preciso definir dos reglas –igualmente duplicadas– para la semántica de la elección, mientras que aquí basta con una sola ($\textsc{Sum}_k$); la conmutatividad de la suma ha sido ya definida por la relación de congruencia estructural (véase la Tabla 2).

Para profundizar en la semántica de πP habría de realizarse un análisis detallado de la noción de *equivalencia* entre sus procesos, ya que como resulta evidente, la definición de prioridad le afecta de manera significativa. Así, por ejemplo, en el caso de CCS^{sg}, Cleaveland y Lüttgen [4] demostraron que la equivalencia se mantiene para la bisimulación fuerte, pero no para la bisimulación débil. Dada la similitud de la definición de una equivalencia observacional para πP, se puede esperar un resultado parecido; no obstante, este tipo de comprobaciones, así como la extensión a otros tipos de bisimulación más específicos, requieren todavía un estudio más exhaustivo.

Otro aspecto particularmente interesante sería el dado por la comparación con el modelo semántico del cálculo-π estocástico, basado en la noción de computación por demostración [10]. Una diferencia fundamental consiste en que en éste el uso de probabilidades restringe el indeterminismo –esto es, aumenta el determinismo– únicamente en las elecciones –es decir, en las sumas–. En cambio, la condición básica del cálculo con prioridad no se aplica únicamente en la elección, sino en toda operación de composición de procesos. A diferencia del modelo probabilístico, que se bifurca en las decisiones, el principio de precesión siempre es conservado.

5. Ejemplo: Tolerancia a Fallos en una Red P2P

Con el fin de ilustrar el uso de este cálculo, en esta sección presentamos un pequeño ejemplo de uso del mismo. Aunque este ejemplo se basa en un caso real, la presentación que se hace aquí no intenta ser completa ni exhaustiva, sino únicamente ilustrar el modo en que se utiliza la prioridad para implementar una señal de emergencia, esto es, el equivalente distribuido de una *interrupción*.

$$
\begin{array}{rcl}
E(\underline{h_1}, \underline{h_2}, \underline{c_1}, \underline{c_2}) & \triangleq & \underline{h_1}?(\underline{f}).\ (\nu\, \underline{g})(\ \underline{c_2}!(\underline{g}).\ \underline{g}?(n, \underline{e}).\ \underline{f}!(n, \underline{e}).\ E\langle \underline{h_1}, \underline{h_2}, \underline{c_1}, \underline{c_2}\rangle) \\
& & +\ \underline{h_2}?(\underline{f}).\ (\nu\, \underline{g})(\ \underline{c_1}!(\underline{g}).\ \underline{g}?(n, \underline{e}).\ \underline{f}!(n, \underline{e}).\ E\langle \underline{h_1}, \underline{h_2}, \underline{c_1}, \underline{c_2}\rangle) \\
\\
R(\underline{h}, \underline{c}, r, \underline{d}) & \triangleq & r!(m).\ \tau.\ R\langle \underline{h}, \underline{c}, r, \underline{d}\rangle \\
& & |\ (\ \underline{c}?(\underline{g}).\ \underline{g}!(r, \underline{d}).\ R\langle \underline{h}, \underline{c}, r, \underline{d}\rangle \\
& & \ +\ (\nu\, \underline{f})(\ \underline{h}!(\underline{f}).\ \underline{f}?(n, \underline{e}).\ \underline{d}!(n, \underline{e}).\ \mathbf{0}\ )) \\
\\
A(r, \underline{d}) & \triangleq & r?(m).\ \tau.\ A\langle r, \underline{d}\rangle \\
& & |\ \underline{d}?(n, \underline{e}).\ A\langle n, \underline{e}\rangle \\
\\
\textsc{Sistema} & \triangleq & (\nu\, \underline{h_1}, \underline{h_2}, \underline{c_1}, \underline{c_2})(\ E\langle \underline{h_1}, \underline{h_2}, \underline{c_1}, \underline{c_2}\rangle \\
& & \ |\ (\nu\, r_1, \underline{d_1})(\ R\langle \underline{h_1}, \underline{c_1}, r_1, \underline{d_1}\rangle\ |\ C\langle r_1, \underline{d_1}\rangle\ ) \\
& & \ |\ (\nu\, r_2, \underline{d_2})(\ R\langle \underline{h_2}, \underline{c_2}, r_2, \underline{d_2}\rangle\ |\ C\langle r_2, \underline{d_2}\rangle\ )\ )
\end{array}
$$

Figura 1. Definición (Parcial) de una Red P2P de Emisoras (R) de *Streaming*

El sistema es una red P2P de Emisoras de audio en *streaming*, concebido de manera similar a PeerCast [2]. Se puede concebir con facilidad como la versión P2P de una red de emisoras de radio convencional. Básicamente, desde una emisora central (que no tiene por qué ser siempre la misma) se emite un programa de radio, que es difundido a través de una red de emisoras locales (repetidores); a cada uno de éstos se conectará un número indefinido de agentes receptores. Un detalle característico es que, en esta red, los *peers* actúan a un tiempo como emisores y receptores; pero esto no será relevante en el desarrollo de este ejemplo, ya que los roles funcionales se mantienen separados, y se especificarán como procesos separados.

Con el objeto de mostrar la utilidad del modelo de prioridad, en este ejemplo se examina únicamente uno de los mecanismos de esta red; a saber, el mecanismo de tolerancia a fallos. El diseño es bastante simple. En un momento dado, cualquiera de los *peers* que actúa como emisora local puede fallar o ser desconectado. Esto implicaría que todos los receptores clientes perderían también la conexión con el sistema. Pero dado que en la red hay múltiples emisoras actuando como repetidores, parece lógico que estos clientes sean simplemente conectados a otro *peer*.

El mecanismo utilizado se describe de manera simplificada en la especificación de la Figura 1; para facilitar su explicación, se facilita un esquema visual en la Figura 2. Por su propia concepción, el sistema P2P es del tipo centralizado; aunque los nodos se comunican directamente entre sí, existe un elemento controlador central (E) que se

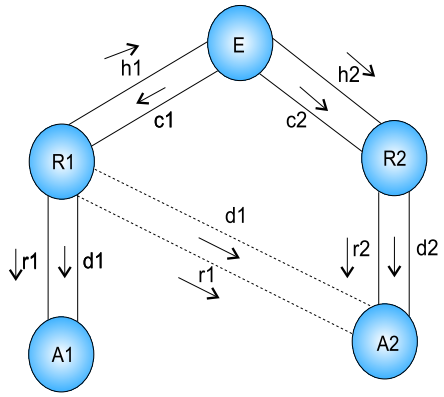


Figura 2. Cambio de Emisora (de R_2 a R_1) para un Cliente (A_2) en la Red de *Streaming*

encarga de la gestión del sistema. Por lo demás, los nodos de la red pueden actuar como emisora (R) o como cliente (A), roles que aquí se describen por separado[5].

Con el fin de simplificar el ejemplo, se utilizarán únicamente dos niveles de prioridad; por tanto, $\mathcal{N} = \{0, 1\}$. Esto significa que el nivel normal de funcionamiento es 1, y que todo canal creado con prioridad 0 es un canal de emergencia. En este caso, y siguiendo a [4], se puede adoptar también la notación $\underline{nb} = nb{:}0$ y $nb = nb{:}1$, de modo que se asume que todo nombre convencional tiene la prioridad mínima, y todo nombre subrayado designa a un canal de emergencia con la prioridad máxima.

Se adopta también la convención –innecesaria, pero a menudo conveniente– de que cuando se crea un nombre para enviarlo a través de un canal c de prioridad k, este nombre tenga también la misma prioridad k. Por tanto, toda señal enviada por un canal de emergencia será también una señal de emergencia. Intuitivamente, el motivo es evitar que la gestión de una "interrupción" activada desde un canal de alta prioridad se detenga a la mitad de su ejecución debido al uso de un canal intermedio de prioridad baja.

Como puede verse en la Figura 2, en el ejemplo se describe una red con un controlador central (E), dos emisoras (R_1 y R_2) y dos clientes de estas emisoras (A_1 y A_2). Se asume que los clientes reciben la emisión de *streaming* de las emisoras a través del canal r. No se muestra el origen de la emisión, por no ser relevante para el ejemplo.

El ejemplo muestra cómo cuando una emisora (R_2, en la Figura) va a desconectarse, emite una señal de emergencia ($\underline{h_2}$, de alta prioridad) al controlador (E) para solicitar la identificación ($\underline{c_1}$) de otra emisora (R_1). A continuación, transmite esa identificación a sus clientes (sólo A_2, en la Figura) a través de un canal de prioridad ($\underline{d}$). Los clientes se reconectan a la nueva emisora, y la transmisión del programa continúa.

[5] En el texto se ha preferido usar las denominaciones de controlador, emisora y cliente para describir a los distintos roles del sistema, por considerar que son más intuitivas en el contexto este ejemplo. Sin embargo, los nombres de los procesos correspondientes en la especificación de la Figura 1 se refieren a sus papeles en la definición original del sistema de *streaming*. Así, se refieren respectivamente a emisora central (E), repetidor (R) y agente (A).

El mecanismo de prioridad de πP se aplica con claridad tres veces en la emisora (R) y una en el cliente (A). Así, cuando una emisora solicita al controlador ($\underline{h}$) la identificación de otra, esto tiene precedencia sobre la emisión (r) de *streaming*. Lo mismo ocurre cuando se recibe la referencia de otra emisora ($\underline{c}$) y cuando ésta se comunica a los clientes ($\underline{d}$). En todos estos casos, el mecanismo de tolerancia a fallos *interrumpe* a la emisión normal

El ejemplo, aunque bastante simple e imperfecto en parte, es también extenso. Se ha elegido éste, en lugar de otros más breves, debido a que, por su propia naturaleza, expresa la necesidad de la definición de un cálculo móvil con prioridad o, aún mejor, de la disponibilidad de un álgebra de procesos con prioridad que, además, sea móvil.

En efecto, en este ejemplo no sólo se usa la *prioridad* para decidir que la reconexión de un cliente tiene precedencia sobre la emisión normal de *streaming*, sino que también se usa la *movilidad* para realizar esta misma reconexión; la definición de la estructura dinámica es propia de los cálculos móviles; pero además, aquí el modelo de prioridad interviene directamente en la modificación de la estructura. En resumen, este ejemplo *no* es representable en $\mathrm{CCS}^{\mathsf{sg}}$, ni en ninguna otra álgebra de procesos con prioridad conocida. La necesidad de describir situaciones como la del ejemplo justifica por sí sola la necesidad de definir cálculos con la expresividad demostrada por πP.

La especificación de la Figura 1 fue concebida principalmente para poder mostrar los aspectos relativos al modelo de prioridad, y cómo éste permite gestionar una situación crítica. Por tanto, se ha puesto especial énfasis en explicar el modo en que las señales de emergencia ($\underline{h}$, $\underline{d}$) tienen precedencia sobre la comunicación convencional (r). Otros aspectos del sistema, como la definición del mecanismo de *broadcast* de las emisoras (R), no serían realistas sin una descripción más compleja y detallada. Por ejemplo, con esta formulación, por ejemplo, los clientes compiten por la emisión en lugar de compartirla, lo que no coincide con los objetivos del sistema. No obstante, este trabajo se centra en otros aspectos, por lo que éstos no se consideran relevantes.

6. Aplicación del πP en Prisma

El enfoque de desarrollo Prisma [8] tiene como objetivo principal dar soporte al desarrollo de arquitecturas software orientadas a aspectos de sistemas complejos. Este enfoque proporciona un Lenguaje de Descripción de Arquitecturas Orientado a Aspectos (LdaOa) [9] para especificar formalmente e independientemente de tecnología dichos sistemas. Este LdaOa se basa en la Lógica Dinámica [5] y el cálculo-π poliádico [6] para modelar los cambios de estado y los procesos que acontecen en los componentes software, respectivamente.

Existen una gran variedad de dominios cuyos sistemas software manifiestan una gran complejidad. Concretamente, Prisma ha sido aplicado al dominio de los sistemas de tele-operación, ya que a diferencia de los ejemplos puramente académicos, este dominio proporciona problemas reales que han de resolverse en sistemas software de carácter industrial.

Los sistemas de tele-operación son sistemas de control que dependen del software para realizar sus operaciones. Prisma ha sido aplicado a sistemas de tele-operación robóticos, cuya funcionalidad consiste en desarrollar aquellas tareas que los operarios no

pueden realizar debido a su peligrosidad. Dichos robots deben proporcionar servicios de parada de emergencia que permitan parar al robot inmediatamente cuando el robot, su entorno o el propio operario están en peligro. Esta necesidad elemental de los sistemas robóticos implica que es necesario priorizar la ejecución de servicios de parada de emergencia, frente a los servicios de computación que realiza el robot de forma habitual. Con el objetivo de dar cuenta de estas necesidades de proceso de los sistemas robóticos tele-operados, el LdaOa de Prisma no utiliza el cálculo-π poliádico puro para especificar sus procesos, sino que utiliza el dialecto πP para especificar los distintos procesos de los componentes software.

La extensión al cálculo-π que realiza el dialecto πP es especialmente útil para el LdaOa de Prisma a la hora de especificar la prioridad de los servicios de emergencia cuando peticiones de diferentes servicios pueden llegar simultáneamente. Los niveles de prioridad que puede haber en un sistema software son ilimitados, ya que pueden variar dependiendo del sistema. Por ese motivo, y al igual que en el cálculo, el nivel máximo de prioridad está representado por el número 0 y el nivel mínimo de prioridad puede venir dado por cualquier número natural. El resultado de la aplicación del πP en Prisma es que la definición de la recepción de una petición de servicio y la solicitud de ejecución de un servicio se realiza como se presenta a continuación:

Solicitud:
nombre_del_servicio ? (parámetros_del_servicio) : nivel_prioridad
Recepción:
nombre_del_servicio ! (parámetros_del_servicio) : nivel_prioridad

Por tanto, en Prisma se asocia la prioridad a las accones, no a los canales; pero como ya se ha explicado en la sección 4, el resultado es totalmente análogo.

Haciendo uso del LdaOa de Prisma se ha especificado de forma completa un sistema robótico teleoperado llamado TeachMover [12]. Dicho sistema dispone, entre otros, de dos servicios, uno que para el robot (*stop*) y otro que permite mover una de las articulaciones de su brazo (*moveJoint*). El servicio *stop* tiene máxima prioridad porque implementa una parada de emergencia; y por lo tanto, el sistema necesita disponer de al menos dos niveles de prioridad, como en el ejemplo anterior.

A continuación se muestra un ejemplo de cómo los procesos de solicitud (S) y recepción (R) de ambos servicios se especifican con el LdaOa de Prisma haciendo uso de una adaptación de πP al contexto arquitectónico.

```
S ::= stop!():0 + moveJoint!(Steps, Speed):1
R ::= stop?():0 + moveJoint?(Steps, Speed):1
```

El comportamiento del brazo robótico, tal como se describe en estos dos procesos, sigue las pautas descritas en los apartados anteriores; así, y dado que se utiliza en el contexto de una elección (suma), siempre se realiza solamente una de las dos actividades. En el caso habitual, se realiza el movimiento del brazo, que es la acción de menor prioridad (1). Sin embargo, cuando aparece cualquier situación de emergencia, la orden de parada *stop* resulta prioritaria (0), y cualquier orden de menor nivel se ignora por completo. En definitiva, el mecanismo de prioridad no deja opción al sistema.

El ejemplo presentado es análogo al descrito en la sección anterior; en ambos casos se presenta una orden de emergencia (una *interrupción*) con mayor prioridad que el comportamiento normal, y se trabaja únicamente con dos niveles. Sin embargo en éste se define una solución aplicada al mundo real, mientras que en el anterior se definía de manera abstracta una situación hipotética. La diferencia entre ambos planteamientos se debe al cambio del nivel de abstracción. Así, la aplicación del πP en el contexto de Prisma permite utilizarlo a nivel arquitectónico; de este modo pierde su carácter teórico para convertirse en una herramienta útil y tangible en un sistema real.

7. Conclusiones

A lo largo de este artículo se ha presentado la definición de πP, lo que supone, hasta donde alcanza nuestro conocimiento, la primera vez en la que se establece un cálculo móvil con mecanismos de prioridad. Como ha podido verse, la definición del cálculo muestra unas características similares a las de otras álgebras de procesos con prioridad, pero manifestando además unas propiedades específicas.

El ejemplo de la sección 5 muestra con claridad que existe una necesidad real de un formalismo con estas características, ya que hay situaciones, que combinan prioridad y movilidad, en las que sólo podría utilizarse un cálculo de este tipo. Por su parte, el uso del cálculo en el contexto de Prisma, tal como se describe en la sección 6, demuestra que no se trata simplemente de una curiosidad teórica sino que, por el contrario, tiene una aplicabilidad clara en un sistema real, en el nivel arquitectónico.

Referencias

1. Jan A. Bergstra, Alban Ponse, y Scott A. Smolka, editores. *Handbook of Process Algebra*. North-Holland Elsevier, 2001.
2. PeerCast P2P Broadcasting. http://www.peercast.org.
3. Luca Cardelli y Andrew D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240:177–213, 2000.
4. Rance Cleaveland, Gerald L. Lüttgen, y V. Natarajan. Priority in Process Algebra. En Bergstra et al. [1], págs. 711–765.
5. David Harel. Dynamic Logic. En Dov M. Gabbay y F. Guenthner, editores, *Handbook of Philosphical Logic II*, págs. 497–694. Reidel, 1984.
6. Robin Milner. The Polyadic π-Calculus: a Tutorial. En F.L. Hamer, W. Brauer, y H. Schwichtenberg, editores, *Logic and Algebra of Specification*. Springer Verlag, 1993.
7. Joachim Parrow. An Introduction to the π-Calculus. En Bergstra et al. [1], págs. 479–543.
8. Jennifer Pérez, Nour Ali, José A. Carsí, y Isidro Ramos. Dynamic Evolution in Aspect-Oriented Architectural Models. En *Second European Workshop on Software Architecture*, volumen 3527 de *LNCS*, Pisa, Junio 2005. Springer Verlag.
9. Jennifer Pérez, Nour Ali, José A. Carsí, y Isidro Ramos. Designing Software Architectures with an Aspect-Oriented Architecture Description Language. En *9th Symposium on Component Based Software Engineering (CBSE)*, LNCS. Springer Verlag, Junio 2006.
10. Corrado Priami. Stochastic π-Calculus. *The Computer Journal*, 38(5):578–589, 1996.
11. Davide Sangiorgi y David Walker. *The π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
12. The TeachMover Robot. http://www.questechzone.com/microbot/teachmover.htm.

DEMOSTRACIONES

PRISMA CASE♦

Jennifer Pérez, Ismael Carrascosa, Javier Guillén, José A. Carsí, Isidro Ramos

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
{jeperez, icarrasosa, jguillem, pcarsi, iramos}@dsic.upv.es

La propuesta del Desarrollo Dirigido por Modelos (DDM) pretende mejorar las etapas tempranas del ciclo de vida software automatizando sus tareas en la medida de lo posible. DDM es un paradigma de desarrollo software basado en modelos que usa técnicas de generación automática para obtener el producto software final. El uso de modelos para el desarrollo de software proporciona grandes ventajas como la obtención de soluciones independientes de tecnología, trabajar con modelos, metamodelos y modelos específicos de dominio de igual forma, obtener a partir del mismo modelos códigos para distintas plataformas tecnológicas y lenguajes de programación mediante la aplicación de mecanismos de generación automática de código, etc. PRISMA sigue el DDM para ofrecer sus ventajas en los procesos de desarrollo y mantenimiento de sus arquitecturas software. PRISMA aplica DDM mediante el enfoque Software Factories propuesto por Microsoft. Dicho enfoque es soportado por el conjunto de técnicas y herramientas que proporciona DSL Tools [1].

El enfoque de desarrollo PRISMA tiene como objetivo principal el dar soporte al desarrollo de arquitecturas software orientadas a aspectos e independientes de tecnología. Su enfoque consiste principalmente en modelar y especificar completamente las arquitecturas, para que puedan ser compiladas a diferentes plataformas tecnológicas y lenguajes de programación a posteriori. Para dar soporte a este enfoque, se ha desarrollado un prototipo de herramienta CASE (Computer-Aided Software Engineering) llamado PRISMA CASE [2].

PRISMA CASE ha sido desarrollado utilizando DSL tools y en la actualidad PRISMA CASE es capaz de generar código C# orientado a aspectos a partir de un modelo arquitectónico. Dicho código es ejecutable sobre la plataforma .NET gracias al middleware PRISMANET que incorpora PRISMA CASE. PRISMA CASE esta compuesta por el metamodelo PRISMA, un conjunto de herramientas de modelado gráfico, un compilador de modelos, PRISMANET y una interfaz gráfica de usuario para ejecutar el código generado por la herramienta.

Referencias

1. Domain-Specific Language (DSL) Tools. http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx
2. Proyecto PRISMA, http://prisma.dsic.upv.es/.

♦ Este trabajo ha sido financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), Proyecto PRISMA, TIC2003-7804-C05-01

MOMENT: una herramienta de Gestión de Modelos aplicada a la Ingeniería Dirigida por Modelos♦

Abel Gómez, Artur Boronat, Pascual Queralt, José Á. Carsí, Isidro Ramos

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia. España.
{agomez | aboronat | pqueralt| pcarsi | iramos}@dsic.upv.es

La Ingeniería Dirigida por modelos es un campo en la Ingeniería del Software que, durante años, ha representado los artefactos software como modeles con el objetivo de incrementar la productividad, calidad, y reducir los gastos en el proceso de desarrollo de software. Los modelos proporcionan una descripción más abstracta de un artefacto software que el código final de la aplicación. Las compañías de desarrollo de software han aumentado su interés en este campo. Como por ejemplo encontramos las aproximaciones *Model Driven Architecture*, apoyada por la OMG, así como las *Software Factories*, apoyadas en este caso por Microsoft.

El Desarrollo Dirigido por Modelos ha evolucionado del campo de la Ingeniería Dirigida por Modelos. En él, no sólo las tareas de diseño y generación de código están involucradas, sino que también se incluyen las capacidades de trazabilidad, gestión de modelos, tareas de meta-modelado, intercambio y persistencia de modelos, etc. Para poder abordar estas tareas, las operaciones entre modelos, transformaciones, y consultas sobre ellos son problemas relevantes que deben ser resueltos. En el contexto de MDA se abordan desde el punto de vista de los estándares abiertos. El este caso, el estándar *Meta Object Facility* (*MOF*), proporciona un mecanismo para definir metamodelos. Por su parte, el estándar *Query/Views/Transformations* (*QVT*) indica cómo proporcionar soporte tanto para transformaciones como para consultas. A diferencia de otros lenguajes nuevos, *QVT* se apoya en el ya existente lenguaje *Object Constraint Language* (*OCL*) para realizar las consultas sobre los artefactos software.

Además, dentro de la ingeniería dirigida por modelos se ha propuesto una nueva disciplina denominada Gestión de Modelos. Ésta considera los modelos y las correspondencias entre ellos como entidades de primer orden, proporcionando un conjunto de operadores independientes de metamodelo y basados en teoría de conjuntos para tratar con ellos (Merge, Cross, Diff, ModelGen, etc.). Estos operadores proporcionan una solución reutilizable y componible para las tareas descritas anteriormente.

En esta demo presentamos la herramienta MOMENT, que da soporte a todas estas aproximaciones surgidas dentro de la Ingeniería por modelos. MOMENT proporciona un soporte algebraico a los operadores de gestión de modelos, así como a las tareas de transformación y consulta de modelos mediante un eficiente sistema de reescritura de términos —Maude— y desde un entorno de modelado industrial —*Eclipse Modeling Framework* (*EMF*)—. *EMF* puede ser visto como una implementación del estándar MOF, y permite la importación automática de artefactos software desde orígenes de datos heterogéneos: modelos UML, esquemas relacionales, esquemas XML, etc. En este sentido MOMENT aprovecha las capacidades de modularidad y parametrización de Maude para proporcionar un entorno de gestión, transformación y consulta de modelos de forma genérica e independiente de metamodelo.

♦ Este artículo ha sido financiado por el Proyecto Nacional de Investigación, Desarrollo e Innovación DYNAMICA TIC 2003-07804-C05-01.

LUNA+: Un entorno para validación de modelos basado en prototipado automático♦

Ángel Roche y Patricio Letelier

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
letelier@dsic.upv.es

La generación automática de código a partir de modelos está en pleno apogeo. Etiquetas tales como Model-Driven Architecture (MDA) y Model-Driven Development (MDD) resultan familiares y también otras cercanas como Domain Specific Modeling (DSM) y tecnología Meta-CASE. Esta tendencia enfatiza el modelado en lugar de la programación, el código se obtiene mediante transformaciones a partir de modelos, podremos incluso decir que los propios modelos son ejecutables. Con este renovado protagonismo de los modelos, una validación temprana (acompañando al proceso de construcción del modelo) es importante puesto que el producto final generado (manual o automáticamente) presenta dificultades para validar exhaustivamente el modelo. Tradicionalmente la validación de un producto software es realizada interactuando con el producto o alguna representación cercana a la implementación, por ejemplo mediante su interfaz de usuario, pero esto no permite validar de forma detallada el modelo. LUNA+ es un entorno software que permite validar modelos conceptuales (expresados mediante Diagramas de Clases de UML) mediante la generación automática de un prototipo ejecutable, específico para esta tarea de validación. LUNA+ es el resultado de nuestro trabajo en animación de modelos conceptuales utilizando un soporte subyacente formal [1, 2], pero de forma transparente y utilizando tecnologías actuales [3, 4].

Referencias

1. Letelier P., Sánchez P. y Ramos I. *Prototyping a requirements specification through an automatically generated concurrent logic program*. First Int. Workshop on Practical Aspects of Declarative Languages, New Mexico, USA, LNCS 1551, pp. 31-45, 1999.
2. Sánchez P, Letelier P. y Ramos I. *Animating Formal Specifications with Inheritance in a DL-based Framework*. Journal of Requirements Engineering, vol. 4(4), pp. 198-209, Springer-Verlag London Limited, 1999.
3. Letelier P. y Sánchez P. *Validation of UML classes through animation*. Internacional. International Workshop Conceptual Modeling Quality, IWCMQ'02 (junto con ER'2002), 7-11 Octubre, Tampere, Finlandia. Springer-Verlag LNCS 2784, pp. 300-311, 2002.
4. Roche A., Letelier P., Navarro E. y Llavador M. *Validación de modelos usando escenarios y prototipado automático*. XI Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2006), pp. 337-346, Sitges, Barcelona. 2006.

♦ Este trabajo ha sido financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), Proyecto PRISMA, TIC2003-7804-C05-01.

Una herramienta visual para la generación automática de entornos de desarrollo de software basados en modelos♦

Manuel Llavador, José H. Canós

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46022 Valencia, Spain
{mllavador, jhcanos}@dsic.upv.es

Recientemente han aparecido tres propuestas para la mejora del proceso de desarrollo de software –Desarrollo de software dirigido por modelos (MDD), Gestión de modelos (MM) y Factorías de Software (SF)– que se basan en la sucesiva transformación de descripciones abstractas de requisitos que permiten obtener el producto software final de forma (semi)automática. A las descripciones de requisitos se les llama especificaciones o modelos y a las herramientas que realizan las transformaciones motores de transformación o compiladores de modelos.

Sin embargo, a día de hoy, existen muy pocas implementaciones de esas técnicas y las que existen sólo funcionan parcialmente. De hecho, los expertos consideran que aún pasarán entre 5 y 10 años para que este tipo de herramientas se consoliden en el mercado.

Para acelerar el proceso, recientemente han aparecido entornos –herramientas METACASE– que permiten generar de forma automática este tipo de herramientas de desarrollo CASE (ej. MetaEdit+, Microsoft DSL Tools) basándose en la definición de lenguajes específicos de dominio, para la descripción de los modelos, y plantillas de transformación, para las transformaciones entre modelos y la generación de código. El principal problema de esos entornos es que resultan complejos de utilizar y/o requieren el aprendizaje de nuevos lenguajes específicos de cada herramienta (ej. Visual C# para DSL Tools).

En este trabajo presentamos XMetaCase, una nueva herramienta que permite generar de forma automática entornos de desarrollo de software que hacen uso de las técnicas MDD, MM y SF. Esta herramienta está basada en la definición de los lenguajes específicos de dominio que los ingenieros de software utilizarán para la descripción de sus productos en la herramienta generada, los operadores de transformación que se podrán aplicar a los modelos y los compiladores de modelos capaces de generar el código fuente final. Las principales novedades de este entorno respecto a los ya existentes es que todo el proceso está soportado por herramientas visuales, y que hace uso extensivo de la familia de lenguajes relacionados con XML (XSL, XSD, XPath, etc.).

♦ Este trabajo ha sido financiado por el Ministerio de Educación y Ciencia bajo el programa de formación de profesorado universitario (FPU) AP2005-3356 y la Comisión Interministerial de Ciencia y Tecnología (CICYT), Proyecto PRISMA, TIC2003-7804-C05-01

MDHDM: Un Método de Desarrollo de Hipermedia Dirigido por Modelos

Carlos Solís, José H. Canós, María C. Penadés

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia, Spain
{csolis, jhcanos, mpenades }@dsic.upv.es

MDHDM [1] [2] aplica la ingeniería dirigida por modelos (MDE) al desarrollo de hipermedia y aplicaciones Web. La tecnología MDE usada es la provista por la propuesta Model Driven Architecture (MDA) del Object Managemet Group (OMG). Los modelos independientes de la plataforma (PIM) de la fase de análisis son el modelo de proceso y el modelo conceptual. El modelo de proceso captura la dinámica de las aplicaciones, y el modelo conceptual las relaciones estáticas entre los datos. Debido a la introducción del modelo de proceso, los modelos navegacionales contienen dos tipos de enlaces: los de proceso y los estructurales. En MDHDM se definen las reglas necesarias para que a partir de los PIM de proceso y conceptual se derive el modelo navegacional, el cual se ubica en la fase de diseño de la aplicación. MDHDM cuenta con una herramienta que permite definir los modelos en Ecore, un subconjunto de Meta-Object Facility (MOF), y ejecutar las transformaciones declarativas definidas en Atlas Transformation Language (ATL). Los elementos de hipermedia cuentan con una presentación final diseñada de forma independiente como documentos HTML. La transformación XSLT entre la representación XML de los nodos navegacionales y su presentación final se obtiene a través de XWebMapper [3]. Para cubrir todas las etapas del desarrollo se definieron algunos modelos específicos de la plataforma (PSM) ubicados en la fase de implementación. El modelo conceptual se transforma en un modelo relacional, sin embargo, para implementarlo se usan plantillas de transformación a la herramienta de persistencia objeto-relacional Hibernate y a la base de datos relacional MySQL. A partir del modelo de navegación y de proceso se construye una arquitectura Model View Controller (MVC), otro de los PSM. El modelo de proceso se transforma en controlador del MVC, y el modelo de navegación se convierte en un conjunto de vistas del MVC, que son representadas por nodos implementados mediante Java-Servlets.

Referencias

1. C.Solís, J.H. Canós, M. Llavador, M.C. Penadés. De modelos de proceso a modelos navegacionales. XI JISBD, 2006.
2. C. Solís, J.H. Canós, M.C. Penadés, M. Llavador. A Model Driven Hypermedia Development Method. ICWI, 2006.
3. M. Llavador, J.H. Canós. XWebMapper, a Web-based Tool for Transforming XML Documents. ECDL, 2006.

Índice de Autores