Reuse based Analysis and Clustering of Requirements Diagrams

Oscar López[1], Miguel A. Laguna[2], and Francisco J. García[3]

[1] Technological Institute of Costa Rica, San Carlos Regional Campus, Costa Rica
olopez@infor.uva.es
[2] Department of Informatics, University of Valladolid, Spain
mlaguna@infor.uva.es
[3] Department of Informatics and Automatics, University of Salamanca, Spain
fgarcia@usal.es

Abstract. Requirements reuse is intended for cost reduction by benefiting from reusable requirements elements (*assets*) in software development. In general, successful reuse approaches are based on the identification of commonalities and variabilities between application family members. However, there are many domains where the software development process has not followed an application family focus. Addressing the requirements reuse in these domains requires a process to set up a taxonomy which should be founded on analysis and clustering of requirements information. This paper presents a technique, which is based on metamodeling, Petri nets and facets, for analysis and clustering of requirements diagrams. This technique is supported by a prototype system within a reuse framework.

Keywords: Requirements engineering, requirements reuse, use case, Petri net, facets.

1 Introduction

Software reuse approach has successfully contributed to improve the software development process in restricted and well understood domains. As a consequence, several reuse approaches have appeared, for example FODA [9], PuLSE [1] y ODM [25], which face the software development on the basis of taking advantage of reusable elements (*assets*) among the members of an application family. This reuse approach is usually called “product lines”. In general, the product lines based approach increases the productivity and reduces the development cost for each product. It improves the quality and it also allows the better estimation related to the software development process to be done [2, 3].

The starting point for a product line approach is the domain analysis. This analysis leads to the development of a common software architecture which allows the requirements, design and implementation elements to be shared among different family members. There are, however, lots of domains where software development has been addressed for a long time and without a product line based approach. Also, on these domains usually there are not enough resources to afford a domain analysis from the scratch, e.g. the domain of software applications for handicapped people. An attractive alternative for this kind of domains could be one which tries to sort the existing specifications to be reused in a product family approach, thus making the development management easier.

Establishing a requirements family allows the effectiveness in software reuse to be accomplished. Neighbors [18] points out that effective reuse has to be addressed at a high level of abstraction, thus the concept of domain-oriented reuse appears. Domain orientation means the reusable elements are ones which implement the domain concepts. To implement the domain concepts, the domain knowledge must be classified in a taxonomy. Building of domain taxonomies, on requirements reuse context, has been approached in different works [6, 10, 11, 26] which face the specification of reusable requirements with natural text and formal language. In contrast to works, we propose establishment of a requirements family from existing, semiformal diagrams on a domain.

This paper presents a domain oriented approach to reusing software requirements. The proposal is based on setting up a Diagram Family Model to generate new requirements specifications. Requirements that have been captured and documented based on semiformal diagrams are integrated by a metamodel and they are stored as a family in a repository. To make sure about some quality issues of the stored reusable elements we automatically analyze them in a Petri net environment. After that, we are in a position to apply a mechanism to cluster the requirements in such a way that it allows the domain requirements patterns to be identified. The rest of the paper is distributed as follows: Section 2 deals with the analysis of existing diagrams. In section 3 we explain how to set the diagrams to obtain a requirements reuse structure and it is applied to use cases clustering. Section 4 presents a tool for requirements reuse. Section 5 shows a case study. Section 6 relates our work to other known studies. The section 7 concludes the paper and proposes additional work.

2 Analysis of Requirements Diagrams

The requirements documentation acts as a means of communication of customer needs and it constitutes the information flow from analysis stage towards design and implementation stages. In industrial software production, requirements specification is based on segmenting the market and minimizing the implementation cost, thus trying to reuse assets. Moreover, the requirements can be modified regarding both the understanding of product characteristics and availability of reuse options. On this context, there is a need for taking into account the requirements of all potential family members. Thus, a great deal of requirements information, which is modeled with distinct formality levels, is expected to be processed in complex domains. For this reason, techniques and supporting tools must be available to analyze the requirements information in a product line based approach.

Due to the communication and agreement needs, the more graphical, intuitive and semiformal models, the more widely used in requirements engineering process [23]. However, lack of formality constrains the automatic and systematic analysis of the behaviour of requirements components. This shortage is a critical issue in real reuse environments. As a consequence, it must be found the way of transforming semiformal models to a precise language, a standard one if possible, to describe and manage different requirements models in a domain. We address the analysis of requirements by normalizing the representation of diagrams in metamodeling [15] and projecting diagrams to CPN formalism [14]. Metamodeling allows diverse models to be integrated [7, 17] while CPN [8] supply formal support to modeling of complex systems. The software requirements reflect a complex pattern of rules for static and dynamic issues of the system [20] which may be adequately modeled by CPN. Furthermore, description of requirements diagrams with Petri nets allows the automatic and analytical treatment of requirements information to be done [13].

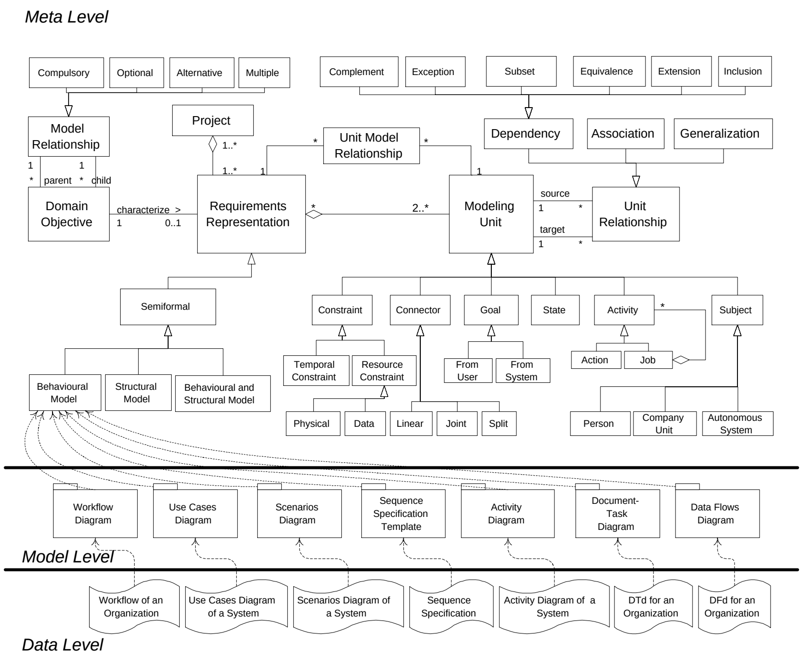


Fig. 1. Requirements Metamodel expressed in UML.

2.1 Metamodel for Requirements Diagrams

We carried out an abstraction process to propose a conceptual scheme describing diverse requirements diagrams in the reuse framework. This conceptual scheme constitutes a metamodel for different semiformal requirements diagrams, see figure 1, more details could be found in [15]. The metamodel is theoretical basis for our reuse framework such that each requirement information element from semiformal diagrams is an object instance of some metaclass.

The main elements of our metamodel are the *Requirements Representation*, which describes different requirements diagrams, the *Modeling Unit*, which describes units that belong to the requirements diagrams, and the *Domain Objective*, which allows the Requirements Representation to be characterized. There are three categories of Requirements Representations whereas six of Modeling Units, three of relationships among Modeling Units and four of relationships between domain Objectives.

The relationships between Modeling Units are described in the metamodel as instances of the *Unit Relationship* class. These relationships represent the association between elements inside a requirements diagram. These relationships between Modeling Units have an essential role in the reuse strategy because they enhance the content of diagrams. All these kinds of relationships between Modeling Units show a *direction* issue establishing that some elements act as a source while others act as a target.

A *Domain Objective* is an essential requirement which must be satisfied by developing a class of applications. There is no restriction about the sort of information to be recorded as a Domain Objective. The only restriction is that Domain Objective has to be a general intention or a domain significant goal to be addressed by developing software. Domain Objectives have several attributes: identifier, complexity, cost, required technology, quality, and additional information. Domain Objectives are related through Model Relationship, whose roles allow a parent-child based taxonomy to be built. Resulting taxonomy is a lattice structure where every Domain Objective might have zero to many children, as well as zero to many parents.

The *Model Relationship* class describes the relationships between Domain Objectives. These relationships give us the basis for the integration of a body of domain knowledge which is related to Requirements Representation. These relationships refer to the way in which diverse diagrams are combined to accomplish the domain description. For example, these relationships allow the Use Case Diagram and Activity Diagram to be used to describe distinct Domain Objectives. Model Relationship has a structural issue which determines the degree of association between two or more related diagrams.

Some modeling units may have such a complexity that they need to be specified in another complete requirements representation. For example, a process in a Data Flow Diagram may be exploded in another Data Flow Diagram, or a use case may be specified as a Sequence Specification Template. These relationships are described as Unit Model Relationships in metamodel.

2.2 Logical Consistency of Requirements Diagrams

Correctness of requirements diagrams which are stored in a repository is strongly important in the systematic reuse context. Reusing diagrams that contain errors can make the requirements negotiation phase more difficult, as well as costs for error correction higher. All of this may lead the enterprise to waste business opportunities. Fails in requirements diagrams lead to both misunderstanding of real necessities and increasing the development effort. Hence, analysis to make sure about logical consistency of reusable requirements diagrams which are stored in a repository is quite important.

In order to make sure about logical consistency of the requirements diagrams we make a projection of diagrams to a rigorous representation in CPN [16]. Analysis of Petri net models can be automatically done with available techniques which allow the deadlock and livelock-free, finite set of states and conflict-free properties to be guarantee [24]. Furthermore, we analyze the CPN models as proposed by Aalst [5], it consists of testing whether classical Petri nets models satisfy the soundness property, so that we obtain the equivalent models in classical Petri nets, and these models are tested as proposed by Aalst to determine the soundness property. If equivalent classical Petri nets models show soundness property then we are sufficiently sure about the soundness property of the initial CPN models.

3 Clustering of Existing Requirements Diagrams

Studying the domain for a software system, being part of a product family, goes beyond technical aspects of software engineering. Different domain issues, such as market strategies, product positioning, risk management, product look-and-feel, architectural aspects, quality standards, etc., have to be taken into account when studying a domain. These issues should be clearly reflected by a domain requirements structure containing clustered requirements information in a reuse context. Sorted information makes browsing and searching of relevant information easier.

3.1 Diagram Family Model

We sort the requirements in a domain taxonomy as a Diagram Family Model based on Domain Objectives. Requirements in a domain taxonomy make the reuse approach easier in such a way that reusing requirements can be aimed both at obtaining better and quicker software specifications and directing the elaboration for reusable design and implementation in a domain under an application family based approach.

The requirements taxonomy is obtained by performing the following activities:

– Identifying the Domain Objectives and establishing a domain dictionary. The goals belonging to stakeholders are identified so that these become into a lattice structure where goals are related by Model Relationship. This activity requires lots of experience and domain knowledge to make decisions about overlapping goals, conflicting goals, and dependencies among goals.

– Associating Domain Objectives to existing requirements diagrams. This stage aims to benefit from current requirements representations which have been created by different projects in the domain. Rewriting or modifying some diagrams may be required through an iterative process which is supported by Requirements Editor we present in section 4.
– Storing the diagram family into repository. The diagram family is physically stored into repository which supplies the operative support for *creation, management, and use* of reusable diagrams. Repository scheme is based on our requirements metamodel.

The Diagram Family Model allows the reuse opportunities to be detected by allowing stakeholders to browse and choose requirements. Each tree on the family is visited and requirements are selected by making corresponding choices.

3.2 Requirements Catalogue

Benefiting from previous modeling effort, as it is sorted on Diagram Family Model, might be enhanced by a catalogue of requirements diagrams. The catalogue allows descriptors to be associated to reusable elements. A hierarchical taxonomy can be combined with different kinds of catalogues, such as thesaurus and facet based schemes. Facet based schemes are widely used, they consist of a set of terms which are grouped into subsets which are called *facets* [21].

In order to catalogue the set of existing requirements documents in a domain, we make a *Facet based Requirements Catalogue* which allows the requirements patterns to be identified regarding to dependency and association relationships. These relationships are described in our metamodel as shown in figure 1. The supporting idea is the domain has been modeled across a set of applications so that clustering the requirements diagrams allows a characterization of domain requirements to be obtained. The distinct software applications share similar features which are called *commonalities* [11, 10]. Sorting of requirements information is vital to detect these commonalities. We propose automatic clustering of requirements diagrams to make the commonalities identification easier.

Requirements metamodeling [15] reflects the fundamental element of the different requirements diagrams we have investigated is any instance of the Job metaclass (for example, each use case in a Use Case Diagram). The instances of Job metaclass show dependency relationships between them. They are also composed of other Jobs and Actions which have different relationships between them. Hence, Jobs, which are expressed as Sequence Specification Template, are main elements to catalogue the requirements on the context of systematic software reuse. Jobs may be mapped to different patterns as proposed in [22]. The decision tree of figure 2 allows Jobs to be classified by considering both association and dependency relationships. A Job *S* is assigned to an element from the set $P = \{production, collaboration, service, negotiation\}$. Within the Negotiation term there exist different sub-categories: Negotiation Step, Negotiation

Step with Jobs Triggering, End of Negotiation, Complete Negotiation, Service, Unfinished Negotiation with Jobs Triggering, Unfinished Negotiation.

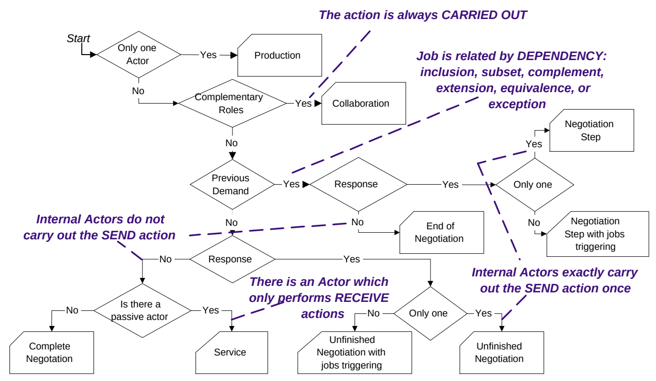

The action is always CARRIED OUT
Start
Only one Actor
Yes
Production
No
Complementary Roles
Yes
Collaboration
Job is related by DEPENDENCY: inclusion, subset, complement, extension, equivalence, or exception
No
Negotiation Step
Yes
Previous Demand
Yes
Response
Yes
Only one
No
End of Negotiation
No
Negotiation Step with jobs triggering
Internal Actors do not carry out the SEND action
No
No
Response
Yes
There is an Actor which only performs RECEIVE actions
Internal Actors exactly carry out the SEND action once
No
Is there a passive actor
Yes
No
Only one
Yes
Complete Negotiation
Service
Unfinished Negotiation with jobs triggering
Unfinished Negotiation

Fig. 2. Decision tree to catalogue the jobs of diagrams.

The facet of Diagrams Patterns, cataloguing a particular diagram, is a subset of P. As Jobs which are contained into a requirements diagram correspond to different terms of P, the requirements diagrams may be catalogued by composed facets such as Production plus service plus collaboration, and Negotiation (whichever of its kinds) plus production or collaboration or service, etc. In this way, a characterization of typical sequences of tasks in a domain is obtained. It could be thought of a great dispersion of combinations of terms, however in [22] it is shown that a high proportion of tasks corresponds to a catalogue of patterns.

The Diagram Pattern allows the diagrams sharing similar structures among different projects and modeling techniques to be identified. One can benefit from this pattern identification by taking it as a guideline to the construction of reusable design and implementation elements in the domain.

4 A Tool for Requirements Reuse (R^2)

Process to build the requirements taxonomy is supported by R^2, a prototype system which is implemented using relational tables (ORACLE) and Java language. This environment is composed by five main elements with corresponding sub-elements: (1) *User Interface* including menu options and necessary views to support the interaction between reusers and the environment, (2) *Requirements Editor* which supplies needed functionality for creation and modification of requirements diagrams, (3) *Data Manager*, it allows the requirements information to be stored, classified, retrieved and updated, (4) *Repository*, which physically contains the information related to requirements diagrams regarding to the conceptual scheme of our requirements metamodel, and (5)

Data Exchange, it is a module which allows the information to be directed to and directed from external applications, for example the Petri net applications to address the verification of logical consistency of diagrams.

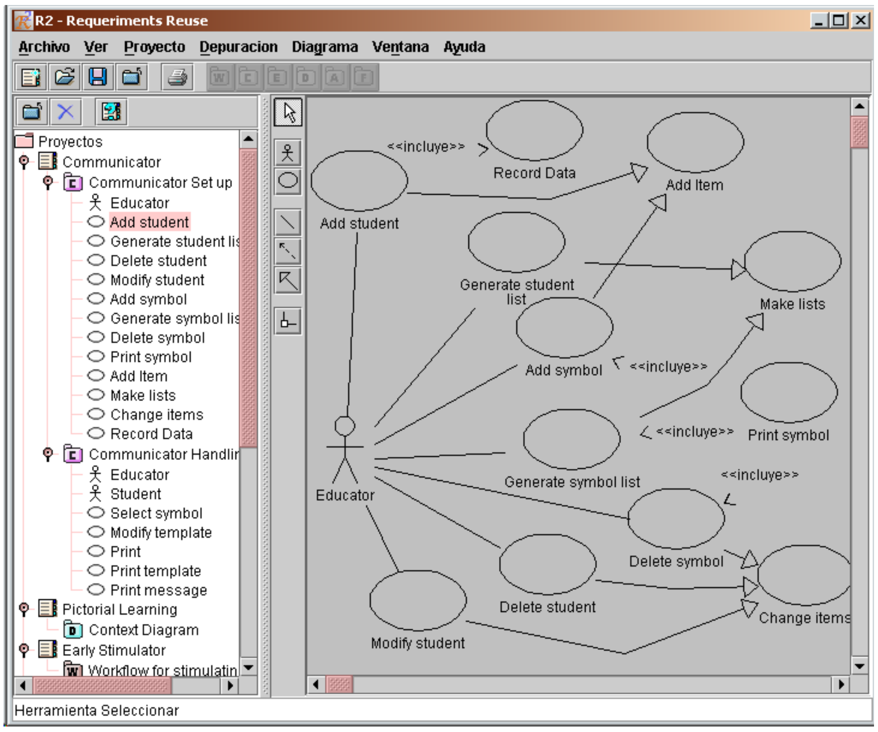


Fig. 3. Requirements Editor interface of R^2 environment.

Figure 3 shows the Requirements Editor interface of R^2. The identifier of project and their elements are on the left side of the window. Graphical notation corresponding to current element appears on the right. The figure shows the selected diagram inside the so called project "Communicator".

5 A Case Study: Preliminary issues

We currently apply our proposed scheme for analysis and clustering of requirements on the domain of software applications for handicapped people. This domain has been addressed with different projects conducted by undergraduated students at Computer Science Department, University of Valladolid. These projects have been developed without product line perspective, and for this reason there are different requirements diagrams from distinct notations.

We start by building a domain taxonomy as a Diagram Family Model based on Domain Objectives. Semiformal diagrams were linked to Domain Objectives, thus setting up the requirements family. This process has required the support from domain experts through several work sessions. It had been necessary to rewrite some diagrams. Furthermore, we had to create some new diagrams as domain experts was suggested. All of these new diagrams were incorporated to requirements family. All the semiformal diagrams being part of the family were tested for logical consistency, as we explained in section 2.2.

We address the construction of Diagram Family Model through a subjective process which demands for knowledge and experience with regard to application domain. Sometimes we needed to make compromise solutions regarding how to link requirements on a family. Although similar works related to making requirements families have been published [10, 26, 11, 6], these do not provide enough details of the construction process. All of these make the need for a time to stabilize the requirements family evident. Furthermore, it confirms Leite's idea [12] about the need for conducting experiments in the process of domain construction.

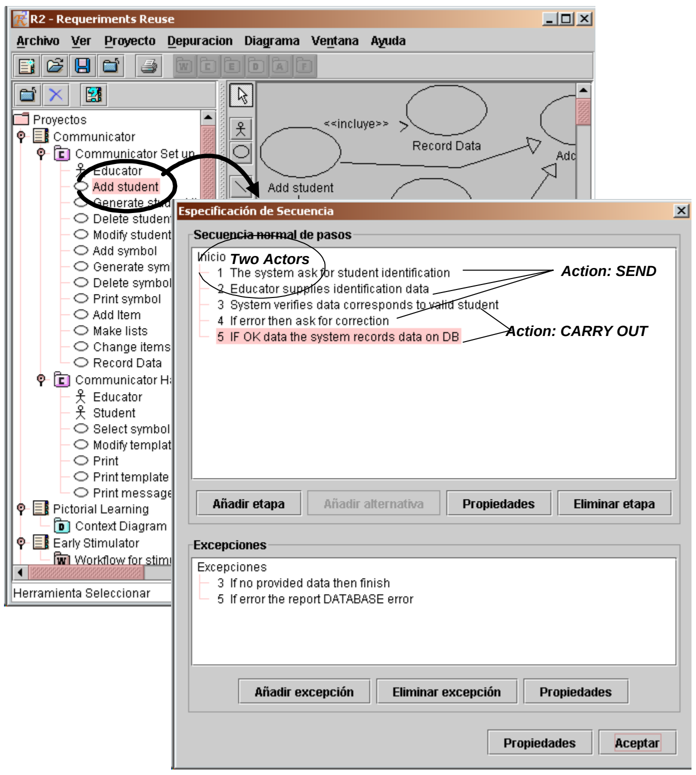
The corresponding clustering of requirements diagrams is obtained by applying the guidelines we have mentioned in section 3.2. By following the decision tree of figure 2, the tasks inside a diagram are classified as different subcategories. As an example, the use case Add Student, see figure 4, is classified as *Negotiation Step with Jobs Triggering*. In this use case there exist two actors (System and Educator) whose have not complementary roles (the action is not always "carry out"), there exits previous demand (job is related by dependency) and more than one responses have to be produced.

We are now ready to experiment with our Diagram Family Model in generating new requirements specifications. Having a Requirements Reuse prototype system we expect to obtain quantitative data with respect to reusing requirements and making an increase of requirements specifications. We believe that reusing requirements on our reuse framework can make better requirements specification because of requirements engineers might focus on decisions about taken a set of requirements regardless the details of definition, logical consistency, and documentation.

6 Related Work

The organization of requirements has been proposed by different authors. Kuseela [11] proposes a hierarchy in which the design objectives are defined by other design objectives or by design decisions. Lamsweerde [27] has also proposed an objectives hierarchy. Mannion [10] proposes a discrimination based arrangement. The main difference between these proposals and ours consists of we are proposing a way to benefit from existing information while those claim for developing the proper structure as whole.

On the other hand, we know that some authors such as Cybulski [4], Palmer and Liang [19], have been addressing the classification of requirements specifications. These


R2 - Requeriments Reuse

Proyectos
Communicator
Communicator Set up
Educator
Add student
Generate stud
Delete student
Modify student
Add symbol
Generate sym
Delete symbol
Print symbol
Add Item
Make lists
Change items
Record Data
Communicator H
Educator
Student
Select symbol
Modify templat
Print
Print template
Print message
Pictorial Learning
Context Diagram
Early Stimulator
Workflow for stimu

Herramienta Seleccionar

Especificación de Secuencia

Secuencia normal de pasos

Inicio
1 The system ask for student identification
2 Educator supplies identification data
3 System verifies data corresponds to valid student
4 If error then ask for correction
5 IF OK data the system records data on DB

Excepciones

Excepciones
3 If no provided data then finish
5 If error the report DATABASE error

Fig. 4. Clustering of Requirements Diagrams.

proposals focus on indexing of text based requirements specifications. Our proposal is based on diagram based requirements specifications. However, we do not mean that requirements are only made as diagrams. In fact, the real requirements are in natural language, as we express the Domain Objectives. Our approach is aimed at reusing diagrams which are linked to textual requirements.

7 Conclusions and Future Work

In this paper we propose a technique for analysis and clustering of reusable requirements. The proposal allows the domain description to be obtained in terms of the association and dependency relationships related to the modeled services in different techniques under a requirements metamodel. We address the rigorous description of the static and dynamic issues of the system in order to test the logical consistency of reusable requirements elements which are stored in a repository. The rigorous description is based on the CPN formalism. The analysis and clustering of requirements are automatically made with our prototype system for requirements reuse.

The Diagram Descriptor allows the similar structures of requirements in a domain to be identified. By applying these descriptors we are in a position to classify requirements and to discover commonalities and variabilities from requirements diagrams which are modeled in domain applications. This classification can be aimed both at obtaining better and quicker software specifications and directing the elaboration for reusable design and implementation in a domain under an application family based approach.

Our future work is aimed at defining process for developing the requirements specifications in a domain. This process should be founded on the structure for requirements analysis and organization we proposed in present paper. Furthermore, from this proposal for requirements clustering, we aim to explore the possibilities for developing the core assets for requirements, design and implementation in a product line based approach.

Acknowledgments

This work is sponsored by the DOLMEN Project within CICYT-TIC2000-1673-C06-05, Ministry of Technology and Science, Spain.
Oscar López wishes to thank the Spanish Agency for International Cooperation (AECI) and the Ministry of Technology and Science of Costa Rica.

References

1. J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A methodology to develop software product lines. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99)*, pages 122–131, Los Angeles, CA, USA, May 1999. ACM.
2. Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. ACM Press. Addison-Wesley, May 2002.
3. Paul C. Clements and Linda N. Nothrop. *Software Product Lines: Practices and Patterns*. The SEI series in software engineering. Addison-Wesley, 2002.
4. J. L. Cybulski and K. Reed. Requirements classification and reuse: Cross domain boundaries. In *Sixth International Conference on Software Reuse*, Vienna, Austria, June 2000.
5. V. d. Aalst. The Application of Petri Nets to Workflow Management. Eindhoven, University of Technology, Netherland, URL: wwwis.win.tue.nl/ wsinwa/jcsc/jcsc.html, 1997.
6. Stuart R. Faulk. Product-line requirements specifications (PRS): an approach and case study. In *Proceedings of 5th IEEE International Symposium on Requirements Engineering*, pages 48–55, Toronto, Canada, August 2001. IEEE Computer Society.
7. R. Geisler, M. Klar, and C. Pons. Dimensions and dichotomy in metamodeling. In *Proceedings of the Third BCS-FACS Northern Formal Methods Workshop*. Springer-Verlag, September 1998.
8. Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 3*. Springer-Verlag, Berlin, Germany, 2nd. edition, 1997.
9. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Feasibility study. Technical Report CMU/SEI-90-TR21 (ESD-90-TR-222), Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, November 1990.
10. Barry Keepence, Mike Mannion, Hermann Kaindl, and Joe Wheadon. Reusing single system requirements from application family requirements. In *Proceedings of the 21st International Conference on Software Engineering*, pages 453–462. ACM Press, May 1999.

11. J. Kuusela and J. Savolainen. Requirements engineering for product lines. In *Proceedings of ICSE 2000, IEEE*. IEEE Computer Society, 2000.
12. J. Leite. Are domains really cost effective? In *Proceedings of the Workshop on Institutionalizing Software Reuse (WISR'9)*, January 1999.
13. O. López, M.A. Laguna, and F.J. García. Automatic generation of uses cases from workflows: a petri net based approach. In *European Joint Conferences on Theory and Practice of Software ETAPS*, pages 279–293, Grenoble, France, Abril 2002.
14. O. López, M.A. Laguna, and F.J. García. Representación de requisitos mediante redes de Petri coloreadas. Technical Report IT-DI-2002-1. Departamento de Informática, Universidad de Valladolid, 2002.
15. O. López, M.A. Laguna, and F.J. García. Requirements modeling for reuse. In *Actas de Las II Jornadas de Trabajo DOLMEN*, pages 105–114, Valencia, España, Marzo 2002.
16. O. López, M.A. Laguna, and F.J. García. Reuse based requirements clustering. In *Actas de Las II Jornadas de Trabajo DOLMEN*, pages 129–138, Valencia, España, Marzo 2002.
17. S. Mann and M. Klar. A metamodel for object-oriented statecharts. In *Proceedings of The Second Workshop on Rigorous Object-Oriented Methods*, May 1998.
18. James M. Neighbors. The Draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering*, SE-10(5):564–574, September 1984.
19. J. Palmer and Liang Y. Indexing and clustering of software requirements specifications. *Information and Decision Technologies*, 18(4):283–299, 1992.
20. K. Pohl. Requirements engineering, an overview. Encyclopedia of Computer Science and Technology, Vol. 36, Marcel Deccer Inc., 1996.
21. Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.
22. M. Ridao, J. Doorn, and J. Leite. Domain independent regularities in scenarios. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 120–127. IEEE Computer Society, August 2001.
23. Colette Rolland and Naveen Prakash. From conceptual modelling to requirements engineering. Technical Report Series 99-11, CREWS, 1999.
24. Manuel Silva. *Las Redes de Petri en la Automática y la Informática*. Editorial AC, Madrid, España, 1985.
25. Mark Simos, Dick Creps, Carol Klingler, Larry Levine, and Dean Allemang. Organization domain modeling (ODM) guidebook - version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121, June 1996.
26. A. Sutcliffe and N. Maiden. The domain theory for requirements engineering. *IEEE Transactions on Software Engineering*, 24(3):174–196, March 1998.
27. A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Proceedings: 2nd IEEE International Symposium on Requirements Engineering*, pages 194–203. IEEE Computer Society Press, 1995.