Reuse based Requirements Clustering

Oscar López Villegas
Technological Institute of Costa Rica,
olopez@infor.uva.es

Miguel Ángel Laguna
University of Valladolid,
mlaguna@infor.uva.es

Francisco J. García
University of Salamanca,
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. There are many domains where the software development process has not followed an application family focus. Addressing the requirements reuse in these domains requires an analysis to sort the requirements and discover the commonalities and variabilities. This paper presents a coloured Petri net based technique for analysis and clustering of requirements information such that one could take advantage of it within a reuse framework.

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

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 [6], PuLSE [1] y ODM [13], which face the software development on the basis of taking advantage of reusable elements (*assets*) between 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.

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 companies which have been developing software for a long time and without a product line approach. Also, these usually do not have enough resources to afford a domain analysis from the scratch. An attractive alternative for these organizations could be one which tries to sort the existing specifications to be reused in a product family approach. This requirement sorting should be based on analyzing the set of domain requirements as a whole, thus establishing a requirements family such that it makes the development management easier.

For analysis and clustering of requirements we need to address the problem of diverse notations and lack of formality in requirements representations. The documentation of the requirements is basically oriented to being a means of communication between analysts and stakeholders (customers, developers, maintenance personal, managers, etc.). For this reason, it is represented with diverse notations and formats. Furthermore, the needs for communication between many people lead to avoid the higher levels of formality in initial requirements documentation [12]. This diversity and lack of formality imply the need for particular actions to analyze requirements documents and group them together in a repository of reusable artifacts [4].

In this paper we propose a technique for analysis and clustering of existing requirements diagrams in a domain. Requirements that have been captured and documented based on semiformal diagrams are integrated by a common model. Then, the requirements lack of formality is corrected through Petri nets formalism. After that, we are in a position to apply corresponding mechanisms to set the requirements into a coloured Petri net (CPN) based structure. This structure always allow the domain requirements and the application requirements to be viewed. The organization of requirements has been proposed by different authors. Kuseela [8] proposes a hierarchy in which the design objectives are defined by other design objectives or by design decisions. Lamsweerde also [14] has proposed an objectives hierarchy. Mannion [7] 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.

The rest of the paper is distributed in 3 sections. 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. The section 4 concludes the paper and proposes additional work.

2 Analysis of Requirements Diagrams

The requirements specification 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 data is expected to be processed in complex domains. For this reason it must be available tools to analyze the requirements information in domains and then associate it to design information in a software reuse framework.

Due to the communication and agreement needs, the more graphical, intuitive and semi-formal models, the more widely used in requirements engineering process [12]. However, lack of formality constrains the 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 translating requirements semiformal models to a precise language, a standard one if possible, to describe and manage different requirements models in a domain. To do so, in this section the representation of diagrams in CPN formalism is addressed. The software requirements act as a complex pattern of rules reflecting static and dynamic issues of the system [10] which may be adequately modeled by CPN. Furthermore, CPN supplies such a strong formal support that it has been successfully applied in modeling complex systems [5].

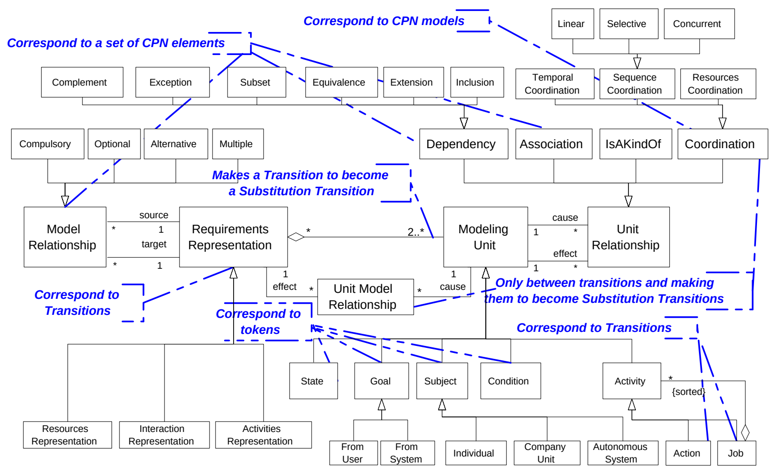


Figure 1: Common Model of Software Requirements, expressed in UML, and its correspondence with CPN elements.

2.1 Common Model for Requirements Diagrams

The requirements diagrams could be describe with a common model, as shown in figure 1. The central elements of this model are the *Requirements Representation*, which is used to describe the different diagrams, and the *Modeling Unit*, which describes units that belong to the requirements diagrams. There are three categories of Requirements Representations whereas five of Modeling Units, four of relationships among Modeling Units and four of relationships between Requirements Representations.

The relationships between Modeling Units are described in the common model 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 describe the content of diagrams. All these kinds of relationships between Modeling Units show a *causality* issue establishing that some elements act as a cause while others act as an effect.

The relationships between Requirements Representation are described in the common model as instances of the *Model Relationship* class. These relationships give us the basis for the integration of different levels of requirements description in the reuse strategy. These relationships refer to the way in which the diagrams are combined to associate distinct descriptions of the domain. For example, these relationships allow a Use Case Diagram or an Activity Diagram to be related to a Class Diagram or a Sequence Diagram. 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.

2.2 Matching the Requirements Model and CPN

In order to translate the common model to a formal language, we have establish the correspondence between the requirements common model and CPN [9]. This correspondence could be expressed in following terms (see figure 1):

1. Modeling Units:
   (a) State, goal, condition and subject match to tokens.
   (b) Job and action match to transitions.
2. A Requirements Representation matches to a transition.
3. Relationships:
   (a) Relationships between Modeling Units:
      i. Dependency and Association match to a set of elements formed by arcs, places, tokens and arc expressions.
      ii. IsAKindOf Relationship is ignored because it is only descriptive in the requirements models we have dealt with.
      iii. Coordination Relationships match to CPN models. These relationships are only established between modeling units which are represented as transitions.
   (b) Relationships between Modeling Units and Requirements Representations (Unit Model Relationship) are only established between transitions and it makes the transition to become a substitution transition.
   (c) Relationships between Requirements Representations (Model Relationship) are translated to a set of elements formed by arcs, places, tokens and arc expressions.
   (d) Relationship of Aggregation between Modeling Units and Requirements Representations makes the container transition to become a substitution transition.

2.3 Algorithm to Translate Requirements Diagrams to CPN

From matching common model and CPN, the projection of requirements diagrams to CPN models might be obtained. To do so, we must apply the matching rules in a given order. In this section we present an algorithm for translation of requirements diagrams (such as Use Case Diagram, Workflow, or Activities) into a CPN model following a top-down direction.

1. Project each requirements representation to a substitution transition.
2. Represent the relationships between requirements representations (instances of the Model Relationship) by following the rule 3c. (Note: Compulsory, Alternative, Optional, and Multiple kinds of relationships are represented as CPN models).
3. While there exist substitution transitions do the representation of their content. The relationships are represented as follows:
    (a) Represent the dependency and coordination relationships between the instances of Job and Action metaclasses.
    (b) Represent the association and dependency relationships between State, Goal, Condition and Subject.
        i. The Cause role in States, Goals, Conditions and Subjects is represented by marks in corresponding previous places and arc expressions.
        ii. The Effect role in States, Goals, Conditions and Subjects is represented by marks in corresponding post places and arc expressions.

This algorithm allows the formalized expression of the described requirements diagrams to be obtained. This CPN formalized expression allows the requirements diagrams to be analytically treated with available CPN based tools. Furthermore, this CPN based requirements formalization provides the necessary support for our requirements organization scheme to approach the requirements reuse.

2.4 Obtaining Data from Requirements Diagrams

The requirements reuse, which is inside the general context of software reuse, aims to take advantage of previous development effort in new products development. 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. For these aims the requirements reuse relies on a domain requirements analysis. Several techniques address the domain analysis (FODA, FORM, PuLSE, etc.) but they need an exhaustive domain analysis and they demand for a great investment. On the contrary, our proposal takes the set of existing requirements documents as the starting point to carry out a *reuse based requirements analysis*, see figure 2. This analysis essentially consists of extracting a *domain requirements description* from the available requirements diagram.

The requirements description is carried out by taken all the domain applications into account and it is expressed in terms of coordination, dependency and association relationships. This description allows the domain generic requirements to be identified. The supporting idea here is the specific domain has been modeled through the time in a series of software applications. From this sample of applications the requirement documentation is analyzed in order to obtain a domain characterization. In other words, if software applications model the domain, analyzing the requirements documents allows the specific characterization in terms of domain requirements to be obtained. This characterization is based on Coordination, Dependency and Association relationships.

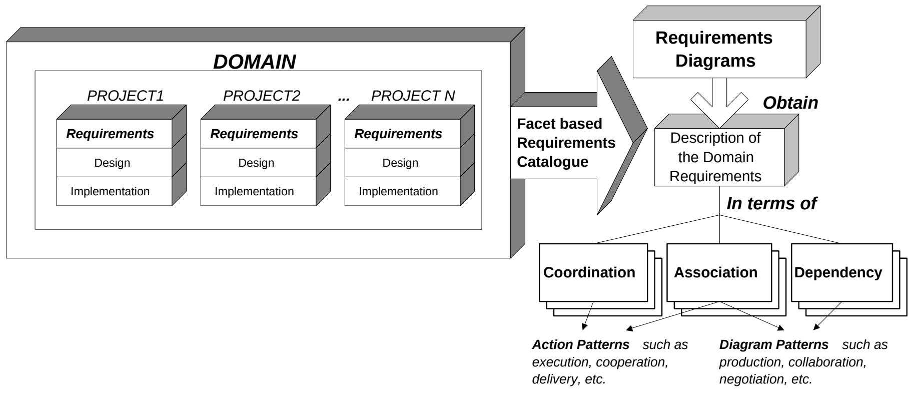

DOMAIN
PROJECT1
PROJECT2
...
PROJECT N
Requirements
Design
Implementation
Facet based Requirements Catalogue
Requirements Diagrams
Obtain
Description of the Domain Requirements
In terms of
Coordination
Association
Dependency
Action Patterns such as execution, cooperation, delivery, etc.
Diagram Patterns such as production, collaboration, negotiation, etc.

Figure 2: Reuse based Requirements Analysis. From Requirements Diagrams, belonging to Domain Software Projects, a Requirements Description is obtained. This description is expressed in terms of Coordination, Dependency and Association.

3 Requirements Clustering

The requirements systematic reuse requires the information to be clustered in an environment (then a set of supporting tools is supposed to be available). In this context, the reuser is one who express the characteristics of the reusable elements to be retrieved from the repository and then the environment must return the reusable elements satisfying those characteristics. In such a case of there is no satisfactory answer to the reuser then the environment might help the reuser to create the desired element, perhaps by partially reusing elements from the repository.

The first necessary element to requirements clustering, in a reuse environment, is diagrams formalization that make us sufficiently sure about precision of modeling data. Having the translation of requirements diagrams into CPN, we propose a scheme where requirements diagrams are arranged in a Petri net itself, as shown in figure 3. The *Select* place is the entering point where the reuser determines which is the desired information. Depending on which information is introduced then the corresponding transition is triggered: *Modeling Technique*, *Domain Pattern* or *Domain Requirements*. Each one of these transitions sends a corresponding signal to the *Begin* place. The signal in Begin makes the *Display* transition to be triggered then producing the corresponding information for *Diagram (Template)*, *Requirements Pattern* and *Requirements Classification* places. The Display transition takes the necessary information from the fusion places (identified with dotted lines) of figure: *T*, *L*, *O*, *R*, *C*, *D*, *A*. These fusion places contain the information that characterizes to domain requirements after the *Sort* transition has been triggered.

The arrangement scheme we propose allows the requirements information to be viewed in different ways. It could be displayed as templates, patterns or classification scheme. In this way the environment is a tool for domain analysis. The information that is stored in fusion places supports for the domain characterization in terms of generic requirements.

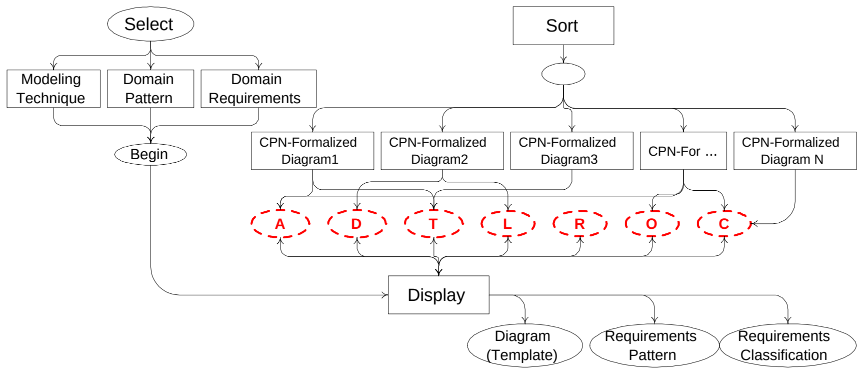


Figure 3: Petri net for requirements clustering. The Sort transition allows the requirements descriptive information to be put in fusion places. The *Select* place allows the sorting process to be triggered.

3.1 Classification Criteria for Requirements

Due to interdependencies between requirements [2], reusing a complete diagram in an application being different from the original one is a very difficult and unlikely scene. Reusing some information from diagrams is more workable. If this information may be (semi) automatically extracted then it makes the clustering of knowledge related to requirements easier.

The fundamental element of the different requirements diagrams is any instance of the Job metaclass (for example, each use case in a Use Case Diagram). The instances of Job metaclass show coordination and dependency relationships between them. They are also formed by other jobs and actions which have different relationships between them. These modeling units are the reusable elements in systems level software requirements. For this reason, they have to be arranged in order to be able to reuse them. The Jobs may be classified by applying two orthogonal criteria: (a) Domain Patterns and (b) Coordination Relationships.

Regarding to the domain patterns, we have adopted the domain independent regularities which are proposed in [11]. The decision tree of figure 4 shows these patterns. By combining the association and dependency relationships, a Job *S* falls in one and only one of following categories: Sensors, Production, Collaboration, Negotiation Step, Negotiation Step with Jobs Triggering, End of Negotiation, Complete Negotiation, Service, Unfinished Negotiation with Jobs Triggering, Unfinished Negotiation. Criteria to assign the jobs to the former categories are shown in the figure.

Regarding to coordination relationships, a Job *S* is assigned to one and only one of the following categories (where i means the number of steps in the job):

Temporal Coordination: $T = \{T_i / i = 2, 3, \ldots, m\}$

Linear Sequence: $L = \{L_i / i = 1, 2, 3, \ldots, n\}$

Selective Sequence: $O = \{O_i / i = 2, 3, \ldots, o\}$

Concurrent Sequence: $C = \{C_i / i = 2, 3, \ldots, p\}$

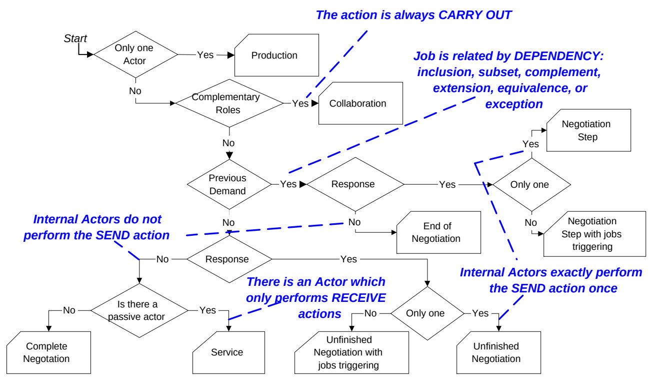

The action is always CARRY OUT
Start
Only one Actor
Yes
Production
No
Job is related by DEPENDENCY: inclusion, subset, complement, extension, equivalence, or exception
Complementary Roles
Yes
Collaboration
No
Negotiation Step
Yes
Previous Demand
Yes
Response
Yes
Only one
Internal Actors do not perform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

Figure 4: Classification of jobs by applying requirements patterns.

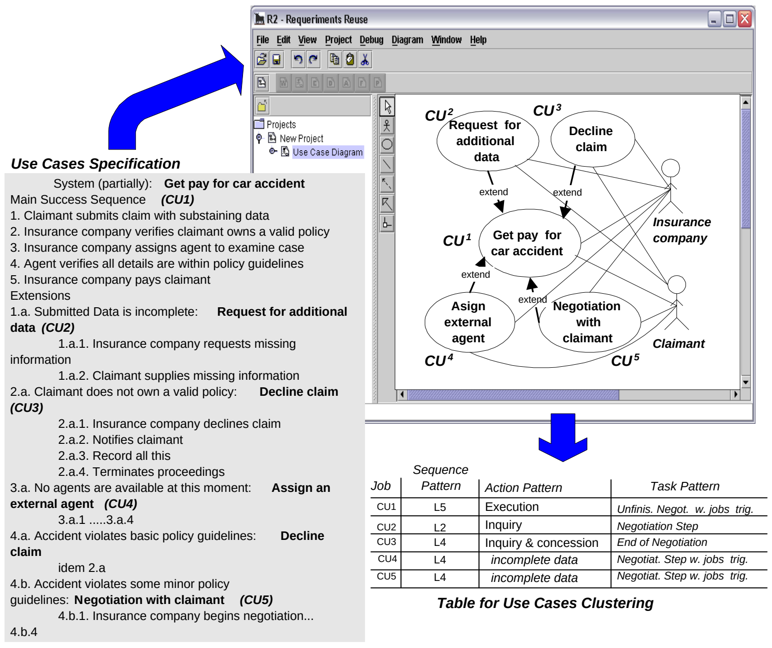
Resources Sharing: $R = \{R_i/i = 2, 3, \ldots, q\}$

The classification criteria (domain patterns and coordination relationships) when applied to a set of requirements diagrams allow a clustering for the jobs or domain services to be obtained. This distribution constitutes a table of classification displaying a view of the services or jobs belonging to each diagram. It constitutes a specification family in a domain. Each cell of the table represents a commonality point in the domain requirements. As a result, applying the classification scheme to a set of requirements projects allows a service taxonomy to be obtained. This taxonomy allows the typical domain pattern of job ordering to be displayed. One can benefit from this taxonomy through two ways, which have been mentioned in subsection 2.4. First, taking it as a guideline to the construction of reusable design and implementation elements in the domain. Second, taking it as reusable elements in development of new specifications in the domain. These two alternatives of requirements reuse should be addressed as a part of our future work.

3.2 An Example of Use Cases Arrangement

Use cases are a well known modeling technique which can be arranged by applying our proposed scheme for analysis and clustering of requirements. The corresponding translation of Use Case Diagram to CPN is obtained by applying the guidelines we have established in sections 2.2 and 2.3.

Figure 5 shows a use case diagram for policies claiming, borrowed from [3]. There are five use cases (CU^n) and two actors in this diagram. By applying the classification scheme to this diagram we assign the use cases to the corresponding categories. By following the decision tree of figure 4, the CU^1 use case is classified as Unfinished Negotiation with Jobs Triggering. In this CU^1 use case there exist two actors whose have complementary roles, there exits no previous demand and several responses are supposed to be produced. The CU^2 use case is classified as Negotiation Step due to there exists a previous demand and it is supposed to be produced a response from this use case. The CU^3 use case falls in the Negotiation Step category due to there exists a previous demand and there is no expected response from this case. The CU^4 and CU^5 use cases are classified as Negotiation Step with Jobs Triggering because of the



Use Cases Specification

System (partially): **Get pay for car accident**
Main Success Sequence ***(CU1)***
1. Claimant submits claim with substaining data
2. Insurance company verifies claimant owns a valid policy
3. Insurance company assigns agent to examine case
4. Agent verifies all details are within policy guidelines
5. Insurance company pays claimant
Extensions
1.a. Submitted Data is incomplete: **Request for additional data** ***(CU2)***
1.a.1. Insurance company requests missing information
1.a.2. Claimant supplies missing information
2.a. Claimant does not own a valid policy: **Decline claim** ***(CU3)***
2.a.1. Insurance company declines claim
2.a.2. Notifies claimant
2.a.3. Record all this
2.a.4. Terminates proceedings
3.a. No agents are available at this moment: **Assign an external agent** ***(CU4)***
3.a.1 .....3.a.4
4.a. Accident violates basic policy guidelines: **Decline claim**
idem 2.a
4.b. Accident violates some minor policy guidelines: **Negotiation with claimant** ***(CU5)***
4.b.1. Insurance company begins negotiation...
4.b.4

Job	*Sequence Pattern*	*Action Pattern*	*Task Pattern*
CU1	L5	Execution	*Unfinis. Negot. w. jobs trig.*
CU2	L2	Inquiry	*Negotiation Step*
CU3	L4	Inquiry & concession	*End of Negotiation*
CU4	L4	*incomplete data*	*Negotiat. Step w. jobs trig.*
CU5	L4	*incomplete data*	*Negotiat. Step w. jobs trig.*

Table for Use Cases Clustering

Figure 5: Use cases clustering for policy claiming system.

existence of previous demand for performing these use cases and, besides the example do not supply enough details, supposedly there exist several responses from these two use cases. Finally, the CU^3, CU^4 and CU^5 use cases are assigned to L_4 category because of they are specified as a linear sequence with four steps. Analogously, the CU^1 and CU^2 use cases are classified into the L_5 and L_2 categories, respectively.

4 Conclusions and Future Work

In this paper we propose a way for analysis and clustering of reusable requirements. The proposal allows the domain description to be obtained in terms of the coordination, association and dependency relationships related to the modeled services in different techniques under a common model for requirements. We address the formalization of the static and dynamic issues of the system in order to obtain the generic requirements which generalize the variability in a domain. The requirements formalization is supported by the CPN formalism.

We have not been looking for a technique for substituting the requirements diagrams but for formally represent them in order to precisely extract the information allowing the reuse based requirements classification to be obtained. The execution or simulation of the Petri net gives us the basis for the process of obtaining the description of diagrams which belong to different projects. All this description provides us a double mechanism for syntactic validation and automatic data extraction to classify the requirements in a domain.

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 design and implementation elements in a specific domain.

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] P. Carlshmre, K. Sandahl, M. Lindvall, B. Regnell, and J. och Dag. An industrial survey on requirements interdependencies in software product release planning. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pages 84–91. IEEE Computer Society, August 2001.

[3] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, Boston, 2000.

[4] J. L. Cybulski. Patterns in software requirements reuse. Technical report, Department of Information Systems. University of Melbourne, July 1998.

[5] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 3*. Springer-Verlag, Berlin, Germany, 2nd edition, 1997.

[6] K. Kang, S. Cohen, J. Hess, W. E. Novak, and A. 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.

[7] B. Keepence, M. Mannion, H. Kaindl, and J. 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.

[8] J. Kuusela and J. Savolainen. Requirements engineering for product lines. In *Proceedings of ICSE 2000, IEEE*. IEEE Computer Society, 2000.

[9] O. López, M. Laguna, and F. García. Representación de requisitos mediante redes de Petri coloreadas. Technical Report IT-DI-2002-1. Departamento de Informática, Universidad de Valladolid, 2002.

[10] K. Pohl. Requirements engineering, an overview. Encyclopedia of Computer Science and Technology, Vol. 36, Marcel Deccer Inc., 1996.

[11] 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.

[12] C. Rolland and N. Prakash. From conceptual modelling to requirements engineering. Technical Report Series 99-11, CREWS, 1999.

[13] M. Simos, D. Creps, C. Klingler, L. Levine, and D. 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.

[14] 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.