INTERNATIONAL WORKSHOP ON REQUIREMENTS REUSE IN SYSTEM FAMILY ENGINEERING

July 5th, 2004, Carlos III University of Madrid, Madrid, Spain

Co-located with International Conference on Software Reuse 8

POSITION PAPERS

Organisers: Juan C. Dueñas, Klaus Schimd

Contents

International Workshop on Requirements Reuse in System Family Engineering

http://www.dit.upm.es/iwreqfam

July 5, 2004, Carlos III University of Madrid, Madrid, Spain
Co-located with International Conference on Software Reuse 8

Goals

Requirements engineering can be nowadays considered as a mature branch of software engineering that deals with the efficient elicitation, description and management of systems and software requirements. Another interesting approach to the development of complex software systems is the system family engineering (also called product line engineering), that proposes techniques and methods for mass production of systems, focusing on the reuse aspects of development and guiding the creation of products (application engineering) from a large software base (domain engineering).

The joint of both disciplines leads to the concept of requirements engineering for system families; the first workshop specifically devoted to the issue was organised in 2002 in conjunction with the International Requirements Engineering Conference. This workshop is the second of that track.

Family engineering adds some axis of complexity to the traditional requirements engineering, especially in the elicitation of requirements, classification and analysis, which are the most advanced fields of work. The management of large sets of requirements, their interaction with stakeholders in the family engineering context, the traceability to other phases in development appear as problems for which research is yet open, especially as regards the provision of practical methods and tool support.

This workshop brings together software product line researchers, requirements engineering practitioners, tool builders, industry and academia, in order to exchange ideas and experiences, explores the state of the practice, discuss current and emerging practices and introduce new concepts in the area of requirements engineering in the context of system families, assisted with tools. A summary of the applicability of tool support in the requirements engineering for product families will be obtained. A list of techniques applied in practice is also sought.

Topics of interest include -among others-:
* tools for requirements engineering in system families,
* variability description in requirements,
* techniques for derivation of application requirements from domain requirements,
* decision models in requirements engineering,
* recovery of requirements in system families,
* traceability of requirements in system family engineering.

The workshop is mainly discussion-oriented; brief presentations by participants (20 minutes) are sought that foster discussions. Second, third and fourth sessions will be devoted to presentations of participants and discussions and further discussion on the issues presented.

Organizers

* Juan C. Dueñas. Universidad Politécnica de Madrid, Spain. jcduenas@dit.upm.es
* Klaus Schmid. Fraunhofer IESE, Germany. Klaus.schimd@iese.fraunhofer.de

Program committee

	Jan Bosch, University of Groningen. The Netherlands. Jan.Bosch@cs.rug.nl
	Juan C. Dueñas. Universidad Politécnica de Madrid, Spain. jcduenas@dit.upm.es
	Julio Cesar Leite, PUC-RIO, Brazil. julio@inf.puc-rio.br
	Frank van der Linden, Philips, The Netherlands. frank.van.der.linden@philips.com
	Birgit Geppert, Avaya, USA. bgeppert@research.avayalabs.com
	Klaus Schmid. Fraunhofer IESE, Germany. Klaus.schmid@iese.fraunhofer.de

Programme

10:00 Introduction and presentation

10:15 Session 1 Requirements reuse
State of the art and practice in requirements reuse in system families. Juan C. Dueñas.

11:00 Discussion 1

11:30 Session 2 Requirements modelling
Product Line Requirements based on Goals, Features and Use cases. Bruno González-Baixauli, Miguel A. Laguna, Yania Crespo.

A Meta-model for Requirements Engineering in System Family Context. Rodrigo Cerón, Jose L. Arciniegas, Jose L. Ruiz, Juan C. Dueñas.

Variability Description in Requirements for Product Family Support. Rafael Capilla.

12:30 Discussion 2

13:00 Lunch break

14:00 Session 3 Traceability
The Use of Satisfaction Arguments for Traceability in Requirements Reuse for System Families: Position Paper. Katrina Attwood, Tim Kelly, John McDermid.

Traceability for Product Family Systems An XQuery Approach. Waraporn Jirapanthong & Andrea Zisman.

SPL Needs an Automatic Holistic Model for Software Reasoning with feature models. David Benavides, Antonio Ruiz-Cortés, Rafael Corchuelo, and Octavio Martín-Díaz.

15:00 Discussion 3

15:30 Break

16:00 Session 4 Tool support
Tools Requirements for Requirements Engineering in System Family Context. Rodrigo Cerón, Francisco Valsera, Jose L. Arciniegas, Jose L. Ruiz, Juan C. Dueñas.

Experience-based Approach to Requirements Reuse in Product Families with DOORS. Antonio Monzón, Juan C. Dueñas.

Requirements Re-Use in the IT Business Process WEB Implementation Product Family. Lawrence E. Day.

16:30 Discussion 4

16:45 Summary and end of meeting

Product Line Requirements based on Goals, Features and Use cases

Bruno González-Baixauli, Miguel A. Laguna, Yania Crespo

Department of Computer Science, University of Valladolid,
Campus M. Delibes, 47011 Valladolid, Spain
{bbaixauli, mlaguna, yania}@infor.uva.es

Abstract. Traditional PL requirements approaches present several problems in requirements analysis, mainly in variants analysis and selection. The reason is fundamentally the difficult of deal with non-functional requirements. These problems can be solved with the introduction of the goal/softgoal paradigm. This paradigm introduces intentionality (the "whys") and allows relating functional and non-functional requirements, the basis of the variant analysis. This proposal improves the PL requirements introducing goal/softgoal paradigm and relating it with well-known techniques as feature modeling, and use case *extend* mechanism.

1 Introduction

Software reuse has been a very promising discipline for many years, but the results have not been so good as expected. Recently, Product lines (PL) appear as the more successful approach in the reuse field, as they combine coarse-grained components, i.e. software architectures and software components, with a top-down systematic approach, where the software components are thought a priori and integrated in a high-level structure [6].

Concerning requisites, the more used techniques are related with features. These techniques are focused in commonality and variability to find the main components (common to the entire PL). However, we think these techniques are too much design oriented, and complex to apply without a more requirements oriented technique.

Recently, requirements engineering (RE) has taken growing importance inside software engineering. One of its most important approach is the goal-oriented RE. It proposes to model explicitly the intentionality of the system (the "whys"). The intentionality has been widely recognized as an important point to the system, but usually it is not modeled. The main advantages of the goal-oriented approach are that can be used to study alternatives in software requirements (it uses AND/OR models that models the different alternatives) and that can relate functional and non-functional requirements in effortlessly.

Then , there are two important characteristics of goals that can be useful to the PL approach: first, they express the intentionality of the system, therefore they give a very natural way to take decisions. We can take decisions from "what we want", versus “what the system do”. Second, they can model alternatives in system requirements, what can be easily mapped to variants in PL.

However, this concept must be linked to PL, mainly with PL architectures. An easy way to do this is to relate goals with traditional features. Here, there is a lot of work done relating goals and use-cases/scenarios and use-cases/scenarios with features. Consequently, use-cases/scenarios can be a good joint point.

In this context, we propose an approach to PL requirements where the concept of goal as a guide for the selection of variants.

2 Using Goals, Features and Use Cases for Requirement Variability

PL Requirements define the products developable in the PL and their features. There are two specific characteristics different from traditional requirements: the main requirements of every PL product should be determined, even the requirements of the products not developed (must be forecasted); and it is fundamental to know their commonality and variability, and the dependencies between them.

To represent this kind of information, the requirements are usually structured in definition hierarchies [7] or feature models as in FORM (Feature-Oriented Reuse Method) [4]. Thus, each PL requirement is a prominent and distinctive concept or characteristic that is visible to various stakeholders, or feature. These features are organized by a graphical AND/OR hierarchy diagram, and set if mandatory, optional, or alternative. The main problems with this technique are that a large domain experience is needed, and it is also more oriented to the architecture definition than to client presentation or requirements definition.

Requirement elicitation can also be based on use case analysis (usually a more familiar technique). The basis for the use case support of the variability is the *extend* mechanism [3]. Although this technique is more oriented to requirements, the extend mechanism is not enough to represent complex variability.

The common characteristic of these two techniques is their solution-space orientation. It is already recognized that an intentional viewpoint that responses the “whys” is also necessary. In order to address this issue, large work has been done in the field of goal-oriented RE. A goal is an objective the system under consideration should achieve [8]. There are two types of goals: (hard) goals and *softgoals*: goals satisfaction can be established through verification techniques [9], but satisfaction of *softgoals* cannot be established in a clear-cut sense (usually used to model non-functional characteristics of the system) [8]. The dependences between goals and *softgoals* can be established. The NFR framework defines these correlations [1].

It is necessary to gather these three viewpoints: intentional (goals), operational (use cases) and functional (features). To achieve this, the use case and scenarios are the natural common point. There are already several works that relate goals with use cases or scenarios in a single system environment [5] and use cases with features in PL approaches [2]. The idea is to use the first techniques where goals help to construct use cases, and use cases assist in goals discovery. Then, with the use cases, find features in an easy way with the later techniques. The features are useful because they are closer to the architecture definition.

Traceability links goals with use cases and use case with features. Subsequently, the goals (intentionality) are related to features, and these to PL architecture and assets. In addition, a direct relationship between goals and features can be found, the goal-oriented task concept (the means for attaining the goals) can be easily mapped to the more implementation type features. Following this reasoning, we are separating the features in two concepts: the goals that model the capabilities features (general functionality and operations and non-functional requirements), and the tasks, that model all the other features types (operating environment, domain technology and implementation technique). In Fig. 1 the relations between the different PL requirement engineering models are outlined.

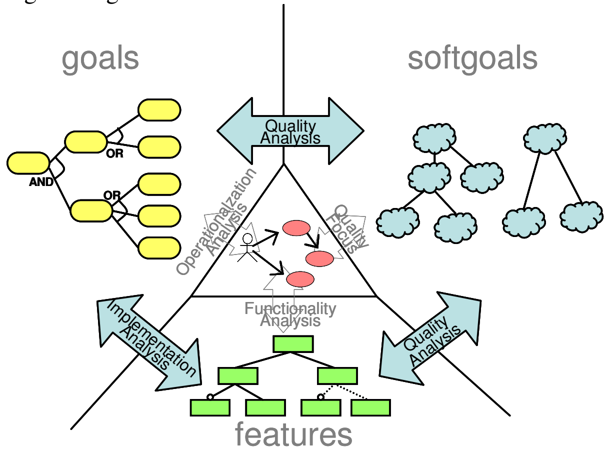

Fig. 1. Relationships between models. Goals are operationalized into use cases, whose variants are focused to different quality factors. Features implement the use cases actions with different contributions to *softgoals*. Also direct relationships (without use cases) are possible.

This approach also helps in the PL variants selection problem. In general, only the functional or operational point of view guides the selection of variants. We think that a more rigorous mechanism of selection is required. In this sense, goal approach allows to relate functional requirements with non-functional ones and to conduct formal analysis [1]. Therefore, goal analysis techniques can be used, and the selection is done from a more natural viewpoint (intentionality).

The idea is that, given a set of requirements variants, encoded in our model, this technique allows to select the better solution (set of features and related assets) according to a basic functional selection (*hardgoals*) and a given quality criteria (*softgoals*). Here, hardgoals limit the variability space and softgoals give the criteria to get the best variant from the limited variability space.

Nowadays, we are focused in the modeling aspect of the approach. We are extending UML to allow modeling goals and features with two new models. In addition, it is important the relationships between models: the goals cover use cases, the use cases are described by features, and features operationalize goals.

3 Conclusions and future work

The main contribution of this approach is the definition of a general model of requirement variability that incorporates goals as guide for the variants selection. This approach makes easier and more complete the PL requirement analysis introducing an intentional viewpoint and relating three well-known techniques. In addition, the variability analysis is improved by means of considering non-functional requirements with goal analysis. Consequently, the product architects have a rationale for the selection of those characteristics that had better support the requirements (functional and non-functional) of a new product line member.

We are working on a tool (adaptation from a previous one) that allow the correlation of goals, use cases and features models. This tool will focus on variant selection from the goals, selecting the desired functionality with the (hard) goals and the preferred quality properties by prioritizing *softgoals*. In this way, the features are hidden, but their relationships and constraints are used to get the variants. These relationships and constraints will be also used to find relationships on goals, comparing the possible variants from goal selections. Therefore, it is possible to know if one goal requires other (the goal is only achieved when the other), two goals are mutual-exclusive (there is no variants that achieve both goals), one hints to the other or they mutually hinders.

References

1. Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers 2000.
2. Griss, M., Favaro, J., and d' Alessandro, M. Integrating feature modeling with the RSEB. In Proceedings of the Fifth International Conference on Software Reuse, pages 76--85. IEEE Computer Society Press, 1998.
3. Jacobson I., Griss M., and Jonsson P.: Software Reuse. Architecture, Process and Organization for Business Success. ACM Press. Addison Wesley Longman (1997)
4. Kang, K. C., Kim, S., Lee, J., and Kim, K.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering, 5:143-168 (1998)
5. Kavakli, E., Loucopoulos, P., and Filippidou, D. Using Scenarios to Systematically Support Goal-Directed Elaboration for Information System Requirements. In Proceedings of IEEE Symposium and Workshop on ECBS'96, Germany:, 1996. pp. 308-314
6. Knauber, P., and Succi, G.: Perspectives on Software Product Lines. ACM Software Engineering Notes, 26(2):29-33 (2001)
7. Kuusela, J., and Savolainen, J.: Requirements Engineering for Product Families. In Proceedings of 22nd International Conference on Software Engineering – ICSE 2000. Pages 60-68. ACM Press (2000)
8. Mylopoulos, J., Chung, L., Yu, E. and Nixon, B.: Representing and Using Non-functional Requirements: A Process-Oriented Approach, IEEE Trans. on Software Eng, 18(6), June 1992 pp:483-497.
9. van Lamsweerde, A. *"Goal-Oriented Requirements Engineering: A Guided Tour"*, Proceedings of the 5 IEEE Int. Symp. on Requirements Engineering, 2001, pp:249-262

A Meta-model for Requirements Engineering in System Family Context

Rodrigo Cerón[1], Francisco Valsera, Jose L. Arciniegas[1], Jose L. Ruiz[1], Juan C. Dueñas[1]
Department of Engineering of Telematic Systems,
Universidad Politécnica de Madrid,
ETSI Telecomunicación, Ciudad Universitaria, s/n, E-28040 Madrid
{ceron, jlarci, jlruiz, jcduenas}@dit.upm.es
fmvalsera@ya.com

Abstract.

Software industries are pursuing the development of software intensive systems with a higher degree of reuse, reduction of costs, and shorter time to market. One of the successful approaches taken is based on the development of sets of similar systems where development efforts are shared. This approach is known as System Families. This position paper discusses an important issue in system family engineering activities: requirements modeling in system family context. The requirements must contain both the common and variable parts. Also, functional and non-functional aspects should be considered. Besides, an organization framework must be taken into account for requirements management. Traceability becomes an important issue in this context in order to be applicable in the industrial context.

1. Introduction

For many years, software industries have been trying to achieve the development of software intensive systems with a higher degree of reuse, cost reduction, and shorten of time to market. System Families (SF) are considered as one of the most successful approaches to do it. It focuses on the reduction of the time to market and development costs by the provision of a set of elements that are common to a number of systems. Therefore, it is a set of systems or services; also, it shares a significant part of their development effort. Since it shares many elements, it is sensible to think that the concepts behind them are also common. The main goal of this position paper is to present one approach to the requirements modelling and management in system families.

The basic work has been performed in the CAFÉ project [1]. The CAFÉ reference framework (CRF) gives a guide to classify the activities and models related with SF development. In summary, CRF can be divided in two main parts CAFÉ Process Reference Model (CAFÉ – PRM) and CAFÉ Assets Reference Model (CAFÉ - ARM). The objective of this model is to represent major activities and methods operating on the core assets, to allow the mapping of the contribution/tools against a common reference.

[1] Rodrigo Cerón and Jose L. Arciniegas are visiting professors from Universidad del Cauca, Colombia. Rodrigo Cerón is sponsored by COLCIENCIAS - Colombia. The work performed by José L. Arciniegas has been partially developed in the project TRECOM, granted by Spanish Ministry of Science and Technology under reference TIC2002-04123-C03-01. The work done by Jose L. Ruiz, and Juan C. Dueñas has been partially performed in the project CAFE and FAMILIES (Eureka 2023, ITEA ip00004, ip ip02009), partially supported by the Spanish company Telvent and by the Spanish Ministry of Science and Technology, under reference TIC2002-10373-E.

2. Related Work

There are many works in this area [2] [3] [4] [5] [6] [7]. There are many concepts. In consequence, there is a need for common understanding. Even more, in our analysis of the requirements management models, we have found that no single model covers in full extent the needs of SF engineering (SFE) currently. Thus, the work presented in this document takes as inputs the published meta-models provided, among others, by:

- SPLIT (Software Product Line Integrated Technology) [8]; whose key issue in the management of product line requirements is the capability to distinguish if a certain requirement is optional, alternative or parameterized for a certain product [4].
- The Helsinki University of Technology HUT model [9] provides a different view of goals, features and constraints. Its focus is on the creation of hierarchies and ordering requirements for priority.
- Siemens model for requirements modelling and traceability [10] allows the division of the set of requirements into shared, discriminated (optional, single-adapter, multiple-adapter, required-adapter and excluding-adapter).

Our attempt is to cover the specific needs in system family engineering regarding requirements management. A view of these needs has been provided by SEI [2], referring to the number of requirements in a system family (higher than for a single system), the number of stakeholders involved, also higher; the risks imposed by wrong requisites affecting the whole family; and the differentiation between common and specific requirements.

3. General Issues

The management of the SF reference requirements is supported by the capability to represent the information into a single model. It must contain both the common and variable requirements. From our viewpoint, there are several key specific issues that must be considered when performing requirements management in a SF, and that are seldom taken into account:

- Traceability: since the SF approach extends the complexity of the development by adding the variable-common dimension in each of the assets, the need of automatic support for navigation, traceability and calculation of coverage matrixes is much higher than in traditional product engineering.
- The separation of common and specific requirements can be further elaborated into several aspects: the explicit representation of the variability in requirements, including the basic relations [11], as well as other more detailed variability options.
- Binding, variability derivation or selection of the system specific requirements is the other side of the requirements management for SF. When a system or service engineering cycle is executed, requirements from the SF repository must be selected, adapted or modified.
- Decisions and conflicts management: keeping the set of requirements that are common for the SF is not enough for its efficient management; also, the requirements that are specific of a certain system must be kept in the common base. Thus, contradictory requirements (for different systems in the family) may coexist and the decision making process must reflect the conflicts that can appear while in the development. In any case, the decisions that lead to the requirements for a system must be recorded and justified.
- Reuse and evolution of requirements: keeping a central database with the requirements for the SF is desirable. From it, we must obtain system specifics requirement. However, the SF cannot be maintained alive if there is no capability for a requirement that appears in several systems to become a common requirement for the SF.

4. Context of Requirements in System Family Engineering

The SF can be described by several models. Models give a way of communication for the people into the organization. Requirement, architectural, design and test models can be distinguished into the SF environment. In our work Requirements is composed of five parts: the “Internal Traceability Mechanism” package, that contains elements related with horizontal traceability. The “Management Environment” package, that contains basic elements needed to organize SF in an organizational framework. The “Documentation” package, that contains the classes used to generate glossaries in relationship with a project. The “Requirement Management” package, that is responsible for managing and controlling requirements and their links with system and SF and finally, the “Requirement” package, which provides the means to define requirements, their types and their relationships.

5. Requirements meta-model

One of the aims desirable in SFE is having a common vocabulary that eases communication among the different members of an organization. This is the main reason for the existence of the “Documentation” package. Stakeholders involved in the PF will create entries in a glossary, in order to have a common vocabulary to unify the language used to define the requirements [12]. Obviously, the glossaries will lie under the umbrella of an organizational framework, described in the “Management Environment” package. An organization will possibly be in charge of several system families, among which these glossaries can be shared. A traceability mechanism will be necessary to manage the complexity associated to the development process inside a SF. This is the main purpose of the “Traceability” package. Functional requirements will be expressed by means of use cases. Decisions in the development process will possibly derive in conflicts. These conflicts will have an impact on the use cases previously built. Relationships among conflicts, decisions and use cases will be kept in this package. Decisions will solve the variability issues [13]. The stakeholders are responsible for providing requirements, and these can be common to all systems or specific to a subset of systems in the SF. In the process of derivation from the SF domain to the specific system domain stakeholders will take decisions. The specific requirements for a system will be obtained by means of decisions taken by the stakeholders. Within this process, conflicts might appear and should be solved again by the stakeholders before going on. These relationships are kept in the “Requirement management” package. Requirement taxonomy will be indispensable in order to organise and effectively manage a big amount of requirements [14]. Taking as resources [15] and [16], we have built a classification of the requirements. As a first criterion, requirements can be classified as functional or non-functional. Functional requirements can be classified again into other three categories: goal, feature and atomic. These categories are composable among them. Non-functional requirements have to be considered and are defined in terms of ISO-9126 quality model (its definition can be found there) [16]. Relationships among requirements have to be taken into account for reusing and maintainability purposes in SFE. As important relationships, kinds considered in the meta-model the following can be mentioned: generalization, consistency and revision. Each requirement should be characterised by a set of properties: priority, cost, importance, state and stability. This information would be useful to automate conflict resolution, decision process and evolution of the SF.

6. Conclusions and Future Work

We have performed this work trying to cope with the industrial requirements for methods, tools and mechanisms for SFE. In this domain, models are organized accordingly to organizational needs. The position paper gives a partial answer for the requirements modelling context. Further research must be done to solve it completely. In any case, the meta-model presented here covers several of the specific needs of the SFE as regards requirements management and the traceability to other systems-models (we have used UML [17] for the meta-model and for the experiments performed so far) in the development.
Back to the specific issues in requirements management for SF, the meta-model allows:

- The full navigation and traceability matrixes generation for internal traces, despite the complexity of requirements in the SF. Experiments performed with tool support for more than forty kinds of traceability matrixes. The usage of a web navigation schema helps in keeping the complexity under control.
- The basic relations between requirements are covered (exclusion, coexistence, mandatory, alternative), as well as relations during evolution (revision); hierarchies (generalisation, specialization) of requirements are also supported.

- The concept of system in the system family appears in this model as a set of requirements. Being these requirements related, the meta-model allows for the calculation of “coherent” systems, containing sets of non-conflicting requirements. As well, the progress in development of a certain system can be represented by the growing set of requirements covered.
- Decisions are represented explicitly. It is possible to navigate from each requirement to the decisions related, and the decisions related to each system. Conflicts are also expressed, so its reasons and stakeholders related can be identified and solved.
- The differentiation between common and specific requirements is not statically wired; instead, as there is navigational capability from-to systems and each requirement, it is possible to know in a certain point in time how many of the system in the SF are affected for one requirement.

We developed ENAGER. It is a tool for requirements management in the System Family context. It uses the meta-model we introduce in section 5. With it, we capture the requirements for an Open Service Gateway system family.

References

1. van der Linden, F.: Software Product Families in Europe: The Esaps & Café Projects. IEEE Software, Vol. 10 No. 4. IEEE Computer Society, Los Alamitos, CA (2002) 41-49
2. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison Wesley, Boston (2002)
3. El Kaim, W.: Managing Variability in the LCAT SPLIT/Daisy Model. In: Gacek C., Jourdan J., Coriat M. (eds.): Product Line Architecture Workshop, First Software Product Line Conference (2000) 21-31
4. Salicki, S., Farcet, N.: Expression and Usage of the Variability in the Software Product Lines. In: van der Linden (ed.): Proceedings of 4th international workshop on Software Product-Family Engineering PFE-4, LNCS 2290. Springer, Berlin (2001) 304-318
5. Griss, M.: Implementing Product-Line features by Composing Component Aspects. In: Donohoe P. (ed.): Software Product Lines: Experience and Research Directions: Proceedings of First International Software Product Line Conference. The Kluwer International Series in Engineering And Computer Science, Volume 576. Kluwer Academic Publishers (2000) 271-288
6. Jacobson, I., Griss, M., Jonsson, P.: Software Reuse, Architecture, Process and Organization for Business Success. ACM Press, Addison-Wesley, New York (1997)
7. van der Linden, F. (ed.): Development and Evolution of Software Architectures for Product Families. Proceedings International Workshop IW-SAPF-3, Las Palmas de Gran Canaria, Spain. Lecture Notes in Computer Science, 1951. Springer, Berlin (2000)
8. El Kaim, W., Cherki, S., Josset, P., Paris, F., Ollagon, J. C.: Applied technology for designing a PL architecture of a pilot training system. In: Knauber, P., Succi, G. (eds.): Software Product Lines: Economics, Architectures, and Implications, Workshop #15 at 22nd International Conference on Software Engineering, Limerick, Ireland. IESE-Report No. 070.00/E. Fraunhofer IESE, Sauerwiesen (2000) 55-64
9. Kuusela, J., Savolainen J.: Requirements Engineering for Product Lines. In: Proceeding of the 2000 International Conference on Software Engineering, ACM-IEEE, New York, NY (2000) 60-68
10. Luttikhuizen P. (ed.): Requirements Modelling and Traceability. ESAPS, WP3, Derivation of Products and Evolution of System Families WP3-0106-01 Technical Report, (2001)
11. Keepence, B., Mannion, M.: Using Patterns to Model Variability in Product Families. IEEE Software, Vol. 16 No. 4. IEEE Computer Society, Los Alamitos, CA (1999) 102-108
12. IEEE Standards Board: IEEE Recommended Practice for Software Requirements Specifications. IEEE std 830, 1993. Institute of Electrical and Electronics Engineers, New York, NY (1993)
13. Alonso, A., León, G., Dueñas, J.C., de la Puente, J.A.: Framework for Documenting Design Decisions in Product Families Development. In: Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems. IEEE Computer Society, Los Alamitos, CA (1997) 206-211
14. Thayer, R., Dorfman, M.: Software Requirements Engineering. 2nd edn. IEEE Computer Society Press, Washington (1997)
15. IEEE-SA Standards Board: IEEE Guide for Developing System Requirements Specifications. IEEE std 1233, 1998. Institute of Electrical and Electronics Engineers, New York, NY (1998)

16. ISO/IEC: Software engineering -- Product quality -- Part 1: Quality model. ISO/IEC 9126-1:2001. International Organization for Standardization, Geneve, (2001)
17. OMG: OMG Unified Modeling Language Specification. Version 1.5. Object Management Group, Needham, MA (2003)

Variability Description in Requirements for Product Family Support

Rafael Capilla

Department of Informatics and Telematics,
Universidad Rey Juan Carlos, Madrid, Spain
rcapilla@escet.urjc.es

Abstract. Product Lines and Product Family Engineering are a very promising area for developing intensive software systems. The construction of these software products and reusable components are supported by a software architecture that provides the means for customizing similar products from the same design. One of the problems for deriving a suitable software architecture is the transition from the requirements to the design because of the existence of requirements common to the domain and a set of different requirements for the variety of products we want to build. This variability makes complex the trace from the analysis to the design phase and vice-versa. In this work we deal with the problem of representing these requirements and how they can be enriched for supporting the variability of the products we want to build. Also, this would have positive effects for traceability purposes under common maintenance processes.

1 Introduction

The development of intensive software systems under product family approaches or product lines [1] constitutes a successful approach for building similar systems which fosters the production of reusable components. The reuse level is achieved because they employ common software pieces that can be used in the entire product family which is derived from a common software architecture [1]. In this scenario, the complexity of maintenance and evolution processes of the product family increases as the number of products and variations grow and as the requirements change over time. In contrast to the traditional software development, we have to manage a set of requirements which are common for the product family, while other specific requirements appear in specific products [3]. Therefore, the management of these requirements and its transition to the design phase becomes more complex than if only one product is developed. In this work we will explore alternatives for improving the representation of the diversity of requirements in order to gather the variability of systems and how this can be represented and managed in the requirements analysis phase. We will base our work on the IEEE std. for Recommended Practice for Software

Requirements Specifications (Std. 830-1998) [6]. One of the motivations of this work comes from the experience gained in a mid-term project for developing web systems for which a lightweight product line model was set-up for building the different web products. Each year with the renewal of the project, part of these requirements change or evolve while others remain stable. In this way, a suitable representation of the requirements and the variability becomes a need in the project.

2 Modeling Variability in Software Requirements

For representing the requirements in the project we used informal notations but close to the standards on requirements engineering. When the lightweight product was set-up for the first year, the software architecture was derived from the requirements and the initial web products were engineered from the core components developed under the product line. As the project evolves over time, part of the requirements change and new ones appear for building new software products. Thus, the need for representing and managing this variability [1] [4] becomes not only a design problem but also a challenge for the requirements specification. Our position in this work is to improve this situation by representing variable issues [4] [7] [8] [9] in the requirements description, not only in the design phase. To do this we evaluated the templates proposed in the standard IEEE 830-1998 as a recommended practice for requirements specification. From the eight templates proposed for describing specific requirements, we thought the organization by features (shown in figure 1) fits better than other templates because many proposals for representing the variability of systems can be done using a featured model, such as FODA [5].

1. Speciifc requirements
    - 1.1 External interface requirements
        - 1.1.1 User interface
        - 1.1.2 Hardware interface
        - 1.1.3 Software interface
        - 1.1.4 Communications interface
    - **1.2 System Features**
        - **1.2.1 System Feature 1**
            - **1.2.1.1 Purpose of feature**
            - **1.2.1.2 Stimulus / Response sequence**
            - **1.2.1.3 Associated functional requirements**
                - **1.2.1.3.1 Functional requirement 1**
    - 1.3 Performance requirements
    - 1.4 Design constraints
    - 1.5 Software system attributes
    - 1.6 Other requirements

Fig. 1. Requirements organized by features based on the standard IEEE Std. 830-1998.

After the requirements analysis phases was performed, we found two main limitations not well represented in the requirements template selected. The first one refers to the separation between common requirements for the entire product family and specific requirements for particular products. Of course, this can be solved describing the common requirements and specific product requirements in different documents. The second problem refers to the ability of the template to include more information in the requirements related to the future variation points. In fact, many software systems include quantitative values or range of values that are associated to functional operational modes. This aspect can be modeled as specific variation points with their own variants. Such as is mention in [2], featured models don't represent this explicitly but the same work describes an approach to extend these models using a simple notation. In order represent the same information at the requirement level, we propose in this work to extend the requirements template shown in figure 1 to include the following information:

1. *Common and product specific requirements*: the requirements template has two main parts. One for representing the common requirements for the domain or the entire product family and one subsection for each products or products who have particular requirements.
2. *Variation points*: This field indicates if the system feature would represent a variation point. The values allowed are: true or false.
3. *Type of value field*: for each system feature of a common or particular product requirement, this field includes the type of value allowed, such as for instance: numeric, string, etc.
4. *Range of value field / List of values*: for certain type of values, ranges or lists of values can be represented in this field. For instance, the database systems allowed for connecting a web system with data are: ms-access, My-SQL and oracle.
5. *Max / Min value field*: the maximum and minimum values allowed for a specific variant.
6. *Repetition value field*: in certain cases is necessary to represent the number of times a element can occur.
7. *Dependencies from common to single parts*: links for establishing dependencies or hierarchies between family and product specific requirements for tracing purposes.
8. *Graphical notation for distinguish product family and specific product requirements*: we propose a simple graphical notation that makes easy the distinction of requirements for the entire product line, specific product families and concrete products. The name of a specific product family or particular product can be attached to the notation or in the requirements list. This notation is shown in table 1.

Table 1. Notation for describing types of PFE requirements

SYMBOL USED	TYPE OF PFE REQUIREMENT
⬭	A requirement of the entire product line.
△△	A requirement of a specific product family (PF)
△	A requirement of a particular product in the PF.

The information previously describe can added to requirements template to achieve a more accurate description of the requirements for PF as well as for representing the variability such requirements. This proposals tries to improve the distinction between common and particular product requirements in the PF and facilitate the identification of the variation points in the design phase. Also, traceability issues can be favored because the transition from the analysis to the design phase results easy.

3 Transition from Requirements to Architectures

Based on the extended template for describing the requirements in product families, we propose a set of simple steps to perform the transition from the requirement analysis phase to the design phase. These steps are the following:

1. Identify and extract the requirements of the products we want to build.
2. Identify and separate the requirements common to the product family.
3. For each product or group of similar products identify specific product requirements a fill a new subsection in the requirements template adding the notation of table 1.
4. For product or group of products, identify which of the requirements associated to system features would be considered as initial variation points. Identify types of values, ranges and other extra information suitable to have variations in the future system and fill the appropriate subsections in the template.
5. Establish dependencies in the requirements list from the product family to products.
6. Check inconsistencies among the values and links already specified.

In addition to this, in the Web systems domain some of the variation points and variants affect to the user presentation layer in the architecture, which has impact in non functional requirements such as the usability of the system. For instance, the definition of font types, size, color or the allocation of the information and menu options if a Web system may affect the usability experimented by the user. In this way, the Web Accessibility Initiative (WAI) [10] outlines design principles for creating web accessible Web contents. These features that have impact in non functional requirements can be also modeled as software system attributes as shown in figure 1 and related to specific variations such as is mentioned in section 2. Therefore, some variation points of the system could be associated to non functional requirements that affect to the overall design of the system.

4 Conclusions

The aim of the position described in this paper encourages the inclusion of variability information in the requirements specification process. The limitations founded in the development of web products under a product line model have shown the lack of variability information in the analysis phase. This approach results useful for several reasons. First, to trace the results from the analysis to the design phase (i.e.: forward approach) and from the design to the analysis phase (i.e.: backward approach). Second, because is not difficult for a CASE tool to implement the information described in section 2. Third, because it relates functional and non functional requirements connected through the variation points defined in the architecture. Fourth, it distinguishes common requirements from specific product requirements. Finally, all these reasons facilitate the maintenance, reuse and evolution of the requirements in product family approaches and in particular for Web products but also opened to other domains or software systems.

References

1. Bosch, J., Design & Use of Software Architectures, Addison -Wesley (2000)
2. Capilla, R., Dueñas, J. C.: Modelling Variability with Features in Distributed Architectures. 4th International Workshop on Software Product-Family Engineering. Lecture Notes in Computer Science, Vol. 2290. Springer-Verlag, Berlin Heidelberg New York (2002) 319–329
3. Clements, P., Northrop, L. Software Product Lines. Practices and Patterns. Addison-Wesley (2002)
4. Dobrica, L., Niemelä, E.: Using UML Notation to Model Variability in Product Line Architectures, International Workshop on Software Variability Management, ICSE'03, Portland, Oregon, USA, (2003) 8-13
5. Kang K. C., Cohen S., Hess J. A., Novak W. E., Peterson A. S. Featured-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21 ESD-90-TR-22, Software Engineering Institute, Carnegie Mellon University, Pittsburgh (1990)
6. IEEE Recommended Practice for Software Requirements Specification, Std. 830-1998
7. Jacobson, I., Griss, M., Johnsson, P.: Software Reuse. Architecture, Process and Organization for Business Success, ACM Press (1997)
8. T. Myllymäki, "Variability Management in Software Product Lines", Tampere University of Technology, Software Systems Laboratory, ARCHIMEDES, http://practise.cs.tut.fi/pub/papers/VarMgnFinal.pdf, 2001
9. Robak, S.: Feature Modeling Notations for System Families, International Workshop on Software Variability Management, ICSE'03, Portland, Oregon, USA, (2003) 58-62
10. Web Accessibility Initiative (WAI). http://www.w3.org/WAI

The Use of Satisfaction Arguments for Traceability in Requirements Reuse for System Families: Position Paper

Katrina Attwood, Tim Kelly, John McDermid

Rolls-Royce University Technology Centre in Systems and Software Engineering, Department of Computer Science, University of York, Heslington, YORK. YO10 5DD United Kingdom
{katrina.attwood, tim.kelly, john.mcdermid}@cs.york.ac.uk

1 Introduction

Refinement of requirements into specifications depends on the concept of requirements 'satisfaction'. This is a recursive process, in which the fulfillment of lower-level requirements is seen as a sufficient condition for the fulfillment of the higher-level statement [1]. Traceability structures record satisfaction relationships between requirements at various levels of abstraction. It is essential for the successful reuse of requirements across system families that they contain sufficient information about the relationships between and context of requirements from early products in the family to assure developers that the requirements are valid for subsequent projects. Failure to observe this principle risks the late, and therefore costly, discovery of erroneous or unimplementable requirements.

In this paper, we offer a critique of standard traceability techniques and propose a method for developing traceability structures for requirements reuse.

2 Satisfaction Arguments and 'Rich Traceability'

Zave and Jackson [2] observe that satisfaction of a requirement (R) can be demonstrated only by a sufficient combination of domain knowledge (K) and specifications (S): $S, K \vdash R$. Jackson suggests that traceability links between requirements and specifications should be supported by textual 'correctness arguments' which explain how the specifications and domain behaviour combine to provide assurance that the engineered system satisfies the requirement in the application domain [3]. Jackson's 'correctness' arguments have been integrated into several industrial-strength processes for requirements engineering [for example, 4]. The 'Rich Traceability' technique [5] represents 'satisfaction arguments' using goal-structures charting AND/OR decompositions, the justifications for which are recorded. Fig. 1 documents the 'rich traceability' relationship between a top-level operational requirement for a vehicle and lower-level technical requirements. The 'satisfaction argument' introduces domain knowledge and indicates that a conjunction of the three specifications satisfies the top-level requirement.

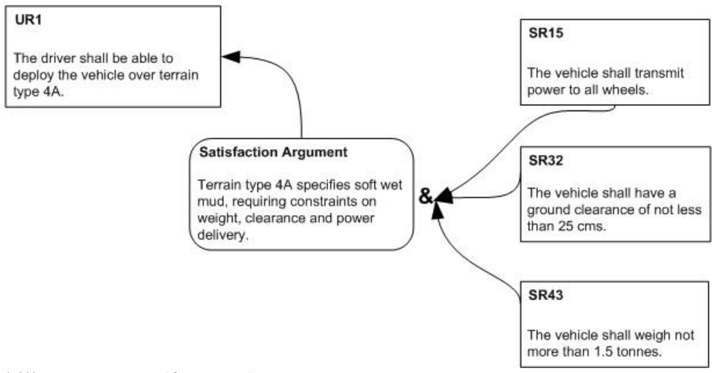
UR1
The driver shall be able to deploy the vehicle over terrain type 4A.
Satisfaction Argument
Terrain type 4A specifies soft wet mud, requiring constraints on weight, clearance and power delivery.
&
SR15
The vehicle shall transmit power to all wheels.
SR32
The vehicle shall have a ground clearance of not less than 25 cms.
SR43
The vehicle shall weigh not more than 1.5 tonnes.

Fig. 1. A 'rich traceability' structure (from [5])

3 Locating the Satisfaction Argument

The inclusion of a 'satisfaction argument' in the 'rich traceability' structure (Fig.1) provides an explicit record of the strategy used to decompose requirements statements and permits tracking of relationships and constraints between requirements statements at different levels of abstraction. In this respect, 'rich traceability' offers considerable advantages over simple traceability structures, which simply postulate links between requirements artifacts, with no attempt to justify or explain the connections between them. We are concerned, however, that 'satisfaction arguments' and traceability structures of this type do not actually demonstrate the satisfaction of the top-level requirement by the lower-level statements.

Fig. 1 puts forward no argument to convince the reader that the refinement tactic adopted will result in a specification which satisfies the requirement. No evidence is presented to support the implied claim that power, clearance and weight are the only concerns which need to be considered in assessing the vehicle's capabilities in the terrain. Nor is there any argument to demonstrate that the technical constraints specified in the system requirements are relevant to the top-level requirement and sufficient to satisfy it. What is offered here is not a 'satisfaction argument', but a series of propositions which need to be supported by arguments to convince the reader that the design meets the requirement. The real 'satisfaction argument', then, is not to be found in the decomposition strategy recorded here, but in an unexpressed meta-argument stating the basis on which this strategy is proposed.

4 Satisfaction Arguments and Safety Cases

Previous work at York has focused on the development of a graphical notation method for the construction of safety case arguments, the Goal Structuring Notation (GSN) [6]. Safety cases are semi-formal arguments which demonstrate how available evidence about a system and its context can be used to show that a system is acceptably safe to operate in its context [7].

As applied in requirements engineering, GSN provides a means for recording traceability links between individual claims and sub-claims. These claims are represented as goals, and equate to the requirements statements and specifications in the 'rich traceability' example (Fig. 1). The notation also records the strategies used to decompose the goals. These strategies map to the 'satisfaction argument' in Fig. 1, in that they seek to provide a basis for the relationship between the goals. As well as presenting a clear record of the goal-decomposition strategy, however, GSN allows this strategy to be validated by the use of an apparatus of justifications and assumptions attached to goals and strategies within the structure, as well as by explicit references to artifacts such as system architectural models or contextual information. This documents the meta-argument capturing the satisfaction relationships which justify the decomposition of requirements. GSN expresses justifications and assumptions as propositions, and there is no opportunity provided for their further development. A more sophisticated treatment of the meta-argument can be achieved by the use of GSN 'away goals', by which reference is made to a parallel goal-structure in which the meta-argument (our 'real' satisfaction argument) is elaborated.

5 Meta-Arguments and Requirements Reuse for System Families

Satisfaction arguments, expressed as meta-arguments on a requirements decomposition, provide assurance that the traceability relationship between requirements at different levels of abstraction is valid within a given application domain. If requirements are to be reused successfully between projects, it is important that satisfaction relationships remain valid for requirements and specifications in the reuse domain. The following example demonstrates how GSN can be used to document satisfaction arguments, and to indicate where changes in design commitments or contexts challenge reused requirements.

Our example concerns a system family of full-authority digital engine controllers (FADECs) for civil airliners. A FADEC provides a computer-controlled management system for the engine, which takes inputs from the cockpit controls and sensors located on the aircraft and produces outputs in the form of digital signals used to control the engine [8]. Engines are developed as a system family, comprising 'marks' and 'variants'. A 'mark' is a specific engine in a series, such as the Rolls-Royce BR-710 and the BR-715, while a 'variant' is a mark produced to the specific requirements of an Airframer. Requirements for a particular 'mark' are expressed in terms of a 'common core', with variations associated with specific 'variants'.

Inadvertent deployment of reverse thrust while an aircraft is in flight can result in severe yawing and, at worst, loss of control. The common core requirements for a small-engine series (we'll call it 'Engine A') have four discrete check systems, all of which must be actively disengaged before reverse thrust is deployed:

1. An isolation valve must be opened, allowing the flow of hydraulic fluid into the thrust reverse system. This valve is controlled by the FADEC, and its default position is 'closed'.
2. The Thrust Directional Control Valve must be switched to 'reverse' from its default 'forward' position. This valve is the joint responsibility of the FADEC and the aircraft, although the aircraft (i.e. the pilot) has ultimate override control.
3. The mechanical Tertiary Locks holding the thrust reverser door in 'closed' position must be opened. The aircraft has responsibility for these locks.
4. Throttle interlocks provide the pilot with tactile feedback concerning the degree of reverse thrust, and prevent him from requesting more reverse thrust than he requires.

Fig. 2 records a simplified decomposition, in GSN, of the common core requirements for the thrust reverse deployment protection system in 'Engine A'. The decomposition strategy providing the traceability links between the checks (R2 - R5) and the top-level requirement (R1) is documented in the rhomboid labelled S1. A rich apparatus of contextual assumptions and definitions is recorded alongside the decomposition.

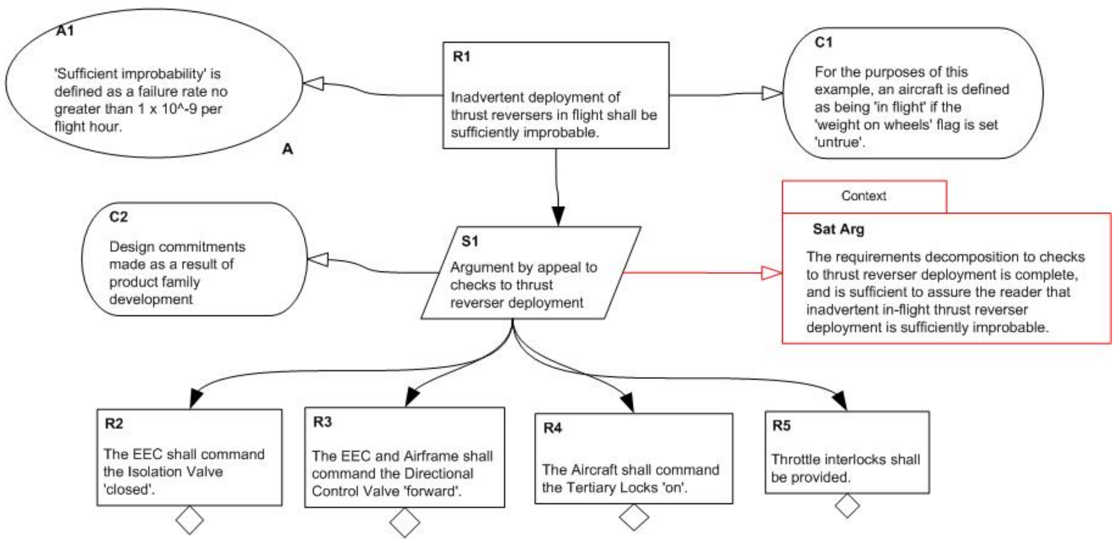
A1
'Sufficient improbability' is defined as a failure rate no greater than 1 x 10^-9 per flight hour.
A
R1
Inadvertent deployment of thrust reversers in flight shall be sufficiently improbable.
C1
For the purposes of this example, an aircraft is defined as being 'in flight' if the 'weight on wheels' flag is set 'untrue'.
C2
Design commitments made as a result of product family development
S1
Argument by appeal to checks to thrust reverser deployment
Context
Sat Arg
The requirements decomposition to checks to thrust reverser deployment is complete, and is sufficient to assure the reader that inadvertent in-flight thrust reverser deployment is sufficiently improbable.
R2
The EEC shall command the Isolation Valve 'closed'.
R3
The EEC and Airframe shall command the Directional Control Valve 'forward'.
R4
The Aircraft shall command the Tertiary Locks 'on'.
R5
Throttle interlocks shall be provided.

Fig. 2. Decomposition of core requirements for the thrust reverse deployment protection in the 'Engine A' family

The decomposition in Fig. 2 does not attempt to justify the decomposition strategy employed. Instead, it contains an 'away goal' reference to a justification claimed stored elsewhere (highlighted in red). This goal is the top-level claim of the meta-argument in Fig. 3, which is the satisfaction argument justifying the requirements decomposition. The argument strategy is a two-pronged one: the left-hand side of the goal structure argues that the checks, taken together, are sufficient to satisfy the top-level requirement, while the right-hand side (not fully developed here) argues that all possible failure modes have been considered and are adequately mitigated by the checks. The GSN structure makes clear what evidence is required to demonstrate the satisfaction of the top-level requirement (PSSA).

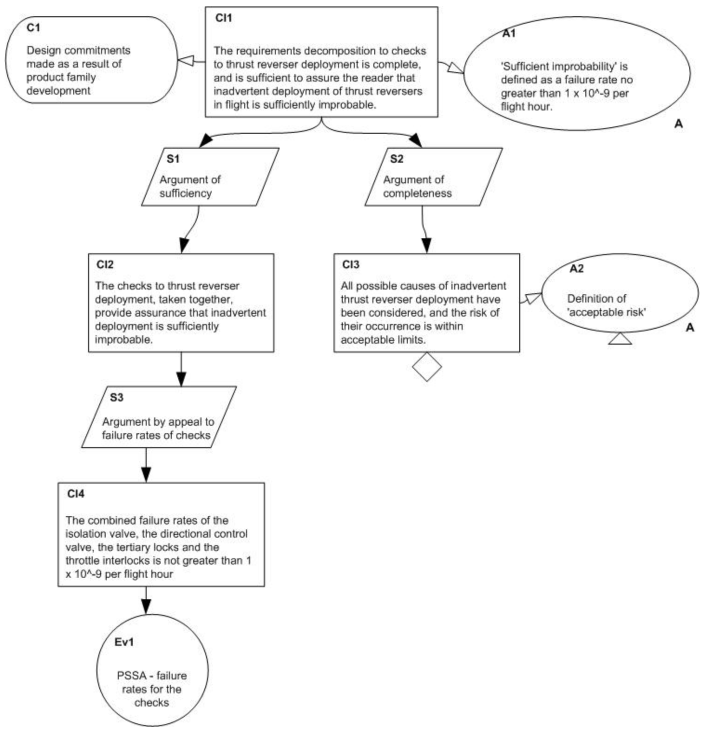
C1
Design commitments made as a result of product family development
Cl1
The requirements decomposition to checks to thrust reverser deployment is complete, and is sufficient to assure the reader that inadvertent deployment of thrust reversers in flight is sufficiently improbable.
A1
'Sufficient improbability' is defined as a failure rate no greater than 1 x 10^-9 per flight hour.
A
S1
Argument of sufficiency
S2
Argument of completeness
Cl2
The checks to thrust reverser deployment, taken together, provide assurance that inadvertent deployment is sufficiently improbable.
Cl3
All possible causes of inadvertent thrust reverser deployment have been considered, and the risk of their occurrence is within acceptable limits.
A2
Definition of 'acceptable risk'
A
S3
Argument by appeal to failure rates of checks
Cl4
The combined failure rates of the isolation valve, the directional control valve, the tertiary locks and the throttle interlocks is not greater than 1 x 10^-9 per flight hour
Ev1
PSSA - failure rates for the checks

Fig. 3. Satisfaction argument for core thrust deployment protection requirements in the 'Engine A' family

For historical and regulatory reasons, Airframer customers for certain variants in the Engine A family do not require the throttle interlocks feature on their engines. Safe reuse of the core requirements with this modification depends on R1 being satisfied by the three remaining checks. The GSN structure provides a straightforward means for assessing which aspects of the satisfaction relationship are threatened by the design change. If R5 (throttle interlocks) is removed from the requirements decomposition, reference to the satisfaction argument demonstrates that the sufficiency argument is compromised. It will be necessary to consult the PSSA to see whether the combined failure rates for the isolation valve, the directional control valve and the tertiary locks can combine to satisfy the overall safety requirement of a failure rate not greater than 1 x 10^{-9} per flight hour.

GSN satisfaction arguments thus allow for the clear record of domain information and assumptions, and indicate which information sources are required for adequate requirements traceability. The severity of the impact of requirements or contextual change can be assessed by reference to meta-arguments on the requirements decomposition. This is of clear benefit in system family development environments, where requirements reuse depends on the assurance of the validity of requirements decompositions in alternative development contexts.

References

1. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. Proceedings of the 5th IEEE International Symposium on Requirements Engineering. IEEE CS Press, Toronto (2001) 249-263
2. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology, Vol. 6 No. 1 (1997) 1-30
3. Jackson, M.: Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, London (2001)
4. Hall, A.: A Unified Approach to Systems and Software Requirements. Proceedings of the 5th IEEE International Symposium on Requirements Engineering. IEEE CS Press, Toronto (2001) 267
5. Hull, E., Jackson, K., Dick, J.: Requirements Engineering. Springer-Verlag, London (2002)
6. Wilson, S., Kirkham, P.: SAM User Manual. University of York, York (1995)
7. Kelly, T., McDermid, J.: Safety Case Construction and Reuse Using Patterns. Proceedings of the 16th International Conference on Computer Safety, Reliability and Security. Springer-Verlag, New York (1997) 55-69
8. Lam, W.: Achieving Requirements Reuse: a Domain-Specific Approach from Avionics. Journal of Systems and Software Vol. 38 (1997) 197-209

Traceability for Product Family Systems
An XQuery Approach

Waraporn Jirapanthong & Andrea Zisman

Department of Computing
City University
Northampton Square, EC1V 0HB, London, UK
{w.jirapanthong | a.zisman}@soi.city.ac.uk

Abstract. Traceability has been proposed as a mechanism to guarantee quality in the development life-cycle of a software system. In this paper we present an approach to support automatic generation of traceability relations between requirements artefacts for product family software systems. Our approach is rule-based in which the rules are represented in XQuery and the artefacts are translated into XML. We present eight different types of traceability relations between feature models, functional requirements specifications, and class diagram.

1. Introduction

Requirements Traceability (RT) has been recognized as an important activity in software system development [9][14][16]. In general, traceability relations can improve the quality of the product being developed, and reduce the time and cost associated with the development. In particular, traceability relations can support evolution of software systems, reuse of parts of the system by comparing components of the new and existing systems, validation that a system meets its requirements, understanding the rationale for certain design and implementation decisions in the system, and analysis of the implications of changes in the system.

Automatic generation and maintenance of traceability relations is not an easy task and support for traceability in software engineering environments and tools is not always adequate [16]. Some existing approaches assume that traceability relations should be established manually [2][11][18], which is error-prone, difficult, time consuming, expensive, complex, and limited on expressiveness. Therefore, despite its importance, traceability is rarely established. In order to alleviate this problem, more recently, other approaches have been proposed to support semi- or fully automatic generation of traceability relations [1][6][7][13][15]. However, in the majority of these approaches, the generated traceability relations do not have strong semantic meanings necessary to support the benefits that can be provided by traceability.

In order to overcome the difficulties above, in our previous work [17][20] we have proposed a rule based approach to allow automatic generation of bi-directional traceability relations between different types of requirements documents (i.e. customer requirements specifications, use-case specifications, and analysis object model). In this previous work we identify three different types of traceability relations, namely *overlap*, *require* and *realize* relations. In the approach, traceability rules are used to match syntactically related terms in customer requirements and use-case specifications with semantically terms in object model. Based on this matching and using other set of rules, the approach also supports generation of traceability relations between the customer requirements and use-cases.

The approach described in this paper is built upon our previous work. Here, we propose an approach to allow automatic generation of traceability relations between documents generated during the development of product family systems, in order to facilitate identification of common and variable functionality in systems composing the family, and reuse of core assets that are available under the product family architecture. We are interested in documents generated in feature-based methodologies due to the fact that when developing product family systems customers and system developers communicate with each other in terms of product features. We also believe that object-oriented methodology is important to support product family development. Therefore, we propose to generate traceability relations between documents produced when applying FORM [12] methodology and some object oriented documents such as component, class, and state chart diagrams. We decided to use FORM due to its simplicity, maturity, practicality, and extensibility characteristics. In this paper we concentrate on traceability for functional requirements, feature models, and class diagrams.

In our approach, the documents are represented in XML and the traceability rules in XQuery [19]. The rationale for using XML are due to several reason: (a) XML has become the de facto language to support data interchange among heterogeneous systems, (b) the existence of applications that use XML to represent information internally or as a standard export format, and (c) to allow the use of XQuery as a standard way of expressing the traceability rules.

The use of traceability to support development of product family systems is not a new thing. Some approaches have been proposed before [2][3][8][10]. However, to the best of our knowledge most of these approaches do not provide ways to generate traceability relations automatically.

The remaining of this paper is structured as follows. In section 2, we describe the main documents used in our work. In section 3 we present an overview of our approach, the different types of traceability rules, and traceability relations. Finally, in section 4 we summarise our approach and discuss directions for future work.

2. Types of documents

In this section, we give an overview of the functional requirements specifications, feature model, and class diagram used in our approach, and present extracts of these documents for a product family application related to mobile phones. Our discussion assumes a familiarity of the reader with the basic features of XML which, due to space limitations, cannot be given in this paper.

```
<FunctionalReqSpec System=”MobilePhone”
                   Product_Member=”MP1”>
 <Use_case UseCaseID=”1”>
 <Title> <VVB>Send</VVB> <NN0>data</NN0> </Title>
 <Status>Common</Status>
 <Region  Name = "All"/>
 <Description>…<AJ0>mobile</AJ0> <NN1>phone</NN1>
   <VM0>can</VM0> <VVI>send</VVI> <NN0>data</NN0>
  <VVN>kept</VVN> <PRP>in</PRP> <AT0> the</AT0>
  <NN1>phone</NN1> <PRP>to</PRP> <DT0>another</DT0>
  <NN1>phone</NN1> <CJC>or</CJC> <NN1>device</NN1>
  <PRP>via</PRP> <NN1>communication</NN1>
  <NN2>channels</NN2> <AV0>i.e.</AV0> <NN1>Bluetooth</NN1>…
 </Description>
 <Level>Primary Task</Level>
 <Preconditions>…</Preconditions>
 <Postconditions>…</Postconditions>
 <Primary_actor>…</Primary_actor>
 <Secondary_actors>…</Secondary_actors>
 <Flow_of_events>…</Flow_of_events>
 <Exceptional_events>…</Exceptional_events>
 <Related_Information>
 <Superordinate_use_case>…</Superordinate_use_case>
 <Subordinate_use_case>…</Subordinate_use_case>
 </Related_Information></Use_case>…</FunctionalReqSpec>
```

(a)

```
<FeatureModel>
  <Feature>
   <Feature_name>  <NN1>Bluetooth</NN1>
   </Feature_name>
   <Description>…<NN1>Bluetooth</NN1>
    <NN1>connection</NN1> <VM0>can</VM0>
    <VBI>be</VBI>  <VVN>used</VVN>
    <TO0>to</TO0>  <VVI>send</VVI>
    <NN0>data</NN0>  <AV0>i.e.</AV0>
    <NN2>texts</NN2>  <NN1>business</NN1>
    <NN2>cards</NN2>  <NN1>calendar</NN1>
    <NN2>notes</NN2>  <CJC>or</CJC>
    <TO0>to</TO0>  <VVI>connect</VVI>
    <AV0>Wirelessly</AV0>  <PRP>to</PRP>
    <NN2>computers</NN2>…
   </Description>
   <Issue_and_decision/>
   <Type>Capability</Type>
   <Commonality>Optional</Commonality>
   <Relation/>
  </Feature>
</FeatureModel>
```

(b)

Fig. 1. Examples of XML documents: (a) functional requirement specification for use case “Send data”; (b) feature model for “Bluetooth”

Functional Requirements: Functional requirements specifications are use-cases defined according to a template proposed in [4]. These specifications are represented in XML, following a DTD that we have created. A use case contains *title*, *status*, *region*, *description*, *level*, *preconditions*, *postconditions*, *primary_actors*, *secondatry_actors*, *flow_of_event*, *exceptional_events*, *superordinate_use_case*, and *subordinate_use_case*. We propose to mark up the words of the natural language sentences present in the use case descriptions by using XML elements that indicate the grammatical role of the words in the sentence. This grammatical role is identified by using the part-of-speech tagger called CLAWS [5]. Figure 1 (a) presents an example of a use case description for sending data from a mobile phone, for a member (MP1) of the product family. In this example, the words that appear as the content of elements <Title> and <Description> are marked-up with the grammatical roles. For instance, the word “Send” is tagged with <VVB> denoting that it is a base form of a lexical verb, while the word “data” is tagged with <NN0> denoting that it is a neutral noun. For a complete description of the tags given by CLAWS, please refer to [5].

Feature Model: In our work we use an XML representation of the feature model proposed in [12]. According to [12], the model is structured like a graphical tree. Each feature of the system is represented in a feature model and includes the following information: *feature_name*; *description; issue_and_decision*, describing issues and decisions that arise during the feature analysis process; *type*, that classifies the feature in terms of *capability*, *operating environments*, *domain technology*, or *implementation techniques*; *commonality*, denoting if a feature is *mandatory*, *alternative* or *optional*; and *relations*, representing relationships with other features. An example of a feature model for a *Bluetooth* feature presented in mobile phone systems is shown in Figure 1 (b). This feature is "optional" (i.e. not every product member in the family needs to have this feature) and of type "capability" (i.e. Bluetooth is concerned with an ability of the system).

Class Diagram: The class diagram is actually a UML class diagram, including classes, attributes, operations, associations, and generalisation relations between classes and is represented in the eXtensible Metadata Interchange (XMI) format.

3. Overview of the approach

Figure 2 presents an overview of the process of our approach. Initially, the documents of our concern are translated into XML by using an XML translator. In the case of the class diagram, the XMI format is generated by using commercial XMI exporter (e.g. Unisys XMI exporter for Rational). The XML translator is also responsible to add the XML POS-tags in the sentences after identifying this tags using CLAWS. These XML documents are used as input to the *traceability generator* that creates traceability relations based on the rules represented in XQuery [19]. The traceability relations are also represented in XML format. As described below, some of the traceability relations will be used to support identification of new traceability relations (*derived relations*). In the figure this is represented by using the XML-formatted relationships documents as input to the traceability generator.

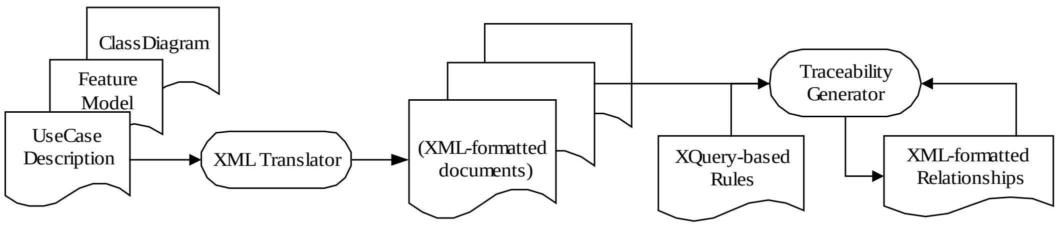

Fig. 2. Overview of the approach

Types of Traceability Relations

According to [16], there are various dimensions for capturing the relations between system elements. We have identified eight types of traceability relations for product family systems for the three types of documents used in our work. We classify the traceability relations in two groups: *direct* and *derived* relations. The *direct* relations represent associations that are identified by matching terms in the documents. The *derived* relations represent associations that are identified based on the existence of other traceability relations identified by our traceability generator. Both *direct* and *derived* relations are sub-classified into other types of relations. Table 1 presents a summary and brief explanation of these different types of relations. Note that some types of relations, particularly *mutual exclusive* and *requires,* can be both *direct* and *derived* relations. Relations *alternative* and *additional* allow identification of variable aspects of the product members in a family, while relation *similar* supports identification of common aspects among these products. In the table, the term *element* is loosely used to refer to a feature, an object, an item, a functionality, or a characteristic of the system.

Table 1. Types of relationships

Type	Relationship	Explanation
Direct	Aggregation	Expresses that an element is composed of other elements.
Direct	Generalisation/ Specialisation	Expresses an abstraction between an element and a set of its specialised elements.
Direct	Implemented	Expresses that an element is implemented by another element.
Direct/ Derived	Mutual Exclusive	Expresses that two elements require the existence of each other.
Direct/ Derived	Requires	Expresses that an element requires the existence of another element.
Derived	Alternative	Expresses different ways of achieving the same functionality in different product members of the family.
Derived	Additional	Expresses extra functionality that may exist between product members of the family.
Derived	Similar	Expresses common elements in the product members of the family.

Traceability Rules

In our approach the traceability relations given above are identified by using traceability rules. We propose to represent the rules in XQuery [19] due to its power and support for retrieving data from XML documents. XQuery allows the location of specific elements and attributes in an XML document by using XPath expressions. Apart from the embedded functions offered by XQuery, it is possible to add new functions and commands. In our work, it is necessary to add new functions in order to cover some of the traceability relations being proposed. Examples of these functions are *synonym*, to identify words with the same meaning, *distance_control*, to measure the distance between words in a text.

Figure 3 shows an example of a rule in XQuery to identify *implemented* relations between use case and feature models. Each rule has a unique identification and type associated to the traceability relation identified by the rule. The XQuery part is composed of two parts. The first part (*for*) identifies the elements of the documents being compared (i.e. <Feature> and <Use_case>). The second part (*where*) contains the conditions that should be satisfied in order to create the traceability relations. The conditions of this rule verify if there is a singular noun (<NN1>) in the name of the feature that appears in the description of the use case, and the title of the use case is composed of a neutral noun (<NN0>) and a verb (<VVI>) that appear in the description of the feature model.

```
<TraceRule RuleID="R1" RuleType="Implemented">
  <Query>
    <![CDATA[ for $fm in doc("file:///c:/FeatureModel.xml")//Feature_model/Feature,
       $uc in doc("file:///c:/UseCases.xml")//Use_case
       where some $t in $fm/Feature_name/NN1 satisfies (contains(string($uc//Description), $t))
         and (some $p1 in $fm/Description/NN0 satisfies contains(string($uc//Title), $p1))
         and (some $p2 in $fm/Description/VVI satisfies contains(string($uc//Title), $p2))]]>
  </Query>
  <Action> <Relation type="Implemented"/>
             <Elements>$fm/feature_name</Elements>
             <Elements>$uc/title</Elements>    </Action> </TraceRule>
```

Fig. 3. Example of a rule in XQuery for *implemented* relations

The parts of the documents to be related when the conditions of the rule are satisfied, is specified by element <Action>. In order to illustrate, consider the use case and feature model shown in Figure 1. In this case, a relation of type *implemented* is created between the use case and the feature model, since the feature name “Bluetooth” appears in the description of the use case, and “send data” appears in both the title of the use case and the description of the feature model.

Another example of a rule to identify *similar* relations between two use cases is shown in Figure 4. In this case, the relation signifies that two different product members of a family have some similarities if they implement the same feature. This relation is of type derived and depends on the existence of previous *implemented* relations between use cases and feature models (i.e. different use case titles and the same feature names).

```
<TraceRule RuleID="R2" RuleType="Similar">
  <Query> <![CDATA[ for $item1 in doc ("file:///c:/traceRelations.xml")//link,
                 $item2 in doc ("file:///c:/traceRelations.xml")//link
                where string ($item1/feature_name) = string ($item2/feature_name)
                   and string ($item1/title) != string ($item2/title)]]> </Query>
  <Action> <Relation type="Similar"/>
                 <Elements>$item1/title</Elements>
                 <Elements>$item2/title</Elements>
                 <Elements></Action> </TraceRule>
```

Fig. 4. Example of a rule in XQuery for *similar* relations

4. Conclusion & Future work

This paper presents a rule-based approach to allow automatic generation of traceability relations in documents for product family systems. We have identified eight different types of traceability relations between feature models, functional requirements specifications, and class diagrams. We propose to use XQuery to represent the rules.

Currently, we are implementing the traceability generator in order to evaluate our work. Before large-scale experimentation and use, we are extending our work to support traceability generation between other documents for product family software development like component diagrams, state chart diagrams, process models, and subsystem models.

References

1 Antoniol G.; Canfora G.; Casazza G.; De Lucia A.; Merlo E.: Recovering Traceability Links between code and Documentation, IEEE Transactions on Software Engineering, Vol. 28, No. 10, pp. 970-983, October 2002.
2 Bayer J.; Widen T.: Introducing Traceability to Product Lines, Proceedings of Software Product-Family Engineering: 4th International Workshop (PFE 2001), Bilbao, Spain, Lecture Notes in Computer Science, ISSN: 0302-9743.
3 Biddle R.; Noble J.; Tempero E.: Supporting reusable use cases, In Proceedings of the Seventh International Conference on Software Reuse, April 2002.
4 Cockburn A.: Structuring Use-Cases With Goals, JOOP, Sep-Oct 1997.
5 CLAWS: https://www.comp.lancs.ac.uk/ucrel/claws.
6 Cleland-Huang J.; Chang C.K.; Sethi G.; Javvaji K.; Hu H.; Xia J.: Automating Speculative Queries through Event-based Requirements Traceability, Proceedings of the IEEE Joint International Requirements Engineering Conference, Essen, Germany, September 2002.
7 Egyed A.: A Scenario-Driven Approach to Trace Dependency Analysis, IEEE Transactions on Software Engineering, Vol. 9, No. 2, February 2003.
8 Esaps: http://www.eso.es/en/projects/esaps/esaps.html, 2001
9 Gotel O.; Finkelstein A.; An Analysis of the Requirements Traceability Problem, the First International Conference on Requirements, England, pp. 94-101, 1994.
10 Halmans G.; Pohl K.: Communicating the variability of a software-product family to customers, Springer Verlag 2003, February 2003.
11 Integrated Chipware; RTM: www.chipware.com
12 Kang K.: FORM: a feature-oriented reuse method with domain-specific architectures, in Annals of Software Engineering, Vol. 5, pp. 354-355.
13 Marcus A.; Maletic J.I.: Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing, ICSE, 2003.
14 Pohl K.; Process-Centered Requirements Engineering, John Wiley & Sons, Inc., 1996
15 Pinheiro F.; Goguen J.: An Object-Oriented Tool for Tracing Requirements, IEEE Software, pp. 52-64, March 1996.
16 Ramesh B.; Jarke M.: Towards Reference Models for Requirements Traceability, IEEE Transactions on Software Engineering, Vol. 37, No. 1, January 2001.
17 Spanoudakis G.; Zisman A.; Perez-Minana E.; Krause P.: Rule-based Generation of Requirements Traceability Relations, Journal of Systems and Software, Vol. 72, No. 2, pp. 105-127, 2004 (to appear).
18 Teleologic; Teleologic DOORS: www.teleologic.com/products/doors
19 XQuery: http://www.w3.org/TR/xquery/
20 Zisman A.; Spanoudakis G.; Perez-Minana E.; Krause P.: Towards a Traceability Approach for Product Families Requirements, 3rd ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Orlando, USA, 2002

SPL Needs an Automatic Holistic Model for Software Reasoning with feature models [⋆]

David Benavides, Antonio Ruiz-Cortés, Rafael Corchuelo, and Octavio Martín-Díaz

Dpto. de Lenguajes y Sistemas Informáticos
University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
{benavides, aruiz, corchu, octavio}@us.es

Abstract The number of features and their relations in a Software Product Line (SPL) may lead to have SPLs with a big number of potential products which may be difficult to manage. This number of potential products widely increases if, as well as functional features, extra–functional features are taken into account. There are several questions that a SPL engineer would like to ask to his SPL model such as: is it a valid model?, how many potential products a SPL has?, is there any product fulfilling the customer needs? and so forth. These types of questions are error prone to answer without an automatic support. The work reported in this position paper glipmses some misconceptions of previous related proposals: we uphold the need to have an holistic product line model were not distinction are made between functional and extra–functional features, we propose a model based on a formalism strong enough to support both type o features: contraint programming.

1 Introduction

The number of potential single products of a SPL increases with the number of features. Thus, a big number of features lead to have SPLs with a big number of potential products. In a extremely flexible SPL where all features may or not appear in all potential products the number of single products is equals to 2^n, being n the number of features. Moreover, current feature models are only focussed on modelling functional features and in the most cited proposals [5,6,8,10,17] there is a lack of modelling artifacts to deal with extra–functional features (features related to quality or non–functional aspects). If extra–functional features are taken into account the number of potential single products increases still more.

A SPL engineer may want to ask several questions to their SPL model such as : it is a valid model? How many potential single products are there? Is there any single product fulfilling the user's requirements (both functional and extra–functional)?, and so on. Answering these types of questions is not a trivial task and doing it by hand is obviously error prone. The current proposals lack from a complete formal base for reasoning on SPL able to automatically answer to these types of questions.

⋆ The work reported in this article was partially funded by the Spanish Ministry of Science and Technology under grant TIC2003-02737-C02-01

In this paper we outline the need for an Holistic SPL Model where the main characteristic of the model regarding others would be: $i)$ using the same formalism for functional and extra–functional features without making distinction between them in order to avoid manichaean discussions [1, pag.76] (holistic point of view). $ii)$ modelling a SPL as a Constraint Satisfaction Problem (CSP) [12] which allows reasoning on SPL with functional and extra–functional features. $iii)$ fixing some misconceptions of [11] regarding to the operators used to model SPLs $iv)$ presenting a running prototyping implementation that has not been presented as far as we know in other proposals.

The remainder of the paper is structured as follow: section 2 gives an overview of current feature models . Section 3 point out the problems we identify in current proposals giving some conclusions.

2 Feature Models

One of the main activities in SPL engineering is to identify commonalities and differences between different single products of a SPL. Feature modelling is an approach for such a purpose. The main output of feature modelling is a feature model which intend to have a compact representation of all potential single products of a SPL.

A feature model describe a SPLs as a set of features and relations between them. Roughly speaking a feature is a distinctive characteristic of a product. Depending on the development stage, a feature may refers to a requirement [7], a component in an architecture [2] or even to pieces of code [15] of a SPL. A feature model is composed of a set of features disposed in an hierarchically structure. Thus, A feature model is considered as the space of potential single products. A single product in a feature model is also called a feature instance. Thus, a feature model has 1 to n feature instances.

Consider the Home Integration System (HIS) domain as an example [1]. Figure 1 represents a possible feature model of a product line of HIS.

There are several notations to specify feature models [5,6,10,8,17]. In figure 1, we use the notation proposed in [5, pag. 86] because we consider it to be the most comprehensible and flexible proposal. In this notation there are four relations types between features:

- **Mandatory features**: a mandatory feature is always present in a feature model instance when its parent feature is present.
- **Optional Feature**: an optional feature may be or not present in a feature model instance when its parent feature is present.
- **Alternative Features**: an alternative feature of a set of alternative features may be present in a feature model instance if its parent feature is included. Then, exactly one feature of the set is present.
- **Or features**: an or–feature of a set of or–features may be present in a feature model instance if its parent feature is included. Then, at least one feature of the set may be present.

In figure 1 three main features can be distinguished for a HIS product line:

[1] This example is partially inspired in [10].

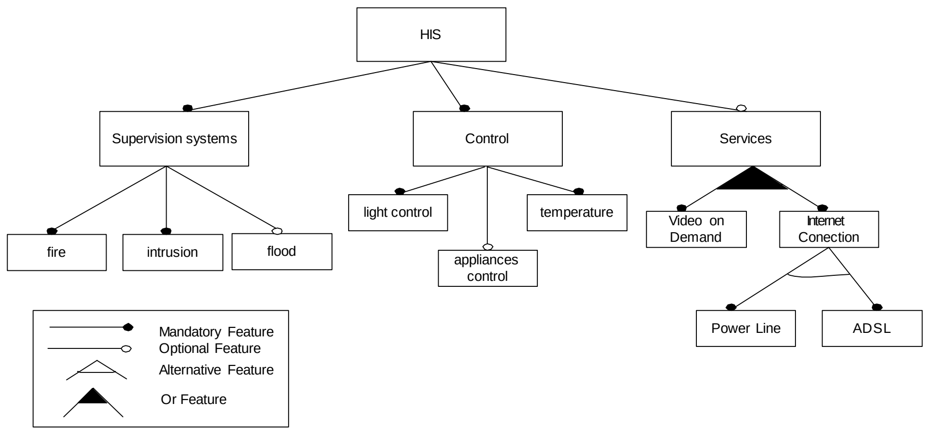

Figure 1. Feature model of a HIS

– Supervision systems: for fire, intrusions and flood alerts.
– Control: electricity, temperature and appliance control.
– Services: optional services like video on demand, Internet connection and so on.

In this feature model it is clearly expressed the capability of having different products for the HIS product line, e.g.:

– **Basic product**: with neither services, nor flood supervision, nor appliance control.
– **Full product**: a product with all the features as well as a power line or an ADSL Internet connection.

3 Problems with current feature models

3.1 Current Only Fucntional Feature Models

In figure 1, every feature refers to functional features of the HIS product line so that, every feature instance differs because of its functional features. However, every feature of figure 1 may have associated extra–functional features. For instance, consider the *services* feature, it is possible to identify extra–functional features related to it, such as *availability, reliability, number of lines of code, time of development, cost* and so forth. Likewise the feature *Internet Connection* can have extra–functional features such as *latency* or *bandwidth*. Furthermore, the value of every extra–functional feature related to every functional feature can vary from one feature instance to another. That is to say, every feature instance differs as well as because of its functional features, because of its extra–functional features.

Consider the full product of the HIS product line example presented formerly with the same functional features. It is possible to offer several product with the same functional features but different extra–functional features, for instance:

– **High quality full product**. A product with full functionality and high quality: high *availability* and *reliability* and high *cost* too.
– **Basic quality full product**. A product with full functionality but lower quality: lower *availability* and *reliability* and lower *cost* too.

Until now, we have not found any proposal dealing with functional and extra–functional features in the same model. However, there are some works in the literature suggesting the need of dealing with extra–functional features: Kang *et. al* have been suggesting the need of take into account extra–functional features since 1990 [8, pag. 38] when they depicted a classification of features but they did not provide a way to do it. Later in 1998 Kang *et. al* [9] made an explicit reference to what they called 'non–functional' features (a possible type of what we call extra–functional features). However the authors again did not give a way to do it. Later in 2001 Kang *et. al* [3], proposed some guidelines for feature modelling, in [3, pag. 19], the authors made again distinction between functional and quality features and pointed out the need of an specific method to include extra–functional features, however they did not provide such a specific way to do it either. In [16], the authors made distinction between functional and what they called "parameters" (another possible type of what we call extra–functional features). The authors marginally introduce a way to deal with this kind of features. Nevertheless there is a main drawback in this proposal: the authors propose to include the values of the parameters in the feature model, this is to say having a pair parameter/value. This way it is not allowed to have ranges or sets of values or even more, it is not possible to have aritmetical relations between attributes. In such a way, the possible instances of the SPL are limited to the values specified in the feature model.

Although we agree that in a SPL there are both functional and extra–functional features, in our opinion, there is no need of separating functional and extra–functional features due that this separation may lead to fall in manichaean discussions [1, pag.76]. For instance, an ADSL connection can have and attribute: the *bandwidth*, this attribute can be seen as a quality feature therefore as an extra–functional feature. Nevertheless, depending on where the attributes takes its values it can also be seen as a functional feature. Thus, if there are only two possible configurations of ADSL connections, 128 or 256, every of this configurations can be seen as an individual functional feature as illustrated in figure 2.

We propose to have an Holistic SPL Model with the following characteristics:

– functional features and the relations described previously: Mandatory, Optional, Alternative and Or features.
– extra–functional features related to any functional feature.

However, we use the same formalism in the model so we do not make distinction between functional and extra–functional features.

3.2 Reasoning on Holistic Feature Models

The number of potential single products in a SPL increases with the number of features the SPL contains. If extra–functional features are also considered this number increases still more. In this context, reasoning on SPL should be considered and some questions can be inquired to the holistic model:

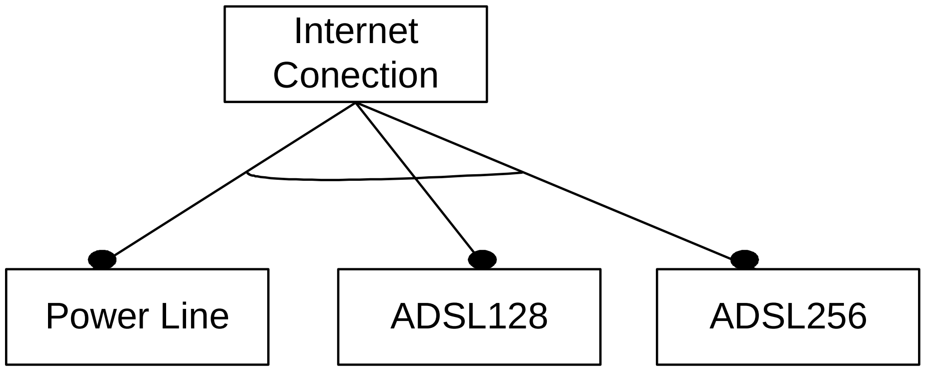

Figure 2. Feature model with two possible ADSL connections

- *Is is a valid model?*: a valid model would be one on which at least a single product can be selected.
- *How many potential single products a SPL has?*: answering this question can give an outlook of the flexibility of the model. Greater the number of potential single products is, more flexibility is supported but greater the complexity of the SPL is too.
- *Is there any product fulfilling the user's requirements (both functional and extra–functional)?* From the potential single products of the SPL, a user can require some features so the space of the potential single products may be reduced for this user.
- *From the products fulfilling the user's requirements which are the products that maximize or minimize an extra–functional feature? e.g.: a user need a product where the* price *or the* number of lines of code *are minimized.*

Excluding a couple of limited proposals [4,11], as far as we know, any other proposal have been presented to automatically reason on feature models until now. Van Deursen [4] and Mannion [11] provides a somehow formal support for automatic reasoning on feature models. However, none of them deals with extra–functional features, so they are focussed on functional aspects of SPLs. We propose to have an holistic model formalized using constraint programming [12] using the background we have [14,13]

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison–Wesley, 1998.
2. M. Bernardo, P. Ciancarini, and L. Donatiello. Architecting families of software systems with process algebras. *ACM Transactions on Software Engineering and Methodology*, 11(4):386–426, 2002.
3. G. Chastek, P. Donohoe, K.C. Kang, and S. Thiel. Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, June 2001.

4. A. van Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.
5. K. Czarnecki U.W. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison–Wesley, may 2000. ISBN 0–201–30977–7.
6. M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of theFifthInternational Conference on Software Reuse*, pages 76–85, Canada, 1998.
7. S. Jarzabek, Wai Chun Ong, and Hongyu Zhang. Handling variant requirements in domain modeling. *The Journal of Systems and Software*, 68(3):171–182, 2003.
8. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
9. K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature–oriented reuse method with domain–specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.
10. K.C. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/August 2002.
11. Mike Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer.
12. K. Marriot and P.J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, 1998.
13. O. Martín-Díaz, D. Benavides, J. Pe na, and M. Toro. Un tratamiento sensible a la calidad para la adquisición de servicios web. In *VIII Jornadas de Ingeniería del Software y Bases de Datos JISBD'03*, pages 209–220, Alicante, España, 2003.
14. O. Martín-Díaz, A. Ruiz-Cortés, A. Durán, D. Benavides, and M. Toro. Automating the procurement of web services. In *1st. International Conference on Service Oriented Computing ICSOC'03*, pages 91–103, Trento, Italy, 2003. Springer–Verlag LNCS 2910.
15. Christian Prehofer. Feature-oriented programming: A new way of object composition. *Concurrency and Computation: Practice and Experience*, 13(6):465–501, 2001.
16. D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003.
17. J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), IEEE Computer Society*, pages 45–54, 2001.

ENAGER – A Tool for Requirements Engineering in System Family Context

Rodrigo Cerón[1], Francisco Valsera, Jose L. Arciniegas[1], Jose L. Ruiz[1], Juan C. Dueñas[1]
Department of Engineering of Telematic Systems,
Universidad Politécnica de Madrid,
ETSI Telecomunicación, Ciudad Universitaria, s/n, E-28040 Madrid
{ceron, jlarci, jlruiz, jcduenas}@dit.upm.es
fmvalsera@ya.com

Abstract.

Software industries are pursuing the development of software intensive systems with a higher degree of reuse, reduction of costs, and shorter time to market. One of the successful approaches taken is based on the development of sets of similar systems; in it, development efforts are shared. This approach is known as System Families. A weakness in system families is the lack of existence of tools. Requirements engineering is a mature area today. However, it needs some improvements in the system family tools field. In this position paper, you can find a study for tool support for requirements in system family context. An analysis of actual tools was made. Some requirements for tool support in this field are proposed. In addition, some metrics for the tool are proposed. A tool we called ENAGER was implemented with this into account.

1. Introduction

For many years, software industries have been trying to achieve the development of software intensive systems with a higher degree of reuse, cost reduction, and shorten of time to market. System Families (SF) are considered as one of the most successful approaches to do it. It focuses on the reduction of the time to market and development costs by the provision of a set of elements that are common to a number of systems. Therefore, it is a set of systems or services; also, it shares a significant part of their development effort. Since it shares many elements, it is sensible to think that the concepts behind them are also common. The main goal of this position paper is to present tool support for requirements modelling and management in SF. Some requirements for tool support in this field are proposed. In addition, some metrics for the tool are proposed. A tool we called ENAGER was implemented with this into account.

The basic work has been performed in the CAFÉ project [1]. The CAFÉ reference framework (CRF) gives a guide to classify the activities and models related with SF development. In summary, CRF can be divided in two main parts CAFÉ Process Reference Model (CAFÉ – PRM) and CAFÉ Assets Reference Model (CAFÉ - ARM). The objective of this model is to represent major activities and methods operating on the core assets, to allow the mapping of the contribution/tools against a common reference.

[1] Rodrigo Cerón and Jose L. Arciniegas are visiting professors from Universidad del Cauca, Colombia. Rodrigo Cerón is sponsored by COLCIENCIAS - Colombia. The work performed by José L. Arciniegas has been partially developed in the project TRECOM, granted by Spanish Ministry of Science and Technology under reference TIC2002-04123-C03-01. The work done by Jose L. Ruiz, and Juan C. Dueñas has been partially performed in the project CAFE and FAMILIES (Eureka 2023, ITEA ip00004, ip02009), partially supported by the Spanish company Telvent and by the Spanish Ministry of Science and Technology, under reference TIC2002-10373-E.

2. Related Work

In our work, we analyze and extract some conclusions about actual tools. We cannot analyse all the tools but we saw the documentation for the most popular in the software market and in one open source projects. The tools in deep analysed are those where we can download an evaluation copy and technical documentation (i.e.: RequisitePro, Integral Requisite Analyzer (IRqA), Distributed Requirement Engineering System (DRES), Computer Aided Requirements Engineering (C.A.R.E) and RMTrak). Other tools were taken into account since we can download technical documentation but we cannot have an evaluation copy. For this position paper, we only summarized the tools in deep analysed.

Rational RequisitePro was developed and it is maintained by Rational (now a division of IBM) [2]. They said that systematic capture, control and change communication is a key point in requirement management. For them, another key point is the transformation from the internal repository format to text documents. In this way, requirements can be managed and communicated. Requirements are managed in a project basis. Three types of relationships are utilized to associate requirements. They are called hierarchical, trace and indefinite. As a conclusion, we found that this tool was not designed for SF.

IRqA was developed and it is maintained by “TCP Sistemas e Ingeniería” (A Spanish software organization). They said that domain analysis and requirement management are its key features. It is based in Meta-IRqA as its framework [3]. The project is the basis for the requirements management. Inside the project, the user defines domains. They are useful for divide the complex problems. The modelling elements are services, requirements, concept or entity and implementation class. These modelling elements are related to the others in predefined ways. Another key point is the definition of terms and facets. One term is a specific vocabulary in the problem context. One facet groups sets of terms. As a conclusion, we found that this tool was not designed for SF. However, it gives us improvements in the way of management of requirements.

DRES is an open source tool for requirements management [4]. It is still in development and the latest version is 0.8. Requirements are stored in a traditional way. The basic idea is to store and to retrieve requirements. Its strength is to retrieve requirements from any place using standard browser. Project is its primary unit. One project contains categories. Requirements are inside one category. The tool can generate documentation for its users. This tool was not designed for SF.

C.A.R.E was developed by the German organization SOPHIST GROUP [5]. The main element is a textual document; it is called specification. Other elements are related to this one. Requirements and definitions are the other elements defined by this tool. Requirements are divided into functional and non-functional (user interface, contract, quality of service, etc.). Definitions are employed to represent concepts. These concepts can be into the domain space or in the solution space. Definitions are grouped in glossaries.

RMTrak was developed by RBC Inc [6]. Project is its primary unit. Projects are composed of a series of documents. Documents contain requirements for a project. Requirement classification can be from user, hardware or software. Inside a document, a label identifies a requirement. Requirements can be related by means of a father-son relationship. This tool was not designed for SF.

DOORS was developed by Telelogic AB [7]. It is one of the most widely used tools. One of its key features is its multi-user support environment. The tool eases communication among its users. Easy navigation and understanding of its information for users are key features. They are obtained by means of object browser, graphics, statistics, etc. Collaborative support for its users is another feature. Tools for discussion and change control are inside it. Finally, trace support that eases validation and requirements tracking are included.

RDT and SLATE were analyzed too. For RDT, SLATE and DOORS we cannot obtain evaluation copy and we cannot do some experiments with them.

3. General Issues

From the tool evaluation, we found the following general issues.

- There is no commercial or open source tool with full support for SF. All of them are thought for single project development.
- Domain Engineering is not taken into account in all tools. IRqA defines domain but for a single project. It hinders reuse into several projects.
- System engineering support is partially covered.
- Internal trace mechanisms for requirements are defined in the tools studied. No external trace mechanisms can be found to other development phases.
- We need to define a system of classification for requirements in SF. They can be classified as functional and non-functional but a further division is necessary.
- We need to define elements that ease requirements management in SF. They must take into account development and evolution of SF. Conflicts and decisions must be elements to take into account.
- Evolution of system gives us another issue known as version control. For SF, this issue is very big. Some of their problems are multiple systems, assets and requirements.
- We need aids that help us in documentation generation.
- Users need ways to assess requirements stored.

In the design of ENAGER (our tool), we try to cope with these general issues found.

4. Requirements for ENAGER

[8], [9], [10], [11], [12], [13] and [14] gives us general ideas for tool requirements in SF context. The following requirements are from this point of view.

- Organizations need to manage more than one system family. In general, large organizations need support for several projects they work in. If they can share most of its products requirements between several projects, it can shorten its product development cycles.
- Stakeholders are in charge of providing definition of domain and application terms. They must be stored in glossaries.
- Glossaries need to be shared across all organizational units. If they can be share across all development units then a common understanding is foster across the organization.
- Variability management for requirements need to be solved by means of conflicts and decisions. Requirements must have been related to others by means of several situations. A conflict can be used to express variability. A decision can be used to solve variability.
- Functional requirements need to be traced to use cases. It closes the gap between requirements and design.
- Conflict impact need to be traced to use cases. The problems detected among requirements can influence following development phases. If developers have aids that help them in tracking those problems to the following phases, they improve their performance in development phases.
- Decisions must be stored. It is necessary to know all the decisions taken into account to derive one product. It helps in change control and evolution.
- Requirements need to be classified. A hierarchical taxonomy must be used. All tools proposed a model of hierarchies for its requirement.
- Functional requirements can be classified as goal, feature and atomic. With three levels and relation among them all, the possibilities can be covered (exclusion, coexistence, mandatory, alternative).
- A software quality model for non-functional requirement is recommended for the tool (i.e.: ISO 9126).
- Each requirement must have properties.
- We need easy navigation in order to keep requirement management under control.

These general requirements were taken into account for the design of ENAGER.

5. Metrics for ENAGER

In order to asses different SF requirements ENAGER defines some metrics. The first group is called Framework metrics. They are intended to asses the quality of the overall specification stored. The second group is called SF metrics. They are intended to asses the quality of one of the SF stored.
Framework metrics are subdivided in two groups called: complexity and consistency. The first one gives us an idea of how big is the information stored. The second one gives us an idea of how coherent is the information stored.
SF metrics are subdivided in five groups called: complexity, consistency, cost, maturity and common/variable. The first two are similar to those ones in framework but for only one SF. The third one gives an idea of the overall cost of SF. The fourth one gives an idea on how mature is the SF under analysis. The fifth one gives an idea of the common requirements compared to the system specific requirements.

6. Conclusions and Future Work

We have performed this work trying to cope with the industrial requirements for tools in SF requirements management. The position paper gives a partial answer for tool support in requirements modelling context. Further research must be done to solve it completely. In any case, the tool presented here covers several of the specific needs of the SFE as regards requirements management and the traceability to other systems-models.
Back to the specific issues in requirements management for SF, the tool allows:

- The full navigation and traceability matrixes generation for internal traces, despite the complexity of requirements in the SF. The usage of a web navigation schema helps in keeping the complexity under control.
- The basic relations between requirements are covered (exclusion, coexistence, mandatory, alternative), as well as relations during evolution (revision); hierarchies (generalisation, specialization) of requirements are also supported.
- The concept of system in the system family appears in this tool as a set of requirements. Being these requirements related, the tool allows for the calculation of “coherent” systems, containing sets of non-conflicting requirements. As well, the progress in development of a certain system can be represented by the growing set of requirements covered.
- Decisions are represented explicitly. It is possible to navigate from each requirement to the decisions related, and the decisions related to each system. Conflicts are also expressed, so its reasons and stakeholders related can be identified and solved.
- The differentiation between common and specific requirements is not statically wired; instead, as there is navigational capability from-to systems and each requirement, it is possible to know in a certain point in time how many of the system in the SF are affected for one requirement.

References

1. van der Linden, F.: Software Product Families in Europe: The Esaps & Café Projects. IEEE Software, Vol. 10 No. 4. IEEE Computer Society, Los Alamitos, CA (2002) 41-49
2. IBM – Rational.: Rational RequisitePro. http://www-306.ibm.com/software/awdtools/reqpro/, (2004)
3. TCP Sistemas e Ingeniería: IRqA integral requisite analyzer. http://www.irqaonline.com/index_irqa.htm, (2004)
4. Kowalczykiewicz, K.: DRES Distributed Requirements Engineering System, http://ophelia.cs.put.poznan.pl/xdre/, (2003)
5. SOPHIST GROUP: C.A.R.E Computer Aided Requirements Engineering, http://www.sophist.de/sopgroupeng.nsf/(ynDK_framesets)/Main, (2003)
6. RBC Product Development: RMTrak, http://www.rmtrak.com/, (2003)
7. Telelogic AB: Telelogic DOORS, http://www.telelogic.com/, (2003)

8. Kuusela, J., Savolainen J.: Requirements Engineering for Product Lines. In: Proceeding of the 2000 International Conference on Software Engineering, ACM-IEEE, New York, NY (2000) 60-68
9. Luttikhuizen P. (ed.): Requirements Modelling and Traceability. ESAPS, WP3, Derivation of Products and Evolution of System Families WP3-0106-01 Technical Report, (2001)
10. IEEE Standards Board: IEEE Recommended Practice for Software Requirements Specifications. IEEE std 830, 1993. Institute of Electrical and Electronics Engineers, New York, NY (1993)
11. Alonso, A., León, G., Dueñas, J.C., de la Puente, J.A.: Framework for Documenting Design Decisions in Product Families Development. In: Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems. IEEE Computer Society, Los Alamitos, CA (1997) 206-211
12. Thayer, R., Dorfman, M.: Software Requirements Engineering. 2nd edn. IEEE Computer Society Press, Washington (1997)
13. IEEE-SA Standards Board: IEEE Guide for Developing System Requirements Specifications. IEEE std 1233, 1998. Institute of Electrical and Electronics Engineers, New York, NY (1998)
14. ISO/IEC: Software engineering -- Product quality -- Part 1: Quality model. ISO/IEC 9126-1:2001. International Organization for Standardization, Geneve, (2001)

Experience-based Approach to Requirements Reuse in Product Families with DOORS®

Antonio Monzón, Juan Carlos Dueñas[1]

Departamento de Ingeniería de Sistemas Telemáticos,

Universidad Politécnica de Madrid

Requirements reuse has revealed as one of the most important issues in the Requirements Engineering (RE) field over the last years, as this discipline has a huge impact on systems production and maintenance costs. For this reason it is required to provide specific solutions to be directly applied to real industrial environments with the help of tools. The objective of this paper is to show a practical approach to requirements reuse in product families supported by a well-known commercial tool (Telelogic DOORS®). For illustration purposes the paper focuses on the particular domain of commercial aircraft product families.

1 Introduction

The proper application of requirements engineering principles is a must for the successful development of software intensive systems; as the size and complexity of these systems grow, it becomes evident that tool support is a key. The development of product families –instead of individual products- is a strategic improvement followed by many companies, pursuing wide reuse of any asset of the product family, including requirements. Many works in the literature have already been produced about requirements reuse but few of them present practical solutions to the problem with the help of commercial tools. This is the aim of this paper.

It is important to remark that this is a practical experience paper with no or very few direct theoretical contributions. The concept of Requirement handled here is the expression of a need represented as a classical natural language description, with some additional attributes for management purposes. Neither formal approach nor domain modeling techniques are proposed.

In order to provide a better understanding of the concepts involved in this paper, several UML diagrams will be used in order to represent the information metamodels behind the problem to be solved. On one hand, the proper supporting tool basic metamodel will be represented. On the other hand, the concrete requirements metamodel used in a particular program (product family) will be represented. And finally,

[1] The work done by Juan C. Dueñas has been partially performed in the projects CAFE and FAMILIES (Eureka 2023, ITEA ip00004, ip02009), partially supported by the Spanish company Telvent and by the Spanish Ministry of Science and Technology, under reference TIC2002-10373-E.

the particular approach to reuse techniques applied will also be represented as an UML diagram.

The reasons for choosing the artifacts and the particular information structures have been mainly based on the experience of requirements engineering teams in the field. Data structure has been proposed and refined over a couple of decades and it has been agreed by several parties. Therefore, it could be said that the models have been validated by their industrial application during years.

The main information sources for requirements are the Functional Requirements Documents (FRD's). These documents contain both textual descriptions of the problem domain (graphical and tabular information included) and also proper requirements. Requirements are tagged through unique identifiers and clearly delimited by short textual descriptions.

2 DOORS® Metamodel

DOORS® is the most extended RE tool in aerospace arena. Although it is not final-user oriented and it is difficult to use the tool without specific training, it shows an extremely adaptation capability, very useful in technology-driven environments. The main concepts involved in this tool are represented in Figure 1.

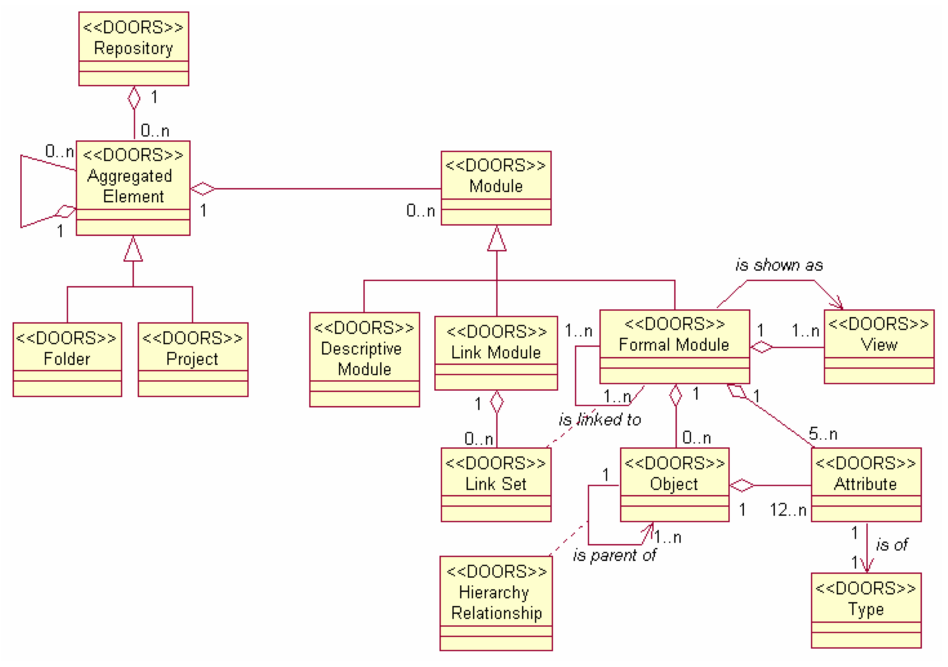

Fig. 1. DOORS® Basic Metamodel

It is just worth to mention that DOORS® Object meta-concept is the information element suitable to represent requirements. But this is so general that can be also used to represent whatever other information artifact conceivable (document paragraphs, diagrams, tables, etc.) The main idea is that they can be extended with DOORS® Attributes and that DOORS® information model allows to organize program documentation into folders and to establish traceability relationships between elements contained in DOORS® Modules (document meta-concept in DOORS®). Although it is an incomplete model, that presented in Figure 1 shows parts of the basic support that the tool offers to some well-know RE activities; management elements such as Folders or Projects, requirements typing with the support given by Attribute and Type, and relations and traceability support with Links and Relationships.

3 The Aircraft Program RE Metamodel

The Requirements Engineering metamodel shown in Figure 2 has been widely used in Airbus as the RE standard reference for different Aircraft Development Programs (lately A380 and A400M Programs).

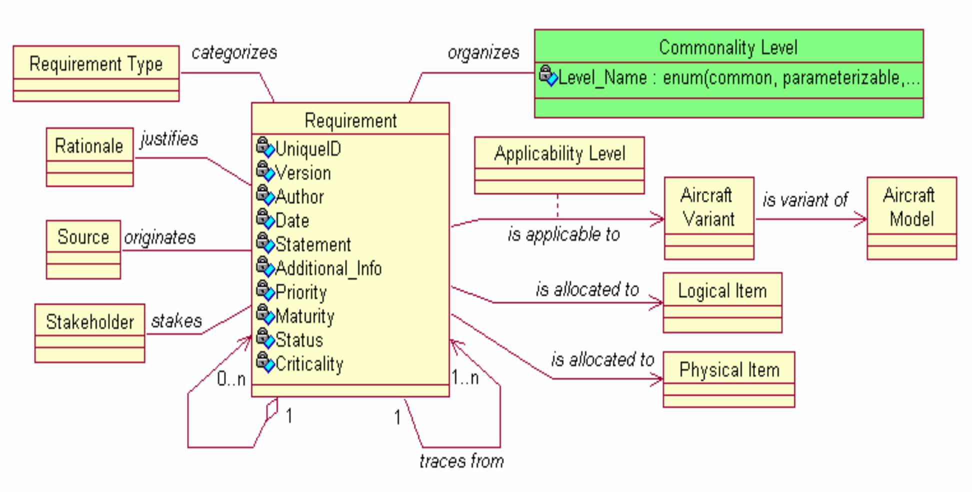

Fig. 2. Aircraft Program RE Metamodel

This model already contains a basic configuration control mechanism to manage system variants, as requirements are applicable to certain Aircraft Variants (domain variability). But this is not enough to effectively reuse parts of the requirements specification.

4 Requirements Reuse Metamodel

In order to establish an effective reuse organization, it is first required to analyze the commonalities and variabilities of the information containers. As previously mentioned, the basic information containers in this field are FRDs. Several FRDs have been studied in order to find the commonality taxonomy more suitable to this problem domain.

After this analysis, it has been concluded that three different kinds of requirements are contained within a typical FRD:

- Common Requirement: This is a requirement literally applicable to different products (i.e. subsystems to be embedded in an aircraft), with no modification.
- Parameterizable Requirement: This is a requirement with a common description, but containing different parameter values depending on the product it affects.
- Specific Requirement: This is a requirement particular to a product that can not be shared.

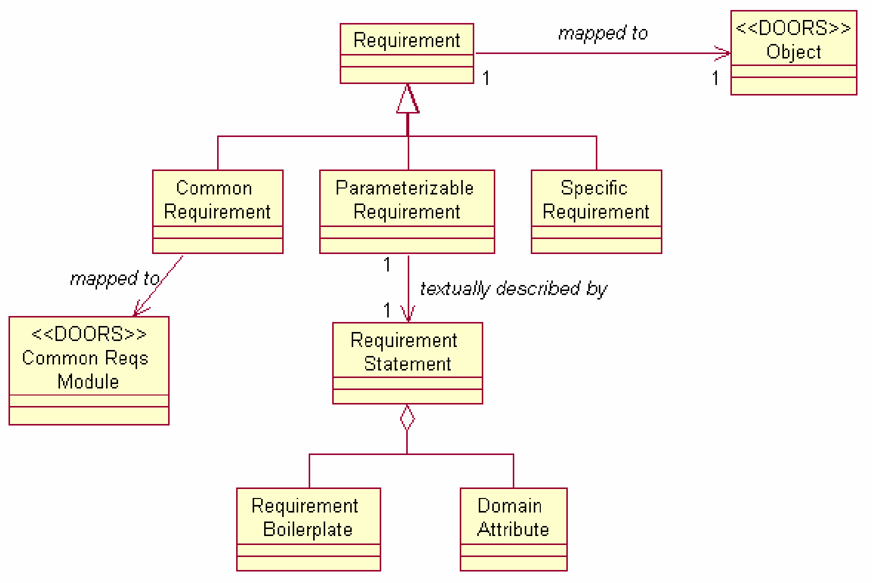

Fig. 3. Requirements Reuse Metamodel

According to this taxonomy, the information organization for the program (Product Family) proposed is as follows.

All the common requirements are contained in just one DOORS® module. When it is decided that a common requirement is applicable to an organizational element (an aircraft variant, a system, a subsystem or equipment), it is copied to the particular FRD module and automatically linked to the original requirements. Common requirements are copied as read-only in order to avoid modifications in the particular module. Modifications are only allowed at Common Requirements module level, thus ensuring they are kept consistent.

The parameterizable requirements are treated as Boilerplates [Dick 2000] with domain attributes. An especial adaptation of the tool has been used to support this boilerplate approach. The idea behind is simple: requirements share a common textual description and only differ in the values of certain parameters. Let us remark that the approach is similar to the definition of usage of frames [Basset 2002]. Boilerplates applicable are also contained in just one module. When a requirement is identified as one of this category, it is copied from the boilerplate module and the values are established as required. Once again the adapted requirements are linked from the original ones in order to identify them as boilerplates.

Specific requirements are handled as usual in this environment. Obviously, a specific requirement may be raised to one of the two previous categories at any time, as the requirements and the systems evolve.

It has not been explicitly mentioned but it is assumed that requirements are under Configuration Control (a default feature of this tool). This means that all categories of requirements may have different versions over the time, and that only certain versions of them conform a particular version of the overall product (version or variant).

5 Conclusions and Future Work

DOORS® provides a reasonable infrastructure to support requirements engineering activities, but it is always required to adapt it in depth to assure a proper support. This is even clearer in the case of adapting the tool for reuse purposes.

An adaptation of existing methodologies has been performed to support reuse in product families applicable to the particular domain of commercial aircrafts.

A combination of common and parameterizable requirements (based on boilerplates approach) has been selected as the basic mechanism to reuse requirements. This approach has demonstrated to be operative enough as it is founded on very intuitive and simple concepts.

Nevertheless, no measurements have been recorded in order to assess the benefits obtained in terms of effort savings. Furthermore the proposed approach has been conceived to be applicable to a particular domain. For this reason no general conclusions may be extracted without a further research. In any case this approach has to be enhanced and matured through its application in ongoing programs. Additionally specific scripts (DOORS® extension mechanism) should be required to properly implement the reuse metamodel.

References

1. Dick, J., "Boilerplates", http://www.requirementsengineering.info/boilerplates, 2002
2. Dorfman, M. "Requirements Engineering", R.H. Thayer and M. Dorfman, Eds. Software Requirements Engineering, IEEE Computer Society, Los Alamitos California, 1997, 7-22

3. Hull, E., Jackson, K., Dick, J., "Requirements Engineering", Springer Verlag, May 2002
4. Mannion, M., Keepence, B., Kaindl, H. (Siemens), Wheadon, J., "Reusing Single Requirements From Application Family Requirements", 21st IEEE International Conference on Software Engineering (ICSE'99), May 1999
5. Object Management Group, Inc., "Unified Modeling Language Specification (UML) – version 1.5", March 2001
6. Telelogic DOORS®/ERS, "Using DOORS - DOORS 6.0", June 2002

Requirements Re-Use in the IT Business Process WEB Implementation Product Family

Dr. Lawrence E. Day, CSQA, PMP
The Boeing Company
Seattle, Washington, USA

www.boeing.com

Tel: +1 425 957 5039
Fax: +1 425 865 3498
lawrence.e.day@boeing.com

ABSTRACT

This paper outlines and discusses the need and advantage of software requirements reuse for implementing business processes on the web. Given the understanding that process is an inherently difficult discipline this paper examines ways of capturing and reusing process implementation requirements so that the implementation project and the customer can leverage the retained knowledge inherent in the reused requirements. The paper goes on to explore the advantages of requirements reuse in the various quadrants of a process implementation business model. The assertions and lessons are empirically derived from a database of project experience.

Keywords: Business Model, Cost, Performance, Process, Quality, Requirements, Reuse, Time-to-Market,

Introduction

Process in inherently difficult. It is so difficult that only 3% of the world's population has the intellectual capacity to understand process". (Conway) Most of the approaches to implementing process and quality in major organizations have focused on getting the employees to become experts as process analysts and quality specialists. Given the 3% capability figure, it is and will continue to be difficult to get an entire organization to effectively embrace and successfully implement real process driven activities. However, for most of us there is hope. While the workers may not be able to fully comprehend the process discipline, they are extremely capable of successfully working within processes if they feel ownership and are given the opportunity to improve the process. The specific family of products that will be examined in this position paper are web based business processes.

Common IT Business Process Requirements

It has been the experience of the author that when asked to document a process, most technically oriented people end up listing as set of activities that they intend to perform. When I ask them what the "object" of the process is, they often appear very confused. They know what they think they are supposed to do, but often aren't aware of what states the "object" of the process is going through. An example of activities would be "review", "approve", "analyze", "purchase", "contact", etc… The object might be a purchase request for a new laptop, a change request, an operating system migration (the object could either be the server or a share on the server), and so on.

Objects in IT business processes have quite a few common attributes and include some of the following:

- Step/State processes
- Reporting
- Search Engines / Results
- CI Listings
- CI Input / Update
- On-Line Documentation
- Site and Data Administration
- User Access

- Help - FAQ

The attributes of the objects are requirements for them. It turns out that quite often, even though objects themselves may be quite different, the requirement for certain attributes is very common.

Benefits of Requirements Reuse

Beneficial results of our requirements reuse include the following:

- Requirements Reuse cut new application development by approximately 80%.
- Code from Common Requirements improvements quickly migrated across sites.
- Requirements errors more readily detected by increased use across multiple sites.
- Requirements errors are fixed prior to discovery by users on other sites.
- Similar requirements at existing sites used to demonstrate proposed functionality to new customers.

Lessons Learned

After developing a certain level of capability in process implementation requirements reuse, the process implementation Subject Matter Experts (SMEs) often end up knowing more of the customers requirements than the customer does. This shows up in requirements every where. The following are two examples that occurred:

1. **Delete a Request capability:** the customer wanted the capability to delete a request. I acknowledged the requirement and indicated that I would implement the request by identifying a common requirement to tag the request a "Deleted" but actually leave it in the database. This design requirement was a well established best practice requirement. The customer insisted on the request being actually deleted. I

explained that if it were really deleted, it would be irrecoverable. They still insisted and so I changed the requirement. Later, when the process was in production, management complained when requests that they had been tracking suddenly disappeared.

2. **"Name" is a Required Attribute**: a customer for a Requirements Tracking process did not identify a Requirement Name as one of the attributes of the requirement. I pointed out that such a field was needed for display purposes (part of the accumulated best practice reuse requirements). They insisted that it was not needed. After going into production, a display showed only the first few characters of the data in the field (say 55). Thus they quickly realized that they needed a "summary" statement to search on and sort by, so we added a title.

The following are some of the Lessons learned in dealing with customers in implementing their business processes:

- The Customer is **NOT** Always Right:
  - Deleting data
  - CI #
  - CI Description
- The "Real" Process is not the one originally described.
- The Data Will Set You Free
- Have Something To Talk About to demonstrate requirements commonality
- Keep Records of Requirements, but be ready for change (constantly!)

Acknowledgments

This paper and accompanying presentation has been supported financially by The Boeing Company of Chicago, Illinois, USA.

Reference

Conway, William E. Chairman & CEO, Conway Management Co. http://www.conwaymgmt.com/

Technical Report, Universidad Politécnica de Madrid, Spain, 2004.

ISBN: 84-688-7314-4.