# REFSQ'03

# PRE-PROCEEDINGS

## 9th International Workshop on REQUIREMENTS ENGINEERING - FOUNDATION FOR SOFTWARE QUALITY

## In conjunction with CAiSE'03
## 16 -17 June 2003, Klagenfurt/Velden, Austria

*Program Co-Chairs:*
Camille Salinesi - Ben Achour,
Björn Regnell,
Erik Kamsties

**www.refsq.org**

REFSQ

# Table of Contents

# Elicitation of Requirements from User Documentation

Isabel John, Jörg Dörr

*Fraunhofer Institute for Experimental Software Engineering (IESE) – Kaiserslautern - Germany*
*{john; doerrj}@iese.fraunhofer.de*

**Abstract**

*This paper describes an approach for elicitation of requirements based on existing user documentation. The approach we describe in this paper supports capturing of the information found in user documentation of legacy systems, e.g., user manuals, and the specification of this information in requirements specifications, using, e.g., Use Cases. We propose a conceptual model describing the transition from user documentation to requirements artifacts describing common and variable elements of a product line model or requirements specification. We present heuristics that allow an easy identification of text elements in user documents that are then used to create a significant part of the requirements specification and product line model, respectively.*

## 1. Introduction

The development of industrial software systems may often benefit from the adoption of a development cycle based on the so-called *system-families* or *product lines* approach [20] [6]. This approach aims at lowering production costs by sharing an overall reference architecture and concepts of the products, but allows them to differ with respect to particular product characteristics in order to e.g. serve different markets. The production process in product lines is therefore organized with the purpose of maximizing the commonalities of the product family and minimizing the cost of variations [14].

In the first stage of a software project, usually called *requirements elicitation* [12], the information and knowledge of the system under construction is acquired. Especially when developing more than one product, requirements elicitation is a complex task, in depth knowledge of the problem domain often is a prerequisite for a successful product family. Normally, domain experts with knowledge in the problem or application domain, have to elicit and model the requirements in an highly interactive and time consuming process. But when a company wants to build a new product, or decides to start a product line, often systems already exist that can be used as a knowledge base for the new product line [16]. The information from legacy systems is a valuable source for building the reusable assets. This information from existing systems can be found in the code, in architecture descriptions and in requirements specifications [16].

If user documentation is present, it is the first choice to start the elicitation process for the information needed in product line modeling as well as in single system development. User documentation that is useful as input for product line modeling can be found in the cases of project-integrating (existing systems under development will be integrated into the product line), reengineering-driven (legacy systems have to be reengineered into the product line) and leveraged product line engineering (the company sets up a product line based on a product line that is already in place) [26]. Furthermore, also in case of creating the requirements specification for a new single system in the product family, user documentation of recent and current products can be available.

Product Line Engineering includes the construction of a reusable set of assets. Constructing such a reusable asset base for specific products in a domain is a more sophisticated task than the development of assets for a single system because several products with their commonalities and variabilities have to be considered. This implies the planning, elicitation, analysis, modeling and realization of the commonalities and variabilities between the planned products. As a result, creating the requirements for a product line puts high load on the domain experts creating them.

In this paper we propose an elicitation approach that is based on a conceptual model. With the elicitation approach common and variable features [19], Use Case elements [7], tasks describing user activities in an interactive system [24] and textual requirements can be elicited from user documentation (e.g., manuals). As existing systems are the basis for this approach, it can be seen as a reengineering method for documents transferring user documentation into basic elements of information for requirements specifications. By reusing the information from the user documentation of the recent and existing systems, one can produce a traceable requirements specification that is more consistent and complete. This kind of approach ensures a systematic connection between the requirements specification and

the recent and current systems. Furthermore, the domain experts have less workload, as basic elements of information are already provided either by non-domain experts or automatically by a tool.

The paper is structured as follows: in Section 2 we describe product line modeling and the benefits of using user documentation for elicitation and specification of requirements. In Section 3, we present the conceptual elicitation model, i.e., models describing the syntactic and semantic types of information found in user documents and requirements specifications. They are the foundation for the elicitation approach we describe in section 4. As part of this approach, we present heuristics that are used to map textual elements in the user documentation to requirements artifacts that are used to build up a significant part of the requirements specification and product line model, respectively. Finally, we conclude the paper in Section 6.

## 2. Motivation

Using legacy system description as input for the requirements engineering phase is on the one hand motivated by product line engineering and on the other hand by reuse principles. In this section we will describe general product line modeling concepts and the influence of legacy documentation on modeling requirements for single systems and product lines.

### 2.1. Product Line Modeling

Product line engineering [6] can be described as a technology providing methods to plan, control, and improve a reuse infrastructure for developing a family of similar products instead of developing single products separately. This reuse infrastructure manages commonality and controls the variability of the different products. Examples for product line approaches are PuLSE [3], FAST [31] and the SEI Product Line Practice Initiative [6].

The goal of product line engineering is to achieve planned domain-specific reuse by building a family of applications. Distinct from single system software development there are two life cycles, domain engineering and application engineering. In domain engineering the reusable asset base is built and in application engineering this asset base is used to build up the planned products. The requirements engineering phase of product line engineering is generally called domain analysis or product line modeling. Domain analysis methods provide processes for eliciting and structuring the requirements of a domain, or product line. The results are captured in a domain model. A domain model must capture both, the common characteristics of the products and their variations. The domain model is the basis for creating

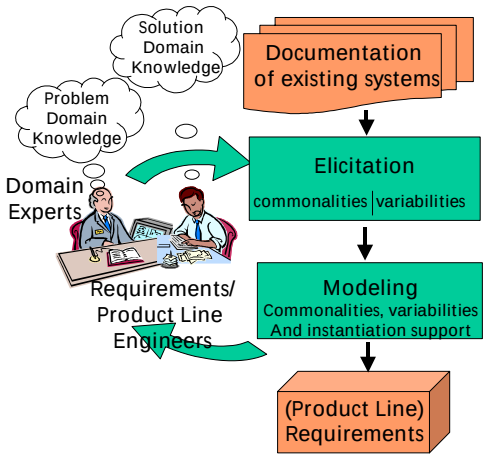



**Figure 1 A requirements elicitation process**

other reusable assets like a domain specific language or a component-based architecture. For a domain analysis method to be applicable it must be appropriate to the specific context of the organisation and the application domain and it must provide enough guidance so that it can be carried out. As in other areas of software development, the context for each domain analysis application varies, and methods that are appropriate in one context will not be in others. This fact is especially important for domain analysis because of the compound effects of inappropriate models over multiple products and over the whole lifecycle. Therefore, a generally applicable domain analysis method should be customisable to the context of the application.

Product line modeling extends requirements engineering for product lines.

Apart from general requirements engineering principles, product line modeling methods have to emphasise further principles:

- Commonality and Variability
  When doing domain analysis the properties of several products have to be modelled at once. As the planned products that are analysed during domain analysis differ in their features and in their requirements, the commonalities and variabilities between those products have to be captured and adequately modelled.
- Traceability
  Providing traceability from the requirements to the product and from the requirements to architecture, implementation and tests is very important in product line engineering. As a product line spans over several products and several releases of the products it has to be ensured that those two dimensions of traceability (traceability through products and through lifecycles) is provided.

Furthermore, decision modeling (building a that model collects and abstracts the information on which requirement is instantiated in which product), and evolution over lifetime of thee product line also have to be supported.

There are several approaches for domain analysis or product line modeling. In most product line modeling approaches, the integration of legacy systems into the domain analysis phase is not described in depth. An overview on domain analysis methods like FODA [19], ODM [28] or Commonality Analysis within FAST [30] can be found in several surveys like [8] or [2]. An approach that is often used is feature modeling [19], where features are seen as common and variable characteristics of a system that have some value to the user. Our elicitation approach supports feature modeling as with the approach, features can be identified in user documentation.

The PuLSE-CaVE (Commonality and Variability Elicitation) approach for elicitation that we describe here is integrated into the PuLSE-CDA [4] approach that builds the domain analysis component of the PuLSE™ (Product Line Software Engineering) [1] framework [3]. With the help of the approach described here, information on legacy systems can be systematically integrated into a product line model developed with CDA or any other approach. As variable and common elements can be elicited with the approach we describe in section 4, this approach supports requirements elicitation for product lines.

## 2.2. Reusing Documentation in Requirements Engineering

The information needed to build a requirements specification for a single system or a product line model is normally elicited interactively with high expert involvement (c.f. Figure 1). As domain experts have a high workload and are often unavailable, high expert involvement is a risk for the successful introduction of requirements engineering processes and methods like a product line engineering approach in an organization. Systematically using existing documentation of former or current products like user manuals to support the elicitation process reduces the expert load and makes the requirements more trustable. So, systematically integrating legacy documentation into the requirements phase has many benefits:

- Benefit 1 – Integration and reuse of textual information:
  This is achieved by integrating existing systems textual information (e.g., user manuals) into product line and requirements specifications. By integrating textual information, not only code can be reused but all assets built during the previous lifecycles.
- Benefit 2 – Feasibility of requirements engineering:
  The feasibility of requirements engineering approaches and of product line modeling will be supported through these document-based techniques (e.g. by finding reasons for missing requirements [12]). A document-based technique can decrease the effort the domain experts have to spend with interviews and meetings and leads to a significant reduction of the expert load. The basic information can be elicited from documents and the experts can concentrate on planned innovative functionality.
- Benefit 3 – Increased acceptance of the product line in the development organization:
  The acceptance of the product line within the organization can be increased by reusing the legacy information, which was produced within the organization. There are two reasons for this. First, the acceptance of the product line is increased because there is confidence in the quality of the legacy products. Second, reusing the legacy information instead of developing everything from scratch reduces the effort to built the product line.
- Benefit 4 – Better traceability from the product line to the existing systems:
  Traceability to the existing system can be established only with a systematic approach which supports linking of legacy assets to the product line model built during domain analysis. Therefore, it is important to document the traces from the legacy documents to the new documents during elicitation and modeling.

There are some methods from single system requirements elicitation that describe how to elicit information from existing documents. Alexander and Kiedaisch [1], Biddle [5] , von Knethen [30] and the REVERE Project [25] focus on reusing natural language requirements in different forms. The QuARS approach [9], the KARAT approach [29] and Maarek [21] apply natural language processing or information retrieval techniques to requirements specifications in order to improve their quality. The approach that we describe here overcomes the shortcomings of other approaches by explicitly considering variability and integrating user documentation into product line modeling and modeling of Use Cases.

For product line modeling, single system elicitation methods cannot be taken as they are, because multiple documentations have to be compared, commonalities and variabilities have to be elicited and additional concepts (e.g. abstractions, decisions) are needed. MRAM [22] is a method that describes how to analyze and select appropriate textual requirements for a product line but

[1] PuLSE is a registered trademark of Fraunhofer IESE

their focus is on the transition from domain engineering rather than on the transition between existing systems and domain engineering. In ODM [28], the primary goal is the systematic transformation of artifacts (e.g., requirements, design, code, tests, and processes) from multiple existing systems into assets that can be used in multiple systems. ODM stresses the use of legacy artifacts and knowledge as a source of domain knowledge and potential resources for reengineering/reuse but doesn't clearly state how to elicit requirements from documents.

With the approach that we present here we overcome the shortcomings of the existing approaches for product line modeling (no explicit elicitation, no systematic integration of existing documents) and for reusing requirements from single systems engineering (no consideration of variability, no use of user documentation).

## 3. Conceptual Elicitation Model

In this section we describe a conceptual elicitation model that is the basis for our elicitation approach described in section 4. The elicitation model consists of four parts (see Figure 2):

- A user documentation model describing the elements that are typically found in user documentations, manuals and technical specifications (e.g., sections, glossaries, and lists).
- A requirements concept model describing concepts that are typically used in requirements specifications (e.g., roles, activities, functions ) independent of the notation used.
- A variability concept model describing the principle commonality and variability concepts that can be found by comparing different documents and that are used for modeling.
- A requirements artifact model describing elements of typical single system requirements specifications and product line models. These elements form a notation that is used to capture requirements (like Use Case elements, features or textual requirements). Those requirements can have, but do not have to have an explicit representation of variability.

The transition from one stage of the model to another stage is described by heuristics (specific rules-of-thumb or arguments derived from experience). These heuristics describe, e.g., which element of user documentation can be typically transformed into which requirements concept. It is also possible to directly transform requirements concepts into requirements artefacts without searching for variabilities (see arrow "single system elicitation" in Figure 2).

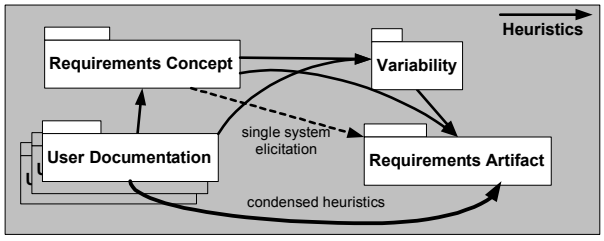



**Figure 2 Overview of the model**

### 3.1. User Documentation Model

Our user documentation model (see Figure 3) describes the principal constituents of user documents. The document types that we analyze are user documentations or user manuals that describe the functions and usage of a system and product descriptions that describe the features and technical details of a product. A document normally has a title, it often has a table of contents and a glossary and it consists of several sections. A TOC entry normally corresponds to a heading in a section. A glossary consists of a list of terms that are described in paragraphs. A paragraph consists of sentences; it can also contain figures, tables and formulas. A sentence is composed of phrases (language constructs consisting of a few words) and/or words. A phrase can also be a link (describing a reference to something inside or outside the document). Most elements of the user documentation model have attributes describing characteristics of this element (like highlighted for paragraphs and words, or numbered for lists), the

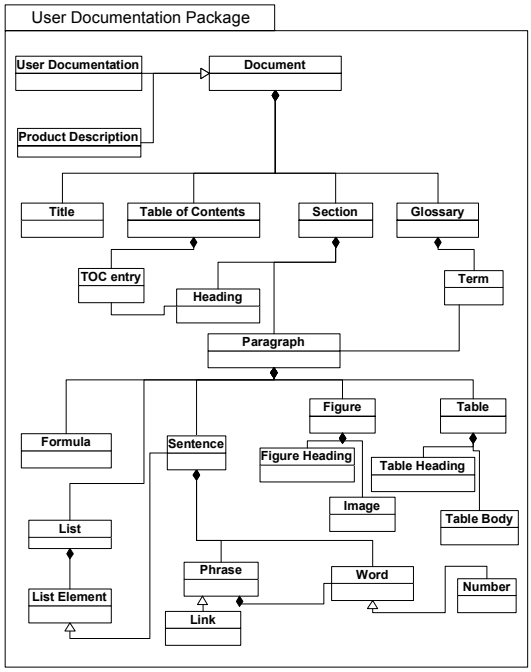



**Figure 3 Model of User Documentation**

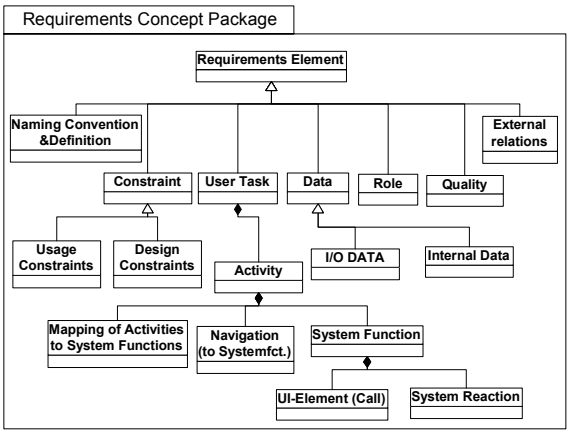



**Figure 4 Requirements Concept Model**

attributes are not shown in the figure. This model describes the elements of a document on an adequate level for eliciting requirements concepts.

## 3.2. Requirements Concept Model

The requirements concept model (see Figure 4) describes concepts that can be elicited from user documentation and that are normally realized or described by requirements artifacts in requirements specifications. The model describes the elements independent of a specific notation (like textual or Use Case representation). The most general requirements concept is a requirements element. A requirements element can be everything that is of value for a requirements specification. A requirements element can be a user task, a role, data, a naming convention, a constraint or a relation to something in the environment of the system to be described. Data can either be I/O data or internal data, constraints can either be usage or design constraints. A user task, that describes the high level task the user wants to perform with the help of the system can be decomposed into activities, activities consist of navigation elements, system functions and a mapping of the activities to functions.

Based on this requirements concept model and the model of user documentation described in section 3.1 we can define heuristics for the transition of elements from one model to another. Example heuristics for transitioning from a user documentation element to a requirements element are: "A heading that contains a verb often is an activity" or "a highlighted sentence containing the phrase "normally" or "with the exception" can describe constraints".

## 3.3. Variability Model

In the variability model, the variation aspects are described. In order to find different variability elements,

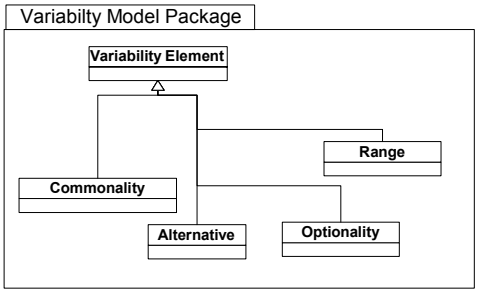



**Figure 5 Variability Model**

the requirements elements (from the requirements concept model) found in different user documentations are compared. We decided to support the following variability elements and kinds of variation:

- **Commonality**
  No variation exists in the requirements element, the same requirements element can be found in all documentations.
- **Optionality**
  A requirements element exists in some of the products, but does not exist in some others.
- **Alternative**
  The requirements element exists in two or more different characteristics in the existing products (e.g. one product supports one database one product supports a different one).
- **Range**
  There is a range of values that is supported by the different products (e.g. the memory size can vary from 10 to 128 MB).

Based on those variability elements, heuristics can be defined that identify different variable requirements concepts by comparing the user documentations of several legacy products. These heuristics are depicted by the two arrows in
Figure 2 from user documentation and from requirements concept to variability.

Examples of such heuristics are "numbers in the document that were identified as data and belong to the same function and that have a different value can be a range variability element" or " "navigation elements that occur only in one documentation can be a hint for an optionality (an optional user interface element)".

## 3.4. Requirements Artifact Model

The fourth package of our conceptual elicitation model is the requirements artifact model. In this model different elements of requirements specifications that can be used for single system modeling and for product line modeling are described. Different from the requirements concept model, that describes the elements on a conceptual or semantic level, the requirements artifact

model describes requirements elements on a syntactic or notational level. In different kinds of requirements specifications, the same conceptual elements can be described with different notational elements, e.g. a role from the requirements concept model can be an actor in a Use Case description or a stakeholder description in a textual requirements specification.

As we also describe the application of our approach for product line modeling, we have an integrated model of variability here. The variability model we use here is the model described in the PhD thesis of Muthig [23]. In product line engineering, variability has to be made explicit in the requirements artifacts. Different extension (e.g. to UML-Use Case diagrams [17][14], or to textual Use Cases [17]) exist that make the variability explicit and give support for instantiation of requirements for application engineering. Some of these extensions use stereotypes or tags to describe variability, some extensions use extra elements to make variability explicit.

As variability is encapsulated outside the requirements artifact model in the product line artifact and the product line artifact element (see Figure 6), the model can also be used for specifying single systems requirements. At the moment we have specified two different kinds of requirements notations: Use Cases and textual requirements specifications. We have also specified artifacts that are more specific to product line modeling like feature models [19] but we will not describe them in this paper. Further requirements artefacts will be integrated into the requirements artifact model. We added different representations here, as our general approach to product line modeling [4] is customisable and highly depends on the requirements elements found in the organization that wants to do product line engineering. For doing product line engineering, we put variability elements on top of the existing notation and so can keep the notation similar to the one used in the organization before [27].

Concerning the elements in Figure 6, a Use Case diagram consists of Use Cases, actors and different relationships between the Use Cases and the actors. A textual Use Case (according to Cockburn [7]) consists of different elements like Use Case goal, precondition post condition, Use Case exceptions and the actual description of the Use Case consisting of steps. The form of requirements specification we describe here follows the IEEE Standard 830 [15]. A requirements specification is a textual document consisting of functional, non-functional and data requirements including project issues and rationales for the different requirements.

We have defined heuristics for transitioning requirements concepts into requirements artifacts (e.g., "a role is described as actor in a Use Case diagram" ) and heuristics that additionally include variability (c.f. Figure 2). An example of such a heuristic also considering variability is "an optional activity can be represented as an optional Use Case in a use diagram".
For the transition between elements of these packages we have found different heuristics. For users of the approach and the conceptual model those heuristics can be integrated to condensed heuristics describing the transition from user documentation directly to requirements artifacts (c.f. arrow "condensed heuristics" from user documentation to requirements artifact in Figure 2). The elicitation approach we describe now uses the heuristics in this direct form to make elicitation easier when applying the approach.

## 4. An elicitation approach using user documentation

In this section, we propose an approach for controlled elicitation, which guides product line engineers and domain stakeholders in how to elicit knowledge from existing documents and how to transform documentation into product line models. This approach, PuLSE-CaVE (Commonality and Variability Elicitation) is an approach for structured and controlled integration of user documentation of existing systems into the product line [18]. The approach is compliant with the conceptual model described in Section 3 and is also very valuable for single system requirements engineering if legacy documentation is available.

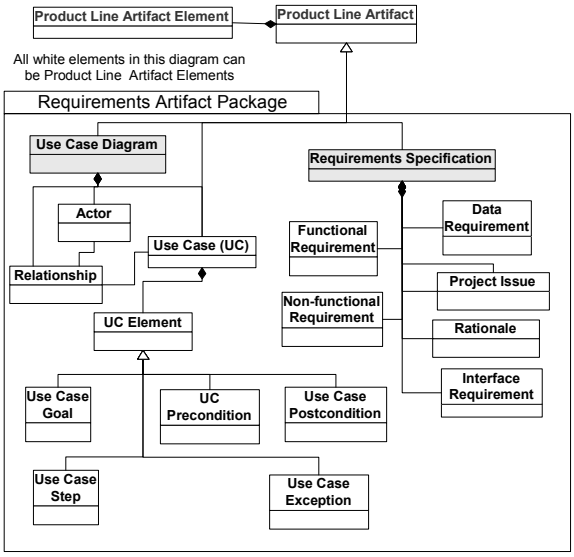



**Figure 6 Requirements Artifact Model**

With the elicitation approach common and variable features [19], Use Case elements [7], tasks describing user activities in an interactive system [24] and textual requirements can be elicited. As existing systems are the basis for this approach, it can be seen as a reengineering method for documents transferring user documentation into basic elements of information for requirements specifications. The approach was applied in three case studies [18] [11], further case studies will follow. The approach consists of the following phases (c.f. Figure 5) :

- Preparation
- Search
- Selection, change and modification.

The first two steps of the approach can be performed by persons who just have a slight domain understanding, they do not have to be domain experts. The third step requires involvement of domain experts as there documentation entities have to be validated and selected. We will now describe the three steps in more detail.

## 4.1. Preparation

Preparation consists of the four sub steps collection, selection, division and browsing. During collection, user documentation for the systems that should be integrated into the product line and of those systems that are related should be collected to have all needed information available. In the case of a project-integrating product line adoption these are all user-documentations of the systems currently under development (as far as they already exist), in the case of a reengineering-driven or leveraged product line adoption all user documentations of existing systems in the domain have to be considered. As parallel reading of more than one document requires divided and increased attention and leads to lower performance [32], the number of documents to be read in parallel should be reduced to a minimum. So, if there are more than 3 systems, we recommend to select two or three documents that cover the variety of systems (e.g., one documentation of a low-end system, one of a high end system and one typical system) to compare for a first search in the documents. The other documents can be used to complete the elicited information after completing the search phase.

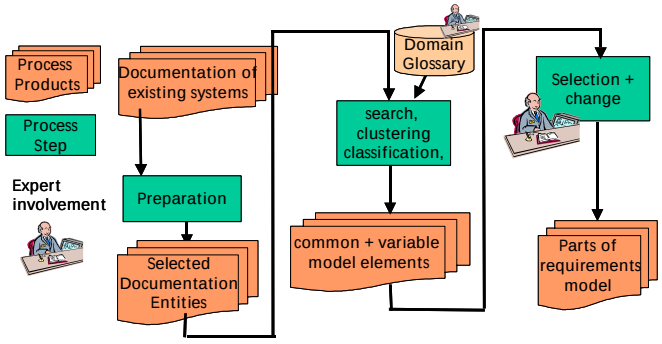



**Figure 6 An outline of the elicitation approach**

After selecting the three typical documentations, divide them into manageable and comparable parts of 3 to 10 pages (e.g., comparable subchapters). In browsing, for each of those manageable parts (or for a subset of those parts that includes typical sub domains) browse through them in order to decide the amount of variability in them. There are two alternatives:

- For those document parts that look obviously different (e.g., that differ in more than 30% of the text), process them one after another in the following phases. Start the analysis with the biggest document
- Compare other documents in parallel in the following phases.

## 4.2. Search

In the search step the identified user document parts containing documentation elements (c.f section 3.1) are analyzed and requirements artifacts are searched. The elements to be identified in the documents, which should be sized from one word to at most 5-6 lines, are marked and tagged in the source documents. Common and variable requirements artifacts that can be identified for Use Cases are for example Use Case names, actors, goals, preconditions, steps of descriptions, success conditions, and extensions. Also features and different kinds of requirements can be defined.

Common and variable requirements artifacts can be identified and marked in the text with the following heuristics The heuristics described here are heuristics that transform user documentation into requirements artifacts, so these heuristics build a connection between user documentation and requirements artifacts by using requirements concepts and variability (c.f. Figure 2, the heuristics described here are condensed heuristics) The heuristics we show here are just examples, the complete heuristics can be found in [18]:

**Use case elements**

- Headings of sections or subsections typically contain names of Use Cases.
- Phrases like “only by”, “by using”, “in the case of” can be markers for Use Case preconditions.
- Phrases like “normally” “with the exception”, “except” can mark Use Case extensions.
- Numbered lists or bulleted lists are markers for an ordered processing of sequential steps and describe Use Case descriptions.
- Sentences that describe interactions with the system in the form of “to do this…do that…” are Use Case descriptions.
- Passive voice is typically a marker for system activity

(e.g. “The volume of the radio is muted” = the system mutes the volume of the radio). These sentences can be used in the Use Case description.

**Requirements**

- Phrases like “press”, “hold”, “hold down” , “press briefly”, “select” , “key in” “scroll” etc. mark a dialogue with the user interface or navigation elements
- Activities or system functions are those elements that were marked as features that contain a verb
- Non functional requirements cannot be found explicitly in user manuals, but hints to non functional requirements and to qualities can be found. Shortcuts are alternative usage scenarios and can therefore be a marker for a non functional requirement like “the system shall be used in two alternative ways….”
- Adverbs and adjectives (longer, fast, quickly….) can mark NFRs, especially if a phrase or sentence appears in the user manual once with the adverb, once without. (e.g. “to turn off the radio” and “to quickly turn off the radio”)
- Technical data can give a clue to non-functional attributes of the system (e.g. size of the display, battery size etc.)
- Numbers in the identified elements can be hint for a non-functional requirement (why was exactly this number chosen?)

**Features**

- Headings of sections or subsections typically contain features
- Features can be found in highlighted phrases (bold or italic font) or in extra paragraphs
- Technical descriptions or short descriptions of a system often contain lists of features

**Commonalities and variabilities**

- Arbitrary elements occurring only in one user manual probably are optional elements.
- Headings or subheadings that only occur in one of the documentations can be model elements that are optional as a whole.
- Headings or subheadings that have slightly different names or headings or subheadings that have different names but are at the same place in the table of contents can be hints for alternative model elements.
- Phrases that differ in only one or a few words can be evidence for alternatives.
- If numerical values in the document differ they can be parametrical variabilities.

These heuristics form a first set of heuristics that will be extended in future when applying more case studies. With the support of these heuristics, which help in finding a significant part of the requirements artifacts (i.e., of the requirements specification or product line model) and variabilities, the user documents should be marked (e.g., with different colors for different model elements and for variabilities) and integrated into an intermediate document. The identified elements should be extracted from the document and tagged with attributes containing the information needed for selecting appropriate elements for modeling the product lines requirements. Table 1 shows the elements of such a notation.

### 4.3. Selection

In the last step, selection, the extracted and tagged elements have to be checked and possibly adjusted by a domain expert. The domain expert will change the elements regarding the following aspects:

- Is a text element that was marked as a possible requirements artifact really a requirements artifact that shall be integrated into the requirements specification and product line model, respectivelly?
- Is an element marked as optional/alternative really an optional/alternative element in the new product line?
- In case we have want to build a product line: Are the product line models to be built out of the elements the right models to describe the systems of the product line?

The relations (see Table 1) are used to make comparisons between the documents easier, to establish traceability to the source documents and, with tool based selection, to support navigation in the elements and between the sets of documents.

### 4.4. Results

The results of the approach are approved requirements artifacts that can easily be integrated in requirements specifications and product line model elements, respectively. Which model elements should be elicited depends on what the modeling approach used needs as primitives. The relations (see last lines of Table 1) are used to make comparisons between the documents easier, to establish traceability to the source documents and, with tool based selection, to support navigation in the elements and between the sets of documents. With these elements the domain expert and the requirements engineer can build Use Cases and requirements using the information about the elements collected in the tags.

We have applied the approach in four case studies with real documentation from different systems in three domains (automotive: user manuals of cars, telecommunication: user manuals of mobile phones, civil engineering: user manuals of calculation software) one of the case studies is described in [11], the others will be described in [18]. The user manuals investigated in the case studies were different in structure and content but applying the heuristics was successful in all cases. In the case studies Use Case elements, functional and non-

**Table 1 Attributes for the elicited document parts**

| Attribute | Values | Description |
|---|---|---|
| ID | e.g. 1…n or docnumber.nr | A unique identifier for the element |
| Value | Text | The text of the element that was found in the document |
| Document | Identifiers | The identifiers of the documents this element was found in |
| Requirements Artifact Type | Requirements Artifact | The reqirements artifacts (Use Case description, precondition, feature, textual requirement..) the text matches to |
| Var Type | commonality, optionality, alternative, range | The hypothesis for the variability type of the element (default is commonality) |
| Parent | ID | The element, this element is part of |
| relations | Requirements Artifact Name | A possible requirements artifact this element is related to |
| Var relations | List of IDs | The IDs of other elements that contain alternatives or different parameters for this element |

functional requirements and features were the primary elements found by comparing the documentation of three to five legacy systems.

## 5. Conclusions

In this paper we described an approach for elicitation and specification of requirements specifications and product line models, respectively, based on existing user documentation. Use Cases, which are quite common in single system requirements engineering are also often used in product line engineering to model requirements on a line of systems. The approach we described here supports capturing of the information found in user documentation of legacy systems to use them in requirements specifications and product line models, respectively. The approach can be used for building single system requirements and product line requirements, only the elicitation of commonality and variability that is described in the variability model part of the conceptual model and in the variability heuristics is product line specific. We presented heuristics that allow an easy identification of text elements in user documents based on a conceptual model. The approach we propose here can support other elicitation activities and can give basic information on the existing systems.

With the help of a supporting tool that will also be based on the conceptual model, the selection of the text elements and the tagging with the attributes could be performed semi-automatically. We plan to support our elicitation process by such a tool but only to generate propositions for elements, not for an automatic analysis. The process of analyzing a user manual in a semi-automated process opens up the possibility to capitalize on the wealth of domain knowledge in existing systems considered for migration to next-generation systems. Converting these existing requirements into domain models can reduce cost and risk while reducing time-to-market.

## Acknowledgements

This work was partially supported by the Eureka Σ!2023 Programme, ITEA , Ip00004, Project CAFÉ and Ip00103, Project Empress.
We want to thank Alessandro Fantechi, Stefania Gnesi and Guiseppe Lami for performing the joint case study described in [11] that influenced the work described here.

## References

[1] I. Alexander and F. Kiedaisch. Towards recyclable system requirements. In Proceedings of ECBS'02, Lund, Sweden, 2002.

[2] G. Arango. Domain analysis methods. In W. Shaefer, R. Prieto-Diaz, and M. Matsumoto, editors, Software Reusability. Ellis Horwood, 1993.

[3] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In Proceedings of the Symposium on Software Reusability (SSR'99), Los Angeles, CA, USA, May 1999. ACM.

[4] J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In Proceedings of GCSE '99, Erfurt, Germany, September 1999

[5] Robert Biddle, James Noble, and Ewan Tempero. Supporting Reusable Use Cases. In Proceedings of the Seventh International Conference on Software Reuse, April 2002.

[6] P. C. Clements and L. Northrop. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, August 2001

[7] A. Cockburn. Writing Effective Use Cases. Addison Wesley, 2001.

[8] J.-M. DeBaud and K. Schmid. A Practical Comparison of Major Domain Analysis Approaches - Towards a Customizable Domain Analysis Framework. In Proceedings of SEKE'98,San Francisco, USA June 1998.

[9] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The linguistic approach to the natural language requirements quality; In Proceedings of the 26th Annual IEEE Computer Society Nasa Goddard Space Flight Center Software Engineering Workshop, 2001.

[10] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, Application of Linguistic Techniques for Use Case Analysis, RE'02, Essen, Germany, September 2002

[11] A. Fantechi, S. Gnesi, I.John, G.Lami, J. Dörr Elicitation of Use Cases for Product Lines. Submitted

[12] D.C. Gause, G.M. Weinberg Exploring Requirements – Quality before Design. Dorset House Publishing, 1989

[13] J. A. Goguen, Charlotte Linde, Techniques for Requirements Elicitation, Proceedings of the 1st International Symposium on Requirements Engineering, p.152-163, 1993

[14] G. Halmans, K. Pohl Communicating the Variability of a Software-Product Family to Customers Journal of Software and Systems Modeling, Springer, 2003 to appear

[15] IEEE-Std 830-1998 IEEE Guide to Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, New York, 1998

[16] I. John. Integrating Legacy Documentation Assets into a Product Line. In: Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4), Bilbao, Spain, October 2001.

[17] I. John, D. Muthig, Tailoring Use Cases for Product Line Modeling, REPL'02, Essen, Germany, September 2002

[18] I. John, J. Dörr. Extracting Product Line Model Elements from User Documentation. Technical Report, Fraunhofer IESE, 2003, to appear

[19] 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, November 1990.

[20] F. van der Linden. Software Product Families in Europe: The Esaps and Café Projects. IEEE Software, 19(4):41--49, July/August 2002.

[21] Y. S. Maarek, D. M. Berry, and G. E. Kaiser. GURU: Information retrieval for reuse. In P.Hall, editor, Landmark Contributions in Software Reuse and Reverse Engineering. Unicom Seminars Ltd, 1994.

[22] M. Mannion, B. Keepence, H. Kaindl, and J. Wheadon. Reusing Single System Re-quirements for Application Family Requirements. In Proceedings of the 21st International Conference on Software Engineering (ICSE'99), May 1999.

[23] Dirk Muthig A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. PhD Theses in Experimental Software Engineering. Dissertation, Fraunhofer IRB, 2002.

[24] B. Paech and K. Kohler. Task–driven Requirements in Object-oriented Development. In Leite, J., Doorn, J., (eds) Perspectives on Requirements Engineering, Kluver Academic Publishers, 2003, to appear

[25] P. Rayson, L. Emmet, R. Garside, and P. Sawyer. The REVERE project: experiments with the application of probabilistic NLP to systems engineering. In Proceedings NLDB'2000. Versailles, France, June, LNCS 1959, 2000.

[26] K. Schmid and M. Verlage. The Economic Impact of Product Line Adoption and Evolution. IEEE Software, 19(4):50--57, July/August 2002.

[27] K. Schmid and I. John. Generic Variability Management and its application to Product Line Modeling in Proceedings of the First Workshop on Software Variability Management, Groningen, 2003.

[28] Software Technology for Adaptable, Reliable Systems (STARS). Organization Domain Modeling (ODM) Guidebook, Version 2.0, June 1996.

[29] B. Tschaitschian, C. Wenzel, and I. John. Tuning the quality of informal software requirements with KARAT. In Proceedings of REFSQ'97, 1997.

[30] A. v. Knethen, B. Paech, F: Kiedaisch und F. Houdek. Systematic Requirements Recycling through Abstraction and Tracability. Proceedings of RE02, Essen, 2002.

[31] D. M. Weiss and C.T.R. Lai. Software Product Line Engineering: A Family Based Software Development Process. Addison-Wesley, 1999.

[32] C.D. Wickens. Processing resources in attention. In R. Parasuraman & R. Davies (eds.), Varieties of attention (pp.63-101). New York, 1984, Academic Press

# Creating Requirements – Techniques and Experiences in the Policing Domain

L. Pennell[1] & N.A.M. Maiden[2]

[1]Police Information Technology Organisation
Lucinda.Pennell@pito.pnn.police.uk
[2]Centre for HCI Design, City University
Northampton Square, London EC1V OHB, UK
n.a.m.maiden@city.ac.uk

**Abstract**

*Processes and techniques to discover and create requirements rather than elicit and acquire them from stakeholders have received relatively little attention in the requirements engineering literature. In contrast, researchers in artificial intelligence and cognitive and social psychology have been researching creativity for some time. More recently we have applied their theories and models to requirements engineering. In this experience paper we report results and lessons learned from 2 creativity workshops undertaken with the UK's Police Information Technology Organisation, in which theories and models of creativity informed creative thinking about requirements and opportunities for bio-metric technologies in policing. The main results are presented as lessons learned for the wider requirements engineering community.*

## 1. Introduction

Requirements engineering is a creative process in which stakeholders and designers work together to create ideas for new systems that are eventually expressed as requirements. The importance of creative product design is expected to increase over the next decade. The Nomura Research Institute [1] argues that creativity will be the next key economic activity, replacing information. Creativity is indispensable for more innovative product development [2], and requirements are the key abstraction that encapsulates the results of creative thinking about the vision of an innovative product. It is a trend that requirements engineering researchers and practitioners, with their current focus on elicitation, analysis and management, have yet to grasp fully.

In this experience paper we describe how we applied creative thinking techniques including random idea combination, analogical reasoning and storyboarding as part of a requirements process. The UK Police's Information Technology Organisation (PITO) was seeking to discover new requirements and opportunities to exploit bio-metric technologies in its applications. To this end, the authors ran two facilitated workshops in which Police stakeholders were encouraged to use techniques to think creatively and discover new requirements and opportunities. Although the workshops were a success, the results suggest that different creativity techniques were more successful than others at discovering new requirements. We report the most important results from the workshops as lessons learned that the wider requirements community can learn from and apply in their own activities.

The remainder of the paper is in 6 sections. Section 2 reports work on creativity in other disciplines and its limited application in requirements engineering. Section 3 describes PITO, its bio-metric applications and the rationale behind its 2 creativity workshops. Section 4 describes the workshops themselves, then section 5 reports the main results and section 6 the important lessons learned from the workshops. The paper ends with a brief discussion and future work for both PITO and the authors.

## 2. Related Work

In spite of the need for creative thinking in the requirements process, requirements engineering research has largely ignored creativity and few processes, methods and techniques address it explicitly. Brainstorming techniques and RAD/JAD workshops [3] make tangential reference to creative thinking. Most current brainstorming work refers back to Osborn's text [4] on principles and procedures of creative problem solving. The CPS method describes six stages of problem solving: mess finding, data finding, problem finding, idea finding, solution finding and acceptance finding. The model was originally intended to help people understand and use their creative talent more effectively [5]. It has been through several waves of development. To better describe how problem solving occurs, and to improve the flexibility of the model, the six stages were arranged into three groups – understanding the problem, idea generation, and planning for action. A recent CPS manual [6]

describes activities for supporting each of model stage. Examples to support combinatorial creativity include *the matrix*, which involves making lists then selecting items from each list at random and combining them to generate new ideas, and *parallel worlds*, which uses analogical reasoning to generate new ideas. However, there are few if reported applications of the CPS model in the requirements domain.

Robertson [7] argues that requirements analysts need to be inventors to bring about the innovative change in a product or business that gives competitive advantage. Such requirements are not often things that a stakeholder directly asked for. Nguyen et al. [8] have observed that requirements engineering teams restructure requirements models at critical points when they re-conceptualize and solve sub-problems, and moments of sudden insight or sparked ideas trigger these points. However, elsewhere, there is little explicit reference to creativity in mainstream requirements engineering journals and conferences.

One exception, our RESCUE scenario-driven requirements process [9], incorporated creativity workshops to encourage creative thinking about requirements and the earlier stages of design for new ATM systems. Creative activities were grounded in the referenced theories of creativity from cognitive and social psychology, then data from the workshops was analysed to determine the relevance of these theories to creative requirements processes. We designed each workshop to encourage 4 essential processes based on existing theories of creativity [10,11]: preparation, incubation, illumination and verification. We also encouraged exploratory creativity by encouraging stakeholders to reason with analogical textile design and musical composition tasks. Likewise we encouraged combinatorial creativity through random idea generation and parallels with fusion cooking [12].

The RESCUE creativity workshops benefited the requirements process in two ways. Firstly, the candidate design space reduced the number of requirements to consider by rejecting requirements that could not be met by current technologies. Secondly, high-level decisions about a system's boundaries enabled the team to write more precise use cases and generate more precise scenarios that, in turn, enable more effective requirements acquisition and specification [9]. Lessons learned from these workshops were applied in the design of the PITO creativity workshops reported in this experience paper.

## 3. Creating Requirements to be Satisfied by Bio-metric Technologies

PITO provides information technology, communications systems and services to the police within the United Kingdom. It gathers requirements for these systems and services from the UK Police Forces. One of the problems experienced during PITO's requirements acquisition processes is the tendency for stakeholders to think in terms of solutions that can unnecessarily constrain the system design. In order to encourage innovation in new policing systems, stakeholders need to stop thinking about solutions during the requirements process and focus on creative thinking about their business needs. As such, PITO are currently looking for methods and techniques to support their requirement engineering processes. Creativity workshops based on the RESCUE process workshops were trialed as a source of innovation for producing more creative PITO requirements and systems.

PITO trialed the creativity workshops as part of its bio-metrics program to gather requirements for bio-metric technologies as a basis for future police applications. PITO is aware of the diverse sources of new requirements for information systems, for example reports of problems with existing systems or changes in business process, but new technologies are increasingly a source for new requirements [13]. For example the existence of web technologies led the UK Police Service to develop a web site that enabled citizens to report non-urgent minor crimes on-line – this requirement would not have existed as a viable requirement had the internet not been widely available to the UK public. As such, the PITO bio-metrics program was in a position to benefit from a new approach to discovering requirements which would encourage stakeholders to think creatively.

## 4. Two Creativity Workshops in PITO

This section describes the 2 prototype creativity workshops that were designed and ran for PITO's bio-metrics program.

### 4.1. Sequence and Structure

The 2 prototype creative workshops were based on the RESCUE process workshops designed and ran by City University's Centre for HCI Design and the Atlantic Systems Guild, but tailored to meet PITO's local needs. This meant that the principles of the RESCUE workshops could be re-applied but the workshop designs could not be.

**Participants**: the two workshops were attended by six participants representing a cross-section of roles often involved in producing requirements for a PITO project. These participants were two technical experts (in this case 2 bio-metrics experts), two experienced police officers, and two experienced requirement analysts. The objective set for these participants was to produce new and creative ideas for the use of bio-metric technologies within the UK police service. Each participant was chosen to represent one of these domains of expertise, but each also had knowledge of at least one of the other domains.

These six people had never worked as a team before, although the two technical experts had worked together before, as had the police officers and the requirement engineers. In addition, each workshop had an experienced facilitator and a scribe. The facilitator was Neil Maiden from City University who had facilitated the earlier RESCUE workshops for the CORA-2 project. The scribe was Alexis Gizikis, also from City University, who had also scribed for some of the RESCUE workshops. Unfortunately Alexis was unable to attend the first workshop and a participant acted as the scribe. Neil and Alexis also acted as pseudo-experts in air traffic management during the second workshop as part of the analogical reasoning activity reported in section 4.4. Both had considerable exposure to and knowledge of the air traffic management domain during the RESCUE CORA-2 project with Eurocontrol. However, this role should normally be preformed by a domain expert who would have more detailed knowledge of the domain.

**Environment**: Both workshops took place on PITO premises in a usability laboratory that enabled them to be recorded onto video. Presentations were displayed on a large LCD Screen and 2 monitors so the images could be seen from any part of the room.

The room was set up with two tables around which two groups of participants sat. The ideas that were generated during the workshops were placed on pin boards on the walls of the room so that the participants were able to see them and add to them throughout the workshop. Sufficient room was left for the participants to move around the room during both workshops.

**Facilitation**: The workshops were facilitated to encourage a fun atmosphere so that the participants were relaxed and prepared to generate and voice ideas regardless of how silly they may seem, without fear of criticism. Standard RAD/JAD facilitation techniques and rules [14], for example avoiding criticism of other people's ideas and time-boxing each topic under discussion, were applied throughout both workshops.

**Figure 1. The bio-metrics workshop environment**

**Information Capture**: Participants were supplied with snow cards, post-it notes, A3 paper, felt pens and blu-tack with which to capture the results from the workshops. Everything captured on the posters was subsequently documented electronically and sent to all participants.

**Workshops Agenda**: Each workshop lasted 3.5 hours. The second workshop took place one week after the first one. Each workshop was divided into two distinct creative activities. There was an introduction phase at the start of the first workshop, and an interim phase in the week between the two workshops, during which the participants were encouraged to undertake further creative thinking as input into the second workshop. The timings, structure, activities and deliverables of the most activities from the two workshops are shown in Figure 2.

| Timings and Activities | Activity Description | Intended Outcome |
|---|---|---|
| Workshop 1, 30mns Introduction | Introduce creativity. Define creativity. Elicit participants' opinions about creativity. | Participants have a shared understanding of creativity as a starting point for the workshops. |
| Workshop 1, 30mns The how, why and where of people identification in policing | Participants brainstorm current ways that the police services identify people and the problems that occur when identifying people. | Knowledge of how, where and why the police services currently identify people. A baseline for subsequent creative thinking in the 2 workshops. |
| Workshop 1, 60mns Combinatorial creativity by combining current ideas together | Participants combine the problems identified in the previous activity with capabilities provided by bio-metric technologies as identified by the technology experts. | Participants start finding new ideas for how biometrics can help solve policing problems, from combinations of known and new ideas and technologies. |
| Workshop 1, 30mns Prioritising the new ideas from the preceding activities | Participants vote on the ideas generated from the first workshop, as a basis for focusing creative activities in the second workshop. | A prioritised list of ideas generated from creative thinking in the first workshop. |
| Between workshops Interim activities | The participants are encouraged to continue the combinatorial creativity activity between the workshops. | Participants continue finding new ideas for how biometrics can help solve policing problems, from combinations of known and new ideas and technologies. |
| Workshop 2, 75mns Analogical reasoning with a similar domain | The participants reason analogically about an air traffic management domain in order to create new ideas for a policing system. | Participants understand their domain from a different perspective, and generate new ideas for their domain from that perspective. |
| Workshop 2, 75mns Generate storyboards that encapsulate all creative ideas from the 2 workshops | The participants construct structured storyboards that depict scenarios that include as many ideas as possible that were generated during the preceding activities. | Complex and rich storyboards that integrate the created ideas into coherent potential solutions. |

**Figure 2. The agenda for the 2 workshops**

Each workshop was designed to support the divergence then convergence of ideas as described in the CPS model [5]. Each workshop began with one divergence activity (random idea generation, analogical reasoning) and ended with one converge activity (idea voting, storyboarding).

The following sections describe some of these key activities, and relevant background literature, in more detail.

### 4.2. Brainstorming: How to Detect People

Nickerson [15] reports that one of the earliest attempts to develop a structured approach to the enhancement of creativity started with the promotion of brainstorming by Osborn. Brainstorming is intended to allow participants to produce lots of ideas and to enable their imagination to be stimulated by others ideas. It relies on creating an environment in which participants feel free to suggest any idea without fear of criticism. This can be difficult if the participants have only just met and may take a little time to get going. The PITO brainstorming activity was in 2 parts. In the first participants were asked:

- How do we (the police) identify people (e.g. by recognising face or voice)?
- Why and where do we identify people?

Answers were intended to focus participants on and share knowledge about the business domain. Using answers to these questions the participants were then asked to brainstorm answers to the question:

- What problems do the police have with identifying people?

The purpose of this was to focus the participants on the problems to be solved later in the workshops.

### 4.3. Combinational Creativity

Combinational creativity is, in simple terms, the creation of new ideas from combination and synthesis of existing ideas. As Boden [16] describes, models of creativity fall into two broad categories, because creativity itself is of two types. The first type is combinatorial creativity, where the creative act is an unusual combination of existing concepts. Examples of combinatorial creativity are poetic imagery, free association (e.g. viewing the sun as a lamp), metaphor and analogy. Combinatorial creativity is characterised by the improbability of the combination, or in other words, the surprise encountered when such an unusual combination is presented. Association and analogy are the main mechanisms for combinatorial creativity. Association is the recognition of similar patterns in different domains, sometimes in the presence of noise or uncertainty. The association may be retained and reinforced either by repetition or by systematic comparison of the internal structures of the two concepts. Koestler [17] describes association as the "biosociative act that connects previously unconnected matrices of experience". He states that most creative moments in science are the result of recognising a novel analogy between previously unrelated fields.

Combinatorial creativity by association was applied in the first workshop to create new ideas based on the problems and ideas generated in the preceding brainstorming session. Participants were familiarised with the combinatorial creativity process using an example from the RESCUE workshops, in which the organisers invited a fusion chef to talk about combining unusual ingredients, and to demonstrate fusion cooking. In our workshop the participants worked in 2 groups to generate new ideas to enable police services to identify people more effectively. Throughout the activity the facilitators randomly introduced new biometrics technologies that the participants had to include in the new ideas. The outcome was 2 sets of ideas that incorporated unusual bio-metric technologies in previously unforeseen ways.

### 4.4 Analogical Reasoning

Analogical reasoning is a useful but challenging technique for creative thinking. Analogical reasoning has been the subject of extensive research in both cognitive science and artificial intelligence. However, studies of analogical problem solving suggest that similarity-based reasoning is difficult [18]. Recognising analogies often needs syntactic similarities between problems [19] while inducing mental schemata during analogical matching has proven difficult even for expert software engineers [20].

We have already investigated analogical reasoning in requirements engineering. We define 2 requirement domains as analogous if the domains share a network of knowledge structures that describe goal-related behaviour in both domains [20]. Studies have shown that people can exploit such analogies to reuse requirements if they are given support to recognise, understand and transfer the analogies [20]. In the creative workshops we provided this support but encouraged the participants to go one step further and use the transferred knowledge from the non-policing domains to provoke creative thinking about requirements ideas in the bio-metrics policing domain.

We encouraged analogical reasoning to think creatively about one use of bio-metrics technologies that was prioritised as important by the stakeholders – detecting and monitoring the movement of people in crowds using technologies such as CCTV. The facilitators applied the NATURE Domain Theory [21], of which one of them was an author, to identify and elaborate an analogical match with air traffic management (ATM). Both domains are prototypical instantiations of 3 key NATURE object system models:

- OBJECT SENSING: detecting the complex movements of remote objects in the environment;
- AGENT MONITORING: agents monitor the movement of objects in a remote space;
- OBJECT-AGENT CONTROL: a designed agent seeking to control the movement of remote objects to achieve the goal state of keeping the objects apart in space and time.

The participants again worked in 2 groups of 3 participants. The facilitators encouraged analogical reasoning in 2 stages:

1. Identify and list mappings between agents, objects, actions, constraints and goals in the 2 domains;
2. Use each mapping in turn to generate one or more new ideas about the policing domain by transferring knowledge about problems or solutions from the ATM domain.

To support this process the facilitators used a simple example of analogical reuse between the two rental domains shown in Figure 3. The new ideas were recorded on snow cards and shared between the 2 groups at the end of the activity.

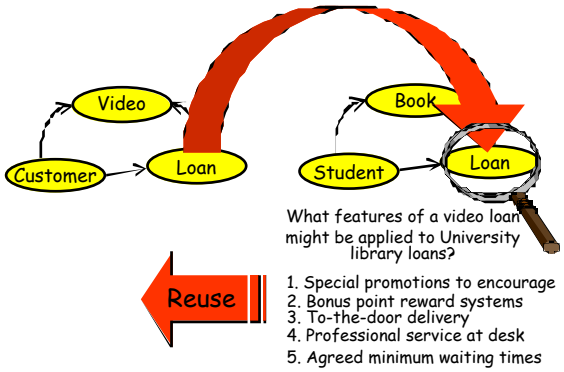



**Figure 3. The rental example used to demonstrate and explain analogical reasoning to the participants**

### 4.5 Storyboards

Storyboarding is a technique that is often used to elaborate creative ideas without constraining the creative process. Participants again worked in 2 groups of 3 participants. Each group was asked to produce a storyboard that described the integration and practical implementation of the ideas generated during the earlier analogical reasoning activity and documented from earlier activities. A practical idea was one that could be implemented in the next 5 years. As such the 2 storyboards were the culmination of the creative process during the 2 workshops, encapsulating many of the creative ideas generated during them.

To structure the storyboarding process each group was given A1-size pieces of paper which were annotated with 16 boxes to contain a graphical depiction of each scene of the storyboard and lines upon which to describe that scene. Examples of a blank and a completed storyboard are shown in Figure 4.

### 4.6 Expert Presentations

Creative thinking requires knowledge from other sources to be successful. One premise behind the workshops is that most people are creative. More creative thinkers search for new ideas by manipulating the knowledge and experience to see different problems, opportunities and solutions. Therefore we used short expert presentations to communicate the relevant domain knowledge to the participants. Each workshop had one such presentation. In the first workshop, one of the bio-metrics experts gave a 15-minute presentation of available bio-metric technologies – these technologies were then used in the subsequent combinatorial creativity activity. In the second workshop, the facilitator gave a 15-minute presentation on air traffic management systems based on his considerable expertise in this analogical domain.

## 5. Results from the Workshops

Both workshops took place and ran to schedule. All planned activities were followed without participant disruption or disagreement, thus making workshop management a relatively straightforward activity. Throughout both workshops we successfully applied standard facilitation rules and guidelines, for example ensuring and controlling all stakeholder involvement. The strong use of the reported techniques meant that potential conflicts about requirements and ideas from different stakeholders arose within each technique, were discovered across groups during presentations, and were resolved during voting at the end of the workshop.

The brainstorming session revealed 18 basic problems that needed to be overcome using bio-metrics technologies. These were: reliability, disguise, quality of information, legislative constraints, admissibility, cost budget, time, resources, memory, limitations of the technology (lack of automation), face blindness, linking to other systems, limitations (human memory, organisational, technological), individual differences, time limitations, sharing of knowledge, false memories, and change of appearance. These problems provided the baseline for subsequent creative thinking in the workshops.

All of the creative activities were undertaken. Both groups combined different problems and bio-metric technologies together according to random permutations generated by the facilitators to generate new ideas. Both groups reasoned analogically with the ATM to generate new ideas about people sensing and location systems in the policing domain – see the example analogical mapping table in Figure 4. Both groups also produced structured storyboards using all of the ideas – as shown in Figure 5.

| Air Traffic | Policing |
|---|---|
| Norm = pattern - identifying abnormalities | Searching for norms and patterns to search for abnormalities |
| Surveillance space activities produce a trigger | Alert and face recognition/CCTV system from motion on a scene |
| Radar | CCTV |
| Mid air collision | Unusual group of people. Man U versus Spurs fans |

**Figure 4. Example analogical mappings produced by one of the groups**

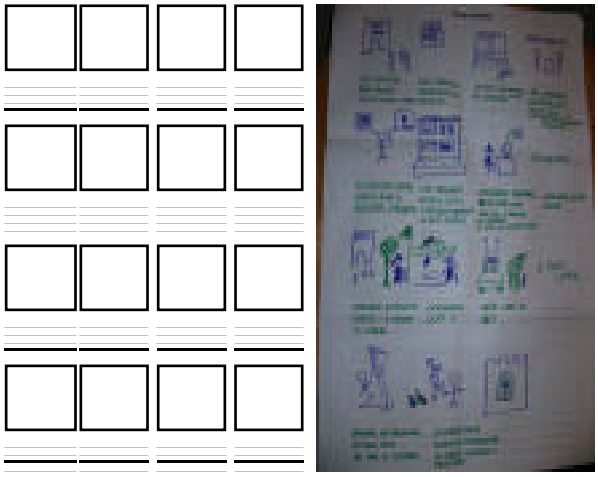


**Figure 5. A blank and completed storyboard template from the second workshop**

The 2 workshops generated a total of 29 new ideas for using bio-metric technologies in PITO applications – 14 in the first workshop and 15 in the second. The ideas from the second workshop tended to be more complete and developed than those from the first. Figure 6 shows the first 4 ideas from either workshop, to demonstrate this difference in the quality of the ideas.

1. Suspect
   Link with scene CCTV-Bio-metric
   Tracking – CCTV – Bio-metric
   Post event Analysis
2. Monitor Event
   2.1. On-line
   2.2. Face in crowd
3. Biometrics in the use of travel e.g. driving into London
4. The use of biometrics on the roadside may shift the balance of a business process

---

1. Reducing paper while maintaining non-repudiation work by using a biometric signature instead of having to print off hard copies which can be signed.
2. Biometric device on a digital camera confirms that scene of crime officer was the person who took that picture at that time.
3. Human rights – A person uses their biometric to release their own information to prove to the police who they are. The person therefore makes the choice whether to release that information.
4. The police could make use (in the form described in 3) of biometrics captured by private organisations for the persons convenience e.g. supermarket privilege card. The public may be less resistant to this then the idea of the police or Home Office keeping this information.

**Figure 6. The first 4 ideas generated from the first and the second workshop – ideas from the first workshop are above and ideas from the second workshop are below**

Post-workshop interviews conducted with all of the 6 participants individually within 24 hours of the end of the second workshop revealed their perceptions about the level of creativity of the ideas. Of the 29 ideas, 25 were considered by at least one participant to be creative. However, participants had quite different views about what makes an idea creative. For example, one participant believed that a creative idea must be surprising but not necessarily useful. Another practical and another felt that to be creative the idea must not have existed before anywhere in any domain.

Figure 7 shows the number of ideas that each participant thought was creative. The participants identified more creative ideas from the analogical reasoning and storyboarding activities in the second workshop than from the brainstorming and combinatorial creativity activities in the first workshop. Another difference is the level of agreement between participants as to which ideas were creative. No more than two participants agreed that a particular requirement from the first workshop was creative, however most of the ideas from the second workshop which were identified as creative had at least 3 three participants in agreement.

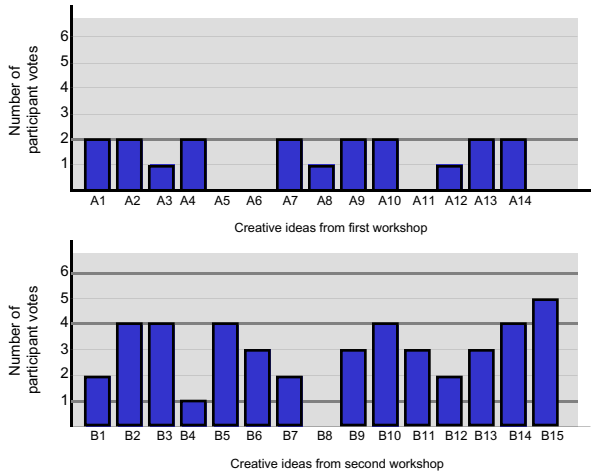



**Figure 7. Creativity of ideas produced during the workshops**

Although the workshops as a whole were seen to be successful, the post-workshop interviews revealed that the effectiveness of the activities varied. Participants claimed the analogical reasoning activity was particularly effective, even though it was new to the participants, which caused some of them to have reservations at the beginning of the activity. The storyboarding activity was also used to good effect. In contrast, the combinatorial creativity activity did not appear as effective at producing creative ideas. However, the fact that the two activities that produced the most creative ideas took place in the second workshop suggests that the activity ordering within the workshops could be a factor.

The brainstorming activity in the first workshop was intended both to stimulate creative thinking and to provide participants with the pre-requisite knowledge about the bio-metrics domain. The importance of allowing time for activities that encourage both

knowledge building and team building may be supported by the fact that the outcome of the combinatorial creativity, which also took place during the first workshop, was knowledge building rather than creative ideas as intended.

All participants agreed that they would consider running this type of workshop in the future due to the potential benefits that it can provide to PITO's requirements engineering processes. This suggests that, from a practical point of view, whether or not the workshops generated genuinely creative ideas was less important than the fact that the workshops enabled participants to produce ideas for requirements that would not normally have been elicited.

## 6. Lessons Learned

In this section we describe 5 lessons learned from the creativity workshops that both inform PITO's future use of creativity workshops in their requirements processes and provide more general lessons learned from this experience paper. Space precludes the inclusion of detailed data that underpins each lesson learned. Rather each lesson is presented as process advice that the reader can use in planning and running creativity workshops.

### 6.1. Explicitly Encouraging Creativity Works

In PITO the 2 prototype creativity workshops succeeded in encouraging the participants to generate new ideas for policing applications that they believe would not have generated using existing requirements processes in the organisation. As such the ideas generated were new to the participants were often new rather than radical and innovative – nonetheless this was perceived by the participants as valuable to PITO. Requirements acquisition techniques often lead to divergent activities [22], in which requirements engineers seek to elicit, acquire, surface, discover and create as many new stakeholder requirements as possible. In this regard the creativity workshops enhance the divergence of requirements early in the requirements process.

### 6.2. Creative Thinking Needs to be Built Up

Results from the retrospective interviews with the participants suggested that the second workshop was more successful at creating ideas than the first, in that more participants believed that more ideas generated in the second workshop were creative. A similar result was found in the RESCUE workshops – the first workshop involved several periods in which the participants cleared the air and understood each others' positions, before effective creative thinking could take place [9]. This finding, combined with our observations and anecdotal evidence from the 2 workshops, suggests that creative thinking requires a period of preparation and incubation [11] during which the participants build up knowledge of the problem domain, a team approach, and confidence in each others' abilities. Therefore, do not expect to encourage creative thinking from the start – it takes time to happen.

### 6.3. Making People Uncomfortable Can Make Them Think Differently

The 2 workshops placed the 6 participants in an unusual environment, working with different people with different roles in PITO to undertake unusual tasks. During some workshop activities some of the participants found the experience uncomfortable – some found it challenging to reason analogically, while others were not used to being told to combine problems, ideas and technologies together against the accepted rules and constraints of the domain. However, responses obtained during the retrospective interviews suggest this discomfort might be essential for creative thinking. The creative activities have the advantage of shaking people out of tried and tested ways thinking about requirements – an important precursor to creative requirements engineering.

### 6.4. Analogies Worked for Some People

The introduction of the analogical reasoning activity using the ATM domain was greeted by some skepticism from the participants. However, once explained, the analogical reasoning worked well for some participants, but not others. Some participants demonstrated the ability to create analogical mappings and transfer knowledge between the domains using these mappings, while others could not. Previous research suggests that analogical reuse is cognitively difficult [20]. This experience supports that, but reveals that effective facilitation of the mapping process as described in this paper can enable some participants to exploit analogical reasoning very effectively. Structure is critical, in contrast to our previous experience. Step-wise, very structured approach to creativity, leaving little to chance. Mapping-by-mapping reduces cognitive effort, but nonetheless difficult for some people, so allow for individual differences.

### 6.5. Storyboarding Worked for Some People

The storyboarding activity was successful in both groups, however one group was able to use storyboards more quickly and effectively than the other. There is a range of possible reasons for this, and we do not have all of the data needed to analyse them. However one observation of this activity was that the more successful group quickly allocated roles to the participants – one participant led the storyboard authoring process, even producing a storyboard of the storyboard in order to structure ideas. The second participant acted as a critic to the storyboard as it was developed, while the third participant, who had good

artistic ability, produced the storyboard in response to instructions from the first 2 participants. In contrast, the other group delayed the start of its storyboarding due to an uncertainty over the structure of the story, and differences of opinion about how to draw the storyboard. Future storyboarding activities should impose a clear structure and role allocation on groups to provide a framework for the thinking creatively and documenting the results of that thinking. Indeed, this apparent dichotomy runs throughout our ongoing planning of creativity workshops – the more successful you want the workshop to be, the more background planning and control is needed to ensure that the right style of creative thinking is encouraged.

## 7. Conclusions and Future Work

This experience paper reports the prototyping of adventurous creativity workshops in a real-world project funded by PITO, the owner organisation. The main finding was that the workshops were effective, in that both generated new ideas that might not have emerged using more traditional requirements acquisition techniques. . The workshop results were accepted by PITO as useful. Retrospective interviews revealed the benefits of some of the activities to the participants.

The improved design of the workshops overcame some of the reported problems in the earlier RESCUE creativity workshops, for example more facilitation for analogical reasoning and structure for storyboarding [9], suggesting that our understanding of how to encourage creative thinking about requirements is increasing. More specifically, some techniques more successful than others. During creative thinking it is important to allow for individual differences between people. One solution is to design in complementary but overlapping techniques might be useful.

We will apply the results and lessons learned from these workshops to a new series of RESCUE creativity workshops as part of a process with Eurocontrol to determine requirements for a Departure Manager system for major European airports. More generally the experience, as part of an effort to improve requirements processes in a large UK organisation, reveals both the opportunities and benefits from thinking about requirements engineering as a creative process. It is one more brick in the wall of evidence that the requirements engineering community needs to think about requirements processes in new and exciting ways.

## Acknowledgements

The authors wish to thank PITO, and particularly all of the workshop participants for their time, involvement and feedback.

## References

1. Nomura 2001, http://members.ozemail.com.au/~caveman/Creative/Admin/crintro.htm
2. Haradon & Sutton 2000.
3. Floyd, C., Mehl, W.-M., Reisin, F.-M., Schmidt, G., & Wolf, G. (1989). Out of Scandinavia: Alternative Approaches to Software Design and System Development. Human-Computer Interaction, 4(4), 253-350.
4. Obsorn A.F., 1953, 'Applied Imagination: Principles and Procedures of Creative Problem Solving', Charles Scribener's Sons, New York.
5. Isaksen, G. & Dorval, K. (1993) Changing views of creative problem solving: Over 40 years of continuous improvement. ICN Newsletter. 3 (1).
6. Daupert, D. (2002) The Osborn-Parnes Creative Problem Solving manual. Available from www.ideastream.com/create.
7. Robertson J., 2002, 'Eureka! Why Analysts Should Invent Requirements', IEEE Software July/August 2002, 20-22.
8. Nguyen L., Carroll J.M. & Swatman P.A., 2000, "Supporting and Monitoring the Creativity of IS Personnel During the Requirements Engineering Process," Proc. Hawaii Int'l Conf. Systems Sciences (HICSS-33), IEEE Computer Society.
9. Maiden N.A.M., Jones S. Flynn M., 2003, 'Innovative Requirements Engineering Applied to ATM', to appear in the ATM'2003 Conference, June 2003, Budapest.
10. Hadamard, J., 1954, 'An essay on the psychology of invention in the mathematical field', New York: Dover.
11. Poincare H., 1982, The Foundations of Science: Science and Hypothesis, The Value of Science, Science and Method, Univ. Press of America, Washington 1982.
12. Maiden N. & Gizikis A., 2001, 'Where Do Requirements Come From?", IEEE Software September/October 2001 18(4), 10-12.
13. Stevens R., Brook P., Jackson K. & Arnold S., 1998, 'Systems Engineering: Copying with Complexity', Prentice-Hall.
14. Andrews D.C., 1991, 'JAD: A Crucial Dimension for Rapid Applications Development', Journal of Systems Management, March 1991, 23-31.
15. Nickerson, R.A. (1999) Enhancing creativity. In The handbook of creativity, edited by R.J. Sternberg (New York: Cambridge University Press) 392-430.
16. Boden M.A., 1990, The Creative Mind, Abacus, London
17. Koestler
18. Gick M.L. & Holyoak K.J., 1983, 'Schema Induction & Analogical Transfer', Cognitive Psychology 15, 1-38.
19. Ross B.H., 1987, 'This is Like That: The Use of Earlier Problems and the Separation of Similarity Effects', Journal of Experimental Psychology: Learning, Memory and Cognition 13(4), 629-639.
20. Maiden N.A.M. & Sutcliffe A.G., 1992, 'Exploiting Reusable Specifications Through Analogy', Communications of the ACM. 34(5), April 1992, 55-64.

21. Sutcliffe A.G. & Maiden N.A.M., 1998, 'The Domain Theory for Requirements Engineering, IEEE Transactions on Software Engineering, 24(3), 174-196
22. Maiden N.A.M. & Rugg G., 1996, 'ACRE: Selecting Methods For Requirements Acquisition, Software Engineering Journal 11(3), 183-192.

# Eliciting Efficiency Requirements with Use Cases

J. Dörr, D. Kerkow, A. von Knethen, B. Paech
Fraunhofer IESE, Sauerwiesen 6, 67661 Kaiserslautern, Germany
{doerrj, kerkow, vknethen, paech}@iese.fraunhofer.de

## Abstract

*Non-functional requirements provide the glue between functional requirements and architectural decisions. Thus, it is important to elicit and specify the non-functional requirements precisely. In practice, however, they are mostly neglected. In this paper, we sketch an approach developed in the context of the EMPRESS project, which allows efficiency requirements to be elicited in conjunction with use cases. This is part of a more general, experience-based approach to elicit and specify non-functional requirements in concert with functional requirements and architecture.*

## 1. Introduction

The last few years have seen a growing awareness of the requirements engineering community for architectural issues and vice versa. Several authors argued convincingly for the tight interdependencies between functional requirements (*FRs*), non-functional requirements (*NFRs*) and architectural options (*AOs*) that need to be made explicit early, for example, [16], [10].

While there are many established methods for the specification of FRs, for instance, use cases [3], and several approaches for specifying AOs, for example, patterns [9], there is little guidance available on how to elicit and specify NFRs in concert with FRs and AOs. The problem is that different kinds of NFRs, such as efficiency or security requirements, need to be treated differently. The different communities concentrating on the different NFRs exemplify this. Thus, it seems difficult to define one method to cope with all NFRs.

In this paper, we propose an approach for specifying efficiency requirements in concert with use cases and, if available, a high-level architecture. This method is so far tailored to efficiency requirements, but we believe that it can be generalized also to other NFRs, such as reliability requirements. We believe this because our approach is based on some general characteristics that can then be used for each type of requirement (e.g., efficiency, reliability, maintainablity requirements).

The main goal of our approach is to achieve a minimal, complete and focused set of measurable and traceable NFRs. The quality criteria on NFRs mentioned are a subset of the general quality criteria on requirements defined by the IEEE-Std. 830 [9].

- *Minimal* means that only necessary NFRs are stated so that the design space is not restricted prematurely.
- *Complete* means that all NFRs of the stakeholders (e.g., customer and developer) are captured.
- *Focused* means that the impact of the NFRs on the solution is clear. A NFR, for example, may concern the system context (namely the customer processes), the system, a FR, or an AO. This supports unambiguity in the sense of the IEEE Std. 830.
- *Measurable* means that a metric is given on how to verify that the system satisfies the NFRs. This supports verifiability and unambiguity in the sense of the IEEE Std. 830.
- *Traceable* means that rationales are given that describe why the NFR is necessary and how it is refined into subcharacteristics. This also supports modifiability in the sense of the IEEE Std. 830.

Our main focus has not been on eliciting consistent NFRs so far. However, our approach includes a consolidation step, where dependencies between elicited NFRs are checked. When specifying means to achieve certain NFRs, consistency has to be treated with more attention.

To accomplish the different quality criteria of the IEEE Std. 830, our approach provides:

- a quality model that captures general characteristics of efficiency (quality attributes), metrics to measure these quality attributes, and means to achieve them. In particular, this model reflects views of different stakeholder roles, such as customer and developer. This quality model supports measurability, completeness as well as focussedness due to the views.
- a distinction of different types of quality attributes, which gives guidance on how to elicit NFRs. This specific treatment for the various types supports focussedness of the NFRs.
- detailed elicitation guidance in terms of checklists and a priorisation questionnaire. The former are derived from the quality model and the types of quality attributes and help to elicit efficiency NFRs in concert with use cases and a high-level architecture. The latter is used to prioritize high-level quality attributes (i.e., maintainability, efficiency, reliability, usability). The

checklists support completeness, the priorization questionnaire supports the focussedness of the NFRs.

- a template, which embeds use cases into a full-fledged requirements document and provides specific places for documenting NFRs. This template supports traceability from NFRs to FRs, completeness and focussedness.
- the use of rationales to justify each NFR. Using rationales supports minimality of the set of NFRs.

The paper is structured as follows. In Section 2, we sketch our terminology and explain the notation of the quality model. Then, we present our approach by way of an example. Section 4 summarizes our experience and Section 5 discusses related work. We conclude with an outlook on future work.

## 2. Terminology

This section describes the foundation of our approach. Subsection 2.1 points out a metamodel that describes the basic concepts of our approach. Subsection 2.2 gives an overview on the "quality model", which instantiates parts of the metamodel.

### 2.1. Metamodel

The metamodel (see Figure 1) describes the main concepts of our approach. Our experience showed, that certain decisions have to be made during the elicitation of NFRs (e.g. does a quality aspect affect a user task, or rather AOs?). The concepts described in the metamodel support these decisions. In the following, we explain the most important elements.

- A *quality attribute* (*QA*) is a non-functional characteristic of a *system*, *user task, system task*, or *organization.* Quality attributes of the organization include development process specific aspects.

  The distinction between different types of quality attributes is important for our elicitation process. Each type of quality attribute is elicited differently (see Section 3). QAs can be *refined into* further QAs. In addition, QAs can have positive or negative *influences* on each other. A more detailed description of the types of QAs and their relationships can be found in Section 2.2.

- A *system* (e.g., "wireless control and monitor system") can be refined into a set of subsystems (e.g., "wireless network", "mobile device"). *Architectural requirements* (e.g., "the system shall have a database") constrain the system.

- We distinguish between two types of *tasks: user tasks* and *system tasks*. *User tasks* are tasks, a certain user has to perform. They are supported by the system (e.g., "monitoring of certain machines"), but include some user involvement. *System tasks* are tasks the system performs. In contrast to user tasks, the user is not involved in system tasks. Tasks can be refined into further tasks. User tasks can be refined into more fine-grained user tasks. Furthermore, user tasks can be refined into parts carried out by the user and system tasks (e.g., a user task "monitoring machine x" is refined into a set of system tasks such as "system displays alarm message if machine runs out of filling"). A *task* is described by one or more *FRs*.

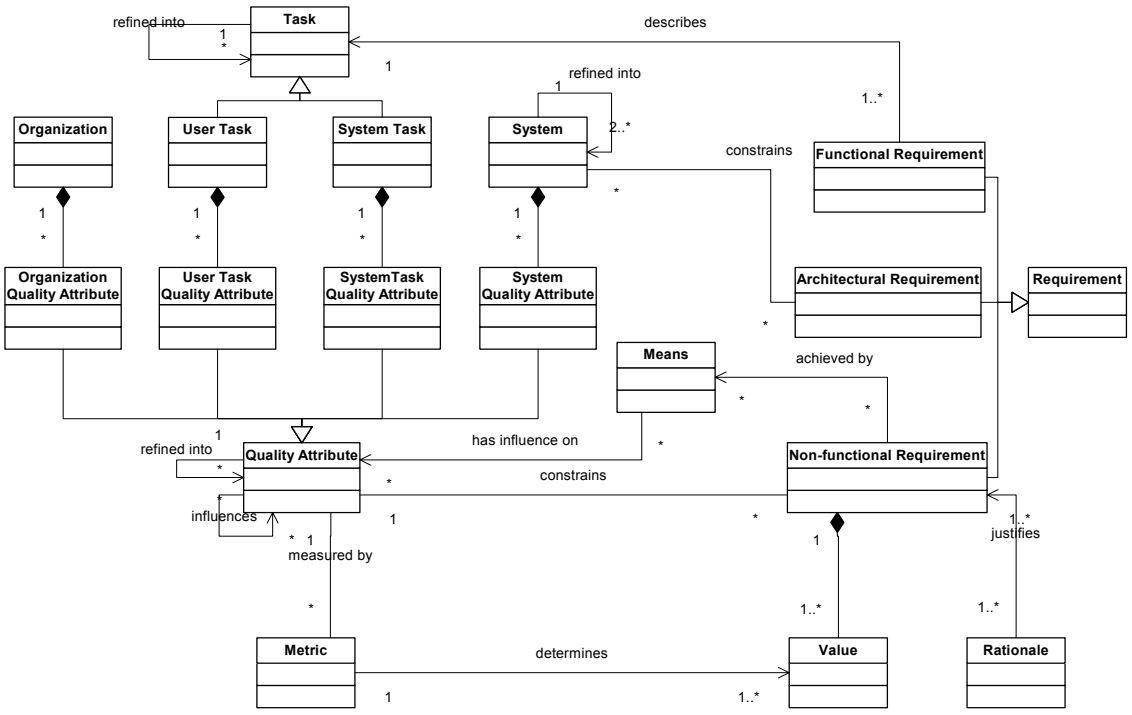



**Figure 1: The metamodel**

- A *NFR* describes a certain *value* (or *value domain*) for a QA that should be achieved in a specific project. The NFR constraints a QA by determining a *value* for a *metric* associated with the QA. For example, the NFR "The database of our new system shall handle 1000 queries per second." constraints the QA "workload of database". The value is determined based on an associated metric "Number of jobs per time unit". For each NFR, a *Rationale* states reasons for its existence (e.g., "the user will be unsatisfied if it takes more than 2 seconds to display alarm message").
- We distinguish problem-oriented refinement (refinement of NFRs according to the constrained QAs) from solution-oriented refinement of QAs. The latter is made explicit in terms of *means*. A *means* is used to achieve a certain set of NFRs. In many cases, a means describes an AO that can be applied to the architecture to achieve a certain QA (e.g., "load balancing" is used to achieve a set of NFRs concerning the QA "workload distribution"). However, a means can also be process related (e.g., the means "automatic test case generation" is used to fulfill NFRs regarding "reliability").

## 2.2. Quality model

A quality model instantiates parts of our metamodel. It describes typical refinements of high-level QAs into more fine-grained QAs, metrics, and means. The idea of the quality model is to refine QAs into QAs that are measurable, i.e., to QAs to which a metric can be associated. In addition, it describes relationships between different QAs. Therefore, it captures experience of previous projects. Our quality model is similar to the goal graphs of, for instance, [12], but emphasizes dependencies, and distinguishes between different types of QAs. Figure 2 gives an example for such a quality model for the QA "efficiency".

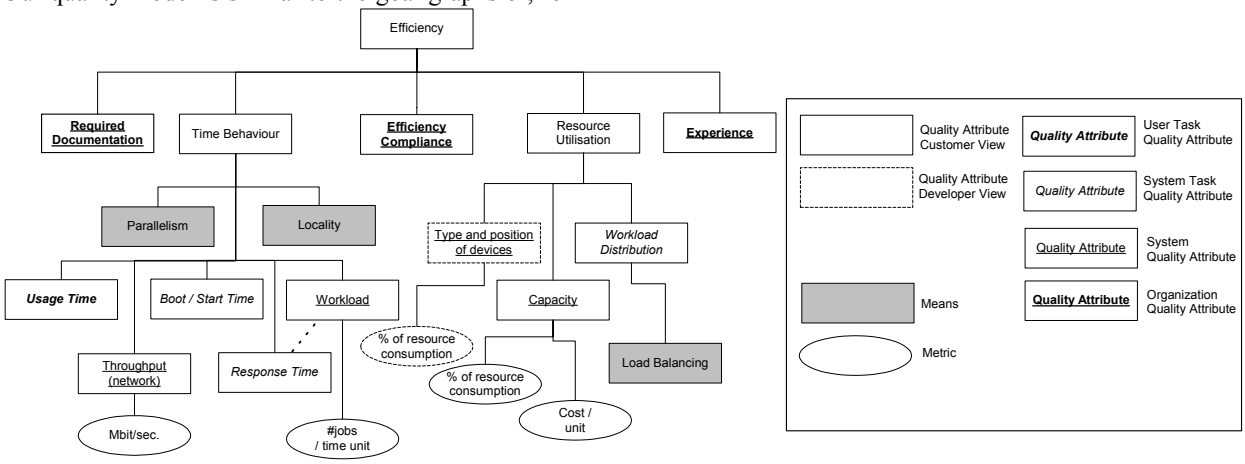



**Figure 2: Quality model for efficiency**

In Figure 2, QAs are represented by white rectangles. Grey rectangles are means that have influence on the related QA and ovals are metrics to measure the related quality attribute. There are five different types of QAs in this quality model (see also metamodel):

- General *QAs* such as "Time Behaviour" are used to structure the QAs on lower levels.
- *Organizational QAs,* such as "Experience", concern the organizational aspects. This also includes development process related aspects, such as required documentations, reviews, etc.
- *System QAs,* such as "Capacity", are QAs related to the system and its subsystems (e.g., related to the database, secondary storage or network).
- *User Task QAs,* such as "Usage Time", are related to tasks in which the system and the user are involved.
- *System Task QAs,* such as "Response Time", are related to system tasks, i.e., tasks that are carried out by the system, not including the user any more (e.g., calculation of results).

Only the latter four QAs are constraint by NFRs. The first type of QA serves as a structuring for the hierarchical decomposition of the more fine-grained QAs. This structure is also used for the template for documenting the NFRs. How the NFRs for the QAs are elicited, depends on the type of the QA they constrain. This is described in Section 3.

Four types of relationships can be found in such a quality model that relates the various kinds of QAs, means and metrics. The metamodel in Figure 1 describes the general types of relationships.

- A QA, such as "efficiency", is *refined into* more detailed QAs, such as "time behaviour" and "resource utilization".

- A means *has influence on* a QA, i.e., it is used to achieve the NFRs constraining the QA. "Load balancing", for example, is influencing "workload distribution" and used to achieve the constraining NFRs (e.g., "The workload for computing the results must be equally distributed on the two processors").
- A QA is *measured by* a metric. The "workload" can, for example, be measured by the metric "number of jobs per time unit".
- A QA can be positively or negatively *influenced by* another QA. If the "workload", for instance, is higher, the "response time " will increase (negative influence).

Our approach provides a default quality model that can be used without adaptations by a company. Reasons for doing so can be a lack of time or money. We recommend tailoring the quality model to the context of each company and project. In addition, a company might have an own quality model that shall be used. In this case, it is very important to agree on the meaning of the different QAs in the quality model. Our recommendation is to build a quality model together with the company in a workshop. By doing so, the quality model benefits from the already integrated experience of our reference quality model and it is tailored to the project and company.

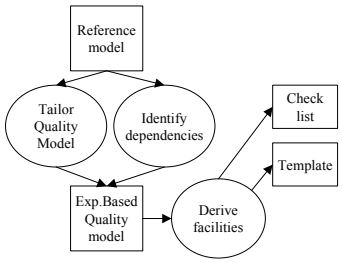



**Figure 3: Experience based creation of a quality model**

Figure 3 describes the process of tailoring the quality model to the project and company. The tailored quality model (experience based quality model) is used as input to develop checklists and templates for documenting NFRs.

The structure of the checklists is given by the hierarchy of the quality model. General QAs (e.g., time behaviour) are, therefore, a means for structuring the checklist, while the QAs at the lowest level (e.g., usage time) are directly used to elicit the NFRs constraining them. The type of the QA influences the way the questions in the checklist are phrased:

- Organizational QAs are used in initialization checklists that focus at general aspects in contrast to the concrete system or its task.
- User task QAs are iterated over the use cases (e.g., use case 1, then use case 2)
- System task QAs are iterated over the use case steps (e.g., step 1, then step 2)
- System QAs are iterated over the various subsystems in the system (e.g., database first, then network1)

The structure of the template is also strongly influenced by the quality model. The NFRs constraining the different types of QAs are denoted at different places in the template:

- NFRs constraining the organizational QAs are documented in an organizational requirements section.
- NFRs constraining user task QAs are attached to the use case diagrams and are, therefore, documented in a use case diagram section.
- NFRs constraining system task QAs are directly attached to each use case in the textual use case description section. Therefore, the use cases have a field "NFRs", where each system task oriented QA is listed. Below such a system task oriented QA, there is a list of the use case steps that express system tasks (e.g., response time: step2, step4). The NFRs for each system task are then expressed at this use case step (e.g., response time: step2 - "The system has to respond within 2 seconds", step4 - "…").
- NFRs constraining system QAs are denoted at two places in the template. First, if a NFR constrains a system QA of a subsystem (e.g., "the database has to store 100000 entries") that is used in a use case, the NFR is attached to that use case. Therefore, each use case also includes a list of system QAs in the field NFRs. Below such a system QA, there is a list of all subsystems (e.g., capacity: database, memory). The NFRs for each subsystem are then expressed at this subsystem (e.g., capacity: database – "the system has to store 100000 entries", memory – "…"). Second, the system NFRs are documented in the section of task overspanning NFRs. The structure is similar to the structure in the use cases (i.e., there is a list of all system QAs, below each system QA there is a list of all subsystems), but it aggregates the NFRs from all use cases and the ones that are not specific for one use case. This is done because a consolidation step searches for dependencies between NFRs concerning one subsystem.

## 3. The elicitation process

As a result of the process "derive facilities" described above, the requirements template is created. Figure 4 shows a subset of this template.

1. Organizational requirements
   1.1. Process requirements
   1.2. Stakeholder requirements
2. Task descriptions
   2.1. UC diagram
   2.2. Textual UC description
3. Task overspanning requirements
   3.1. Textual description of Task overspanning NFR´s

**Figure 4: Subset of the requirements template**

The elicitation process is guided by our experience that various entities (e.g., user task, system task) have different types of QAs. Each NFR has to be elicited under consideration of this entity. In addition, if an entity is described by one or a set of documentation elements (e.g., a user task is described by a use case, a system task is described by a step of a use case), the NFR has to be documented together with this entity.

In the following sections, we describe the activities to be performed within the elicitation process. We use examples from a case study of the CWME project from Siemens about a wireless framework for mobile services. The application enables up to eight users to monitor production activities, manage physical resources, and access information within an industry plant. The user can receive state data from the plant on his mobile device, send control data from the mobile device to the plant components, position the maintenance engineer and get guidance to fix errors on machines. The case study is based on a real system and was provided by Siemens in the context of the Empress project.

## 3.1. Prerequisites

The elicitation process is based upon the documentation of

- the system´s functionality (behavior) described by use cases (Ucs),
- the physical architecture, if available, and further implementation constraints (e.g. constrained HW-resources or constraints derived from the operating systems), and
- assumptions about the average and the maximum amount of data used in the system. The amount of data for each use case is determined under consideration of the amount of data for the entire system.

Since some activities of the elicitation and documentation process are closely related to the functionality, the completeness of the NFRs is limited by the completeness of the FRs.

As described above, some of the QAs are associated to user tasks and system tasks. Therefore, we recommend use cases to describe the FRs. This seems to be beneficial, because QAs associated to user tasks can directly be related to use cases. QAs associated to system tasks can directly be related to use case steps. However, we believe that our approach can be applied to other notations as well.

Figure 5 shows the pre-required documents and the activities to create them.

- *Activities "Prioritize" and "Chose quality models":* Many times, budget and time limitations oblige to prioritize and select a subset of high-level QAs most important for a project. This activity is supported by a prioritisation questionnaire developed at IESE. It builds a ranking order for the QAs described in ISO 9126 (e.g., maintainability, efficiency, reliability, and usability). The questionnaire is described in more detail in [17]. Based on this ranking order, quality models for certain high-level QAs relevant for the project can be chosen.
- *Activity "Elicit FRs":* In this step, the FRs are elicited and documented in form of a graphical use case-diagram. Each use case included in the diagram is later associated to NFRs that constrain QAs of user tasks. In addition, each use case is described textually. The textual description includes an interaction sequence between actor and system. This description allows us later to associate NFRs that constrain QAs of system tasks to use case steps.

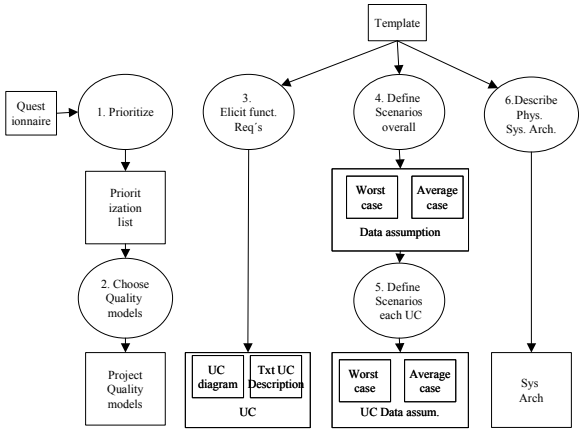



**Figure 5: Development of prerequisites**

- *Activities "Define scenarios" and "Define scenarios for each UC":* In order to be able to imagine NFRs, maximum and average usage data for the overall system, as well as for each use case are elicited and documented.
- *Activity "Describe physical system architecture":* Some NFRs can only be elicited if the detailed physical system architecture is known. So the architecture

must be elicited and documented, whenever it is available.

### 3.2. Elicitation and documentation of NFRs

Figure 6 shows the activities and documents needed to elicit and consolidate NFRs. A checklist that is derived from the quality model as described in Section 2.2 guides each activity. Activities are explained in more detail in the following. We distinguish between different elicitation activities: user task NFR elicitation, system task NFR elicitation and system NFR elicitation. Each activity focuses on eliciting NFRs that constrain one certain type of QA (i.e., organization QA, user task QA, system task QA, and system QA). The user task NFR elicitation is based on use cases. The system task NFR elicitation is based on the interaction sequence described for each use case. The system NFR elicitation is based on physical subsystems and interaction sequences.

*Activity "Elicit organizational NFRs"*

In this activity, NFRs are elicited that constrain QAs of the organization. The customer, for example, might have certain requirements concerning the organizational structure and experience of a supplier. The customer is asked to phrase these requirements. This process is guided by a set of clues in form of a checklist. These clues suggest thinking about domain-experience, size, structure or age of the supplier organization, as well as required standards (e.g. RUP), activities (e.g. inspections), documents or notations (e.g. statecharts). In our case study, some of the requirements expressed were:

- "The supplier needs at least three years of experience in the domain of access-control."
- "The supplier has to create a specification document."

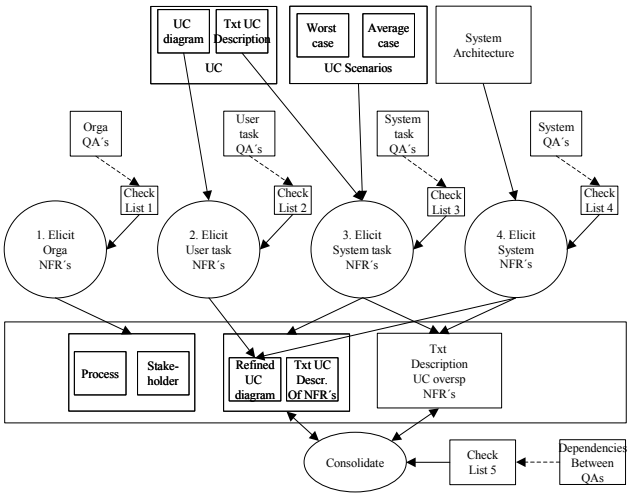



**Figure 6: Elicitation process for NFRs**

To avoid unnecessarily design decisions, the customer is instructed to scrutinize this NFR again, just as Socrates used to try to get to the bottom of statements over and over. This form of Socratic dialogue serves to uncover the rationale behind that NFR and prevents the customer from constraining the system unnecessarily. NFRs are reformulated until they reflect the rationale. It is a good practice to document the rationale as well [5].

As soon as the now elicited and justified NFRs are phrased in a measurable way (this is the case if the metric attached to the QA in the quality model can be applied to the requirement), it is documented in the chapter "organizational requirements" of the template.

*Activity "Elicit user task NFRs"*

In this activity, NFRs are elicited that constrain QAs of user tasks. In our case study, the QA "usage time" included in the quality model is a user task QA. These QAs are documented for each use case included in the use case diagram, because each use case represents a user task. As shown in Figure 7, NFRs are added to use cases with the help of notices.

In our case study the requirement "the use case shall be performed within 30 min." was attached to the use case "Handle alarm". Again, a justification as described above is performed to prevent unnecessary anticipated design decisions. The resulting rationale "breakdown of plant longer than 30 min. is too expensive" is documented in parenthesis behind the NFR.

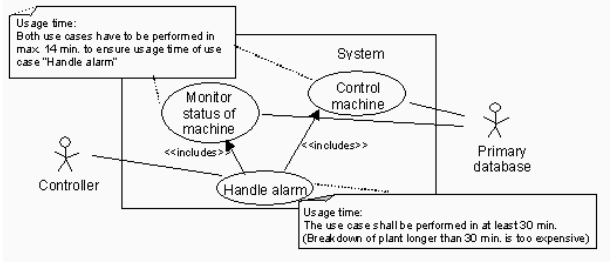



**Figure 7: Use cases with attached user task NFRs**

*Activity "Elicit system task NFRs"*

In this activity, NFRs are elicited that constrain QAs of system tasks. The elicitation is based on the detailed interaction sequence (also called flow of events) documented in the use case. For this activity, maximum and average usage data (Figure 5 shows the development process of this information) are needed. The checklist gives clues of thinking of scenarios where the maximum and the average amount of data are processed in the system. With these scenarios in mind, every step and every exception described by the use case description are checked. Elicited NFRs are documented. Figure 8 shows the textual description of the use case "handle alarm". It

describes that the system shows an alarm and where the alarm was produced. As reaction to this, the user acknowledges the alarm, so other users know s/he is taking care of it.

| | |
|---|---|
| **UseCase** | Handle alarm |
| **Actors** | Controller |
| **Intent** | Actor removes a warning send by a certain machine |
| **Preconditions** | Use Case "Boot up system" |
| **How of events** | 1. System requests alarm messages from the first database regularly<br>2. System shows alarm and where the alarm was produced.<br>[Exception: Multiple alarms]<br>3. Actor acknowledges alarm to let others know he/she is going to take care of it.<br>[Exception: Another actor has acknowledged the alarm]<br>4. Actor moves to the machine following the path displayed by the system.<br>5. During the walk, actor monitors the status of this machine by the system.<br>(-> use case "monitor status of machine")<br>6. Actor removes the problem by controlling the machine<br>(-> use case "control machine")<br>7. System sends control data to first database. |
| **Exceptions** | 1.1 Multiple alarms: System opens a window for each alarm message.<br>3.1 Another user has acknowledged the alarm: System removes alarm message from screen and shows acknowledgement. |
| **Rules** | The alarm warning will always have the highest priority. |
| **NFRs** | **Response time (assumed times):**<br>1. every 5 sec.<br>2. at least in 5 sec.<br>3. just one click<br>4. at least 15 min. (worst case)<br>5. -> usage time of use case "monitor status of machine"<br>6. -> usage time of use case "control machine"<br>7. at least 5 sec. |

**Figure 8: UC steps with attached system task NFRs**

As a result of the elicitation and documentation process, NFRs that constrain the system task QA "response time" were documented. The NFR "at least in 5 sec." was attached to the use case step 2 "System shows alarm and where the alarm was produced" and the NFR "just one click" was attached to the users reaction described in use case step 3. Both requirements were documented in the NFRs field within the textual description of the use case, after being justified by the customer in the Socratic dialogue. The rationale lead to the statement, that the NFRs elicited were assumed times only and could be changed, if necessary. As shown in Figure 7 and Figure 8, the rationale was documented in parenthesis.

*Activity "Elicit system NFRs"*

In this activity, NFRs are elicited that constrain QAs of the system and subsystems. In this activity, again maximum and average usage data is needed. Additionally, the architecture of the physical subsystems is used, if available. The subsystems and architecture constraints on our case study are shown in Figure 9.

Constraints on overall architecture:

- Windows CE OS at PDAs
- Standard PDAs (replaceable)
- Standard Network components (replaceable)
- Throughput: WLAN 11Mbit/sec
- Server: Windows 2000
- Secondary Database -> PDAs: Wireless Network required
- Downloading and monitoring at the same time is not possible

**Figure 9: Constraints on system-architecture**

The checklist gives instructions on how to consider the scenarios while phrasing NFRs for each use case description and physical subsystem of the system architecture. As Figure 10 shows, the NFR field of the use case description is segmented into NFRs related to every physical subsystem.

**Throughput requirements:**
**Network between secondary database and PDA:**

- shall be able to deal in average case with 2 alarms every 10 minutes with 16 machines (assumed average number of alarms)
- shall be able to deal maximal with 8(1/PDA) * 60 alarms at the same time (assumed maximal number of alarms)
- shall be able to deal maximal with 8 people that download 1 doc (size of 8 docs constrained to: <55MBit) /person within 5-10 secs (assumed maximal number of downloads)

**Capacity requirements:**
**PDA:**

- shall have a maximum capacity of 64 MB (standard components shall be used to reduce costs)
- shall be able to handle up to 60 alarms at the same time (assumed maximal number of alarms)

**Workload requirements:**
**PDA:**

- shall allow 5 programs to be opened at the same time (assumed maximal number of programs that will be opened by the user)

**Figure 10: UC with attached system NFRs**

In the use case "handle alarm", NFRs for the QA "capacity" could only be phrased for the physical subsystem "PDA". The subsystem shall have a maximum capacity of 64 MB and shall be able to handle up to 50 alarms at the same time. The rationale for this NFR is the need for usage of standard components available at the consumer market. This rationale is documented as well.

The QA "throughput" does only apply to the subsystem "Network" by definition. Our experience shows, that some QAs are related to only a subset of subsystems. This relationship is documented in the quality model.

The elicited NFRs for single subsystems are documented within the textual use case description as well as in the section "use case overspanning textual description of NFRs". This is done to be able to consolidate the requirements over several use cases.

*Activity "Consolidate"*

In this activity, the NFRs are analysed for conflicts. This activity includes two sub-activities. In the first, NFRs for one physical subsystem are analysed over all use cases. The checklist gives hints on how to identify conflicts and how to solve them. It has to be checked, for example, whether NFRs can be achieved if use cases are executed in parallel. In the second sub-activity, NFRs that constrain different QAs are validated under consideration of the dependencies documented within the quality model.

The consolidation activity discovered an important conflict between the determined throughput requirements and the defined hardware constraints. As shown in Figure 10 one of the throughput requirements stated:

- "The network between secondary database and PDA shall be able to deal in worst case with 8 people that download 1 doc (size of 8 docs constrained to <55Mbit) / person within 5-10 secs."

The restriction of the total size of 8 documents to 55 Mbits was added because the hardware constraints shown

in Figure 9 constrained the network to a 11Mbit/sec WLAN. The additional requirement would not been found without the consolidation activity.

## 4. Experience

We have used this approach so far in a case study with Siemens in the Empress project and in a workshop with 10 practitioners. In the case study, we spent half a day with the customer in discussing and tailoring the default quality model to the case study project and half a day in eliciting the NFRs. The customer acknowledged that the time was very worthwhile as he discovered many new NFRs he had not been aware of before. Also, it helped him to specify them more precisely. In the workshop, we spent one hour explaining our method and then within another two hours we interactively went through the checklists and filled the template. Again, the feedback was very positive as the participants acknowledged that this was the first systematic method they had seen to elicit efficiency NFRs. They particularly liked the idea of the quality model, checklists, and template to capture experience on NFRs. In addition, they liked the use of use cases and the architecture to ensure completeness and ease traceability. They also pointed out the need for capturing the rationale and a supporting tool environment.

## 5. Related work

At last year´s REFSQ we presented the following challenges for a method for the integrated elicitation and specification of FRs, NFRs and AOs [16]:

- Issue 1: Adequate abstraction levels for the elicitation and alignment of FR, NFR and AOs
- Issue 2: Views of different stakeholders in the elicitation of NFRs, FRs and AOs
- Issue 3: Identification of dependencies among FRs, NFRs and AOs
- Issue 4: Compact description of the solution space

In this paper, we concentrate on the first two issues The quality model contains abstract descriptions of NFRs (in terms of QAs) and AOs (in terms of means). Thus, to solve issue 1, we provide two main levels of abstraction. Within the quality model, QAs are refined on as many levels as necessary to distinguish different aspects. With respect to issue 2 (views), we distinguish developer and customer view. We do not support negotiation explicitly. However, by providing a standardized terminology in terms of the quality model, we help reducing conflicts and misunderstanding. The checklists make sure that all relevant aspects are considered.

We also give some hints on how to deal with issue 3 and 4. For dependencies again the quality models helps identifying typical dependencies. This is elaborated in the checklists. With respect to issue 4 (assessment), we use rationale techniques to capture decision making. The framework for the full-fledged method is described in [17]. The main achievement of this paper is a detailed description of the elicitation of efficiency requirements with the help of the checklists.

Further related work can be found in the communities of requirements engineering, architecture design and performance engineering:

Within requirements engineering, [2] provides a general method for specifying NFRs. It also gives specific advice for how to capture performance requirements with goal graphs. However, the emphasis is on the satisfycing step where means are elicited to achieve performance. In contrast, we focus on using use cases to elicit the customer view. [4] seems to be most similar, since it also combines use cases and NFRs. There are, however, essential differences. While we focus on elicitation of NFRs, Cysneiros and Leite focus on *satisfycing* NFRs. This term was coined in [2] to describe the fact that NFRs are not satisfied, but there are several ways to achieve them. Thus, in [4] use cases and NFRs are elicited separately and then combined to make sure that the use cases satisfice the NFRs. For example, because of an NFR new functionality is added into the use case diagram or into the steps of the use case description. In contrast, we use the use cases to elicit measurable NFRs. The same comment applies to [15] which also relates use cases and NFRs after both have been elicited. Furthermore, they only use high-level quality attributes, such as efficiency.

Another approach developed for security requirements is the elicitation of NFRs by using Mis-Usecases [1]. Those are an excellent means to analyze security threats, but inappropriate to express security requirements explicitly in a measurable way, as discussed in [7]. The approaches presented by [7] and [20] are very similar to our activities of user task level elicitation. Our results show, that not every quality attribute can be expressed by documenting a task. There are NFRs that can only be elicited and expressed using descriptions of system components and flows of events not explicitly related to the quality attribute. We have also learned, that a quality attribute such as efficiency can be understood in many different ways by different stakeholders. The definition of the concept and finding dependencies to other attributes are also part of the elicitation process.

As exemplified by last year´s STRAW workshop, in the architecture community several approaches rely on goal graphs for specifying NFRs and FRs and their dependencies [10][6][8][11][13]. In these approaches, the graph captures the actual FRs and NFRs. In contrast, we only

use the graph to represent dependencies between quality attributes and we place the NFRs in the template.

In the performance community it is emphasized, that performance issues are not suitably integrated in regular software engineering processes [14]. This is attributed to education issues, single-user and small database mindsets and in particular, lack of scientific principles and models. The main emphasis of this community is to create just these models, e.g., queuing models. So, for example [18] also uses use cases in the representation of use case maps in combination with efficiency NFRs. As for [4], however, it is already presupposed that the NFRs have been elicited adequately. The main emphasis is then to create a queing network reflecting the paths of the use case maps and the NFRs.

## 6. Conclusion

In this paper, we have presented an approach for eliciting and documenting efficiency requirements in concert with use cases and a high-level architecture. There are two major innovations. One is the use of a quality model and quality attribute types to capture general knowledge on NFRs, while specific NFRs are captured in a template. The other are detailed checklists on how to elicit NFRs in concert with use cases and architecture. With this approach, we achieve a minimal, complete and focused set of measurable and traceable NFRs. There is first evidence from practitioners that this approach is worthwhile.

While so far we have concentrated on efficiency, we believe that this approach can be generalized to other high-level quality attributes, such as reliability or maintainability. This is because of the use of our meta model and our quality model. We assume that the defined concepts, such as the different types of QAs, metrics, and means can be applied to other high-level quality attributes as well. The main open question is whether the distinction between task and system-oriented QAs also gives helpful guidance for eliciting specific NFRs for other quality attributes. This question is the focus of our current work. After that, we will continue working on the other issues mentioned above, for example, notations that support the identification of dependencies between NFRs.

## 7. Acknowledgements

We thank our colleagues for fruitful discussion within the IESE-Empress-Team. We acknowledge the ITEA project EMPRESS for partly funding our research. Furthermore, we want to thank all partners in the ITEA project EMPRESS that contributed to our research. In particular, we want to thank Ricardo Jimenez Serrano (Siemens) for providing a case study to validate our approach and for taking over the role of the customer.

## References

[1] Alexander, I., “Misuse Case Help To Elicit Nonfunctional Requirements”, IEE CCEJ, 2001,

[2] Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.,“Non-Functional Requirements in Software Engineering”, Kluwer Academic Publishers, 2000

[3] Cockburn A., Writing Effective Use Cases, Addison Wesley 2001

[4] Cysneiros, L.N., Leite, J.C.S.P, “Driving Non-Functional Requirements to Use Cases and Scenarios”, XV Brazilian Symposium on Software Engineering, 2001

[5] Dutoit, A. H., B. Paech, P., “Rationale Management in Software Engineering. In: S.K. Chang (Ed.), “Handbook of Software Engineering and Knowledge Engineering. World Scientific, December 2001.

[6] Egyed, A., Grünbacher, P., Medvidovic, N., refinement and evolution issues in bridging requirements and architecture – the CBSP approach, STRAW 2001

[7] Firesmith, D., “Security Use Cases”, in Journal of Object Technology, vol. 2, no. 3, May-June 2003, pp. 53-64.

[8] Gross, F., Yu, E., “Evolving system architecture to meet changing business goals: an agent and goal-oriented approach”, STRAW 2001

[9] IEEE Recommended Practice for Software Requirements Specifications, IEEE Std. 830-1998

[10] In, H., Boehm, B.W., Rodgers, T., Deutsch, W., "Applying WinWin to Quality Requirements: A Case Study", ICSE 2001, pp. 555-564, 2001

[11] In, H., Kazman, R., Olson, D., From requirements negotiation to software architectural decisions, STRAW 2001

[12] ISO/IEC 9126-1:2001(E), “Software Engineering - Product Quality - Part 1: Quality Model”, 2001

[13] Liu, L., Yu, E., From requirements to architectural design – using goals and scenarios, STRAW 2001

[14] Menasce, D.A., “Software, Performance or Engineering”, Workshop on Software and Performance, pp. 239-242, 2002

[15] Moreira, A., Brito, I., Araújo, J., "A Requirements Model for Quality Attributes", Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, workshop da 1st International Conference on Aspect-Oriented Software Development, University of Twente, Enschede, Holland, 22-26 April , 2002

[16] Paech, B., Dutoit, A., Kerkow, D., von Knethen, A.: „Functional requirements, non-functional requirements and architecture specification cannot be separated – A position paper”, REFSQ 2002

[17] Paech, B., von Knethen, A., Doerr, J., Bayer, J., Kerkow, D., Kolb, R., Trendowicz, A., Punter, T., Dutoit, A., „An experience based approach for integrating archiecture and requirements engineering “, accepted for ICSE-workshop STRAW 2003

[18] Petriu, D., Woodside, M., “Analysing Software Requirements Specifications for Performance”, Workshop on Software and Performance, p.1-9, 2002

[19] Shaw, M., Garlan, D., “Software Architecture – Perspectives on an emerging discipline” Prentice Hall, 1996

[20] Sindre, G. & Opdahl, A., „Eliciting Security Requirements by Misuse Cases, Proc. TOOLS Pacific 2000, pp 120-131, 20-23 November 2000

[21] Trochim, W. M. K., “The Research Methods Knowledge Base”, Atomic Dog Pub Inc., Cincinnati, 2001

[22] von Knethen, A., Paech, B., Houdek, F., Kiediasch, F., "Systematic Requirements Recycling through Abstraction and Traceability", RE 2002

# Post-Release Analysis of Requirements Selection Quality - An Industrial Case Study

Lena Karlsson[1], Björn Regnell[1], Joachim Karlsson[2], Stefan Olsson[2]

[1]*Department of Communication Systems, Lund University, Sweden*
*{lena.karlsson, bjorn.regnell}@telecom.lth.se*

[2]*Focal Point AB, Linköping, Sweden*
*{joachim.karlsson, stefan.olsson}@focalpoint.se*

**Abstract**

*The process of selecting requirements for a release of a software product is challenging as the decision-making is based on uncertain predictions of issues such as market value and development cost. This paper presents a method aimed at supporting software product development organisations in the identification of process improvement proposals to increase requirements selection quality. The method is based on an in-depth analysis of requirements selection decision outcomes after the release has been launched to the market and is in use by customers. The method is validated in a case study involving real requirements and industrial requirements engineering experts. The case study resulted in a number of process improvement areas relevant to the specific organisation and the method was considered promising by the participating experts.*

## 1 Introduction

This paper presents a method for identifying improvement areas of the requirements selection process in a market-driven software product development context. The method is called PARSEQ (Post-release Analysis of Requirements SElection Quality) and is based on retrospective examination of decision-making in release planning, at a time when the consequences of requirements selection decisions are visible. PARSEQ is applied in a case study where the requirements selection for a particular release of a specific software product is analysed and improvement areas that are relevant to the studied software organisation are identified.

PARSEQ is intended to be used by software organisations that operate in a market-driven context, offering software products to many customers on an open market. Market-driven requirements engineering (RE) differs from customer-specific RE in several ways, for example in the characteristics of stakeholders and schedule constraints [15, 17]. Requirements are often invented by the developers as well as elicited from potential customers with different needs [13], and it is common to use a requirements database that is continuously enlarged with new candidate requirements [6, 14]. Commonly, market-driven software developing organisations provide successive releases of the software product and release planning is an essential activity [2, 3]. A major challenge in market-driven RE is to prioritise and select the right set of requirements to be implemented in the next release [13], while avoiding congestion in the selection process [14]. This decision-making is very challenging as it is based on uncertain predictions of the future, while crucial for the product's success on the market [2, 10].

Given issues such as uncertain estimations of requirements market value and cost of development, it can be assumed that some requirements selection decisions are non-optimal, which in turn may lead to software releases with a set of features that are not competitive or satisfy market expectations. It is only afterwards, when the outcome of the development effort and market value is apparent, it is possible to tell with more certainty which decisions were correct and which decisions were less accurate. But by looking at the decision outcome in retrospect, organisations can gain valuable knowledge of how to improve the requirements selection process and increase the chance of market success.

In [16, 4], post-mortem evaluations are discussed in a project management context. An evaluation of the project's performance after it has been completed is useful both for personal and organisational improvement and can be conducted as an open discussion of the strengths and weaknesses of the project plan and execution. Furthermore, much can be learned about organisational efficiency and effectiveness by this kind of evaluation, which offers an insight into the success or failure of the project. The lessons learned can be used when planning forthcoming projects to improve project performance and prevent mistakes. Continuous process improvement is important in the maturity of software development and, in particular, requirements engineering is pointed out as a critical improvement area in a maturing organisation [12]. A recent process improvement study based on analysis of defects in present products is reported in [11].

The PARSEQ method is evaluated in a case study, where requirements selection decisions for an already released software product were revisited by the decision-makers of the specific organisation. The market value and

development cost of the requirements that were candidates for a previous release that was launched 18 months earlier, were re-estimated based on the knowledge gained during the two following releases. The re-estimation resulted in a new priority order, which in turn suggested that some selected requirements should have been postponed and some deferred requirements should have been selected for that release. Each such suspected inappropriate selection was analysed in order to understand the grounds for each decision, which in turn lead to the identification of several areas of process improvements.

The paper is structured as follows. Section 2 presents the PARSEQ methods and its main steps. In Section 3, the case study operation is described and the main results are reported. Section 4 discusses the validity of the findings and the generality of the approach outside the specific case study context. Conclusions and directions of further research are given in Section 5.

## 2 The PARSEQ Method

Retrospective evaluation of software release planning may give a valuable input to the identification of process improvement proposals. In particular, post-release analysis of the consequences of previous decision-making may be a valuable source of information when finding ways to improve the requirements selection process.

The PARSEQ method is based on a systematic analysis of candidate requirements from previous releases. By identifying and analysing a set of root causes to suspected incorrect requirements selection decisions, it is hopefully possible to find relevant improvements that are important when trying to increase the specific organisation's ability to plan successful software releases.

In order to perform the PARSEQ method the following foundation practices are required:

- A database with continuously incoming requirements that are dated at arrival and tagged with a refinement state.
- Methods for estimating each requirements' cost and value. The estimations are saved in the database.
- Multiple releases of the product and the requirements from prior releases are saved in the database.
- Employees who have decision-making experience from prior releases are available.

PARSEQ is divided into 4 steps: requirements sampling, re-estimation of cost and value, root cause analysis, and elicitation of improvements, as shown in Fig 1. The method uses a requirements database as input and assumes that information is available in the database regarding when a requirement is issued and in which release a requirement is implemented. The output of the method is a list of process improvement proposals. Each step in PARSEQ is subsequently described in more detail.

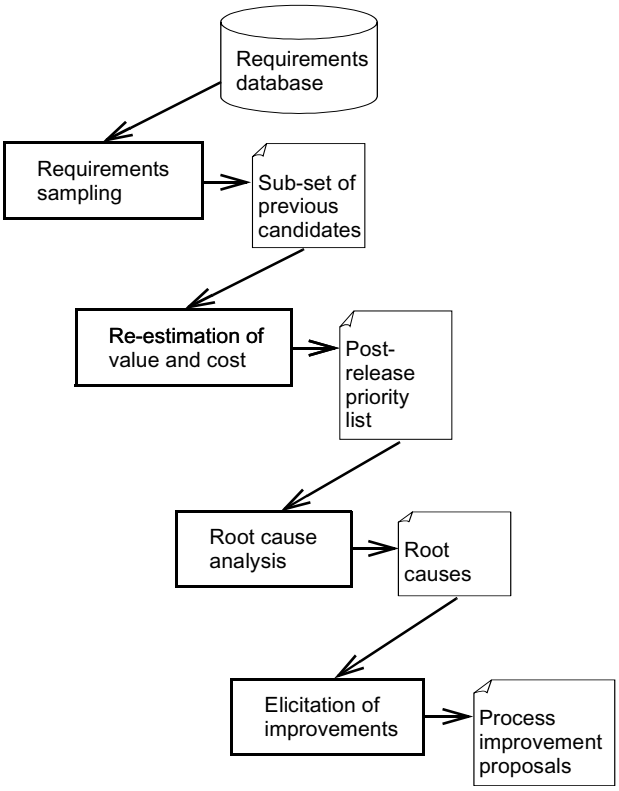



**Fig. 1.** An outline of the activities and products of the PARSEQ method.

**Requirements sampling.** The main input to the post-release analysis is a list of requirements that were candidates for a previous product release that now has been out on the market for a time period long enough to allow for an assessment of the current market value of its implemented requirements. First, such a relevant previous release is selected (subsequently called *reference release*). Secondly, the requirements database is examined and those requirements that were candidates for the reference release are retrieved. The previous candidates are requirements that were suggested and dated prior to the reference release, but were not implemented before the reference release, i.e. the candidate requirements were either implemented in the reference release or in a subsequent release, or they were rejected.

The purpose of the sampling is to compose a reasonably small but representative sub-set of requirements, since the complete database may be too large to investigate in the post-release analysis. The sample should include requirements that were selected for implementation in the reference release as well as postponed or rejected requirements. The requirement set is thereby useful for the analysis as it consists of typical examples of release planning decisions.

The requirements sampling can be performed in a number of ways, such as concentrating on a special market segment or on a difficult part of the product or on particu-

larly difficult decisions. However, if the sample is supposed to represent the whole product and its market, the sample should be as broad as possible. The following types of requirements may then be excluded:

- Very similar requirements, since they do not extend the sample.
- Requirements dated several releases ago, as they may have evolved out of scope.
- Requirements dated recently, since their cost and value are not yet estimated.
- Requirements estimated to have a very long or very short implementation time, as they are atypical and likely to be split or joined.

The output from the requirements sampling is a reasonable amount of requirements, high enough to be representative, yet low enough to allow the following steps of PARSEQ to be completed within reasonable time.

**Re-estimation of value and cost.** The requirement sample is input to the next step of PARSEQ, where a re-estimation of current market value and actual development cost is made in order to find suspected inappropriate decisions that can be further analysed. As the reference release has been out on the market for a while, a new assessment can be made, which applies the knowledge gained after the reference release was launched, which presumably should result in more accurate priorities. The re-estimation is made to find out how the organisation had decided for the reference release, i.e. which requirements that would have been selected, if they knew then what they know now. With todays knowledge, about market expectations and development costs, a different set of requirements may have been selected for implementation in the reference release. If this is not the case, either the organisation has not learned anything since the planning of the reference release, or the market has not changed at all.

The implemented requirements have a known development cost (assuming that outcome of the actual implementation effort is measured for each requirement), but the postponed or rejected requirements need to be re-estimated based on the eventual architectural decisions and the knowledge gained from the actual design of the subsequent releases.

By using, for example, a cost-value prioritisation approach with pairwise comparisons [8, 9], an ordered priority list can be obtained where the requirements with a higher market value and a lower cost of development are sorted in the priority order list before the requirements with a lower market value combined with a higher development cost.

The purpose of the re-estimation is to apply the knowledge that has been gained since the product was released, to discover decisions that would have been made differently today. The discrepancies between the decisions made in the planning of the reference release and the post-release prioritisation are noted and used in the root cause analysis. The output of this step is thus a list of requirements that was given a high post-release priority but were not implemented in the reference release, as well as requirements with a low post-release priority but still implemented in the reference release.

**Root cause analysis.** The purpose of the root cause analysis is to understand on what grounds release-planning decisions are made. By discussing the decisions made in prior releases, it may be possible to create a basis for the elicitation of process improvement proposals.

The output of the re-estimation, i.e. the discrepancies between the post-release prioritisation and what was actually selected for implementation in the reference release, is analysed in order to find root causes to the suspected inappropriate decisions. This analysis is based on a discussion with persons involved in the requirements selection process. The following questions can be used to stimulate the discussion and provoke insights into the reasons behind the decisions:

- Why was the decision made?
- Based on what facts was the decision made?
- What has changed since the decision was made?
- When was the decision made?
- Was it a correct or incorrect decision?

Guided by these questions, categories of decision root causes are developed. Each requirement is mapped to one or several of these categories to illustrate the decision disposition. This mapping of requirements to root cause categories is the main output of this step together with the insights gained from the retrospective reflection.

**Elicitation of improvements.** The outcome of the root cause analysis is used to facilitate the elicitation of improvement proposals. The objective of this last step of PARSEQ is to arrive at a relevant list of high-priority areas of improvement. The intention is to base the discussion on strengths and weaknesses of the requirements selection process and to identify changes to current practice that can be realised. The following questions can assist to keep focus on improvement possibilities:

- How could we have improved the decision-making?
- What would have been needed to make a better decision?
- Which changes to the current practices can be made to improve requirements selection in the future?

The results of PARSEQ can then be used in a situated process improvement programme where process changes are designed, introduced and evaluated. These activities are, however, out of the scope of the presented method.

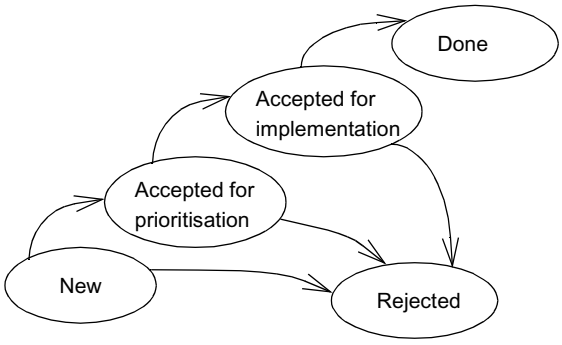



**Fig. 2.** A simplified version of the requirement state model in the database.

## 3 Case Study

PARSEQ was tried out in a case study to investigate its feasibility and gain more knowledge for future research on post-release analysis of requirements selection as a vehicle for process improvement. In the first section of this chapter, the case study site and context is described as well as the tool used in the study. Next, the realisation of the PARSEQ method is described, i.e. how each step of the method was carried out in the case study. Finally, the results from the case study are reported, including a number of improvement proposals.

### 3.1 Background

The case study site is a small-sized organisation developing stand alone software packages. The organisation stores the requirements for the software package in a database that contains already implemented requirements as well as suggestions for new requirements. Each requirement is tagged with a certain state to describe its level of refinement. Examples of states include New, Accepted for prioritisation, Accepted for implementation and Done, see Fig. 2. When a requirement for some reason is not appropriate for the package, its state is set to Rejected. Other states include Clarification needed, Insignificant improvement, Badly documented, Duplicate and Draft.

To analyse the requirements in the database a commercial tool for product management and requirements management, Focal Point[1] was applied. Focal Point has capabilities for eliciting, reviewing, structuring, and prioritising requirements as well as for planning optimal releases that maximise the value for the most important customers in relation to development time and available resources. One prioritisation method in Focal Point is pair-wise comparisons [8]. It is helpful for keeping up concentration and objectivity and Focal Point also provides solutions for reducing the number of comparisons and motivating the priorities. This tool also aids in visualising the decision in a number of different chart types. Due to redundancy of the pair-wise comparisons, the tool also includes capabilities such a consistency check that describes the amount of judgement errors that are made during the prioritisation.

1. For more information see www.focalpoint.se.

### 3.2 Operation

The participating anonymous organisation was given the task to use PARSEQ to reflect on a set of decisions made during prior releases. The case study was executed during a one-day session, with approximately 5 hours of efficient work.

**Requirements sampling.** A release that was launched 18 months ago was selected as reference release, and since then another release has been launched and yet another one is planned to be released in the near future.

The requirements database contains more than 1000 requirements that were issued before the reference release and implemented in either that release or postponed to one of the following ones. Of these requirements, 45 was considered a reasonable number to extract. The requirements were equally allocated over the three releases: A, B and C, i.e. 15 were implemented in the reference release A, 15 in release B and another 15 were planned for release C.

Note that the releases were not equally large in terms of number of requirements, i.e. the samples are not representative. The 15 requirements from release A were selected among 137 requirements, while the releases B and C only consisted of 28 and 26 requirements, respectively as shown in Fig. 3.

The requirements were selected randomly from a range where the ones estimated as having a very high, or very low, development effort had been removed, since they are not considered as representative. Very similar requirements had also been excluded to get an as broad sam-

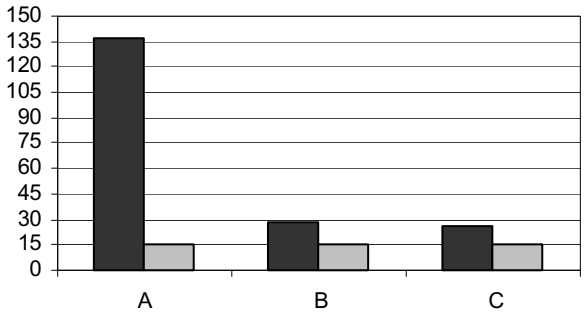



**Fig. 3.** Number of implemented requirements (dark grey) in each release compared to the sample (light grey).

ple as possible, as well as very new ones as development costs had not been estimated.

All market changes, architectural decisions and new knowledge gained during the 18 months between the reference release A and release C could be applied. The selected requirements are all in the states Done or Accepted for implementation; no rejected or postponed requirements were considered in the analysis. The requirements sampling took approximately one hour and was performed by a developer before the session.

**Re-estimation of cost and value.** The re-estimation was performed to find out what requirements the organisation would have selected for release A if they knew then what they know now. With the knowledge gained since the reference release was planned, it is possible that a different set of requirements would have been selected. However, it is important to note that one additional requirement in the release would imply that another one has to be removed, in order to keep the budget and deadline.

The market value was estimated using pair wise comparisons and the cost was estimated in number of hours, based on expert judgement. The following question was used in the pairwise comparison of the candidates to the reference release: "Which of the requirements would, from a market perspective, have been the best choice for release A?". This question was carefully chosen with the objective of enforcing focus on the retrospective nature of the estimation. Thus, the assessment concerned the market value given what is known today, and not whether the decisions made during the reference release were correct or not, given the knowledge available at that time.

The 45 requirements were re-estimated by using the Focal Point tool and pair-wise comparisons to prioritise them based on the selected question. The prioritisation was performed by a marketing person, who has good knowledge of customer demands, guided by a developer, and was attended by the two researchers. When uncertainties or disagreements of a comparison were discovered, the issue was briefly discussed to come to an agreement. The consistency check showed that the prioritisation was carefully performed and only two comparisons had to be revised and changed.

The total time of the prioritisation was just over one hour, in which 70 comparisons were made. The short time is thanks to the algorithms in the tool, which reduces the number of comparisons and points out the inconsistencies among the comparisons [1, 7]. Otherwise, the number of comparisons would have been $n(n-1)/2$, which in this case equals 990.

The development cost of the requirements that were actually implemented was known, while the development cost of the requirements that are planned for a coming release had to be re-estimated. However, it was decided to use the available cost estimations, since the estimates recently had been reviewed and updated.

A bar chart was created in the Focal Point tool to visualise and facilitate analysis of the decisions, see Fig. 4. The grey bars illustrate the requirements implemented in release A, and the white bars represent requirements implemented or planned for release B or C. The prioritisations are performed on a ratio scale and normalised to a relative value in the range between 0 and 1. Thus, it is possible to subtract the cost from the value, getting a *resulting priority*, which is marked by the black arrows in the bar chart [5]. The bars are sorted on their resulting priority from top down. Thus the bar chart shows the ideal order in which requirements should be implemented if only customer value and development costs were to be considered.

Some of the requirements were not identified in release A, but turned out to be important when they later were identified. Furthermore, requirements interdependencies, release themes and architectural choices complicate the situation and thus this ideal order is not the most suitable in reality.

In an ideal case, the requirements at the top of the bar chart would have consisted of requirements from release A. The requirements at the top of the bar chart are estimated as having the highest value and the lowest cost and should therefore be implemented in an as early release as possible. The requirements at the bottom are estimated as having the lowest value and the highest cost and should therefore be implemented in a later release or, in some cases, not at all.

The bar chart illustrates the discrepancies between the two estimation occasions and points out the requirements to discuss.

**Root cause analysis.** The bar chart is used in the Root cause analysis, to find out the rationale for the release-planning decisions. The discussion was attended by three representatives from the organisation: one marketing person and two developers, as well as the two researchers.

The top 15 requirements were scanned to find the ones that were estimated differently in the re-estimation, i.e. the ones that originate from release B or C. These were discussed to answer the main question "Why wasn't this implemented earlier?" and motivations to the decision was stated by the participants. In a similar manner, the 15 requirements at the bottom of the bar chart were investigated, to find the ones that originate from release A and B. These requirements were discussed concerning the question "Why did we implement this so early?". Notes were taken of the stated answers for later categorisation of the release-planning decision root causes.

After the meeting, the researchers classified the stated decision root causes into a total of 19 different categories, inspired by the notes from the meeting. A sheet with the requirements that had been discussed during the root cause analysis was compiled, which the organisation representatives used to classify the requirements. The result from the classification is displayed in Table 1 and Table 2, where 4 categories have been removed as they were not used.

**Elicitation of improvements.** Another purpose of the case study was to capture improvement proposals by en-

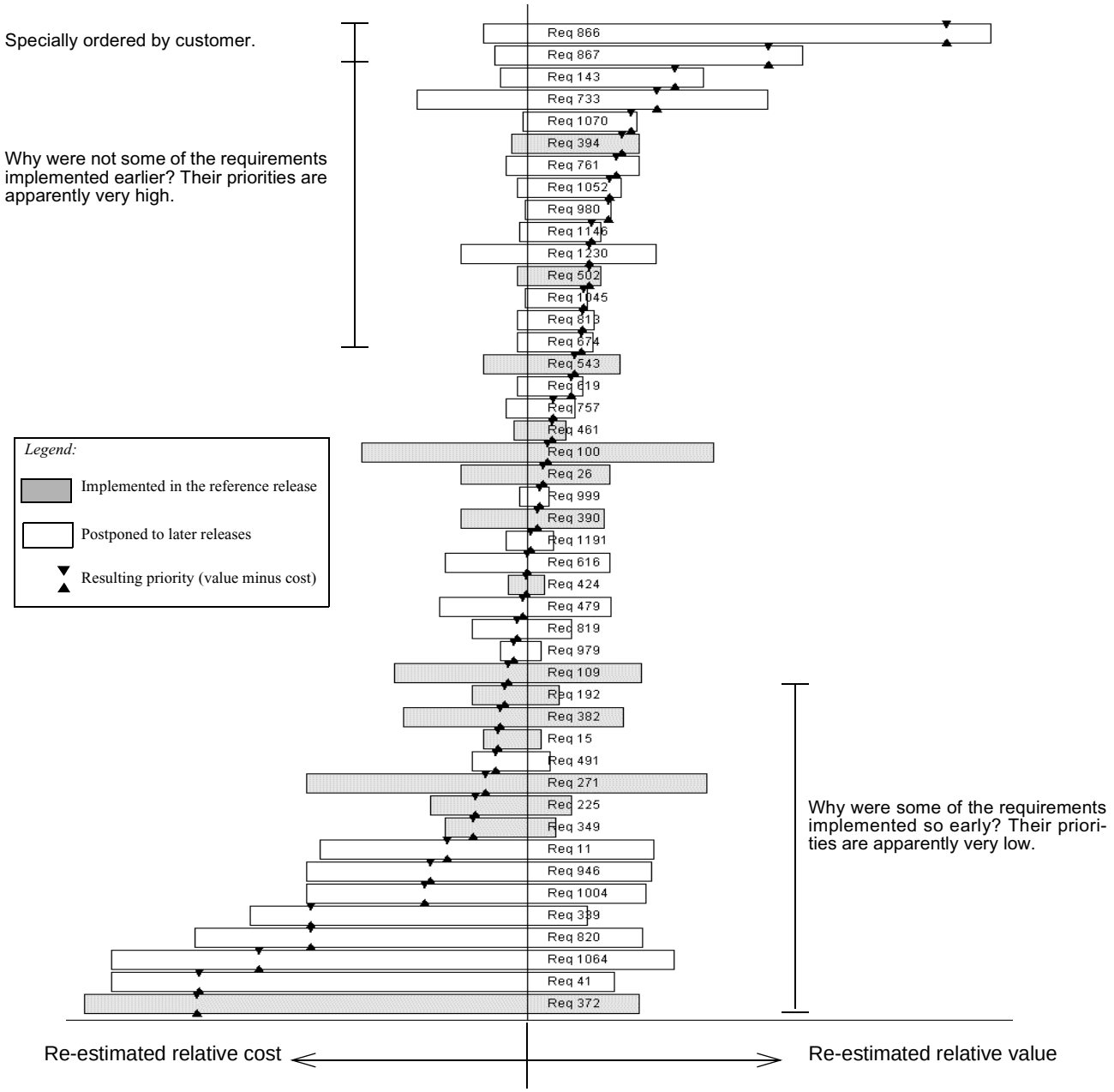



**Fig. 4.** Bar chart from the post-release analysis of the requirements in the database using the Focal Point tool.

couraging the participants to, in connection with each requirement, state some weak areas in need of improvement. This also appeared to be difficult since each decision was dependent on the specific context or situation. Therefore, no list of improvement proposals was compiled at this stage. Instead, more generic improvement proposal areas were elicited by investigating Table 1 and Table 2 and the notes taken from the root cause analysis discussion. This is described below.

### 3.3 Results

The case study showed that it was possible to use the proposed method in practice. The release-planning decisions that were made in prior releases could be categorised and analysed and process improvement areas could be identified. The results indicate that the organisation has gained a lot of knowledge since the planning of the reference release, which is a promising sign of evolution and progress.

**Table 1.** "Why was this requirement implemented so early?"

| | Root Causes | Req 192 | Req 382 | Req 15 | Req 271 | Req 225 | Req 349 | Req 372 | Req 41 |
|---|---|---|---|---|---|---|---|---|---|
| Implem. issues | RC1: Under-estimation of development effort | ■ | ■ | | ■ | | | | |
| | RC2: Part of release theme | | ■ | | ■ | | | ■ | ■ |
| | RC3: A quick fix to provide customers opportunity to give feedback | | ■ | | ■ | | | | ■ |
| Customer issues | RC4: Requirement ordered by a specific customer | | | | | | | | ■ |
| | RC5: Requirement specifically important for a key customer | | | | ■ | | | ■ | ■ |
| | RC6: Over-estimation of customer value | | | ■ | | ■ | ■ | ■ | ■ |
| | RC7: Impressive on a demo | | | | | ■ | | ■ | ■ |
| | RC8: Competitors have it, therefore we must also have it | | | | | | | ■ | |
| | RC9: Competitors do not have it; gives competitive advantage | | | ■ | | ■ | | | ■ |

**Table 2.** "Why was this requirement not implemented earlier?"

| | Root Causes | Req 143 | Req 733 | Req 1070 | Req 761 | Req 1052 | Req 980 | Req 1146 | Req 1045 | Req 813 | Req 674 | Req 866 | Req 867 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Implementation issues | RC10: Over-estimation of development effort | ■ | | | | | | | ■ | ■ | | | |
| | RC11: Insufficient understanding of scale-up effects | | | ■ | | | | ■ | | | | | |
| | RC12: No good design solution available | | ■ | | ■ | | | | | | | | |
| | RC13: Sub-optimal decision based on requirements partitioning | | ■ | | | | | | | | | | |
| | RC14: Only partial implementation in a first increment | | | | ■ | ■ | ■ | | | | ■ | | |
| Cus. issues | RC15: Requirement ordered by a specific customer | | | | | | | | | | | ■ | ■ |

The causes for implementing requirements earlier than necessary are shown in Table 1. Most of the root causes originate from wishing to satisfy customer demands, either one specific customer or the whole market. However, the evaluation showed that the customer value was not as high as expected. On the other hand, it is difficult to measure "good-will" in terms of money, and therefore these decisions may not be essentially wrong. Other root causes of implementing requirements earlier than necessary concern implementation issues, such as incorrect effort estimations, which lead us to believe that estimations ought to be more firmly grounded. Another reason concerns release themes which is a kind of requirements interdependency that is necessary to respect. Developing and releasing small increments of requirements, in order for customers to give feedback early, is a good way of finding out more exactly what customers want, while assigning a low development effort.

As Table 2 shows, the reasons for implementing requirements later than optimal mainly apply to implementation issues. The category complying with the most requirements regards partial implementation in a first increment, which means that it was implemented earlier, but only partially and therefore the requirement remains.

The root cause tables and the material from the discussion were used in the investigation of possible improvement areas. Five areas were found, which could be linked to the root causes, which are described below.

**Trim the division of large requirements into smaller increments.** The manner in which large requirements, affecting several components or having a large implementation effort, are divided into smaller increments can be more thoroughly investigated. The division can be done for several reasons: to get customer feedback at an early stage, to investigate alternative design solutions or to make small incremental improvements of the functionality. Root causes number 3 and 14 deal with requirements developed in increments and the discussions resulted in the idea that the organisation would benefit from an improved increment planning.

**Enhance the overall picture of related requirements.** Some requirements were acknowledged as being related to other requirements due to involving the same feature. These would probably have benefited from creating an overall picture of the release so that all aspects of the specific feature were accounted for. In some cases a feature involved several requirements and after implementing some of them the developers felt content. The related requirements could instead have been designed concurrently in one larger action to avoid sub-optimal solutions. It would also have helped in identifying the most important requirements for that feature. These requirements relations could be taken into consideration more carefully as root cause number 13 describes.

**Additional elicitation effort for usability requirements.** It was recognised that the requirements dealing with the user interface did not fulfil some special customer needs, as described by root cause number 11. The problem concerned scale-up effects and could have been discovered through a more thorough requirements elicitation. Actions to take include building prototypes and asking customers with special user interface needs.

**Improve estimations of market-value of features in competing products.** It seems that many requirements were implemented with the objective of outperforming competitors, as reflected in root cause number 7, 8 and 9. However, looking too much at what competitors have or what may look nice on a prototype or demo may bring less value to the product than expected. The value estimations of the competitors' products may need to be improved.

**Improve estimations of development effort.** Root causes number 1 and 10 concern over- and underestimations of the development effort. Results from an earlier study indicate that the release plan is very dependent on accurate time estimates, since the estimates affect how many of the requirements that are selected [10]. Under-estimation may result in an exceeded deadline and over-estimation may exclude valuable requirements. Improving this area may enhance release-planning and requirements selection quality.

## 4 Discussion

The case study participants found the one-day exercise interesting and instructive. They all agreed that it was valuable to reassess previous releases and reflect on the decisions made. It was during the root cause analysis that the most learning occurred since the discussions between the participants were very fruitful. A set of improvement issues to bear in mind during requirements selection was assessed as valuable for future releases.

Despite the fact that 20 out of 45 requirements were assessed as belonging to the wrong release, there were few decisions that were essentially wrong. Keeping in mind the knowledge available at the time of the reference release, most release-planning decisions were correct, i.e. market opportunities and risks have to be taken, incremental development is applied and only a limited amount of time can be assigned to requirements elicitation and evaluation. However, no matter how successful organisation or product, there are always room for improvements.

There are a number of validity issues to consider in the case study. First of all, the data was not extracted from a representative sample because the releases varied in size. Therefore there are probably many more requirements from the largest release that would be interesting to consider. Since the data only included requirements that were implemented or postponed and no rejected requirements, there would be more decisions to consider in a more thorough evaluation.

The criterion that was used to capture the true value of the requirements appeared to be somewhat difficult to use. Since the development cost was known in most cases, it was difficult for the participants to concentrate on the customer value only, without implicitly taking the cost into account. It was also difficult to, in retrospect, consider the reference release and the value at that particular time without regard of the situation today.

The prioritisation itself is also a source of uncertainty; when not performed thoroughly, the bar chart may not show the appropriate requirements priorities. Nevertheless, the consistency check proved that the prioritisation was performed carefully and few judgment errors were made [8, 9].

Finally, the decision categories that emerged during the root cause analysis may not reflect the typical kinds of decisions. A different set of requirements would probably generate a different set of categories, and therefore these shall not be used by themselves. It is also possible that the categories are formulated vaguely or incorrectly, so that their interpretations differ.

The presented improvement areas are specific to the particular case study organisation and need to be examined in further detail to point out the exact measures to take. However, the participants state that the exercise itself, imposing thought and reflection, may be more fruitful than the particular improvement proposals.

## 5 Conclusions

The presented method for post-release analysis of requirements selection quality, called PARSEQ, was tested in a case study where candidate requirements for a previous release were evaluated in retrospect. The case study demonstrated the feasibility of the method in the context of the specific case and the results from the case study encourage further studies of the method. This may support the hypothesis that the method is generally applicable in the improvement of industrial processes for market-driven requirements engineering in product software development.

The following areas are interesting in further investigations of PARSEQ:

- *Include rejected requirements*. The case study only included requirements that were planned for implementation in the reference release or postponed to coming releases. It would be interested to go through the set of rejected requirements and see if there exist suspected inappropriate rejections, which may be of valuable input to the elicitation of improvements.
- *Selection quality metrics*. Given that the requirements sample is representative to the distribution of appropriate and inappropriate decisions, it may be possible to use PARSEQ to provide numerical estimations of the selection quality in terms of fractions of "good" and "bad" decisions.
- *Connect improvement proposals and root-causes*. It is fairly easy to extract root-causes from the discussion on misjudged requirements. However, advancing from root-causes to improvement proposals appeared more difficult. More investigation into support for finding improvement proposals is needed.
- *Generalisation of root cause categories*. If many case studies applying PARSEQ are carried out in various contexts, it may be possible to derive a complete and generally applicable set of root cause categories that are common reasons for inappropriate decisions. This knowledge may be very valuable in the research of requirements engineering methods in the product software domain.

## Acknowledgements

The authors would like to thank the participating anonymous organisation for the industrial requirements engineering expertise and confidential data, without which this study would not have been possible. We would also like to thank Magnus Höglund at Focal Point for contributing to this work with his valuable time and knowledge.

## References

[1] Carmone, F.J., Kara, A., Zanakis, S.H., "A Monte Carlo Investigation of Incomplete Pairwise Comparison Matrices in AHP", *European Journal of Operational Research*, Vol 102, pp. 538-553, 1997.

[2] Carlshamre, P., Regnell, B., "Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes", *IEEE International. Workshop on the Requirements Engineering Process* (REP'2000), Greenwich, UK, September 2000.

[3] Carshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J., "An Industrial Survey of Requirements Interdependencies in Software Release Planning", *IEEE International Conference on Requirements Engineering* (RE'01), pp. 84-91, 2001.

[4] Cleland, D.I., *Project Management*, McGraw-Hill, 1995.

[5] Fenton, N.E., *Software Metrics - A Rigorous Approach*, Chapman & Hall, 1994.

[6] Higgins, S. A., de Laat, M., Gieles, P. M. C., Guerts, E. M., "Managing Product Requirements for Medical IT Products", *IEEE International Conference on Requirements Engineering* (RE'02), pp. 341-349, 2002.

[7] Karlsson, J., Olsson, S., Ryan, K., "Improved Practical Support for Large-scale Requirements Prioritising", *Requirements Engineering*,Vol 2, pp. 51-60, 1997.

[8] Karlsson, J., Ryan, K., "A Cost-Value Approach for Prioritizing Requirements", *IEEE Software*, Sept/Oct 1997, pp. 67-74.

[9] Karlsson, J., Wohlin, C., Regnell, B. "An Evaluation of Methods for Prioritizing Software Requirements", *Information and Software Technology*, Vol 39(14-15): 939-947, 1998.

[10] Karlsson, L., Dahlstedt, Å.G., Natt och Dag, J., Regnell, B., Persson, A., "Challenges in Market-Driven Requirements Engineering - an Industrial Interview Study", *International Workshop on Requirements Engineering: Foundations of Software Quality* (REFSQ'02), Essen, Germany, September 2002.

[11] Lauesen, S., Vinter, O., "Preventing Requirements Defects: An Experiment in Process Improvement", *Requirements Engineering* Vol 6:37-50, 2001.

[12] Paulk, M. C., Weber, C. V., Curtis, B., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley, 1995.

[13] Potts, C., "Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software", *Proceedings of the Second IEEE International Symposium on Requirements Engineering* (RE'95), pp. 128-30, 1995.

[14] Regnell, B., Beremark, P., Eklundh, O., "A Market-Driven Requirements Engineering Process - Results from an Industrial Process Improvement Programme", *Requirements Engineering*, 3:121-129, 1998.

[15] Sawyer, P., "Packaged Software: Challenges for RE", *Proc. 6th Int. Workshop on Requirements Engineering: Foundations of Software Quality* (REFSQ'00), Stockholm, Sweden, pp 137-142, June 2000.

[16] Ulrich K.T., Eppinger, S.D., *Product Design and Development*, McGraw-Hill, 2000.

[17] Yeh, A., "Requirements Engineering Support Technique (REQUEST): A Market Driven Requirements Management Process", *IEEE Second Symposium of Quality Software Development Tools*, pp. 211-223, New Orleans USA, May 1992.

# Modelling Dependencies between Variation Points in Use Case Diagrams

Stan Bühne, Günter Halmans, Klaus Pohl
*Institute for Computer Science and Business Information Systems (ICB)*
*Software Systems Engineering*
*University of Duisburg-Essen*
*45117 Essen; Germany*
*E-mail: {buehne, halmans, pohl}@sse.uni-essen.de*

## Abstract

*Software product family variability facilitates the constructive and pro-active reuse of assets during the development of software applications. The variability is typically represented by variation points, the variants and their interdependencies. Those variation points and their variants have to be considered when defining the requirements for the applications of the software product family. To facilitate the communication of the variability to the customer, an extension to UML-use case diagrams has been proposed in [9].*
*In this paper we identify common dependency types used within the feature modelling community to express interdependencies between variation points and variants. We then propose a differentiation of those interdependencies to be used to reduce complexity and to facilitate selective retrieval of interdependencies between variation points and the variants. Finally, we extend the notations proposed in [9] for representing interdependencies between variation points and variants in use case diagrams and use a simple example to illustrate those extensions.*

## 1 Introduction

Software product families facilitate pro-active and constructive reuse of software product assets and thereby reduce the development costs of customer applications. The development of product families is characterized by two processes (cf. Figure 1). During domain engineering the common assets are realized and the variability of the product family is defined. During application engineering the product family variability is exploited to define and implement different products by reusing the shared assets defined during domain engineering [21].

Obviously, making the costumer aware of the product family capabilities and variability is a key factor for the successful reuse of the product family assets. The communication of the product family variability to the customer is thus essential. Communicating the capability and the variability to the customer is the main difference between requirements engineering for single software products and requirements engineering during domain engineering.

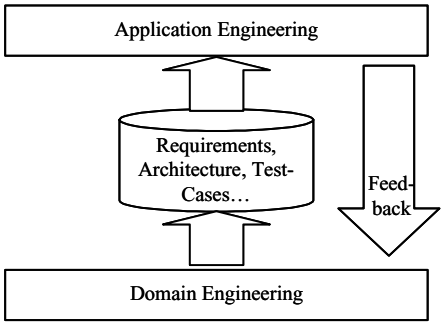



Figure 1: Domain and Application Engineering in Product Family Development

Most research contributions on product family variability focus on the architectural level [1], [2], [20]. For example, they deal with aspects like the realization of variability on a technical level and technical binding of the variation points. From the customer perspective those aspects of variability are not so important. In contrast, the customer is rather interested in the essential aspects of the variability, i.e. the customer wants to know how the variability contributes to his needs. Thus the customer is e.g. interested in the functional and quality aspects of variability, i.e. which different payment methods are available for an internet-based booking system. As argued in [9] one should thus distinguish between *essential* and *technical* variability.

Essential variability comprises all types of variability from the customer viewpoint. This includes of course all functional aspects. For example, the provider of a navigation system provides two variants for determining the address of the next destination, via a menu or via voice input. The customer would be interested in having the choice between the different "input" methods, whereas the engineer is more interested in implementation aspects, e.g. how the voice input is technically integrated in the navigation systems.

Obviously, when defining the requirements for a customer specific application during application engineering, at least the essential variability aspects have to be communicated to the customer. In other words, the variation points and variants have to be considered when defining the requirements for the applications of the software product family. To facilitate the communication of the essential variability aspects to the customer we proposed an extension to UML-use case diagrams (cf. [9]).

Adding to the complexity, besides the variants and the variation points also the interdependencies between variants and variation points have to be considered. For example, a variant can exclude another one, e.g. when you are buying a roof-less car, you cannot choose a sun-roof. Or, a variant might require the choice of another variant, e.g. if you choose an engine with high power, you have to take bigger wheels. Obviously, only if the customer is aware of those interdependencies he is able to recognize the consequences of his selection. Since there are generally many of such interdependencies within a product family, dealing with the complexity of those interrelations is a big challenge.

As the major step forward regarding our work in [9] in this paper we focus on how those interdependencies between variants and variation points can be communicated to the customers. We propose a classification as a mean to deal with the complexity of the interdependencies and suggest a representation using common requirements engineering techniques for supporting the communication.

In section 2 we briefly describe the extensions to use case diagrams required for representing variation points and variants.

In section 3 we give an overview on the state of the art of considering interdependencies of product family variability at the requirement level.

To be able to deal with the complexity of those interdependencies we suggest to consider the origin of the dependencies and to clearly indicate derived interdependencies in section 4.

In section 5 we outline how those interdependencies can be visualized in UML Use Case Diagrams and sketch potential tool support.

In section 6 we summarize the main contribution of this paper and provide an outlook on future work.

## 2 Representing Product Family Variability in Use Case Diagrams

To facilitate the communication of variation points and their variants to the user, both the variation points and the variations must be explicitly represented. In [9] we suggested to apply Use Cases for communicating the variability of the product family to the customer and showed why standard UML notations are not suitable for this purpose. Therefore, we proposed extensions to Use Case Diagrams, which enables the explicit representation of variation points and their variants (see [9] for details). In the following we briefly introduce these extensions.

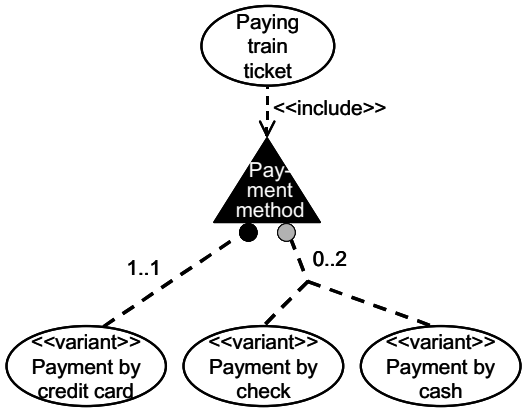



Figure 2: Explicit Representation of Variation Points in Use Case Diagrams

- ***Variation point***: For making a variation point visible in use case diagrams we represent a variation point as a triangle. The variation point is itself included by another use case (cf. Figure 2)
- ***Variant***: To make variant use cases explicit we use the stereotype <<variant>> (cf. Figure 2, e.g. the variant *payment by credit card*)
- ***Cardinality of the relationship between variation point and variant***: Regarding the relationship between variants and variation points one has to consider, that a variant can be mandatory for a variation point A and optional for a variation point B. To express these kinds of relationships we represent the relationship between variants and variation points by a dashed line and a circle on the variation point side. Moreover, we define cardinality for the relationship to express if a variant is mandatory or optional regarding the specific variation point. More general the cardinality expresses, how much of the related variants *must* and how much of the related variants *could* be selected. In Figure 2 for example the variant *payment by credit card* is mandatory, which is represented by the cardinality 1..1.
- ***Mandatory and optional variation points***: To make clear if one of the related variants according to a specific variation point has to be chosen we distinguish between mandatory and optional variation points. A variation point is mandatory, if at least one of the related variants has to be selected. Mandatory variation points are represented by a black filled

triangle (like the one in Figure 2) whereas optional variation points are represented by a light-grey triangle.

With the defined extensions variation points and variants are explicitly visible in the use case diagram. Furthermore the cardinality of the relationship between variant and variation point is documented. It is now explicit, whether a variation point is mandatory or not and one is able to recognize how much variants from a set of variants have to be selected.

The new notations do not involve the possibility to represent dependencies, e.g. between variants related to another variation point. Furthermore, so far it is not possible to express dependencies between variation points and to represent the type of dependencies. These issues will be our focus in the next sections.

## 3 Dependencies in Feature Modelling

The need to consider the interdependencies between different variation points at the requirements level has been identified already by the feature modelling community. In this section, we provide an overview on the state of the art of representing dependencies between features. In Table 1 we first give an overview of dependency-types used in feature modelling approaches.

| Dependency Type | Description | Used in |
|---|---|---|
| composed of | The composed of dependency is used when a parent feature is consisting of a set of child features (e.g. mobile is composed of voice-transfer, data-transfer, shot messages, etc.). | FORM [15]<br>FOPLE [16] |
| implemented by | The implemented by dependency is used to describe that one feature is needed to implement another feature (e.g. UMTS needs specific transfer protocols). | FORM [15]<br>FOPLE [16] |
| generalization | The generalization / specialization dependency is used to generalize or specialize features (e.g. data transfer can be specialized as wap, UMTS, fax, etc.). | FORM [15]<br>FOPLE [16] |
| refinement | The refinement link is used to structure features in the same way as the composed of dependency | FODA [14]<br>FeatuRESB<br>Riebisch |
| requires | The requires dependency is used to describe if one feature needs another (e.g. satellite-navigation requires GSP-signal) | FODA<br>FeatuRESB [12]<br>Riebisch [19] |
| (mutual) exclusive | The exclusive (or exclude) dependency is used when one feature conflicts with another (e.g. the textual cellular display excludes mobile photography) | FODA [14]<br>FeatuRSEB [12]<br>Riebisch [19] |
| hints | The hint dependency is a strategic dependency, and is used to express that the choice of another feature increases the system usage (e.g. mp3 player for mobile) | Riebisch [19] |
| mathematical | The mathematical dependency describes the relative impact from one feature to another. | Riebisch [19] |

Table 1: Dependency-Types used in Feature Modelling

FODA [14] was one of the first feature-oriented approaches. For the representation of dependencies FODA uses refinement-links to partition commonalities and variable aspects among features and semantically describes constraints for requires and mutual exclusive dependencies. FOPLE [16] employs composition, generalization, and implementation links to structure variable aspects among features. The FORM – Method [15] supports the same kind of links as the FOPLE approach. The FeatuRSEB approach [8] is a combination of FODA and Jacobsons RSEB [12], and therefore uses the same dependencies as in FODA to describe variability. Riebisch et al. [19] use refine links to express variability aspects and further differentiate between hard constraints, as requires and mutual exclusive, and soft constraints as hints and mathematical relations to describe dependencies between features.

The various dependency types proposed in the feature modelling literature can be characterized by two main categories:

- *Realization-dependencies*: This category subsumes all dependencies, which deal with realization aspects of a feature. Such dependency types are used to define the restriction in the choice of the variants associated to one variation point. E.g. the representation of a

navigation-system can be by voice, graphics, and or text whereas the representation is either colored or black and white. These dependencies show in which different ways an aspect can be realized.

- *Constraint-dependencies*: This category subsumes all other feature interdependencies. For example, one feature can exclude another, or one feature requires another feature like the choice of colored-graphical representation within a navigation-system requires a colour display and geographical information to fulfil this demand.

# 4 Considering the "Why" of Product Family Variability Interdependencies

A prerequisite to find a suitable representation for product family variability dependencies is to know why these dependencies exist.

In this section we elaborate on different reasons (sources), which cause a dependency concerning the product family variability. Based on dependency types used in feature modelling (see last section) we first define a set of dependency types to be used to represent interdependencies between variation points and variants (Section 4.1). In Section 4.2 we discuss the various types of possible interdependencies between variation points and/or variations. In Section 4.3 we elaborate on the reasons why a certain type of dependency is introduced. We thereby distinguish between so-called core-dependencies and derived-dependencies. In Section 0 we differentiate between several types of derived-dependencies.

## 4.1 Dependency types

Based on the dependency types used in feature modelling and our experience with modelling dependencies concerning product family variability, we suggest differentiating at least between the following four types of interdependencies:

- *requires-dependency*: describes that the binding of one variant implies the need of another variant (required variant). For example if one wants to lock up his car via remote control this variant requires a centralized door locking.
- *exclusive-dependency*: describes that the binding of one variant excludes the selection of another variant (excluded variant) – means that only one of those variants can be selected. As mentioned earlier the choice to get a cabriole for example excludes the variant "sunroof".
- *hints-dependency*: In analogy to the approach of Riebisch et al. [19] this dependency type comprehends dependencies where the binding of one variant has some positive influence on another variant. For example a T-DSL-connection might have a positive influence on choosing the online shopping variant (in contrast to a modem-connection).
- *hinders-dependency:* describes that the binding of one variant has some negative influence on another variant. If one chooses the mobile phone for using the internet this might have a negative influence on online shopping because of the little mobile phone displays.

Note, that the requires dependencies and the hint dependencies are uni-directional, whereas the exclusive-dependency and the hinders dependencies are bi-directional. For example, if a variant A requires a variant B it does not imply that the variant B requires the variant A. In contrast, an exclusive-dependency between variant A and variant B represents that if variant A is chosen than variant B can not be chosen and vice versa.

| **Dependency type** | **Direction** |
|---|---|
| Requires-dependency | Uni-directional |
| Exclusive-dependency | Bi-directional |
| Hint-dependency | Uni-directional |
| Hinders-dependency | Bi-directional |

Table 2: The Direction of Dependency Types

## 4.2 Dependencies between Variants and/or Variation Points

The following three principle interrelations between variants and/or variation point exists:

- *Dependency between variant and variation point*. As described in [2] and [9] a variant is obviously related to one or more variation points. For example the variant *paying a ticket by credit card* is a variant of the variation point *payment methods for train tickets* (in addition to the variants *payment by check and payment by cash*). A variant can be related to more than one variation points (and has thus more than one dependencies to a variation point). The variant *paying a ticket by credit card* for example could also be a variant of the variation point *payment method for merchandising articles* (of the train company).
- *Dependency between variant and variant*: Consider the variant *paying by credit card.* If this variant will be selected for paying via internet one has also to select the specific variant for encrypting the credit card information. In this category one has to consider two different cases: (1) the depending variants are related to one variation point and (2) the depending variants are related to different variation points (see the example mentioned above).
- *Dependency between variation point and variation point*: For example, if a mobile phone product family provides the choice to use the mobile phone in different countries (variation point *countries*) it also

has to provide the choice between different protocols (variation point *protocol*).

The dependencies described in this Section are orthogonal to the dependency types in Section 4.1. For example, the dependencies between a variant and a variation point can be a require dependency or an exclusive dependency and a require dependency can occur in all three categories, i.e. between variation points, between variants and between a variant and a variation point.

### 4.3 Core Dependencies

Each dependency is of course introduced for a particular reason. For being able to understand the reason why a dependency was introduced in the first place, we suggest to add a textual attribute to each dependency for describing the reasons.

We thereby distinguish between two types of dependencies with respect to the product family variability:

- *core-dependencies,* which subsume all dependencies introduced due to the context of the system, i.e. the system context enforces some constraints on the system represented as core-dependencies.
- *Derived-dependencies,* which subsume all dependencies derived from the core-dependencies. One reason why a derived-dependency exists lies in the refinement of features and functions. But there are others (see section 4.3).

For the core-dependencies we further suggest to classify the influence the context of the systems has on the dependencies existing between features (variation points, variants) according to three main views (see Figure 3; cf. [11], [18] for a detail description of those views/worlds):

- *Usage view*: from the usage point of view the system under consideration has to support needs respectively solve problems of the users (whereby a user can be the end-user, the provider or administrator of the system). Thus the usage view deals with all functional demands to the system – in other words it deals with the whole functionality the system should provide to any user.
- *Domain view*: from the domain point of view the system under consideration needs to map the corresponding real world domain to a system. That means all necessary information of a specific scope (domain) has to be represented in a system. This data is driven by relevant subject properties (e.g. name, address, etc of a vendee), domain specific standards (e.g. financial standing) or driven by laws (e.g. full name and address of vendor).

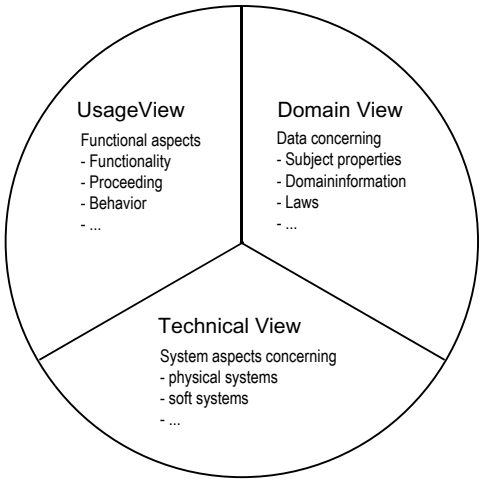



Figure 3: Contextual Views of a System

- *System/technical view*: from the technical point of view the system under consideration depends on physical systems, means given and requested hardware (e.g. architecture, communication interfaces) and soft systems means given and requested software (e.g. computer- and user-interfaces). The term physical systems enfold all environmental systems the product has to deal with (e.g. signal of GPS-Satellite), as well as physical restrictions (e.g. size of display, number of buttons, etc.).

Those views are obviously interrelated, e.g. facts about the domain are represented by systems to support the end-user with needed functionality.

| **Source** | **Category** | **Example** |
|---|---|---|
| Usage View | Dependency regarding the functionality of a system | the payment method via credit card and internet depends on the security functionality of the web application |
| Domain View | Dependency regarding the data of the system | the choice to represent a signature may be depend on the representation of additional authentication information |
| Technical View | Dependency regarding the IT of a system | a chosen graphical user interface (e.g. mobile phone) affects the needed implementation language (e.g. WAP). |

Table 3: Dependency Categories of Core Dependencies

Table 3 summarizes the three different contextual sources for a dependency: the domain view focuses on the data (information), the usage view focuses on functional aspects of the system, and the technical view focuses on computer systems, networks, telecommunication systems, etc. the system run on or interacts with.

## 4.4 Derived Dependencies

Many dependencies concerning the product family variability are not introduced due to contextual reasons, i.e. there are many dependencies, which are not core-dependencies, but derived from the core dependencies or from so-called high-level features.

### *4.4.1 Dependencies Derived from High-Level-Features*

A main reason for introducing variation points and/or variants can be the existence of a so-called high-level features. Examples for such high-level features are "sports-edition" in the car industry or "internet-shop" in a procurement system. For example, when buying a car you might be able to select the "sports-edition". By making this decision, you select a lot of "details", i.e. you bind a whole set of variation points with a predefined set of variants defined for this high-level feature. For example, you get a strong engine, an aluminium interior (instruments, knops, etc.), special tires etc.

Thus, a high level feature often predefines the "binding" of a whole set of variation points to specific variants. Moreover, a certain variant or even a whole variation point might only exist due to the high-level feature.

We thus suggest to specify explicitly that a variation point and/or a variant were introduced due to a high level feature. This can be achieved by adding a dependency link stating the high-level feature was responsible for the introduction of the variant or variation point and by adding an attribute to each dependency link between variants/variation points introduced due to high-level feature.

### *4.4.2 Transitive -Dependencies*

Transitive dependencies are derived from other dependencies. A functionality required from the usage perspective (usage view) might require specific information/data, which in turn require some it-solution. For example, to support the evaluation of the actual stock market situation, one requires the actual stock exchange rates which in turn require a IT-solution providing an update of the stock exchange data every 10 seconds. Figure 4 illustrates this situation. There are two core-dependencies (between the functionality and the data, and between the data and the IT) and one indirect dependency, specifically between the functionality and the IT. The latter one we call (derived) *transitive-dependency.*

To make the reasons traceable, we argue that the derivation of transitive-dependencies should be visible to the product family engineer and the customer. We thus indicate the fact that a dependency was derived due to other core-dependencies by a "transitive-dependency" attribute, which relates the derived dependency to the core-dependencies it was derived from.

Knowing where a dependency was derived from (e.g. to retrieve the sources for the derivation) enables an engineer to understand why a specific dependency exists. This information facilitates the selection of the variants during application engineering, and – even more important – is very helpful for dealing with evolution of the product family itself (e.g. when adapting a specific variant for a customer or even changing a variant for the whole product family).

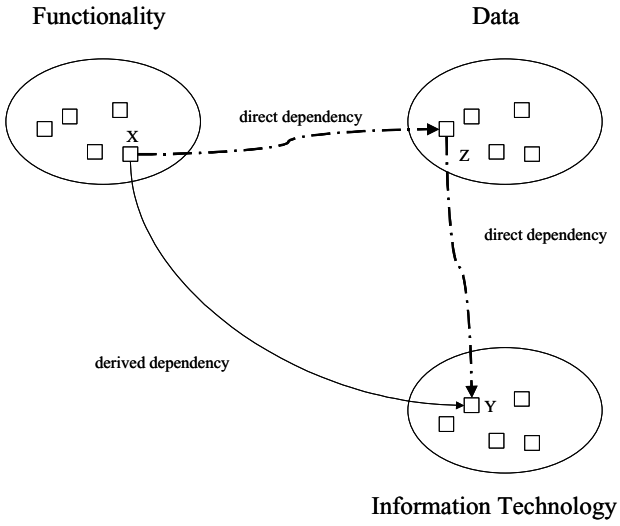



Figure 4: Derived Transitive Dependencies

## 4.5 Classification of Dependencies: Summary

Differentiating between derived and core dependencies empowers the user to focus on a special type of dependencies at a time and thus decreases the complexity of dependencies within product family variability significantly (see section 5 for examples). This differentiation is thus essential for facilitating the communication of product family dependencies to customers. In Figure 5 the necessary attributes for the dependency-links are depicted.

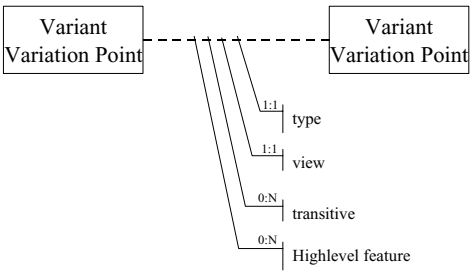



Figure 5: Dependency Attributes

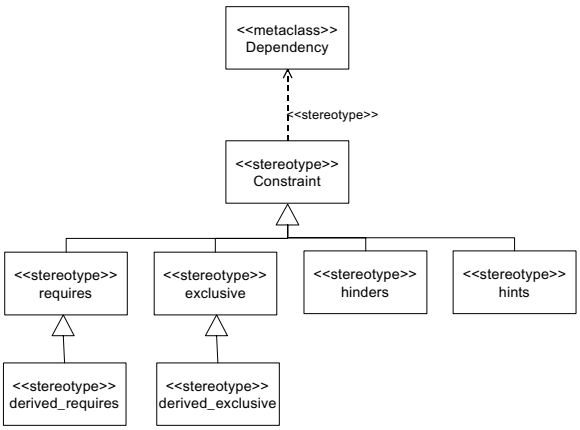



Figure 6: Dependency-Stereotypes

## 5 Representation of Dependencies in Use Case Diagrams

In Section 4 we described the different sources of dependencies within product family variability. In this section we suggest a representation of the different dependencies to support the communication to customers. Motivated by the positive experiences with the representation of variability aspects by use case diagrams (see [9]) we suggest completing this representation according to dependency aspects. The intention of use case diagrams is to model the interaction of users and/or other systems with the system under consideration. Therefore, in the following section we concentrate on the functional aspects according to the product family dependencies.

### 5.1 Representing the Four Types of Dependencies

To express dependencies in use case diagrams, we need to extend the notation among the mentioned dependency-types. The UML [17] allows extending the language by refining consisting elements by stereotyping. To express the dependencies we created a <<requires>>, <<exclusive>>, <<hints>>, and <<hinders>> stereotype. To express the derived dependencies we also specialized the requires and exclusive dependency type to <<derived_requires>> and <<derived_exclusive>> (see Figure 6).

The <<requires>> type is expressed by a dotted line with an arc to the element that is required by the other. The <<exclusive>> type is expressed by a dotted line with arcs on each end that illustrate the exclusive relation between these elements. The <<hints>> type is expressed by a pointed line with an arc to the useful element. The <<hinders>> type is illustrated by a pointed line with an arc on each end, that illustrates the hindering relation between those elements. (Figure 7)

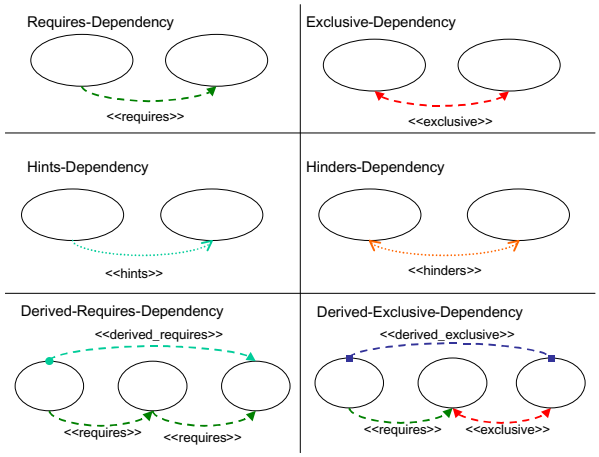



Figure 7: Dependency-Types

### 5.2 Representing the Dependencies According Variant and/or Variation Points

**Dependencies between Variants of the same Variation Point**

Dependencies among variants of the same variation point, like the selection of one variant requires another, have to be expressed by dependency-links. For example,

the selection of the variant *viewpoints* <<hints>> the selection of *city maps* (Figure 8).

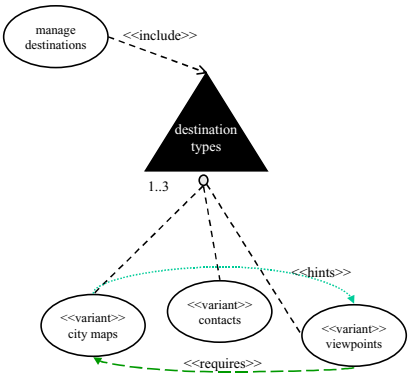



Figure 8: Dependencies between Variants of one Variation Point

Beside the represented requires- and hints-dependency, also exclusive- and hinders-dependencies have to be represented.

**Dependencies between Variants of different Variation Points**

As already mentioned before, dependencies between variants of different variation points also have to be handled. E.g. the selection of a specific variant *"route selection by viewpoint"* causes a requires-dependency to the variant *"manage viewpoints"* of another variation point (Figure 9).

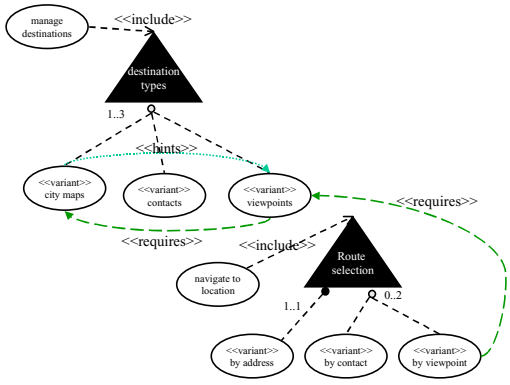



Figure 9: Dependencies between Variants of different Variation Points

**Dependencies between Variation Point and Variation Point**

The selection of one variation point can cause dependencies to other variation points. For example the selection of a movie-player requires the selection of an audio-player. In Figure 10 we illustrate an example where the selection of a navigation system *"navigate to location"* requires the use case variation point *"manage destinations"*.

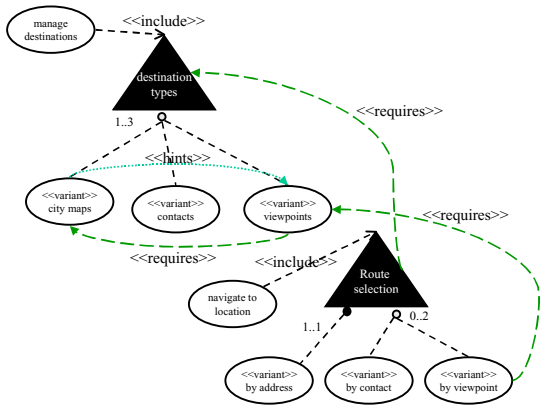



Figure 10: Dependencies between Variation Points

**Dependencies between Variants and Variation Points**

The selection of a variant can cause dependencies to other variation points (without a restriction to the selected variants). This means that when selecting a variant another variation point is required or even excluded, without consideration of the variants. Figure 11 illustrates this case: the variant "*manage contacts*" requires the variation point "*admin categories*", without specifying any variants of the required variation point.

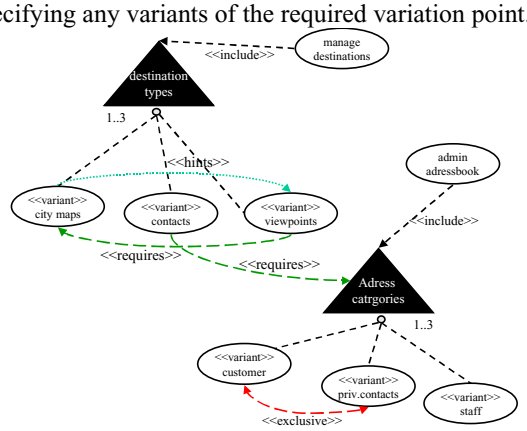



Figure 11: Dependencies between Variants and Variation Points

## 5.3 Representing the Core of Dependencies

In 4.3 we introduced origins of dependencies and differentiated among three categories: usage, domain, and technology. The example (Figure 12) shows the origin of the requires-dependency between *"viewpoints"* and *"city maps"*. The depicted requires-dependency has its origin in the domain, because to manage the viewpoints of a city, one needs city maps to locate those viewpoints.

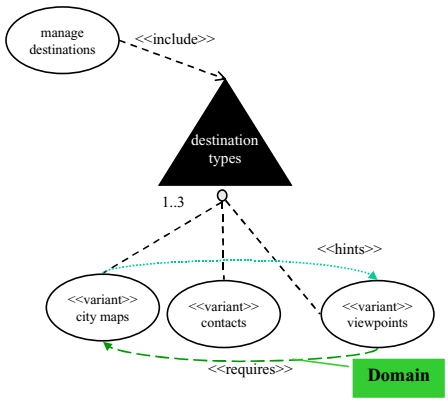



Figure 12: Dependencies with Core Description

The origins of the dependency-links will be illustrated as additional attribute to the link-type. By an adequate tool-support one can generate views that express which use case dependencies have their origin in the domain (i.e. in data), in information technology, or in the usability of a system. As a result of that, one has the opportunity to discover which use cases change when for example the domain, or information technology changes.

## 5.4 Representing derived dependencies

### *5.4.1 Derived Dependencies form High Level Selections*

When selecting a high level feature the binding of a whole set of variants or variation points can be predefined and/or the user has to make a selection for several variation points. For example, if the user selects for a shop system the variant *online system*, he can choose between payment and transaction methods, and delivery different delivery services. But if he selects the *local store* variant, he is not able to choose different transaction methods. The selection he can make therefore depends significantly on the choice he made on the higher level. The example in Figure 13 shows that as a result of a high level selection ("online shop") new (derived) dependencies appear that have to be considered.

In the example, both requires-dependencies are the reason for the "new" derived dependency.

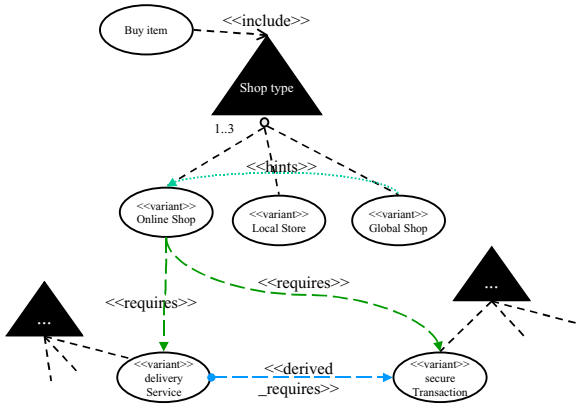



Figure 13: Derived Dependencies Evoked by High Level Selections

### *5.4.2 Transitive Dependencies*

We discussed the fact that a dependency is transitive (derived from other dependencies) in 4.4.2. As depicted in Figure 14, UC_1 requires UC_2 and UC_2 requires UC_3. Therefore it is obvious that UC_1 indirect requires UC_3. The representation of derived dependencies in use case diagrams generates a huge complexity of those diagrams. Figure Figure 14 illustrates the complexity on a simple but abstract example..

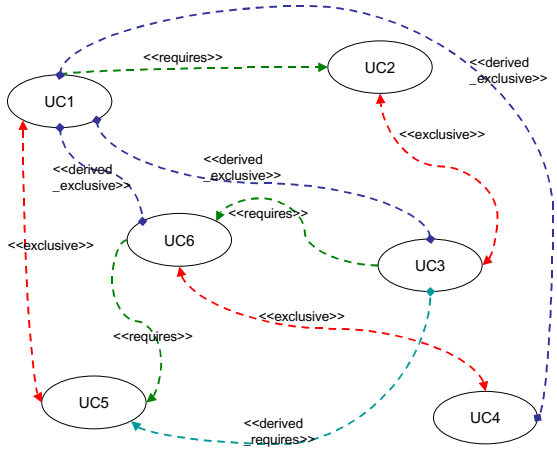



Figure 14: Derived Transitive Dependencies

Figure 15 illustrates the overall complexity of dependencies among use case variants and variation points. Even if only three variation points are illustrated as in this example the result is quite complex. When we think of product families one has to deal with hundreds of variation points. To handle this complexity assistance by tools is necessary.

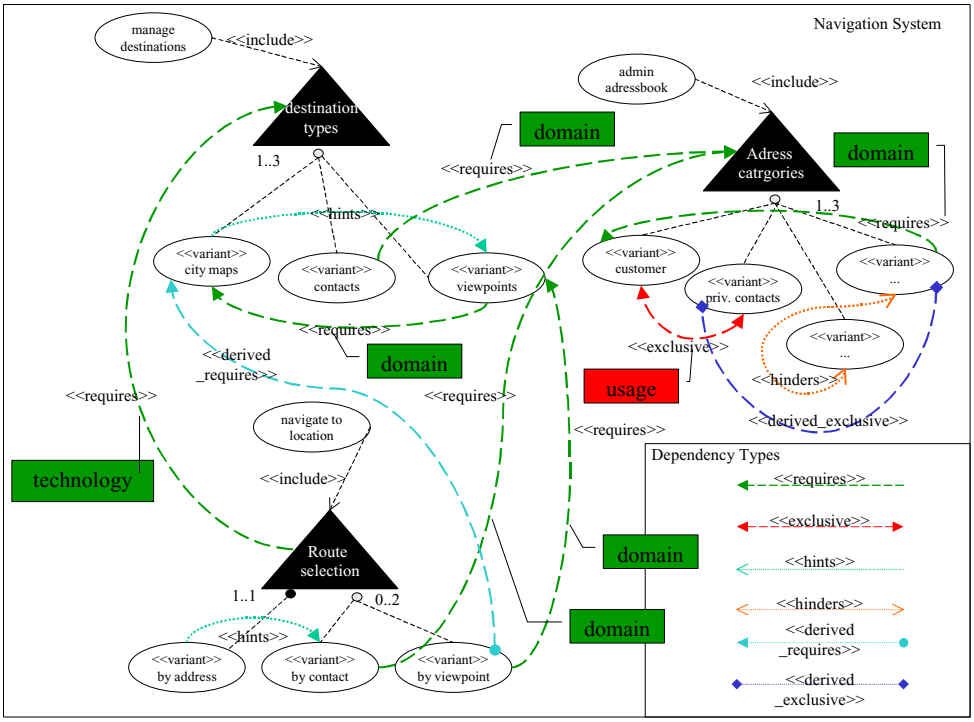



Figure 15: Overall Dependency View

### 5.5 Tool Support

Considering the representations of the different dependencies (sources and types) one has to recognize the enormous complexity of product family variability dependencies. Thus for the communication of these dependencies to customers it is essential to reduce this complexity. Thus the communication to customers leads to the following two problems:

- On the one hand, one has to avoid that the use case diagrams are overloaded when representing the interdependencies.
- On the other hand, the customer has to recognize the important ones. Only if he is aware of the important dependencies he will be able to recognize the consequences of his selections.

The problems can be solved by tool support and by using the different categories of dependencies outlined in Section 4. The tool should fulfil the following requirements:

- In the standard mode the tool only should visualize the four dependency types described in Section 4.1. The tool should provide a separate view for each dependency type, so that it is possible for example to see all exclude-dependencies within one view. In this standard mode no sources of core dependencies (usage view, domain view, IT view) or derived dependencies are shown. (5.2)
- Then the tool should be able to visualize the different sources of dependencies so that the customer is able to recognize if a dependency is based on functional or data aspects. (5.3)
- Furthermore the tool must be able to visualize the derived dependencies, from high-level selections and transitive dependencies (5.4). This implies to
  - show the dependencies (core and derived) in a graphic form. It must be possible to fade in and fade out the derived dependencies.
  - highlight the origin of derived dependencies
  - represent the derived dependencies in a tabular form

With these capabilities the tool would be able to support the communication of product family variability dependencies. One would be able to generate the specific view according to the current topic of the product definition. The tool thus is able to reduce complexity and to make the customer aware about the essential dependencies.

## 6 Summary and Outlook

One of the main concepts of product families is to facilitate the constructive reuse of product family core assets during the development of customer specific products. The main goal of reusing core assets is to reduce costs and time in the process of developing

products. In order to reach this goal it is essential to communicate the product family variability and capabilities with the customer.

In this paper we argue, that explicit representation of product family variability and the variation point dependencies with use case models help the requirements engineer to communicate variability aspects to the customer. For finding suitable representations of dependencies we elaborated on different types of dependencies, in analogy to the dependencies within feature modelling. We characterized the dependencies between variants and/or variation points. Then we described why dependencies occur: The origin of dependencies can be caused by functional, data, or IT aspects. Moreover, dependencies can be derived from other dependencies and thus they are very difficult to recognize and to communicate.

Regarding the special interest of customers in functional aspects of variability and motivated by the successful representation of functional variability in use case diagrams we suggested a representation of dependencies in use case diagrams. The explicit dependency representation is a prerequisite (in addition to variant and variation point representation) of a successful communication of product family capabilities to customers. In addition it supports change management of variants because now one is able to identify all depending variants (or variation points) of the changed variant.

But the suggested representation of dependencies in use case diagrams also illustrates the enormous complexity of this issue. Therefore we suggest tool-support for reducing complexity without loosing the essential dependency information for the communication to the customers.

However, according to the definition and implementation of customer specific products many open issues have to be handled. Thus our future work deals with the following questions:

- How to represent non-functional aspects of variability, including dependencies?
- How to support the derivation of products using use cases and scenarios?
- How influences the cardinality of relationships between variants and variation points the dependencies between variation points?

## 7 References


[1] Felix Bachmann, Len Bass: Managing Variability in Software Architecture; ACM Press, NY, USA, 2001

[2] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, Henk Obbink, Klaus Pohl: Variability Issues in Software Product Lines; Fourth International Workshop on Product Family Engineering (PFE-4), Bilbao, Spain, 2001

[3] K. Czarnecki, U. Eisenecker; Generative Programming: Methods, Techniques, and Applications; Addison-Wesley 1999

[4] J. Carroll: The Scenario Perspective on System Development, Scenario-Based Design: Envisioning Work and Technology in System Development; John Wiley & Sons, 1995

[5] Paul Clements, Linda Northrop: Software Product Lines, Practices and Patterns; SEI Series in Software Engineering; Addison Wesley, 2001

[6] Alistair Cockburn: Writing Effective Use Cases; Addison Wesley, 2001

[7] Coriat, M., Jourdan, J., Boisbourdin, F. : The SPLIT method. Proceedings of the First International Software Product-Line Conference (SPLC-1), August 2000

[8] M. Griss, J. Favaro, M. d'Alessandro: Integrating Feature Modeling with the RSEB; 5th ICSR Conference Proceedings, Vancouver, Canada 1998

[9] Günter Halmans, Klaus Pohl: "Communicating the Variability of a Software Product Family to Customers", *Software and Systems Modeling*; Vol. 2, 15-36; Springer; Hamburg; March 2003;

[10] Günter Halmans, Klaus Pohl: Modellierung der Variabilität einer Produktfamilie; *Modellierung 2002: Modellierung in der Praxis - Modellierung für die Praxis*, 2002

[11] Matthias Jarke, Klaus Pohl; Establishing Visions in Context: Towards a Model of Requirements Processes (1993) EMISA Forum

[12] I. Jacobson, M. Griss, P. Jonsson : Software Reuse Architecture, Process and Organization for Business Success; Addison-Wesley:Longman, May 1997

[13] Ivar Jacobson: Object-Oriented Software Engineering: A Use Case Driven Approach; Addison Wesley, 1992

[14] K. Kang et al; Feature-Oriented Domain Analysis (FODA) Feasibility Study; Technical Report No. CMU/SEI-90-TR-2, November 1990, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

[15] K. Kang et al; FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures; Annals of Software Engineering 5, 1998

[16] Kyo C. Kang, Kwanwoo Lee, JaeJoon Lee, and SaJoong Kim: Feature-Oriented Product Line Software Engineering: Principles and Guidelines; to in Domain Oriented Systems Development-Practices and Perspectives, Gordon Breach Science Publishers, UK, 2002

[17] Object Management Group (OMG); OMG Unified Modelling Language Specification, Version 1.4; September 2001 OMG, http://www.omg.org/uml

[18] Klaus Pohl; Process-Centered Requirements Engineering; Research Studies Press 1996

[19] Detlef Streiferdt, Matthias Riebisch, Ilka Philippow; Formal Details of Relations in Feature Models; In: Proceedings 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems, Huntsville Alabama, USA, April 7-11, 2003

[20] Mikael Svahnberg, Jilles van Gurp, Jan Bosch: On the Notion of Variability in Software Product Lines; Proceedings of The Working IEEE/IFIP Conference on Software Architecture, 2001

[21] Frank van der Linden: Software Product Families in Europe: The Esaps & Café Projects; IEEE Software, 19(4); 41-49, July/August 2002

# Requirements Interdependencies
# - Moulding the State of Research into a Research Agenda

Åsa G. Dahlstedt
*Department of Computer Science*
*University of Skövde,*
*Box 408, SE-541 28 Skövde, SWEDEN*
*asa.dahlstedt@ida.his.se*

Anne Persson
*Department of Computer Science*
*University of Skövde,*
*Box 408, SE-541 28 Skövde, SWEDEN*
*anne.persson@ida.his.se*

**Abstract**.

Requirements relate to and affect each other, i.e. they are interdependent. This paper provides an overview of the current state of research on requirements interdependencies and formulates a research agenda for the area. The research agenda, which is based on a new classification drawn from the literature and intermediary results from an ongoing interview study, addresses a number of unresolved issues concerning the identification, documentation and use of requirements interdependencies in the software development process.

## 1. Introduction

Most requirements cannot be treated independently, since they are related to and affect each other in complex manners [1, 2]. Actions performed based on one requirement may affect other requirements in ways not intended or not even anticipated. Dependencies between requirements may also affect various decisions and activities during development, e.g. requirements change management [3, 4], release planning [2, 5], requirements management [6], requirements reuse [7] and requirements implementation [8]. This implies that there is a need to take interdependencies into consideration in order to make sound decisions during the development process (for examples, see Section 3.1). Despite this, little is known about the nature of requirements interdependencies, and further research is needed in order to understand the phenomenon better [5, 9,10].

The overall aim of our research is to identify which types of requirements interdependencies that are critical to take into consideration in specific development situations, such as e.g. release planning or requirements management. Also, we aim to propose approaches for managing dependencies according to the needs in each specific situation. This paper provides a first step towards this research goal, by providing an overview of the current state of requirements interdependency research, by developing an integrated classification of fundamental interdependency types discussed in the literature, and formulating a research agenda for further research.

The amount of literature addressing requirements interdependencies is fairly small and it approaches the area from different perspectives. Pohl [4] as well as Ramesh and Jarke [6] discuss the topic as part of requirements traceability, focusing on requirements management as well as change management. The effect requirements interdependencies have on requirements selection or release planning is discussed by Karlsson et al [5], Carlshamre and Regnell [9] and Carlshamre et al [2]. Robinson et al [8] reports on requirements interaction management, which deals with identifying how requirements may affect each other's achievement.

The aim of this paper is, as stated above, to provide an overview of the current state of requirements interdependency research. Our first step was to explore which types of interdependencies that are currently known. This was done by compiling the

different views found in the literature, by identifying common patterns among described types to discover fundamental ones. The result is a classification of known interdependency types presented in Section 4. The classification is neutral with respect to development situations. It needs to be further elaborated with respect to the specific needs within the different development situations where requirements interdependencies affect the work. We have also formulated a research agenda, where fundamental problems when dealing with interdependencies have been identified as well as identified some initial development situations where it is considered important to take requirements interdependencies into account.

The paper is organised as follows. Section 2 places requirements interdependency into its context – requirements traceability. We also provide a brief overview of the area an a discussion about the term interdependency. An overview of the literature found addressing requirements interdependencies are provided in Section 3 together with some preliminary findings from an ongoing interview survey. This is then compiled into a neutral classification of fundamental interdependency types presented in Section 4. This section also includes the research agenda developed for requirements interdependency research. The paper ends with some concluding remarks in Section 5.

## 2. Traceability: a Basis for Understanding Requirements Interdependencies

Requirements traceability has been acknowledged as an important part of software and information systems development [4, 11, 12] supporting various activities during the life of a software system. We view the area as a basis for addressing requirements interdependencies. The topic is well-explored, judging by the large amount of literature describing both theoretical and empirical studies (see e.g. 4, 13, 11, 14, 15, 16, 17]. Ramesh and Jarke [6] present an extended overview of the current state of research within the area, based on several years of research.

There are several definitions of the term traceability [see e.g. 6, 18, 19, 4]. In this paper, we have chosen to define traceability as the "*ability to describe and follow the life of a requirement, in both forward and backward direction, ideally through the whole system life cycle*" [20, pp. 32, based on 14]. The definition indicates that requirements traceability can be divided into two main types: pre-traceability and post-traceability (Figure 1). Pre-traceability refers to those aspects of a requirement's life before it is included in the requirements specification [14] and is focused on enabling a better understanding of the requirement. Post-traceability, on the other hand refers to those aspects of a requirement's life from the point in time when it has been included in the requirements specification [14] and is focused on enabling a better understanding and acceptance of the current system/software.

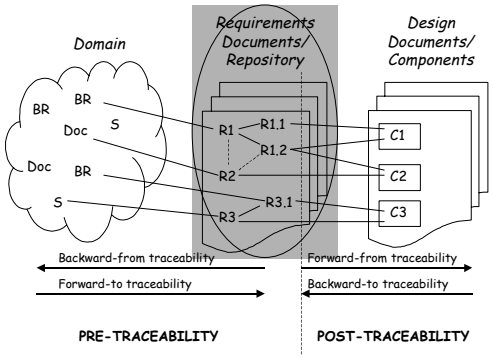



**Figure 1: Different types of traceability**

Requirements pre-traceability is hence concerned with *requirements production* and focuses on the *domain* with which we interact when the requirements are developed and in which the systems is to be installed. Requirements post-traceability is concerned with *requirements deployment* and is focused on the *software* that is developed based on the requirements. Pre- and post-traceability may also be divided into four traceability types, which are presented in [21]. According to [6] traceability information provides important support within requirements engineering, design, systems evolution, and test procedures.

The various types of traceability links presented in Figure 1 support different situations and activities during the development and maintenance of the software system. None of these will alone give full traceability support (see [3]). Different stakeholders are also usually interested in different types of traceability information. Despite this, current literature and standards provide few guidelines regarding which type of information should be captured and used in what context [6].

Traceability is concerned with tracing relationships between trace objects of various types,

e.g. requirements, rational, document, process stages etc. In this paper, we focus on relationships between a specific type of trace object – namely explicitly stated requirements (showed by the shaded area in Figure 1). The term dependency is used in fairly different manners by different authors. Pohl [4] has a broad view of the term and has defined 18 different dependency types (see Figure 2). Ramesh and Jarke [6], on the other hand, use the term in a more specific sense, distinguishing between dependencies and other types of relationships. This implies that the term dependency can either be seen as a synonym for the term relationship, or as a stronger connection between two objects, where the objects affect each other in some way, e.g. in case of changes. In this paper, we will not distinguish between dependency and relationship. We are interested in exploring the different manners by which requirements can relate to each other, which may mean that they affect each other as well. We have also chosen to use the term *interdependency* to emphasise that the relationships that we focus on are those that exist between trace objects of the same type.

## 3. Requirements Interdependencies – Current State of Research

This section aims at providing an overview of the current state of research on requirements interdependencies by outlining findings from the literature concerning requirements interdependency types and affected development situations as well as findings from an ongoing interview survey. The complete set of requirements interdependency types found in the literature are presented in [22]. These are discussed and compiled into a neutral classification of fundamental requirements interdependencies presented in Section 4. We have delimited our survey to literature explicitly discussing interdependencies between requirements.

### 3.1. Requirements Interdependencies – a Literature Review

The area of requirements interdependencies is fairly unexplored judging by the relatively small amount of literature discussing it. However, there are some milestones within this field of research.

In the early days of traceability research, Pohl [4] developed a traceability framework, which included a dependency model defining 18 different types of possible dependency links (Figure 2). Pohl's model describes dependency types that can exist between any type of trace object used in the requirements engineering process. We focus on requirements interdependencies, but there are most certain some correlations between these general dependencies and requirements interdependencies, which motivate why this dependency model is relevant for our investigation.

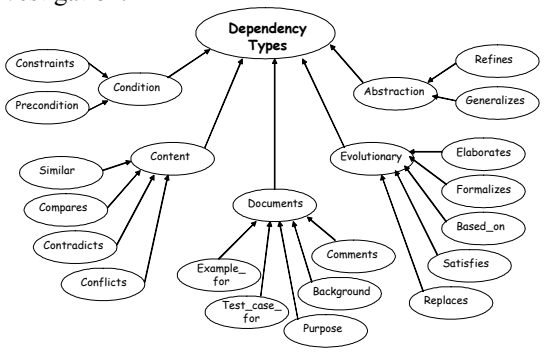



**Figure 2: The dependency model [4]**

However, Pohl's dependency model must be somewhat adapted and specialised towards requirements interdependencies to be useful in our research. There are some dependency types included in Pohl's model that clearly cannot exist between requirements (see [22] for a description of the categories and dependency types in the model). These are the category "Documents" and the dependency type "Compare", which are therefore excluded from further discussion regarding this dependency model. In the other cases, the term trace object in the description of the dependency types may be replaced by requirement and we will use this interpretation in the forthcoming discussion.

Even though Pohl's model is a valuable starting point for our research, the categories and dependency types presented in Pohl's model are sometimes difficult to clearly distinguish from each other. There are also additional requirements interdependency types found in subsequent literature. There is hence a need to adapt and revise this model in order to develop a model focusing specifically on requirements interdependencies and also to incorporate recent research.

Pohl mentions that knowledge about how the requirements have evolved, and hence relate to each other, is considered to be important when dealing

with changes and change integration. Kotonya and Sommerville [3] agree with this view and states that the notion of requirements interdependency is one of the most important aspects of traceability, from a change management perspective. These dependency types are a considerable part of Pohl's model (both abstraction and evolutionary). Pohl also identifies requirements interdependencies as an enabler of identifying reusable software components. If similar requirements are detected when the stated requirements are compared with existing requirements, this indicates a reusable component. The dependency type "Similar" is included in the model.

Karlsson et al [5] have developed an approach for requirements selection, through pair-wise comparison. They state that requirements prioritisation approaches must include means for managing requirements interdependencies in order to fully support developers. Due to these interdependencies, requirements cannot be treated as stand-alone artefacts. For example, if you choose to implement a high priority, low cost requirement, you may also have to implement a low priority, high cost requirement. Requirements can hence not be selected based solely on priority. Karlsson et al [5] concludes that there is a lack of support for requirements interdependencies, one particular where the impact of including or excluding requirements can be observed. They have identified an initial set of interdependency types, which they considered as relevant in the context of requirements selection (see [22]).

Carlshamre and Regnell 98] agree with [5] and conclude that release planning is a very complex task, due to requirements interdependencies. Management of requirements interdependencies are considered to be especially important when the requirements are "fostered asynchronously in a life cycle model", since they connect the requirements fragments. Future research is claimed to be needed concerning the different types of interdependencies that exist between requirements. Carlshamre and Regnell [9] describe some types of interdependencies (see [22]).

Carlshamre et al [2] have continued the work of [5] and [9], and conducted an industrial survey on requirements interdependencies within release planning. Six different types of interdependencies were identified (see [22]), partially based on the types presented in [5], and analysed in relation to 20 high priority requirements within five different companies. The findings from this survey are that there are few single requirements, i.e. requirements with no relationship to other requirements. It was sometimes fairly difficult for the respondents in the study to choose interdependency type for a relationship between two requirements, because more than one interdependency type could be used. There was hence a need to prioritise the interdependency types. It was also concluded that requirements interdependencies are rarely identified explicitly. There are several reasons for this. The large amount of interdependencies results in difficulties to identify and manage dependencies. Requirements interdependencies are also fairly fuzzy, meaning that the relationship they describe can be more or less critical. If R1 increases the implementation cost of R2, it could be a large increase or an insignificant. This problem is also discussed by [6], who states that it is fairly difficult to identify the strength of an interdependency link. Even though pair-wise analysis of requirements also supports identification of other problems with the requirements, it requires much time. It is important to find ways of reducing the assessment time and Carlshamre et al discuss some approaches to this end.

Ramesh and Jarke [6] have taken the first steps towards reference models for requirements traceability. They do not focus on requirements interdependencies, but, as we stated above, requirements interdependencies is a traceability problem. According to [6] companies with a simplistic traceability practice also document traceability links between requirements in order to model requirements traceability. Most of the interdependency types discussed are related to requirements management and requirements evolution (see [22]). Ramesha and Jarke [6] also state that the decomposition of high level requirements into more detailed requirements, is important to keep track on, e.g. in order to manage the explosion in the number of requirements as well as facilitating understanding of the requirements by mapping them back to their sources.

Ramesh and Jarke [6] also emphasise that it is neither feasible nor desirable to maintain links between all related requirements and output produced during the development process, due to the overheads involved in maintaining traceability links. Instead, it is more feasible to identify the critical requirements and to concentrate on storing the relevant traceability information for those.

Robinson et al [8] report on an area called requirements interaction management. This area focuses on managing relationships between requirements, which may interfere with each other's achievements. The idea is to identify requirements that cannot be satisfied simultaneously. Robinson et al has hence taken an implementation or realisation oriented view on requirements interdependencies. The main aim is to manage conflicts between requirements, and identify the problems with satisfying requirements at requirements definition time. Robinson et al have also defined a number of different requirements interdependency types (see [22]).

An approach for systematic recycling of requirements between requirements documents referring to product variants is presented by von Knethen [4], who also considers it important to ensure that all related requirements to a copied requirement are transferred to the recycled document. There are some interdependency types presented and used within this approach (see [22]).

We can hence conclude that several different types of interdependencies are presented in the literature and that different activities or development situations are in focus (see Section 4).

### 3.2. Some findings from an Ongoing Interview Study

This section presents some preliminary results regarding requirements interdependencies from an ongoing interview study. The study focuses on current practice and challenges concerning requirements engineering in Swedish software industry, and one part of the study is more specifically focused on requirements interdependencies. For more information about the study, see [23, 24].

Generally, most of the respondents in the study acknowledge that requirements do relate to and affect each other. However, not many of the participating companies documented requirements interdependencies explicitly. Instead, the requirements were clustered, usually with respect to which requirement that should be implemented together. This could e.g. depend on whether the requirements concerned the same part of the system, if it would be cost efficient to implement the requirements at the same time, or if they should be implemented by the same person.

The interdependency types mentioned by the respondents were mainly conflict and cost of implementation. Conflicting requirements affect each other's achievements, and the main work is to make trade-offs regarding how to implement the different requirements. Cost of implementation is concerned with identifying requirements that can/should be implemented at the same time, since this decreases the implementation cost. Duplicates and similar requirements were also mentioned.

Requirements interdependencies that are easy to discover are also considered easy to manage without documenting them. These interdependencies are handled ad-hoc through experienced and knowledgeable personnel. Instead, it is those requirements interdependencies that are difficult to identify that are problematic to deal with. Also, it is sometimes possible to identify that there is an interdependency, but the consequences of the dependency is difficult to comprehend. Usually these interdependencies exist between non-functional requirements.

## 4. Towards a Model of Fundamental Interdependency Types

Before we can enter deeply into addressing how to manage requirements interdependencies in different situations, we first need to compile the different views expressed in the literature into an integrated model, which is neutral with regard to development situation. One identified problem is to choose between different types of interdependencies and Pohl's dependency model alone comprises 18 types. Also, judging by the discrepancies between requirements interdependency types presented in the literature, there is still some work to be done.

In trying to penetrate the ideas behind the different contributions in the literature it has become clear to us that the perspective that the authors take on the area results in slightly different classifications. In essence, these classifications seem to be influenced by what some stakeholder wants to do with the requirements as part of the development process, e.g. requirements selection or release planning. Also, the various classifications overlap and the meaning of certain terms, which denote the types, are not clear in the area as a whole. E.g. the term "temporal

dependency" is given different meanings by different authors. The complete list of interdependency types on which we base our analysis can be found in [22].

Based on the literature and also on some intermediary results from an ongoing interview study, we have developed a classification (Figure 4), which could be considered to be a first step towards developing an overall, neutral model of fundamental requirements interdependencies.

This classification will most likely need to be further elaborated and most of all validated, e.g. using a number of different sets of requirements. Since we have focused on identifying a few types that we so far consider to be fundamental, these may later be adjusted or extended to suit different needs in the software development process, e.g. in requirements selection or release planning.

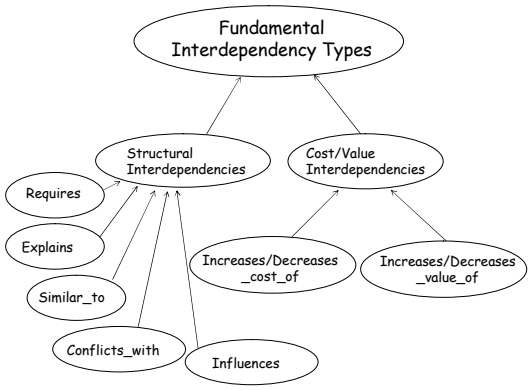



**Figure 4: The new classification**

Taking this stance we have identified two categories of interdependencies that could be considered to be fundamental and more or less neutral. We tentatively call them *STRUCTURAL* and *COST/VALUE* interdependencies.

## 4.1. Structural Interdependencies

Structural interdependencies are concerned with the fact that given a specific set of requirements, they can be organised in a structure where relationships are of a hierarchical nature as well as of a cross-structure nature. Often high-level business requirements are gradually decomposed into more detailed software requirements. Also, requirements from different parts of a hierarchy may influence each other across the overall hierarchy. We find that the following interdependency types fall into this category:

***Requires***

The fulfilment of one requirement depends on the fulfilment of another requirement. This type can be used to describe a hierarchical relation between two requirements, but also relations across hierarchical structures.

This dependency type is derived from the interdependency types "requires" [2], "and" [2], "logical" [9] and "must-exist"[5]. This relationship can also be viewed in the opposite direction i.e. instead of R1 requires R2, R2 is a prerequisite for R1 [2]. The interdependency type *Requires* then also covers "precondition" mentioned by Pohl [4]. Carlshamre et al [2] concludes in their investigation that the temporal dependency type [9, 2, 8] is seldom interesting. It may either be viewed as a *Requires* dependency or an *Increase/decrease_cost_of* dependency (see 4.2). We have chosen to agree with this view, since a temporal interdependency also is useful from the perspective of which activity that should be performed and is hence not neutral.

The "or" dependency [2] is difficult to categorise, since it can be related to different interdependency types. The "or" dependency relates alternative solutions to each other, which e.g. may be required by another requirement i.e. R1 requires some of the following requirements {R2, R3, or R4}. Clearly, this dependency type requires more research in order to be fully understood.

"Satisfy" [4] and "positive correlation" [8] can be viewed as a weaker dependency of the type require. They both concern linking requirements, which support the fulfilment of another requirement. In this context, the require dependency type is used when R1 *must* be implemented in order to fulfil R2, while "satisfy" and "positive correlation" is weaker and describes a situation where the fulfilment of R1 have a positive effect on the fulfilment of R2.

***Explains***

A general requirement is explained by a number of more specific requirements. This dependency type is used to describe hierarchical structures of a weaker nature than *Requires* and relates more detailed requirements to their source requirements. If a detailed requirement is derived from a high level requirements, but it is not a prerequisite for this requirement, the relation is of the dependency type explain.

This dependency type covers "elaborate", "part_of", "is_a" and "derive" from Ramesh and Jarke [6], and "elaborate", "formalise", "replaces",

"generalises", "refines" and "based_on" by Pohl [4] as well as "refinement" from von Knethen et al [7]. As stated above, we seek to identify basic interdependency types, and these are fairly similar and may be difficult to distinguish. We have therefore chosen to summarise them into one overall dependency type.

***Similar_to***

One stated requirement is more or less similar to one or more other requirements.

This interdependency type corresponds with "similar" [4] and "structure" [8]. This interdependency type is also mentioned within the interview study. Natt och Dag [25] presents an evaluation of the feasibility to use natural language processing techniques to identify duplicates within a requirements set.

***Conflicts_with***

A requirement is in conflict with another requirement if they cannot exist at the same time or if increasing the satisfaction of a requirement decreases the satisfaction of another requirement.

This interdependency type includes both situations were it is impossible to implement both requirements, and situations were these have a negative influence on each other's achievements and a trade-off between the resolution of the requirements must be made. It hence covers "constraint" [4], "negative correlation" [8], "conflict" [4] and "cannot_exist" [5]. Conflict is also one of the most frequently mentioned interdependency types in the interview survey. Robinson et al [8] has a strong focus on conflict dependencies, and present some relations, which can be interpreted as reasons for the conflict e.g. "resource", "tasks" and "causality".

***Influences***

A requirement has an influence on another requirement.

It is indicated in the literature that a requirement may affect or influence another requirement in other ways than requires, explains and conflicts. Both [6] and [7] has a fairly general interdependency type, termed "depend_on" and "dependency". Our hypothesis is that more dependencies can be identified, especially when this classification is further elaborated with respect to different development activities or situations. However, at this stage we choose to include a general interdependency type which can be used if a relationship between two dependent requirements is not of the type "requires", "explains" or "conflicts_with".

## 4.2. Cost/value Interdependencies

Cost/value interdependencies are concerned with the costs involved in implementing a requirement in relation to the value that the fulfilment of that requirement will provide to the perceived customer/user.

The following interdependency types fall into this category:

***Increases/Decreases_cost_of***

If one requirement is chosen for implementation then the cost of implementing another requirement increases or decreases.

This interdependency type includes "icost" [2], "positive cost" and "negative cost" [5] as well as "value-related" [9].

***Increases/Decreases_value_of***

If one requirement is chosen for implementation then the value to the customer of another requirement increases or decreases.

This interdependency type covers "cvalue" [2] as well as "positive value" and "negative value" [5].

## 4.3. A Research Agenda

Apart from developing a reference model of fundamental requirements interdependencies and extending this to cater for specific needs in the software development process, we can identify three major issues for research in the area of requirements interdependencies:

- *How can we identify requirements interdependencies?* The problems within requirements interdependencies are not only concerned with how to record and maintain links between related requirements. These relationships must also be identified somehow. Some interdependencies may be easy to discover when analysing the requirements set, but there are interdependencies, which are more difficult to identify. In addition, it can also be difficult to identify how the requirements affect each other, especially regarding non-functional requirements. We need to investigate how to identify requirements interdependencies as well as to explore how requirements affect each other. Pohl [4] has proposed a method for automatically recording traceability links. Carlshamre et al [2] describe how to use pair-wise analysis of the requirements to discover interdependencies, and they also discuss several

alternatives regarding how to decrease the time required performing this analysis. Both these approaches assume that the developers know how the requirements affect each other. There is, however, a need for approaches focusing on how the explore the consequences of an interdependency, i.e. how the requirements affect each other.

- *How can we describe requirements interdependencies?* When the different relationships between requirements have been identified we must also provide support for storing and managing them. A common problem in current traceability tools is that they provide means to store a relationship between requirements but they provide very little guidance regarding the semantic and inherent meaning of the relationship [6]. There is also a need for mechanisms identifying the most critical interdependencies, because it is not feasible to link every related requirement. It must hence be possible to show the strength of the interdependencies [6, 2].

Requirements traceability research includes several alternative approaches for recording and managing traceability links. One important research issue is to investigate which of those are suitable for recording and managing requirements interdependencies. Also, Carlshamre et al [2] presents one approach for describing requirements interdependencies. This approach is built on visualisation, which is considered as an important feature for this issue. It could also be relevant in this context to look at different techniques for goal modelling (see e.g. [26]) as a means to model and describe interdependencies, since requirements could be considered to be low-level goals. The $F^3$ Enterprise Modelling language [27], more specifically a sub-model denoted the Information Systems Requirements Model, also includes means of describing requirements interdependencies, based on this notion.

- *How do we use requirements interdependencies in the software development process?* According to Ramesh and Jarke [6], literature and standards within requirements traceability provide few guidelines regarding what type of information that must be captured and used in what context. An important research issue is, therefore, to investigate what it means in different contexts when we state that there is an interdependency. As indicated by the literature, different types of interdependencies are important in different development activities or as basis for various decisions. Another important research issue is to explore which types of interdependencies are critical to consider in different situations. The first step towards this is to investigate what types of activities are affected by requirements interdependencies. As a starting point the following activities, mentioned in the literature, can be used.

**Requirements Management** is concerned with managing the large amount of requirements and information elicited during requirements engineering [10]. Capturing requirements interdependencies may be useful in this phase since it provides an overview of how the high level requirements are decomposed into more detailed requirements [6]. Keeping track of the derived requirements is also a way of managing the fast increasing number of requirements.

**Change management.** One of the major challenges in software development is the constant evolution and change of requirements [6]. Requirements interdependencies are shown to be useful in this context since it shows the evolution of requirements. Requirements interdependencies also allow us to view the major assumptions behind a requirement, by relating it to the originating requirement. However, one of the most important benefits of requirements interdependencies is that they show how requirements relate to and affect each other, which, hence, facilitates impact analysis of change proposals [3, 4].

**Release planning** is an activity concerned with selecting an optimal collection of requirements for implementation in the next version of a system. The selection is usually based on requirements priority. However, due to the fact that requirements are related to and affect each other, priorities cannot be the only basis [9]. Knowledge about how requirements interact and restrict each other is, therefore, an important basis for these decisions.

**Reuse of components.** If similarity between requirements is documented, these interdependencies can be used to identify reusable components by comparing the stated requirements with the requirements of the existing system [4].

**Reuse of requirements.** When requirements are reused in requirements documents describing various variants of a product, it is considered as relevant to ensure that all requirements related to a copied requirement is transferred to the recycled document [7].

**Implementation.** Software design is to a large extent concerned with decision-making. Many trade-

offs are made e.g. to decide the scope and functionality of the system as well as between implementation cost and other resources [6]. Requirements interdependencies may support these types of trade-offs and decisions, e.g. by revealing interaction between requirements which may interfere with their achievement [8].

**Testing.** A potentially interesting area where requirements interdependencies may be a relevant aspect to take into consideration is software testing. During this activity, test cases are developed based on the requirements which fulfilment is supposed to be ensured. Since requirements relate to and affect each other, their knowledge about requirements interdependencies may affect the ability to create purposeful and complete test cases.

**Maintenance.** Few software and information systems are stable once they are implemented in the organisation. Most systems continuously evolve due to changes in organisation or users needs, or due to errors made during the development [4]. Requirements interdependencies are useful in this context, since it shows how changing requirements affect other requirements already implemented in the software.

## 5. Concluding Remarks

Keeping track of requirements interdependencies is essential in order to support several situations and activities within the system development process. However, there is little known about the nature of requirements interdependencies, which is shown by the relatively small amount of literature discussing the phenomenon.

This paper compiles the different views of requirements interdependencies found in the literature. It also takes a first step towards what we call a neutral classification of fundamental requirements interdependencies. In this classification, interdependencies are grouped in two main categories; structural and cost/value interdependencies.

A research agenda for requirements interdependencies has also been outlined. The first step is to further elaborate and validate the classification framework presented in this paper in relation to development activities or situations affected by requirements interdependencies. Other research issues are related to the identification, documentation and use of requirements interdependencies.

We have mainly addressed the area of requirements interdependencies from a theoretical point of view in this paper. The main concern for the future, however, is to focus on empirical research that gives a useful contribution to solving pressing problems in the field of software development practice.

## References


[1] Regnell, B., Paech, B., Aurum, A., Wohlin, C., Dutoit, A. and Natt och Dag, J. (2001). Requirements Mean Decisions! – Research issues for understanding and supporting decision-making in Requirements Engineering, First Swedish Conference on Software Engineering Research and Practise (SERP'01), October 25-26, Ronneby, Sweden

[2] Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B. and Natt och Dag, J. (2001) An Industrial Survey of Requirements Interdependencies in Software Product Release Planning, Fifth International Symposium on Requirements Engineering, 27-31 August, Toronto, Canada.

[3] Kotonya and Sommerville (1998) *Requirements Engingeering – Processes and Techniques*, John Wiley & Sons.

[4] Pohl, K (1996) *Process-Centered Requirements Engineering*, John Wiley & Sons Inc.

[5] Karlsson, J., Olsson, S. and Ryan, K. (1997) Improved Practical Support for Large-scale Requirements Prioritisation, *Requirements Engineering Journal*, 2(1), p. 51-60

[6] Ramesh, B. and Jarke, M. (2001) Toward Reference Models for Requirements Traceability, *IEEE Transactions on Software Engineering*, Vol.27, no1., p. 58-93

[7] von Knethen, A., Peach, B., Kiedaisch, F. and Houdek, F. (2002) Systematic Requirements Recycling through Abstraction and Traceability. Proc. of IEEE Joint International Conference on Requirements Engineering, 9-13 Septermber, Essen, Germanny, pp. 273-281.

[8] Robinson, W.N., Pawlowski, S.D. and Volkov, V. (1999) Requirements Interaction Management, GSU CIS Working Paper 99-7, Department of Computer Information Systems, Georgia State of University, Atlanta.
(Printed 041201 from http://cis.gsu.edu/~wrobinso)

[9] Carlshamre, P. and Regnell, B. (2000) Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes, Second

International Workshop on the Requirements Engineering Process, Grenwich, London.

[10] Grehag, Å. (2001) "Requirements Management in a Life-Cycle Perspective - A Position Paper". In Ben Achour-Salinesi, C., Opdahl, A.L., Pohl, K. and Rossi, M. (Eds) Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ'01, Interlaken, Switzerland. Essenere Informatik Beiträge, pp. 183-188.

[11] Gotel, O. (1995) *Contribution Structures for Requirements Traceability*, PhD Thesis, Department of Computing Imperial Collage of Science, Technology and Medicine, University of London.

[12] Maciaszek, L.A. (2001) *Requirements Analysis and System Design – Developing Information Systems with UML*, Addison Wesley.

[13] Jarke, M., Rolland, C., Sutcliffe, A. And Dömges, R. (1999) *The NATURE of Requirements Engineering*, Shaker Verlag, Aachen.

[14] Gotel, O. and Finkelstein, A. (1994) An Analysis of the Requirements Traceability Problem, In *Proc. of the 1st international Conference on Requirements Engineering*, Colorado Springs, Colorado, USA, p. 94-102

[15] Ramesh, B. (1993) A Model of Requirements Traceability for Systems Development, Technical report, Naval Postgraduate School, Monterey, CA, USA, September.

[16] Ramesh, B., Powers, T., Stubbs, C. and Edwards, M. (1995) Implementing Requirements Traceability: A Case Study, In *Proc. of the 2nd International Symposium on Requirements Engineering*, York, England, p. 89-93.

[17] Gotel, O. and Finkelstein, A. (1997) Extended Requirements Traceability: Results of an Industrial Case Study, In *Proc. 3rd International Symposium on Requirements Engineering* (RE97), IEEE Computer Society Press, p. 169-178.

[18] IEEE-830 (1994) *Guide to Software Requirements Specification*, ANSI/IEEE Std. 830, Institute of Electrical and Electronics Engineers, New York

[19] Johnson, W.L., Feather, M.S. and Harris, D.R. (1991) Integrating Domain Knowledge, Requirements, and Specifications, *Journal of Systems Integration*, 1, p. 283-320.

[20] Jarke, M. (1998) Requirements Tracing, *Communication of the ACM*, December 41(12).

[21] Davis, A.M. (1990) The Analysis and Specification of Systems and Software Requirements, In *Systems And Software Requirements Engineering*, IEEE Computer Society Press, p. 119-144

[22] Dahlstedt, Å. and Persson, A. (2003) "An Overview of Requirements Interdependency Types". http://www.ida.his.se/ida/~asa/ReqInterdependencies.pdf

[23] Karlsson, L., Dahlstedt, Å., Natt och Dag, J., Regnell, B. and Persson, A. (2002) Challenges in Market-Driven Requirements Engineering - an Industrial Interview Study, Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), September, Essen Germany.

[24] Dahlstedt, Å. G., Karlsson, L., Persson, A., Natt och Dag, J. and Regnell, B. (2003) "Market-Driven Requirements Engineering Processes for Software Products - a Report on Current Practices." Submitted to Development of Product Software, DoPS-03, 20 and 21 June 2003, Velden, Austria.

[25] Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M. and Karlsson, J. (2002) "A feasibility study of automated natural language requirements analysis in market-driven development. " *Requirements Engineering, 7*, pp. 20-33.

[26] Bubenko J.A. jr, Persson A., Stirna J. (2001) *User Guide of the Knowledge Management Approach Using Enterprise Knowledge Patterns*, deliverable D3, IST Programme project HyperKnowledge -- Hypermedia and Pattern Based Knowledge Management for Smart Organisations, project no. IST-2000-28401, Dept. of Computer and Systems Sciences, Royal Institute of Technology, Stockholm, Sweden,
available on http://www.dsv.su.se/~js/ekd_user_guide.html

[27] Bubenko jr., J. A., *"Extending the Scope of Information Modelling"*, Fourth International Workshop on the Deductive Approach to Information Systems and Databases, Lloret, Costa Brava (Catalonia), Sept. 20-22, 1993. Department de Llenguatges i Sistemes Informatics, Universitat Politecnica de Catalunya, Report de Recerca LSI/93-25, Barcelona.

# A Relation-based Approach to Use Case Analysis

A.Fantechi*, S.Gnesi^, G.Lami^

*Dip. di Sistemi e Informatica - Università di Firenze – Italy
^ISTI - C.N.R. - Area della Ricerca C.N.R. di Pisa - Italy

## Abstract

*Use Cases are an effective tool for modeling functional requirements of software systems. A well written Use Case allows to depict a large amount of information regarding the behaviour of the system, as perceived by the actors. Use Cases have the advantage to be expressed using Natural Language expressions that have a fixed structure and this can mitigate some of the usual, NL-inherent, problems of interpretation. In this paper, we present an approach that, starting with the application of NL processing techniques to the Use Case scenarios, derives semantic information on the relations between the actors. This information, largely achievable in an automatic way, can be used to support the analysis of Use Case requirements document and it represents a starting point towards the formal verification of some relevant aspects.*

## 1. Introduction

The problem of the analysis of software requirements with respect to some consistency and correctness parameters has been extensively exploited in several ways in recent years [18]. For example, formal methods and tools have been used for this purpose when a formal representation of software requirements has been adopted.
Currently, in the common practice formal notations are not always used in the first description of the system. More frequently Natural Language (NL) expressions are used to represent software requirements [15, 20]. It is hence quite important to provide methods and tools for the consistency and correctness analysis of them starting from their NL representation.
Use Cases are a powerful tool to capture functional requirements for software systems. They allow structuring requirements documents with user goals and provide a means to specify the interactions between a certain software system and its environment. In his book [5], Alistair Cockburn presents an effective technique for specifying the interactions between a software system and its environment. The technique is based on natural language specification for scenarios and extensions. Scenarios and extensions are specified by phrases in plain English language. This makes requirements documents easy to understand and communicate even to non-technical people.
The typical structure of Use Cases makes their analysis easier and more effective than classic Natural Language sentences. Quality is determined by the fulfilment of some predefined characteristics (target qualities) that in the case of Use Cases can be classified into three main groups: Expressiveness, Completeness and Consistency.

- *Expressiveness* category: it includes those characteristics dealing with the understanding of the meaning of Use Cases by humans, such as *Ambiguity* or *Understandability*
- *Consistency* category: it includes those characteristics dealing with the presence of semantic contradictions and structural incongruities in the NL requirement document.
- *Completeness* category: it includes those characteristics dealing with the lack of necessary parts within the requirement specifications.

The quality of NL components of Use Cases (typically sentences), may be analysed from a lexical, syntactical or semantical point of view [10]. For this reason it is proper to talk about, for example, lexical non-ambiguity or semantical non-ambiguity rather than non-ambiguity in general [12,14]. For instance, a NL sentence may be syntactically non-ambiguous (in the sense that only one derivation tree exists according to the syntactic rules applicable) but it may be lexically ambiguous because it contains wordings that have not a unique meaning.
In the practice expressiveness-related issues can be addressed by means of existing NL-based techniques and tools [2,8,11,17, 22]. For example, ambiguity mitigation may be addressed in the following ways:

- By lexical evaluation: using lexical parsers to detect and possibly correct terms or wordings that are ambiguous.
- By syntactical evaluation: using syntactical analysers to detect sentences having different interpretations.

Understandability improvement may be addressed by lexical and syntactical evaluation as well: using linguistic parsers both to detect poorly understandable or complex parts of the document and to achieve readability indicators (metrics) based on the count of elements of the sentences (e.g. the number of characters or words of the sentences, the average length of the sentences, …).

It is more difficult to exploit NL-based techniques and tools able to provide some help in addressing Consistency and Completeness issues, because it is necessary to capture, at least at some level, the semantics of the Use Case under evaluation.

In this paper we present a methodology for the analysis of the Use Case based requirements documents. This methodology is aimed to extract semantic information from the Use Cases, based on NL processing techniques. This information regards the functional relations between the actors of the Use Case based requirements specification. From the elementary relation between two actors determined by the verb in a sentence we discuss how to derive more complex relations between the concepts present in the Use Cases description, which may help in assessing consistency and completeness issues. This derivation can be largely supported by automatic tools.

This paper is structured as follows: in section 2 we describe the kind of Use Cases we will consider and in section 3 we present our approach to support the analysis of consistency and completeness based on linguistic techniques. In section 4 we discuss how to derive the relations, presenting an integrated environment for the lexical, syntactical and semantic analysis of NL requirements that can be used to this aim. In section 6 we discuss the possible use of the relational approach and the related opportunities to improve the analysis of Use Case-based requirements documents.

## 2. Use Cases

A Use Case describes the interaction (triggered by an external actor in order to achieve a goal) between a system and its environment. A Use Case defines a goal-oriented set of interactions between external actors and the system under consideration. The term *actor* is used to describe the person or system that has a goal against the system under discussion. A primary actor triggers the system behaviour in order to achieve a certain goal. A secondary actor interacts with the system but does not trigger the Use Case.
A Use Case is completed successfully when that goal is satisfied. Use Case descriptions also include possible extensions to this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure in completing the service in case of exceptional behaviour, error handling, etc.. The system is treated as a "black box", thus, Use Cases capture *who* (actor) does *what* (interaction) with the system, for what *purpose* (goal), without dealing with system internals. A complete set of Use Cases specifies all the different ways to use the system, and therefore defines the whole required behaviour of the system. Generally, Use Case steps are written in an easy-to-understand, structured narrative using the vocabulary of the domain. The language used for the description is English. Any other natural language can be used as well, and although our analysis focuses on English, the same reasoning can be applied to other languages (considering the obvious differences in syntax and grammar rules). A scenario is an instance of a Use Case, and represents a single path through the Use Case. Thus, there exists a scenario for the main flow through the Use Case, and other scenarios for each possible variation of flow through the Use Case (e.g., triggered by options, error conditions, security breaches, etc.). Scenarios may also be depicted in a graphical form using UML sequence diagrams. Figure 1 shows the template of the Cockburn's Use Case taken from [6].
In this textual notation, the main flow is expressed, in the "Description" section, by an indexed sequence of NL sentences, describing a sequence of actions of the system. Variations are expressed (in the "Extensions" section) as alternatives to the main flow, linked by their index to the point of the main flow in which they branch as a variation.
Developers have always used scenarios in order to understand what the requirements of a system are and how a system should behave with respect to its environment. The work we present in this paper is an attempt to provide means to identify possible flaws in the textual scenario descriptions.

| **USE CASE #** | < the name is the goal as a short active verb phrase> | |
|---|---|---|
| **Goal in Context** | <a longer statement of the goal in context if needed> | |
| **Scope & Level** | <what system is being considered black box under design><br><one of: Summary, Primary Task, Subfunction> | |
| **Preconditions** | <what we expect is already the state of the world> | |
| **Success End Condition** | <the state of the world upon successful completion> | |
| **Failed End Condition** | <the state of the world if goal abandoned> | |
| **Primary, Secondary Actors** | <a role name or description for the primary actor>.<br><other systems relied upon to accomplish use case> | |
| **Trigger** | <the action upon the system that starts the use case> | |
| **Description** | **Step** | **Action** |
| | 1 | <put here the steps of the scenario from trigger to goal delivery, and any cleanup afterwards> |
| | 2 | <...> |
| | 3 | |
| **Extensions** | **Step** | **Branching Action** |
| | 1a | <condition causing branching> :<br><action or name of sub.use case> |
| **Sub-Variations** | | **Branching Action** |
| | 1 | <list of variation s> |

**Figure 1. Use Case template**

## 3. Towards Linguistic Evaluation of Consistency and Completeness

To effectively address the Consistency and Completeness aspects of requirements specification, we should resort to their formalization, [11,23]. Indeed formal methods are a powerful mean to evaluate requirements because they provide a theoretical framework to verify their correctness. Formal methods require, however, a specific skill and this has increased their cost and prevented their wide application.

In this paper we investigate methods and tools that may provide an effective support to deal with Consistency and Completeness issues, but that are still based on NL analysis and then are more user-friendly.

Other works aiming at the improvement of the correctness of requirements relying on the Use Cases structure exist [1,7]. The methods and tools we are presenting in this paper also rely on the structure of the Use Cases and are based on the study of the relations between actors of Use Case-based description of a system.

The method we describe in this paper can be placed between a "lightweight" parsing [13] and a "full-fledged NL" approach [16] and aims at demonstrating that the extraction of "semantic" information for a text is possible also without using tools and methods too heavy.

The relations we are interested in are the "functional" relations, i.e. the relations or dependencies between two actors. These relations can be determined looking at the syntactical structure of each sentence of the Use Case scenarios defining a set of items (quadruples) where each primary actor (the subject of the sentence) has been put in relation with the secondary actor (the complement) according to the verb. The canonical form of these relations is:

(1.) *(Actor_1, verb_i, Actor_2, Use_Case_id).*

Each item compliant with (1.) describes an occurrence of a functional relation between two actors established by the verb and indicates the Use Case in which this relation occurs.

The functional relations between two actors, in the form (1.), can be extended, by transitivity, to other actors when two items with the following form exist: ($A_i$, $v_1$, $A_j$, $UC_x$) and ($A_j$, $v_2$, $A_k$, $UC_y$). In this way, hence, an indirect functional relation between the actor $A_i$ and the actor $A_k$ is also established by transitivity. Starting from this consideration, chains joining different actors can be built, where each item ($A_i$, $v_x$, $A_j$ ,$UC_x$) of the chain is such that the previous item has the form ($A_k$, $v_y$, $A_i$, $UC_y$) and the following has the form ($A_j$, $v_z$, $A_h$, $UC_z$). The collection of all the items derivable from a Use Case-based requirements document is said *Relations core*. The Relations core embeds all the elementary functional relations between actors that can be extracted directly by the NL description.

We can derive specific, non-elementary, relations from the relations core. In the following we provide some definitions and define some properties based on the elementary relations (1.).

The *ignores* relation, denoted by A~B, holds if no relation *(A, verb_i, B, Use_Case_id)* exists.

The relation (1.) between actors can be used to build the *Relation Graph*. The nodes of this graph represent the actors and an oriented arc connecting two nodes (A and B) indicates that A drives B.

Two nodes are adjacent if an arc from A to B exists.

A *path* from the node A to the node B on this graph is a sequence of adjacent nodes in a graph starting from the node A and arriving to the node B. On the basis of the Relations graph some further relations between actors can be defined:

The *is connected to* relation, denoted by A => B , holds if at least one path from A to B exists on the graph.

The *is chief* relation, denoted by A =>> B, holds if B ignores A and A is connected to B.

The *chief graph* (derived from the chief relation) is an acyclic graph composed of nodes (the actors) and oriented arcs connecting two actors, an arc originating from the node A and arriving in the node B means that A is chief of B.

Nodes of the chief graph having no incoming arcs are said *leader* nodes and nodes having no out arc are said *executors* nodes. An example of Relation and Chief graphs are provided in Figure 2.

The availability of the functional relations and of the graphs derived from them enables the capturing of some semantic information on the system we are describing. In particular, this information can be used to support the detection of critical points (in terms of consistency and completeness) in the interactions between different actors. These critical points can be revealed by analysing the set of derived direct and indirect relations.

## 4. Derivation of the relations between actors

The derivation of the relations core and the consequent construction of the relation chains, relations graph and chief graph, can be supported by automatic tools based on NL processing techniques.

In fact, the basic relations (1.) are detectable by using a syntactical parser able to identify the different components of a NL sentence. To our purpose, the key components to be identified are the subject(s), the verb and the complement(s) associated to the verb. Once this information is achieved, it is possible to define the relations and to build a data base containing the relations core derivable from the collection of Use Cases under analysis.

## 4.1 An example of Derivation of Relations

In this section we present an application of the relational approach to a sample Use Case document, with the aim to clarify the concepts discussed above.
The example we present in this section is derived, with few changes, from a sample System Requirements Document available on the web at the Cockburn's home page [24], which is provided in Appendix 1.
This document, describing a Purchase Request Tracking System, has the purpose to provide the functional requirements of a basic system for the official Buyers of the company, to track what they have ordered from Vendors against what they have been delivered. The documents is organised as a set of Use Cases.
The primary actors of this document are:

- Approver: typically the requestor's manager, who must approve the request.
- Authorizer: person who validates the signature from the Vendor.
- Buyer: person who manages the order, talking with the Vendor.
- Vendor: person or company who sells and delivers goods.
- Requestor: person putting in a request to buy something.
- Receiver: takes care of the arriving deliveries

The document contains fifteen Use Cases describing the behaviour of the system. We slightly modified it by adding two new Use Cases to make it more precise and suitable for the analysis. The Use Cases included into the document are compliant with the Cockburn's style, and they include several data such as, for example, Preconditions, Post-conditions, Trigger, Extensions, etc. We simplified each Use Case by reducing the information associated to them. In particular, we took into account only the Primary Actor, the statement of the Goal and the description of the Scenario. These data represent the minimum set of information necessary to save the essential meaning of the Use Case. In Appendix 1, the set of the simplified Use Case we used for the experiment is shown.
The outcomes of the application of the relational approach to the simplified Use Cases of the case study are summarized in a collection of relations items between actors and a set of relations chains derived from the relations items.
For simplicity let us identify the actors of the case study by a letter:

A. Authorizer
B. Approver
C. Requestor
D. Buyer
E. Vendor
F. Receiver

Figure 2. contains the relations derived from the case study where each actor is identified by the corresponding letter along with the corresponding relation graph and chief graph. In the following a possible set of relations chains starting from the relation (A, notify, B, UC3), is provided:

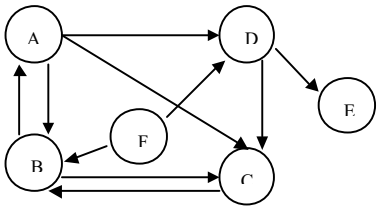
- (A, notify, B, UC3), (B, send, C, UC6), (C, send, B, UC7).
- (A, notify, B, UC3), (B, send, C, UC6), (C, send, B, UC8).
- (A, notify, B, UC3), (B, send, C, UC9), (C, send, B, UC7).
- (A, notify, B, UC3), (B, send, C, UC9), (C, send, B, UC8).
- (A, notify, B, UC3), (B, send, A, UC12).
- (A, notify, B, UC3), (B, send, A, UC12), (A, change, B, UC3).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, C, UC9).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, C, UC9), (C, send, B, UC7).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, C, UC9), (C, send, B, UC8).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, change, E, UC4).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, send, C, UC9).
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, send, C, UC9), (C, send, B, UC7)
- (A, notify, B, UC3), (B, send, A, UC12), (A, send, D, UC16), (D, send, C, UC9), (C, send, B, UC8).

The table in figure 2. also shows that an ignore relation between C and A occurs, and it means that C doesn't influence directly the A's behaviour.

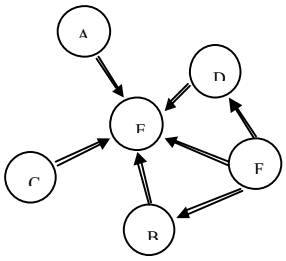
## 4.2 An integrated environment for Use Case Analysis

In this section we provide a description of MAIGRET, an integrated environment for natural language analysis. MAIGRET was built with the goal to provide an automatic support for the analysis framework of expressiveness, consistency and completeness aspects of natural language requirements documents. To reach this goal we have realized and integrated a set of tools, each one dedicated to a specific linguistic analysis purpose of NL sentences. In particular (figure 3), the involved tools are a lexical analyzer (QuARS) [8,9], a syntactical analyzer (SyTwo/Cmap) [4,21] able to extract items representing relations between the entities (actors) described in the requirements document and a third tool (Relation Manager) having the aim of storing and managing these relations. Figure 3 shows the high level architecture of MAIGRET.

| Primay Actor | verb | Secondary Actor | UC |
|---|---|---|---|
| A | change | B | 3 |
| A | notify | B | 3 |
| A | notify | B | 3 |
| D | change | E | 4 |
| C | send | B | 5 |
| B | send | C | 6 |
| C | send | B | 7 |
| C | send | B | 8 |
| B | send | C | 9 |
| A | send | C | 9 |
| D | send | C | 9 |
| D | return | E | 11 |
| B | send | A | 12 |
| D | send | E | 14 |
| F | notify | D | 15 |
| A | send | D | 16 |
| F | send | B | 17 |




Relation Graph




Chief Graph

**Figure 2 : Relations between actors and graphs**

MAIGRET is composed of a processing part (the integrated tools) and of a static part (composed of the following modules: the WordNet English dictionary; a semantic network modelling the domain; and a requirements-specific dictionary) that represents its knowledge base.

In the following we provide a more detailed functional description of the integrated tools.

**QuARS** (Quality Analyzer for Requirements Specifications) is a sentence analyser aiming at reducing linguistic defects by pointing out those wordings that make the document ambiguous or not clear from a lexical point of view. The tool points out these defects without forcing any corrective actions, leaving the user free to decide whether modifying the document or not. Moreover the sentences are analyzed taking into account the particular application domain, and this is possible through the use of targeted dictionaries. In this sense the tool has been designed to be easily adaptable. The following are examples of expressiveness defects pointed out by QuARS; the underlined wordings are the indicators used by QuARS to point out the sentence as defective:

- `the C code shall be clearly commented (vague sentence)`
- `the system shall be as far as possible composed of efficient software components (subjective sentence)`
- `the system shall be such that the mission can be pursued, possibly without performance degradation (optional sentence)`

The first sentence contains the word "clearly" that makes the whole sentence vague. The second contains the wording "as far as possible" that makes it subjective. The third sentence is pointed out as defective because contains the word "possibly" that determines an option in it.

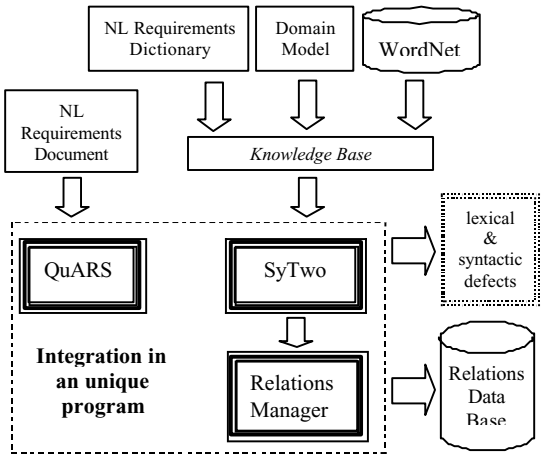



**Figure 3: high level architecture of MAIGRET**

**SyTwo/Cmap** is a syntactical analyser that relies on the Knowledge Base described before. The Knowledge Base is exclusively used by SyTwo/Cmap to perform the syntactical analysis and to derive the relations. Starting with a Use Case description (possibly already analyzed by QuARS), SyTwo/Cmap performs a syntactical analysis making the detection of syntactically ambiguous sentences possible. A syntactically ambiguous sentence can be pointed out if it has more than one derivation tree (i.e. the sequences of syntactical rules to be applied to build the sentence): this implies that the sentence may be understood in different ways. For example the sentence "*The system shall not remove faults and restore service*" may be syntactically understood at least in these two ways:

1. The negation *not* of the auxiliary verb *shall* is related to the first verb *remove* only, and not to the other verb *restore*. In this case, the meaning of the sentence is that the system shall not remove the faults and it shall restore the service.
2. The negation *not* of the auxiliary verb *shall* is related to both the verbs *remove* and *restore*. In this case, the meaning of the sentence is that the system shall not remove the fault and shall not restore the service.

The functionalities described above for both tools allow therefore expressiveness characteristics to be analyzed. More interesting for the methodology presented in this paper is the additional functionality offered by SyTwo/Cmap, which is able, on the basis of the knowledge of the syntactical structure, also to extract the relations between subjects, verbs and objects in a sentence, in a format compliant with (1.). This allows to capture all the relations between two actors in the problem domain.

**Relations Manager**: it receives as input the set of relations derived by SyTwo/Cmap and it puts them in a database for analysis. Once such a database has been populated it is possible to:

- extract, by means of queries, sub-sets of relations.
- extract, by means of queries, sets of relation chains.

The outcomes derived from the data-base, can be used in support of the analysis of the requirements document in order to extract the interactions between actors and build the relation and the chief graphs.

## 5. Applying the Relational Approach

In this section we discuss possible applications and developments of the relational approach to the Use Case-based requirements engineering.

Since relations indicate the presence of a verb in the Use Case relating two actors, they often indicate possible interactions between actors. Hence, relations chains can be interpreted as interaction schemata. Walkthroughs of these interaction schemata may be performed in search of undesired, inconsistent and incomplete dynamic behavior of the system.

These schemata may also form the basis for a formal analysis of interactions, which however we do not address in this context.

Walkthroughs of interaction schemata may be aimed at detecting relation chains containing loops, because loops indicate a more complex kind of interaction, and may point to a possible synchronization problem (such as a deadlock). It is possible, in this way, to point out some potential synchronization problems in a sequence of actions.

Let us consider, for instance, the example of section 4.2. In this case we detect the relation chain: *(A, notify, B, UC3), (B, send, C, UC6), (C, send, B, UC8),* presenting a loop. If carefully walk through this chain, and we represent this interactions sequence on a time scale (see figure 4), it is possible to understand that some potential synchronization problems may occur.

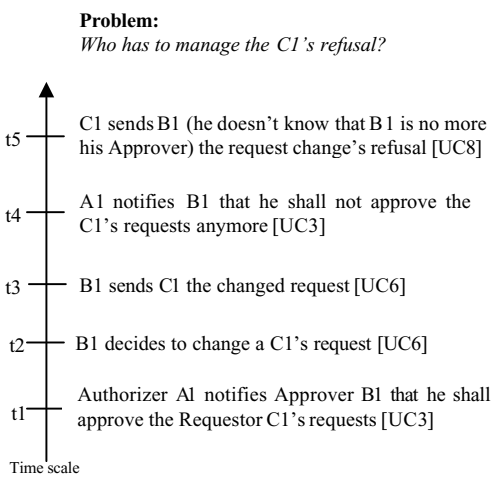



**Figure 4: Interactions sequence**

In fact, if the Approver B1 sends the changed request to the Requestor C1 and, before that C1 tells B1 that this change is refused, Authorizer A1 changes the authorization to B1 and makes B2 the Approver of C1, then who should manage C1's refusal?

In this case, it is possible to detect an inconsistency in the requirements due to an incomplete specification of the requirements because the notification of the changed Approver is not sent also to the associated Requestors. This kind of problems, that are hidden if we consider only the Use Cases-based requirements document, may be easily detected by using the relations chains.

Another possibility of exploiting this information is to point out those pairs of actors that have an higher number of interactions than the others. The pairs that, in the case study, have the highest number of different interactions are Approver-Requestor (5 interactions), Authorizer-Approver (4 interactions) and Buyer-Requestor (3 interactions). The indication that can be derived from these data is that the interactions between these actors are at the core of the functionality of the system, and therefore should be analysed in more detail in order to point at possible problems. Also, this information can give an indication to which parts of the system should be stressed at testing time.

The semantic information that can be extracted from the derived relations and graphs can help the analysis of

correctness and completeness of the requirements by detecting some gaps in the specification of Use Cases.
In fact, the graphs defined above (and in particular the chief graph) allow some interesting considerations to be made. The chief relation is not to be intended as determining a hierarchy in terms of the importance of the role played by the actors. This relation and the information derivable from the graph is a semantic information that allows to enlighten the influence of an actor on the others. In particular, if a node A of the chief graph is connected with the node B by an arc (A, B), then it can be argued that the behaviour of B doesn't influence that of A. This kind of semantic information about the actors, that cannot be directly derived from the set of Use Cases, can play a relevant role for the analysis. In particular, it is possible to easily detect lacks in the relational structure of the requirements.
In the example shown above, the relation F =>> D occurs. This occurrence enlightens a gap in the specifications because the buyer should have the capability of having a relation with the receiver (for instance, to ask the status of an on-going acquisition).

The relational approach can be oriented to achieve a guidance for systematic construction of the Use Case requirements documents. In fact, building the relations graphs in parallel with the definition of the Use Cases impels a continuing series of walkthroughs to check the part of the relations graph completed so far and examine how remaining relations should be added to the graphs themselves.

We wish, in the end to point at another application of the relational approach, which spans outside the context of Use Cases. A concept that has gained importance in the last years, especially in the telecommunication field, is the concept of feature. A feature is a capability of a system which provides value to the users, but is conceived as separate from the other features provided by a system to its users. However, at the system level, features can interact in a complex manner (a problem often referred as "Feature interaction"), so they cannot be treated as separate in the development of the system, and especially in the requirements document. A feature may even prevents other system activities: for instance, in a mobile handset user interface the "keyguard" feature prevents almost all other user-originated activities (but not incoming call handling).
The description of a feature by Use Cases can be trivial (in the keyguard example the scenario might be composed simply of the "set the keyguard on" activity) and the Use Cases may be not able to represent how the system behaviour is affected by a feature.
The knowledge of the influence of the features on the UCs can be important mainly for the testing of the system because the Use Cases are not enough for representing the consequences of the features on the functionalities they describe.
For this reason the relational approach to the Use Case analysis can be of interest to identify those Use Cases affected by a feature.
For example the Use Cases affected by the keyguard feature can be detected because they have in their scenario a sentence like "User digit a key". These UC are influenced in case of the keyguard is set on.

## 6. Conclusions and Future works

In this paper we presented an approach to the analysis of Use Case requirements documents based on the relations between the actors. Starting from the simple relation between two actors derivable from a scenario sentence, by means of NL parsing tools, some more complex, derived relations have been defined. These relations are able to provide semantic information on the content of a requirements document, supporting the completeness and consistency analysis. The semantic information on the Use Case requirements documents that can be captured with this approach is only partial, w.r.t. the semantic of the whole requirements. Anyway, this information is able to provide a concrete support for the analysis. The use of the semantic information derivable with the relation-based approach has been discussed in this paper. In particular, the knowledge of the functional relations between actors expressed by the Use Cases allows to perform walkthroughs in the relation core to detect possible gaps in terms of consistency and completeness. Moreover, a guidance for a systematic construction of the Use Cases requirements document can be obtained by the parallel development of graphs and schemata representing the relations.
A related work to ours is that reported in [19], in which more sophisticated NL techniques are used to extract concept lattices out of Use Cases, which offer a richer information to the analysis. Our approach use simpler, low cost NL techniques to extract useful information: it would be interesting to see whether the benefits obtained by heavier NL techniques balance their higher costs.
The relation-based approach to the analysis of Use Cases is a promising research direction because it can be used as a mean to bridge the gap between the use of the informal NL descriptions typical of requirements documents, and the more formal artefacts typical of later stages of the development process. In particular, the study of the relations between actors, though starting from a light formalism as the Use Cases are, can provide enough information to move towards the application of formal methods with the support of automatic tools and in a user friendly way. We plan to investigate at this regard the

annotation of the relation graph with pre-conditions and post-condition in order to perform simulations of the system and perform a more refined analysis.
Another subject that we are investigating is the extraction of test cases from Use Case scenarios. Also in this case, extracting information from the textual descriptions in the form of relations between actors helps in the definition of test cases covering the most intricate interaction schemes.

## Acknowledgements

Part of the research work described in this paper was performed under the Eureka ? 2023 Programme, ITEA (ip00004, CAFÉ). We wish to acknowledge the work of Carlo Becheri on the Relations Manager component of the MAIGRET architecture.

## 7. References

[1] T. A. Alspaugh, A. I. Antòn, Scenario Networks: A Case Study of the Enhanced Messaging System, REFSQ'01, Interlaken, Switzerland, June 2001.
[2] V. Ambriola, V. Gervasi, Processing Natural Language Requirements, 12th IEEE Conf. On Automated Software Engineering (ASE'97), IEEE Computer Society Press, Nov. 1997.
[3] C. Ben Achour, M. Tawbi, C. Souveyet, Bridging the Gap between Users and Requirements Engineering: the Scenario-based Approach, International Journal of Computer Systems Science & Engineering, 14(6), 1999 (CREWS Report Series 99-07).
[4 ] Cmap tool on-line: see http://www.yana.net/cmap/
[5] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2000
[6] A. Cockburn, Structuring Use Cases with Goals, Journal of Object-Oriented Programming, Sep-Oct 1997 (part I) and Nov-Dec 1997 (part II)
[7] A. H. Dutoit, B. Peach, Developing Guidance and Tool Support for Rationale-based Use Case Specification, REFSQ'01, Interlaken, Switzerland, June 2001.
[8] F.Fabbrini, M.Fusani, S.Gnesi, G.Lami "The Linguistic Approach to the Natural Language Requirements Quality: Benefits of the use of an Automatic Tool", 26th Annual IEEE Computer Society - NASA Goddard Space Flight Center Software Engineering Workshop, Greenbelt, MA, USA, November 27-29 2001.
[9] F.Fabbrini, M.Fusani, S.Gnesi, G.Lami "Quality Evaluation of Software Requirement Specifications", Proc. of Software & Internet Quality Week 2000 Conference, San Francisco, CA May 31-June 2 2000, Session 8A2, pp.1-18.
[10] A.Fantechi, S.Gnesi, G.Lami, A.Maccari "Linguistic Techniques for Use Cases Analysis" Proceedings of the IEEE Joint International Requirements Engineering Conference - RE02. Essen, Germany, September 9 -13 2002.
[11] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, P. Moreschini, "Assisting requirement formalization by means of natural language translation", Formal Methods in System Design, vol 4, n.3, pp. 243-263, Kluwer Academic Publishers, 1994.
[12] N.E.Fuchs, R.Schwitter, Specifying Logic Programs in Controlled Natural Language, Workshop on Computational Logic for Natural Language Processing, Edinburgh, April 3-5, 1995.
[13] V. Gervasi and B. Nuseibeh, Lightweight validation of natural language requirements. Software: Practice & Experience, Feb. 2002
[14] E. Kamsties, B. Peach, Taming Ambiguity in Natural Language Requirements, ICSSEA 2000, Paris, December 2000.
[15] B.Macias, S.G. Pulman. Natural Language Processing for Requirement Specifications. In Redmill and Anderson, Safety Critical Systems, pages 57-89. Chapman and Hall, 1993.
[16] L. Mich, R. Garigliano, The NL-OOPS project: OO modelling using the NLPS LOLITA, NLDB99.
[17] J. Natt och Dag, B. Regnell et al., Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven Development, REFSQ01
[18] B. A. Nuseibeh and S. M. Easterbrook, Requirements Engineering: A Roadmap, In A. C. W. Finkelstein (ed) "The Future of Software Engineering". (Companion volume to the proceedings of the 22nd International Conference on Software Engineering, ICSE'00). IEEE Computer Society Press.
[19] D. Richards, K. Boettger, and O. Aguilera, A Controlled Language to Assist Conversion of Use Case Descriptions into Concept Lattices, LNAI 2557
[20]C. Rolland, C. Proix, A Natural Language Approach for Requirements Engineering. AISE'92, LNCS 593, Springer-Verlag, 1992.
[21] SyTwo/Cmap on-line. See: http://www.yana.net/SyTwo/Cmap/index.html
[22] W. M. Wilson, L. H. Rosenberg, L. E. Hyatt, Automated Analysis of Requirement Specifications, ICSE 1997, IEEE Computer Society Press.
[23] D. Zowghi, V. Gervasi, A. McRae, Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements, Proc. of the 8th Asia-Pacific Software Engineering Conference, Dec. 2001.
[24] http://members.aol.com/acockburn/papers/prts_req.htm

## Appendix 1

**UC1:**

*Goal*: Requestor buys something through the system
*Primary Actor*: Requestor
*Scenario*:

- Requestor: initiate a request
- Approver: check money in the budget, check price of goods, complete request for submission
- Buyer: check contents of storage, find best vendor for goods
- Authorizer: validate Approver's signature
- Buyer: complete request for ordering, initiate PO with Vendor
- Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design)

- Receiver: register delivery, send goods to Requestor
- Requestor: mark request delivered

**UC2:**

*Goal*: Requestor, manager or buyer wants to see the state of the system

*Primary Actor*: Requestor or manager or Buyer or Receiver

*Scenario*:

- Reader asks to see any one or any multiple requests sorted by any imaginable criteria.
- System: show purchases
- Readers asks to get report printed

**UC3:**

*Goal*: Change who can approve purchases, what purchase limits they have

*Primary Actor*: Authorizer

*Scenario*:

- Authorizer notify the current Approver the changes decided
- Authorizer notify the new Approver the changes decided

**UC4:**

*Goal*: Add, delete, change name, address, phone number of vendors

*Primary Actor*: Buyer

*Scenario*: -

**UC5:**

*Goal*: Create a request in the system

*Primary Actor*: Requestor

*Scenario*:

- Requestor asks to initiate a request
- Requestor fills in the request form
- Requestor sends the request to the Approver

**UC6:**

*Goal*: Change any part of a request

*Primary Actor*: Approver

*Scenario*:

- Approver changes the request
- Approver sends the changed request to the requestor

**UC7:**

*Goal*: Accept changed request

*Primary Actor*: Requestor

*Scenario*:

- Requestor send to the Approver the OK on the changed request

**UC8:**

*Goal*: Refuse changed request

*Primary Actor*: Requestor

*Scenario*:

- Requestor send to the Approver the cancellation of the request

**UC9:**

*Goal*: Cause processing on a request to stop

*Primary Actor*: Approver, Authorizer, Buyer

Scenario:

- Actor cancels the request
- Actor send to the Requestor the confirmation of cancelled request

**UC10:**

*Goal*: Finalize a request as delivered OK, no more work need be done on it

*Primary Actor*: Approver

*Scenario*: -

**UC11:**

*Goal*: Initiate process to return goods, keep from paying for them

*Primary Actor*: Buyer

*Scenario*: -

**UC12:**

*Goal*: Complete approval process for a request

*Primary Actor*: Approver

*Scenario*:

- Complete the forms for request
- Send to the Authorizer the request approval

**UC13:**

*Goal*: Complete all parts of request and initiate POs

*Primary Actor*: Buyer

*Scenario*: -

**UC14:**

*Goal*: From one or more Requests, generate PO to a single vendor

*Primary Actor*: Buyer

*Scenario*:

- Buyer generates the Po
- Buyer send the PO to the Vendor

**UC15:**

*Goal*: Notify Buyer that a delivery did not arrive on time

*Primary Actor*: Receiver

*Scenario*:

- Receiver verifies that the due date is past without receiving the delivery
- Receiver send the Buyer notification that a delivery did not arrive on time

**UC16:**

*Goal*: Establish that the request approver really has the needed signature authority

*Primary Actor*: Authorizer

*Scenario*:

- Authorizer validate approver's signature
- Authorizer send to the Buyer authorization to buy

**UC17:**

*Goal*: Mark actual delivery against one or more POs

*Primary Actor*: Receiver

*Scenario*:

- Receiver register delivery
- Receiver send notification of delivered request to the Approver

# Requirements Engineering for Data Warehousing

Mohamed Frendi, Camille Salinesi
*Centre de Recherche en Informatique*
*Université de Paris 1, Panthéon Sorbonne*
*90, rue de Tolbiac, 75013 Paris, France*
*mohamed.frendi@malix.univ-paris1.fr, camille@univ-paris1.fr*

***Abstract***

*Data Warehouses are used in multiple domains such as management and business process performance evaluation, strategic decision making and business planning, or even to support decisions made in business processes. The main purpose of a Data Warehouse is to support decision making based on the analysis of heterogeneous and distributed information. This paper reviews some of the approaches used in practice to develop Data Warehouses. Based on a structured analysis, we demonstrate the existence of several families of approaches and that their main limitations relates to the lack of guidance of requirements engineering activities. The proposal made is to adapt traditional requirements engineering techniques in the specific context of developing Data Warehouses.*

## 1. Introduction

Decision-making requires large quantities of data. Since these data are scattered in and across organizations, it is necessary to gather and integrate them in order to apply complex requests and lay them out in a consistent way. Bill Inmon [Inmo96] defines a Data Warehouse (DW) as a "c*ollection of integrated, non volatile, subject-oriented databases designed to support the decision support system where each unit of data is relevant to some moment in time. It contains atomic data and lightly summarized data"*. This definition underlines several differences with operational Information Systems (IS):

- DW are s*ubject-oriented*. Indeed, they focus on the evolution of high-level business entities (such as employees or financial forecast) in contrast to operational ISs that support business processes which make operational data evolve (e.g. employee registration or rolling up financial accounts).
- DW are *integrated*. This means that data are stored in a single and consistent format (e.g. using naming conventions, domain constraints, unified physical attributes and measurements) even though they originate from different sources in which they have different formats. On the contrary, ISs can be composed of different data repositories (such as relational databases, object oriented databases, files etc), between which exchange interfaces are usually set up.
- Data managed in DW are *time variant.* This means that every data is systematically associated to a time reference (e.g. semester, fiscal year, pay period). This is obviously not the case in IS where the association of time information to operational data depends on the need.
- Data in DW are *non-volatile.* This means that data, once in the DW, do not change (they are historised). Again, data can also be historised in IS. This is however not systematic and depends on the requirements.

Reviews of DW development techniques classify a priori those into families, and teach us very little about their drawbacks and how they could be enhanced other than by better project management [List02]. A literature review was thus undertaken to analyze DW development techniques in a structured and rationale way. A framework was built as formerly done in [Rol98b] [Rol98c]. Each of the analyzed DW development technique was positioned in the framework. This analysis helped uncovering additional families of approaches, and emphasized the general lack of specific guidance for the requirements elicitation process, requirements analysis process and system validation process. In addition to this structured literature review, this paper proposes to adapt traditional RE techniques and integrate them with existing DW development approaches. The result is a first-cut goal-oriented process model that provides a rich picture of how to deal with requirements in the context of a DW project. This process model is of course meant to be evolutionary and therefore, it should be experimented. Our plan is to enrich it by : (i.) introducing new strategies to achieve the goals identified, (ii.) refining and further adapting the techniques already selected, and (iii.) exploring the transition from the requirements view to the DW conceptual model view.

The remainder of this paper is structured as follows: section 2 presents our framework; in section 3, some of the existing DW development approaches are discussed in the light of this framework; section 4 outlines our approach and shows how it proposes to combine the

advantages of different families of approaches. This approach integrates RE activities to solve the main issues identified with the literature review.

## 2. Framework for analysing existing DW development approaches

The data needed by decision makers to found their decisions are often not easily accessible through the conventional IS [Gree01]. On the contrary, DW propose features that allow to :

- *extract* data from scattered sources, such as internal or external databases, enterprise resource planning, other DW, etc.
- *integrate* them in a central repository that differentiates the reconciled data from the operational data stores (ODS) in which data are stored haphazardly,
- *maintain* them through time,
- *customize* and *aggregate* them into data marts that provide synthesised views of the DW according to the decision makers interests.

These features are shown in figure 1. Besides, the figure shows that not only decision makers can use data marts to focus on some aspects of the DW, but also they can directly access the entire DW through complex queries. This usage has two main purposes, which allows to differentiate between two main categories of decision makers : those having operational decisions, and those with strategic decisions [List02]. Operational decisions are those taken in the flow of business process to define how to proceed. On the other hand, strategic decisions have an influence on organisation and they can result in changes on the structure of business processes.

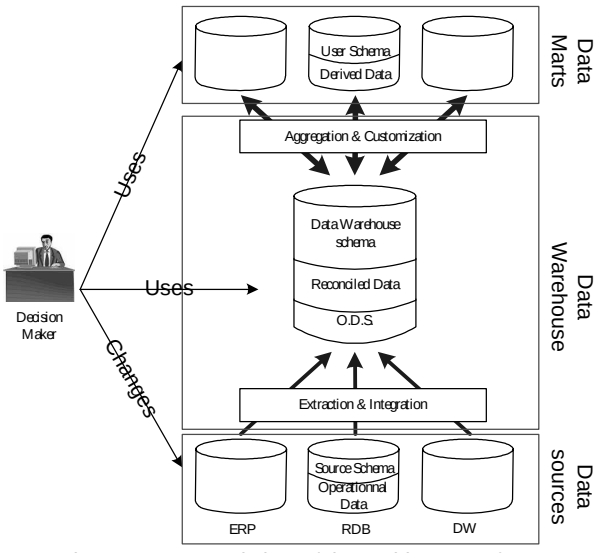



Figure 1: A general view of the architecture of DW

To develop a DW one has to define the structure of its repository and define the operations that allow to feed it in with data as well as to exploit it. A number of approaches can be found in the literature ([Gol98], [Inmo96], [Kim96], [Kim2002], [Moo2000], [Poe96], [Sap98]) and are used in practice ([Inmo96], [Kim96], [Poe96]).

Based on our experience of literature surveys on scenario-based RE approaches [Rol98b] and process engineering approaches [Rol98c], we hypothesised that four perspectives could be used to characterize DW development approaches: (i.) the system perspective, (ii.) the subject perspective, (iii.) the usage perspective, and (iv.) the development perspective.

We developed a framework that views DW development techniques according to these four perspectives. As figure 2 shows, each perspective offers itself a number of options. Choosing among the options offered within each perspectives allows to characterize an approach an to compare it with others.

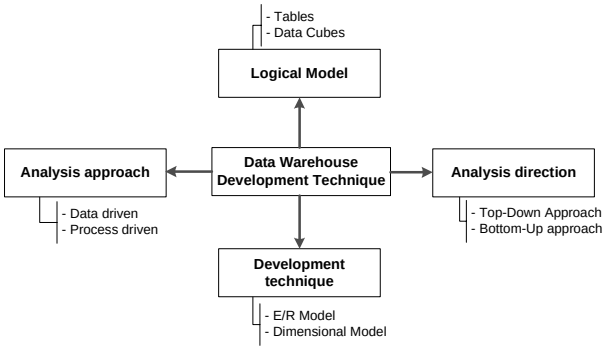



Figure 2: Framework for DW approaches

The remainder of this section is divided into four subsections. Each of those is devoted to the review of the existing DW development techniques within one of the four perspectives.

### 2.1 The System perspective: logical models

This perspectives proposes to categorise DW development approaches according to the logical models they use to implement DWs. Two options are available in this perspective: tabular models and dimensional models.

*Tabular models* are inspired from the relational model. The idea is to use universal tables (denormalized tables); this is particularly useful to avoid joints when accessing data.

The *Dimensional models* introduces the concept of Data Cube. A Data Cube is a multidimensional hierarchy structure. It generally contains summarized data as opposed to tabular models which can also contain detailed data (figure 3). Values higher in the hierarchy are more aggregated than those lower in the hierarchy. This hierarchical organisation is useful to let the user easily navigate between high and low precision views of the same aggregated data.

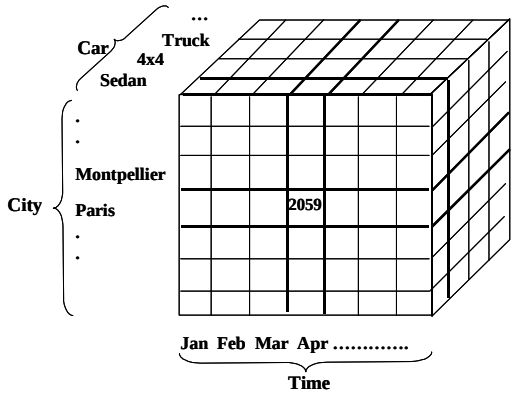


Figure 3: An example of Data Cube

In practice, most of the DWs schema are designed using the tabular model ([Gar2000], [Informix2000] [Craig99]). At the extreme, some approaches reuse the models used at the operational level, and propose universal tables to optimise queries. This approach has the advantage that it allows to reuse the existing operational data model. Although de-normalisation introduces redundancies, there is no maintenance issue in such tabular DW because data are never updated. The main drawback of this model is that it does not easily support the integration of external sources.

This issue is solved by the dimensional model [Kim96]. Indeed, dimension tables can be used to represent the data available in different sources. Kimball argues that besides, the design of DW schema according to the dimensional model is easier. This seems to be reflected in the choice of the dimensional model option in most of the recent approaches ([Kim96], [Bal98], [Hai95], [Jar2002], [Moo2000], [Sap98], [Win96],…).

## 2.2 The Subject perspective: Analysis direction

Two *analysis directions* can be taken while developing a DW : the top-down direction, and the bottom up direction.

In the *Top-Down direction*, the focus is first on the information needed to make decisions prior to the information available at the operational level. This approach is inspired from the Waterfall approach [Roy88] used in the CASE environments. The advantage of this approach is that it allows to limit the scope of the study as well as the system boundary. Experience has shown that this approach has several drawbacks [Nag93] such as: duration of the project, high cost, estimation of ROI difficult before the entire realization of the project.

Approaches that take the *Bottom-Up direction* are recommended for handling legacy systems. They consist in building Data Marts (DM: these are small DWs) first, which is faster than building a whole DW ([Inmo96], [Gol98], [Kim96], [Moo2000] approaches can be used to design DMs). Once DMs are operational, they can be federated into an enterprise wide DW. The advantages of Bottom-Up approach are its speed and easiness of use. Besides, the ROI appears immediately with the DMs. The development of new DMs is an easy way to make the whole system evolve. However, the uncontrolled proliferation of DMs can cause integration problems for building the enterprise DW. Another drawback is the difficulty of exploiting in each DM the data of the other DMs as long as the enterprise DW is not fully developed.

## 2.3 The Usage perspective: Analysis approaches

The purpose of the *Usage perspective* is to categorize the DW development techniques according to the approach used to analyse the system. This perspective is divided into two options: process driven and data driven.

(a) *Process driven* approaches are interested in the business processes by which DWs are populated or used as well as the decision processes during which DWs are exploited. The analysis of these processes can be driven by the requirements of each individual activity that they contain or driven by improvement goals.

*User requirement oriented* approaches generally propose guidelines for conducting user interviews. In these approaches, the DW is not designed to support small-scale query requirements but the decisions made in whole processes [Poe96].

*Goal oriented* approaches assume that the main issue of DW development is to decide which process to improve and what improvement should be made. Identifying the improvement goals helps determining the data needed to achieve the process improvement ([kim96, kim98], [Boe2000]).

*(b) Data driven* approaches consider first the sources of data upon which decisions are made. The priority is therefore to populate the DW. It is only when queries are submitted to the DW that data reconciliation and improvement of the DW schema are achieved. For example, Golfarelli and al. [Gol98] propose a methodology to semi-automatically build a dimensional data warehouse model from E/R models of legacy operational databases. These approaches are also often based on the analysis of the organization data model (e.g. [Inmo96]). However, in such approaches the needs of DW users are only taken into account after the DW is developed, at exploitation and continuous improvement time. The risk in inadequacy of DW developed using the data driven approaches is thus very high.

## 2.4 The Development perspective: Development techniques

This perspective considers the development technique used to design the DW schema. Nowadays, there are two development techniques used, the E/R model and the Dimensional model.

In the *E/R model* based approaches, DWs schemas are usually composed of flat entities (i.e. denormalized). The advantage is that the E/R formalism is well known

by the designers. Extensions to E/R formalism have been proposed for better adequacy to the specific issues of DW development [Sap98].

The *Dimensional model* was introduced by Kimball [Kim96]. It applies a pattern in which there is a large table called "*facts table*" that dominates the others. This central table gathers operational data of the organization. It is combined with a number of "*dimensions tables*" that provide details on the operational data of the facts table. This formalism has the advantage to be clear and to offer an easy way to represent the measurement factors of the organization. Different patterns are proposed in the literature : star schema, snow flake model, constellation model, etc.

As shown in figure 4, in the *Star schema* dimensional model, the facts table is in the center and is connected to other dimensions by [1..n] relationships. This structure has the advantage of limiting the number of joints in queries. Several studies have also proven that the star schema dimensional model is the model that presents the best compromise in term of complexity and information redundancy.

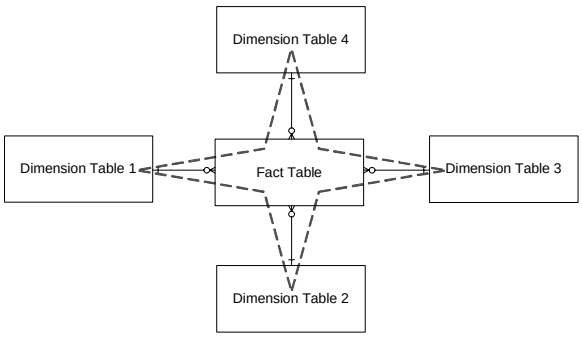



Figure 4: Generic structure of Star schema [Moo2000]

Similarly, in the *Snowflake schema*, there is a central fact table with normalised dimensions around it. Therefore, as shown in figure 5, each dimension is split into sub-dimensions.

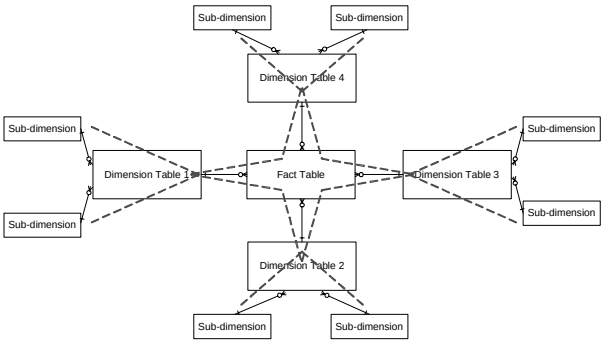



Figure 5: Generic structure of Snowflake schema

## 3. Discussion on existing DW development approaches

Based on the Framework presented in the former section, we reviewed 9 approaches found in the literature. Table 1 shows the characteristics of each approach by affecting yes/no values (ticks are used to characterise a yes) to each option available in the four perspectives provided by the framework.

| Approache / Perspective | Option | Inmo96 | Cabi98 | Gol98 | Kim98 | Boe2000 | Sap98 | West2001 |
|---|---|---|---|---|---|---|---|---|
| System | Table | ➳ | * | * | | * | * | * |
| | Data Cube | ➳ | ➳ | ➳ | ➳ | ➳ | ➳ | * |
| Usage | Process driven/Goal oriented | | | | ➳ | ➳ | | |
| | Process driven / User requirements oriented | | | | | | | ➳ |
| | Data driven | ➳ | ➳ | ➳ | | | ➳ | |
| Subject | Top-Down | | | | ➳ | ➳ | | * |
| | Bottom-Up | ➳ | ➳ | ➳ | | | ➳ | ➳ |
| Development | E/R | ➳ | | | | | | * |
| | Dimensional | * | ➳ | ➳ | ➳ | ➳ | ➳ | * |

Table 1: Classification of some approaches

For example, in the data driven approach proposed by Inmon [Inmo96], the company change goals nor user requirements are neither not taken into account. There is therefore no information on how to ensure the adequacy of the DW schema to the usage of the DW. User needs are integrated only in the exploitation and maintenance phase. This may, of course, force designers to re-design large parts or even the complete DW. Such approaches are thus very likely to become expensive through requirements evolution.

Westerman's approach [West2001] is driven by DW user requirements. It proposes to analyse business processes through the decisions made in these processes. Then, these processes are prioritized and the most important of them are defined in terms of data structure needed to make decisions. The result is a number of services expected from the DW to support these decisions. The assumption is that the actual data will be available at the operational level to feed-in the structures. This is of course a strong assumption, and we believe that legacy data sources should be taken into account earlier in the process to avoid this risk.

Poe [Poe96] proposes a catalogue of high level guidelines for conducting user interviews in order to collect end user requirements. She recommends interviewing different user groups in order to get a complete understanding of the business. This approach is similar to Westerman's approach in that the guidelines that it proposed help understanding the requirements of the operational actors of business process.

On the contrary, Kimball's approach [Kim96] is focused on organizational improvement goals and ignores data constraints and operational users requirements. The idea is to grasp the goals of strategic decision makers that have to decide upon when/how to improve business processes. The object of the analysis is therefore the improvement intentions prior to the improved business processes. The top-down analysis

ends up with the identification of business objects managed in the changed business processes; these are modelled using pre-defined patterns of the dimensional model. Böhnlein and Ulbrich-vom Ende [Boe2000] present an approach that is based on the SOM (Semantic Object Model) process modeling technique in order to derive the initial data warehouse structure. The first stage of the derivation process determines goals and services the company provides to its customers. Then the business process is analyzed by applying the SOM interaction schema that highlights the customers and their transactions with the process studied. In a third step sequences of transactions are transformed into sequences of existing dependencies that refer to information systems. The last step identifies measures and dimensions.
Golfarelli's and Moody's approaches [Gol98] [Moo2000] are both data driven. The guidelines provided help analysing the data models of operational data sources and to transform them into dimensional models of the DW. These are bottom-up approaches in which both improvement goals and user requirements are ignored.

To sum up, three major observations can be drawn from our framework-based review of DW development approaches:
(i.) two main families of approaches can be distinguished : (a) top-down process-driven approaches and (b) data-driven bottom-up approaches;
(ii.) the majority of approaches are data-driven. Indeed, it is considered that while approaches in the first family are more appropriate to build enterprise-wide DW, those from the bottom-up data-driven family are easier to use and generate initial results faster;
(iii.)very few approaches are requirements-driven; this is mainly due to the time constraints imposed to DW projects and general belief that requirements-driven approaches are time consuming.
We observed in this review of (some of the) existing approaches that most of the works done in the DW development domain deal with how data should be structured, stored and managed in DWs. Only few approaches consider the services provided by DWs. Besides, we believe that these approaches have a number of drawbacks in common :

- Their usage is focused on one option at the exclusion of the other options available. All the approaches are either goal-oriented, or user requirements driven or data driven. None proposes to combine change goals to user requirements analysis or data analysis.
- These approaches are not or partially automated. There is for example very few detailed guidelines on how to transform DW requirements into a conceptual DW model, or how to elicit DW requirements through user requirements analysis or goal analysis.
- These approaches are based on the expertise of DW designers; none of them proposes to capitalise knowledge from one project to another.

This review of existing approaches is not complete in the sense that:

- not all existing approaches have been classified yet
- the framework on which it is based is still shallow and can be considerably enriched to refine our understanding of the common aspects of and differences between DW development approaches.

However, we used the aforementioned observations to lay out the process model of our approach.

## 4. Requirements engineering process for DW; proposal

As shown in figure 6, our position is that of a DW development approach that takes into account both business process requirements, requirements from strategic decision processes, and operational data models of existing systems. Besides, we propose to combine DW requirements to DW models.
On one hand, *DW requirements* can be elicited using both business process requirements (i.e. by analysing the usage of the future DW in the decisions made in To-Be business processes) and strategic decision processes and improvement goals (i.e. by analysing the usage of the DW to make strategic decisions about change).
On the other hand *DW models* are produced using a combination of DW requirements and As-Is data models. Therefore, as in [Bruck01], while the DW requirements are elicited in a top down fashion, the operational data models are drawn in a bottom up fashion to produce the DW data models.
Once produced, DW data models can also be used to elicit new requirements. This can be based on the exploitation of the similarities between requirements for DW systems to be used within the same domains, as well as on the possibility to transpose patterns of DW design solutions from one domain to the other .

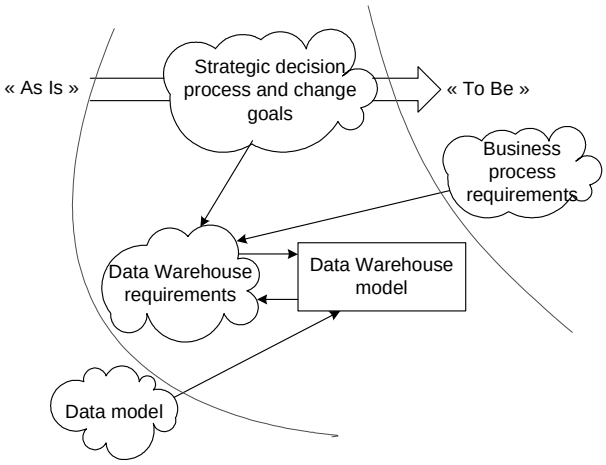


Figure 6: A product view of the position taken on DW development methods

The proposed process is founded on two main goals : (i) elicit requirements, and (ii) design DW models.

The main usage of DW is decision making. Therefore, we believe that *decision analysis* is an important way of guiding requirements elicitation in the DW context. We believe that similar techniques could be used to analyse the decisions made in business processes and the decisions made in strategic decisional processes. The models used can for instance be adapted from the NATURE process meta model (that defines a process as a set of decisions to be taken in the context of different situations and goals) [Grosz97], or from the MAP process meta model (according to which process models should integrate the decisions of which goal to achieve as well as how to achieve it) [Benj99]. Further adaptations for a better adequacy to the needs of DW usage processes can also be inspired from decision theories such as [Roy85], [Roy93], [Ash95].

In addition to the decision analysis strategy, we propose to guide requirements elicitation with a *reuse-based strategy* and an *improvement strategy*. Domain-based reuse strategies are already proposed in packaged solutions e.g. with ERPs. The idea is to provide with the operational systems a pre-defined DW that can be adapted. New requirements can be elicited in reference to the existing solution supplementary however guidance is needed to elicit those requirements and to concretise them into the adapted DW.

The idea behind the improvement strategy is that once developed, a DW should be validated. When it is found that the solution does not adequately support the decision-making processes then a new requirements analysis should be undertaken and changes made to the DW.

Once the requirements are elicited, they can be used to *design the DW model*. This can be done:

(i.) *by matching* them with existing data models of the operational data sources (after transforming them into an equivalent formalism).

(ii.) *by mapping* them into a DW model using mapping rules. Contrary to the former strategy, such mapping rules must be independent of the data available in the operational data sources.

(iii.) by directly using the information available at the data-source level, and without taking requirements into account. This *data driven bottom-up* strategy is the one proposed by [Inmo96]. It is only during the improvement phase that requirements are elicited to make the DW system evolve.

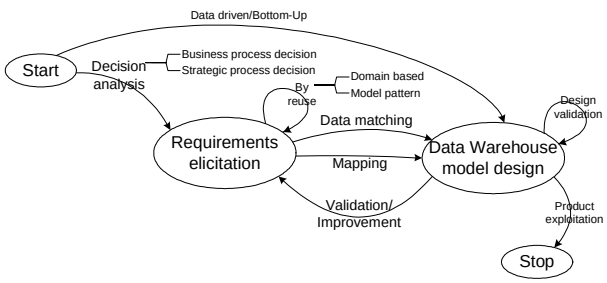



Figure 7: Process view of the position taken on DW development.

We have already developed parts of this approach. In particular, in [Fren2002] we proposed to use the requirements elicitation method *L'Ecritoire* to draw requirements in the form of business process goals. Guidance was proposed under the form of a semi-automated rule that analyses the scenarios documenting goals and maps them into an E/R DW model. Additional guidance was given to translate this E/R model into a dimensional DW model using Moody's approach [Moo2000].

The approach is far from being complete; it has to be improved, refined, and in-depth evaluation is needed. In particular, we would like to investigate its relevance to the issues found with DW development in the industry. Our first investigation showed us that:

- our approach is far from complete;
- it does not yet fully combine the advantages of the requirements-driven approach to those of the data-driven approach;
- it is time consuming; transitory E/R models are un-necessary and the guidance could directly aim at dimensional models; there is also a risk of loss of information in the transition from E/R to dimensional models;
- the goal-scenario approach used in *L'Ecritoire* is efficient to elicit requirements, but could be enriched for better adequacy to the specific issues of decision making in strategic and business processes; the decisions and the information needed to make decisions could for instance be made more explicit;
- this approach does not exploit the information provided by existing operational data-models. We believe that data sources could provide early in the process useful information on the availability of information to support decisions with the DW.

## 5. Conclusion

Data Warehouse development is not new. It started in the early of 80' and all large organizations now have DWs to support their strategic and business decisions. In a recent study [Nag93] of DW development and use experiences, "*managing the expectations of users and management*" was quoted as one of the six "*outstanding issues and challenges associated with data warehouse*".

Our goal in this paper was thus double :

(i.) to improve our understanding of the existing DW development approaches , and

(ii.) to lay out an innovative requirements-based DW development approach.

Our review of DW development is structured according to a framework. This framework is composed of four perspectives on DW development, each perspective identifies a number of options that are used in the existing approaches. The use of the framework in our review allowed to identify two large families of approaches: process-driven top-down approaches and data-driven bottom-up approaches. Both families of approaches have advantages and raise specific issues; there is to our knowledge no approach that tries to solve

these issues by combining the strategies used in each family. Of course, the framework has to be improved (by introducing more options in each view or by refining the existing options); more approaches should also be analysed for a more exhaustive review.
One major issue that we met in all approaches is the lack of guidance of the requirements engineering part of the DW development process. Our position is to guide requirements analysis by combining the DW user point of view and the operational system point of view. The former stands in the top-down analysis of decisions made in strategic or business processes while the latter introduces bottom-up analysis of the operational data used in business processes.
In the system perspective, we aim at an approach that is technology–independent: the DW data models should be usable to produce data cubes or tables indistinctively. This transition is however outside the scope of our proposal.

## 6. Bibliography


[Adam98] : C. Adamson, M. Venerable; Data Warehouse design solutions, Wiley 1998.

[Ant97] : A. Anton; Goal identification and refinement in specification of software-based information systems. Thesis presented to academic faculty, Georgia Institute of Technology, June 1997 URL: www.csc.ncsu.edu/faculty/anton/pubs/thesis/thesis.html

[Ash95] : Robert H. Aston, Alison Hubbard Ashton; Judgment and Decision-Making Research in Accounting and Auditing; Cambridge University Press; 1995.

[Bal98] : C. Ballard, D.Herreman, D. Schau, R. Bell, K. Eunsaeng, A. Valencic; Data Modeling Techniques for Data Warehousing. I.B.M.; February 1998.

[Boe2000] : Boehnlein, M., Ulbrich vom Ende; A.: Business Process Oriented Development of Data Warehouse Structures. In: Proceedings of Data Warehousing 2000, Physica Verlag (2000).

[Benj99] A. Benjamen, Une Approche Multi-démarches pour la modélisation des démarches méthodologiques, Phd Thesis, Université de Paris I, 1999.

[Bruck01] R. Bruckner, B. List, J. Schiefer. *Developing Requirements for Data Warehouse with Use Cases*. In Proc. Of 7th Americas Conference on Information Systems. 2001.

[Cabi98] : L. Cabibbo, R. Torlone; "A logical approach to multidimensional databases," Proc. 6th EDBT 1998, LNCS 1377, 183–197.

[Ben99] : C. Ben Achour; Extraction des Besoins par analyse de scénarios textuels; Janvier 1999.

[Craig99] : Robert S. Craig, Joseph A. Vivona, David Berkovitch; Microsoft Data Warehousing: Building Distributed Decision Support Systems; Wiley and Sons, March 1999.

[Crew] : L'approche CREWS-L'Ecritoire http://crinfo.univ-paris1.fr/CRINFO/CMB/HTML/Paris_chunk_index.html

[Dar91] : A. Dardenne, S. Fickas, A. Van Lamsweerde, "Goal-directed concept acquisition in requirements elicitation", Proc. 6th IEEE Workshop System Specification and Design, Como, Italy, 1991, 14-21.

[Dar93] : Dardenne A., Van Laamsweerde A. and Fickas S., "Goal Directed Requirements Acquisition", Science of Computer Programming, 20(1-2), pp3-50, 1993).

[Dow2001] : Dowling K., Schuff D., St Louis R., Star join schemas versus normalized relational schemas: Does it really make a difference to end-users?, Proceeding Americas conf. on Information systems 2001.

[Fren2002] : Frendi M.; Requirements engineering for building Data Warehouses; Diplôme de DEA information, interaction, intelligence, Université Paris XI, Orsay, France; Septembre 2002.

[Gao99] : Gao J., Requirements analysis concepts and principles, Ph.D. thesis, Jan 1999.

[Gar2000] : Gary Dodge, Tim Gorman, W. H. Inmon; Essential Oracle8i Data Warehousing: Designing, Building, and Managing Oracle Data Warehouses, John Wiley & Sons, September 2000.

[Gol98] : Golfarelli M., Maio D., Rizzi S., Conceptual design of Data warehouses from E/R Schemes. In: Proceedings of the 31st HICSS, IEEE Press (1998).

[Gom81] : Gomez F., Segami C., Delaune C., A system for semiautomatic generation of E/R models from natural language specifications, Data & Knowledge Engineering, Feb. 1998.

[Gree01] L. Greenfield. What Decision Support Tools are Used For. LGI Systems report. www.dwinfocenter.org. 2001.

[Grosz97] : G. Grosz, C. Rolland, S. Schwer, C. Souveyet, V. Plihon, S. Si-said, C. Ben Achour, C. Gnaho; Modelling and Engineering the Requirements Engineering Process : An Overview of the NATURE Approach; Requirements Engineering Journal, (2), 1997, p. 115-131.

[Hai95] : Haisten M., Designing a Data warehouse, InfoDB Vol. 9 Num.2; April 1995.

[Informix2000] : Informix Guide to Designing Databases and Data Warehouses, Informix press, January 2000.

[Inmo96] : Inmon W., Building the DW, Second edition, John Wiley and Sons, 1996.

[Jar2002], M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis; Fundamentals of Data Warehouses, Springer september 2002.

[Kim96] : Kimball R., The Data Warehouse toolkit, Wiley 1996.

[Kim97] : Kimball R., A dimensional modelling manifesto, DBMS, July 1997.

[Kim98] : Kimball R., L. Reeves, M. Ross, W. Thornthwaite ; The Data warehouse lifecycle toolkit: Export methods for designing, developing, and deploying data warehouses, Wiley 1998.

[Kim2001] : Ralph Kimball, Richard Merz, Margy Ross; The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, John Wiley & Sons, April 2002

[List02] : B. List, R.M. Bruckner, K. Machaczek, J. Schiefer; *A comparison of Data Warehouse Development methodologies. Case study of the process Warehouse.* In Proc. Of DEXA'02, LNCS 2453, Springer Verlag. pp203-215. 2002.

[Liu2001] : Liu L., Yu E., From requirements to architectural design using goals and scenarios, micon 2001.

[Mar] : Marsh V., Data warehouse case studies, InfoDB Vol. 9 Num.3.

[Moo2000]: Daniel L. Moody; From enterprise models to dimensional models: a methodology for data warehouse and data mart design; Proceedings of the international workshop on design and management of data warehouses (DMDW'2000), Stockholm, Sweden, June 2000

[Nag93] : Nagraj A., Richards D., Winsberg P., White C., DW in practice, Info DB, Summary 1993

[Opdhal98] A. L. Opdhal, K. Pohl, Workshop Summary REFSQ'98. Proceedeings of the Fourth International Workshop on Requirements Engineering: Foundations of Software Quality, REFSQ'98, Pisa, Italy Presses Universitaires de Namur, (eds, E. Dubois, A.L. Opdhal, K. Pohl), pp.1-11, 1998.

[Poe96] : Poe, V.: Building a Data Warehouse for Decision Support. Prentice Hall (1996)

[Potts94]), C. Potts, K. Takahashi, A. I. Antòn. Inquiry-based Requirements Analysis. IEEE Software, Vol.11, N°2, pp.21-32, 1994.

[Rod] : Rodero J.A., Toval J.A., Piattini M.G., The audit of the Data warehouse framework, Proceeding of the International Workshop on Design and Management of Data Warehouses (DMDW'99) Heidelberg, Germany, 14. - 15. 6. 1999

[Rol98a] : Rolland C., Souveyet C., Ben Achour C., Guiding goal modelling, Crews Report 98-27

[Rol98b] : C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, Dubois, P. Heymans, "A Proposal for a Scenario Classification Framework". Requirements Engineering Journal, Vol; 3, No. 1, pp. 23-47, 1998.

[Rol98c] C. Rolland; "A Comprehensive View of Process Engineering" in the Proceedings of the 10th International Conference CAISE'98, Lecture Notes in Computer Science 1413, B. Pernici, C. Thanos (Eds), Springer, Pisa, Italy, June 1998.

[Rol99] : Rolland C., Ralyté J., Plihon V., Method Enhancement with scenario based techniques, CAISE 99, 14-18, June 1999

[Roy88] : Royce W.W. "Managing the development of Large Software systems"; IEEE 70, 1988

[Roy85] : Roy B.; Méthodologie multicritère d'aide à la décision ; Economica ; 1985

[Roy93] : Roy B., Bouissou D.; Aide multicritère à la décision ; Economica ; 1993

[Sap98] : Sapie C., Blaschka M., Hölfing G., Dinter B., Extending the E/R model for the multidimensional paradigm. Springer Verlag 1998

[Sk94a] : Sk Yu Eric, Mylopoulos J., "Understanding "Why" in Software Process, Modelling, Analysis, and Design". In Proc. of the 16th International Conference on Software Engineering _ ICSE'94, Sorrento (Italy), May 16-21, 1994. IEEE & ACM.

[Sk94b] : Sk Yu Eric, Mylopoulos J., "From ER to AR modelling strategic Actor Relationships for Business Process Reengineering". In Proc. of the 13th International Conference on the Entity-Relationship Approach _ ER'94, Manchester (UK), December 13-16, 1994.

[Sk94c] : Sk Yu Eric, Mylopoulos J., "Towards Modelling Strategic Actor Relationships for Information Systems Development- with Examples from Business Process Reengineering". Proc. of the 4th Workshop on Information Technologies and Systems (WITS'94), Vancouver, B. C., Canada, December 17-18, 1994.

[Taw2001] : Tawbi M., CREWS l'Écritoire: un guidage outillé du processus d'ingénierie des besoins. Thèse de doctorat. Soutenue le 26.10.2001 à l'université Paris I.

[Theo99] : Theodoratos D., T.Sellis, Designing Data warehouses, 20/06/1999 DWDW'99, Heidelberg, Germany 14-15/06/1999

[West2001] : Westerman, P.: Data Warehousing using the Wal-Mart Model, Morgan Kaufmann (2001)

[Win96] : Winsberg P., Modelling the Data warehouses and Data marts, InfoDB Vol. 10 Num.3; June 1996

# Introduction and Application of a Lightweight Requirements Engineering Process Evaluation Method

Tony Gorschek
*tony.gorschek@bth.se*

Mikael Svahnberg
*mikael.svahnberg@bth.se*

Kaarina Tejle
*kaarina.tejle@home.se*

*Department of Software Engineering & Computer Science,*
*Blekinge Institute of Technology, PO Box 520, S-372 25 Ronneby, Sweden*
*Phone: +46 457 385000*

## Abstract

*The lack of an adequate requirements specification is often blamed for the failure of many IT investments. Naturally, the requirements specification is the product of a requirements engineering process.*

*Methods are required to evaluate the current requirements engineering process and identify where improvements are necessary making it possible to produce requirement specifications of high quality. Existing requirements engineering evaluation methods are often large, costly and time-consuming to use. Therefore we introduce a lightweight evaluation method, which we use to evaluate four industry projects. In this paper we outline the evaluation method, describe four industrial applications of the method and present an analysis of the findings.*

*The results suggest that the proposed evaluation method is useful and the studied cases to a large extent have adequate requirements engineering processes although many important aspects are missing from their respective processes.*

## 1. Introduction

According to certain sources the failure rate of IT investments is over 60% [1]. In addition problems introduced through the requirements engineering of a project accounts for something like 50% of the total debugging costs [2]. One of the major causes for this is said to be the lack of a complete and/or adequate requirements specification [2] [3] [4]. As the requirements specification is a direct result of the requirements engineering process it stands to reason that an inadequate specification is a result of a requirements engineering process with a low maturity level [5].

Although there exist several methods for assessing software development processes, (e.g. CMM [6] and ISO9000 [7]), few models focus on requirements engineering, and those that do to some extent (e.g. Sommerville & Sawyer [3] [8], CMMI [9] and SPICE [10]) are large and demand a fair amount of resources in order to be used. This is primarily due to the fact that they are exhaustive and often aimed at large scale evaluations of entire processes, e.g. the whole of a development process. This may not be a problem for large business enterprises, but small and medium size enterprises (SME) often have a limited budget for process evaluation and improvement.

Hence there is no simple and fast way to assess whether or not the requirements engineering processes in a company is inadequate today, or whether other causes are responsible for the aforementioned lack of a complete and adequate requirements specification.

To address this we need a fast and cost effective way to study the status of a requirements engineering process. This initial investigation into the status of requirements engineering in a company need not necessarily be faultless or even exhaustive, instead it should be good enough to give an indication to whether or not a problem exists, and to some extent where the problem areas reside. For this purpose a process evaluation model is introduced, the lightweight model of requirements engineering practices – the *REPM model* [11][1].

In this paper we describe the results from evaluating the requirements engineering process in four companies, using the REPM model. The contribution of this paper is thus as a pilot study into requirements engineering practices in industry as well as the introduction and industrial application of the REPM model. This application is described in detail to illustrate (i) *how* the REPM model was used during these evaluations, (ii) *what* results were obtained, and (iii) *how* the results may be interpreted.

The remainder of the paper is organized as follows. In Section 2 we describe the planning for the case studies conducted and the REPM model, and in Section 3 we describe the execution of the case studies. The results from the study is presented and discussed in Section 4, and the paper is concluded in Section 5.

[1] The REPM model can be downloaded at http://194.47.142.27

## 2. Planning

In this section we describe the context in which we conduct the study and the subjects involved, and present the design of the study as well as address validity issues. Furthermore the section holds a brief introduction to the REPM model itself.

### 2.1. Context and Subjects

The case studies are conducted in industry through in-person interviews by graduate students. The projects evaluated were concluded at the time of the study. This ensures that all of the stages in the RE process are completed at the time of the study.

The case studies involve a total of four companies. We chose to use two medium sized companies, i.e. under 500 employees, and two smaller ones (<150 employees). This to ascertain that the model was tested on both small and medium sized enterprises. The only criterion demanded from the companies selected was that they had projects featuring a customer-developer relationship. Two of the companies are situated in Sweden, and two in Ireland.

These companies were selected because we, or our local contact person on Ireland, had previous relationships with them and because they fit our criteria of large and small companies.

The subjects being interviewed were in senior positions in the projects being evaluated and had knowledge about requirements engineering, and more importantly extensive knowledge about the projects selected for evaluation. The projects' main responsible for requirements engineering was designated "project responsible" for each project evaluation session, but we did not put any limitations on the number of people that were present, as the positive effects of having a discussion with more than a single person in a senior position outweigh any risks that may be involved.

Each of the projects being evaluated was selected by the interview subjects. This made it possible for the companies to avoid questions dealing with projects very sensitive to being exposed to outside parties and also made it possible for them to choose a project that was concluded, of the right sort (customer-developer) and of interest to get evaluated. It is important to realize that the companies participating wanted the evaluation to take place in order to get an evaluation of their requirements engineering process. This alleviated the threat that a "good" project was chosen, i.e. it was in the companies own interest to get a accurate evaluation.

However this way of choosing projects introduces several threats to the study. The sampling is very much tainted by the person choosing the project and it may not be representative for the company at hand - it is a case of convenience sampling. In addition the choosing party can be biased, i.e. trying to portray the company in a positive way. We believe these risks to be small as the project responsible have nothing to loose in being honest and portraying the situation correctly – we informed them at a very early stage that the results of the evaluation would be treated as confidential. In addition the data gathered during the evaluations would have been less usable for the companies themselves if the evaluation was corrupted.

### 2.2. Study design

The study is based around a series of structured interviews (each interview represents one of the four cases). These structured interviews follow a model of requirements engineering practices that has been constructed by the authors, the REPM model [11]. In this section we briefly describe this model and how to interpret the results from it.

#### 2.2.1. Requirements Engineering Process Maturity Model

To assess the state of the requirements engineering processes in the companies we have constructed a model of good requirements engineering practices. This model, the REPM model, is further presented in A Method for Assessing Requirements Engineering Process Maturity in Software Projects [11]. The model is inspired mainly by the work done by Somerville et. al. in the REAIMS project [12] but also other existing work, such as Sommerville & Sawyer [3] [8] CMM [6], ISO9000 [7], Jirotka & Goguen [2] and Kotonya & Sommerville [5]. The REPM model was constructed by combining these sources with personal experiences and including additional experts from academia and industry in the construction process. All of these sources were thus used to determine what should be included in the model and at what maturity level.

For reasons of brevity it is impractical to include the entire REPM model in this paper. Instead, we describe it briefly below and a summary of the actions included in the REPM model is presented in Table 1. This is mainly intended to give a brief overview so that an opinion of the usefulness of the REPM model can be formed. For detailed information about the model and the contents please contact the authors.

The REPM model mirrors what should be done to obtain a consistent requirements engineering process. The individual tasks of which the model is comprised are called *actions*. Actions are the smallest constituents of the model and are in turn mapped to one of three main categories (called Main Process Areas or *MPAs* in the model). The MPAs are: *Elicitation*, *Analysis & Negotiation* and *Management*.

Every action resides on a certain *Requirement Engineering Process Maturity level* (REPM level) spanning from 1 to 5, where level 1 represents a rudimentary requirements engineering process and level 5 represents a highly mature process. The

actions on each level ensure a consistent and coherent requirements engineering process for the particular maturity level.

The maturity levels enable us to evaluate companies with respect to requirements engineering with a better accuracy than if we simply assume that all actions are equally important. By "base-lining" the actions into maturity levels we can assess that a particular company has potential for a certain maturity in its requirements engineering processes and it enables us to see what actions should be focused on to achieve the particular maturity level.

In Table 1 we see the different REPM levels, the goals associated with each maturity level and the actions presented under the relevant level. The actions are divided into groups by the MPAs of Elicitation, Analysis & Negotiation and Management. It is important to notice that achieving REPM level 1 means completing all the actions under REPM level 1, achieving REPM level 2 involves completing all actions under REPM level 1 and all actions under REPM level 2. Thus in order to achieve REPM level 5 one has to complete all actions presented in Table 1.

**Table 1. Action summary, REPM model**

| REPM Level 1 | |
|---|---|
| **Goals:**<br>1. Basic requirements specification | |
| | **Action Name** |
| ***Requirements Elicitation*** | |
| | Ask Executive Stakeholders |
| | Technical Domain Consideration |
| | Executive Stakeholders |
| | In-house Scenario Creation |
| ***Analysis and Negotiation*** | |
| | Analysis Through Checklists |
| ***Management*** | |
| | Document Summary |
| | Term Definition |
| | Unambiguous Requirement Description |
| | Information Interchange Through CARE |
| | Information handling Through CARE |
| **REPM Level 2** | |
| **Goals:**<br>1. Introduction of traceability<br>2. Introduction of validation of requirements<br>3. Introduction of a standardized structure for the documentation produced as a result of the requirements engineering process, i.e. the Requirements Document<br>4. Stakeholder identification | |
| | **Action Name** |
| ***Requirements Elicitation*** | |
| | Research Stakeholders |
| | In-house Stakeholders |
| | Scenario Elicitation - Executive Stakeholders |
| ***Analysis and Negotiation*** | |
| | Requirements Classification |

| REPM Level 2 | |
|---|---|
| ***Management*** | |
| | Requirements Origin Specification |
| | Document Usage Description |
| | Requirements Description Template |
| | Quantitative Requirements Description |
| | Prototyping |
| | User Manual Draft |
| | Requirements Test Cases |
| | Requirements Identification |
| | Backward-from traceability |
| | Backward-to traceability |
| **REPM Level 3** | |
| **Goals:**<br>1. Application domain and processes are studied and taken into consideration<br>2. All stakeholders are consulted<br>3. Dependencies, interactions and conflicts between requirements are taken into consideration<br>4. Requirement categorization and prioritization<br>5. Requirements re-prioritization<br>6. Peer-reviews<br>7. Risk assessment | |
| ***Requirements Elicitation*** | |
| | System Domain Consideration |
| | Operational Domain Consideration |
| | General Stakeholders |
| ***Analysis and Negotiation*** | |
| | Interaction Matrices |
| | Boundary definition through categorization |
| | Prioritizing Requirements |
| | Re-prioritization – New Requirements |
| | Re-prioritization – New Releases |
| | Risk Assessment – selected **OG1.03** |
| ***Management*** | |
| | Global System Requirements Identification |
| | Record Requirements Rationale |
| | Business Case |
| | Descriptive Diagrams usage - **Opt** |
| | System Models |
| | Requirements Review |
| | Forward-from traceability |
| | Volatile Requirements Identification |
| | User documentation |
| | System documentation |
| **Actions REPM Level 4** | |
| **Goals:**<br>1. Human domain consideration<br>2. Business domain consideration<br>3. Advanced risk assessment<br>4. Advanced traceability | |
| ***Requirements Elicitation*** | |
| | Human Domain Consideration |
| | Business Domain Consideration |
| | Scenario Elicitation - General Stakeholders |
| ***Analysis and Negotiation*** | |
| | Ambiguous Requirements refinement **Opt** |
| | Re-prioritization due to Change |
| | Risk Assessment – individual - **OG1.01** |
| | Risk Assessment - sets - **OG1.02** |
| ***Management*** | |
| | Environmental Models |
| | Requirements Inspection |
| | Forward-to traceability |
| | Management documentation |

**Table 1. Action summary, REPM model**

| Actions REPM Level 5 | |
|---|---|
| **Goals:**<br>1. Requirements reuse<br>2. Rejected requirements documentation<br>3. Architectural modeling<br>4. Advanced validation<br>5. Advanced requirements re-prioritization | |
| ***Requirements Elicitation*** | |
| | Requirements Reuse |
| ***Analysis and Negotiation*** | |
| | Re-prioritization with Regularity |
| ***Management*** | |
| | Rejected Requirements Documentation |
| | Architectural Models |
| | System Model Paraphrasing |
| | Version traceability |

### 2.2.2. Structured Interview Design

Based on the REPM model a checklist is constructed, which we use to guide the structured interviews. This checklist takes each action and formulates it as a question which can be answered with one of the three answers: *completed*, *uncompleted* and *satisfied-explained*.

The purpose of the satisfied-explained category is to take model compatibility into consideration. Companies carrying out projects in special environments unlike the traditional customer-developer environment may deem certain actions unnecessary and have compelling reasons for this opinion.

An example can be a company where the developer and the customer both are specialists in a certain domain and hence "speak the same language". The need for extended clarification and validation of requirements may not be needed, e.g. the construction of prototypes can be omitted. Satisfied-explained thus denotes an action that is not completed but the organization doing the evaluation deems the action not applicable to their project.

The organization doing the evaluation makes the distinction when an action is to be considered satisfied-explained. It is important to notice that an action should not be deemed satisfied-explained for reasons like lack of time, lack of money, lack of know-how or just "did not think of it".

### 2.2.3. Results from Interviews

The results of a project evaluation are presented as four tables, one for each MPA and one summarizing all of the results. An example of such a table is found in Table 2. This table is an example of a summary table for all three MPAs for one project evaluation. We see that the actions for each REPM level are listed separately, and that e.g. REPM level 2 contains a total of 14 actions, of which 9 are completed and 4 are satisfied-explained (14 – (9+4) = 1 is uncompleted).

**Table 2. Example of Project Evaluation result**

| *REPM level* | *Total Actions* | *Completed* | *Satisfied-Explained* |
|---|---|---|---|
| *1* | 10 | 8 | 2 |
| *2* | 14 | 9 | 4 |
| *3* | 19 | 11 | 4 |
| *4* | 11 | 4 | 2 |
| *5* | 6 | 1 | 4 |

To assist in the interpretation of the results, we suggest that the results are also presented as graphs, as the example in Figure 1. The graphs represent a simple overview of the results and it is recommended that all results be presented in this way as well. However, due to lack of space, we are unfortunately unable to do this in this paper. The absence of the diagrams should thus not be interpreted as a statement against their worth. A complete list of diagrams can be viewed in [11].

In the graph, the solid gray line represents the total number of actions, the solid black line represents a summary of all actions that are completed and satisfied-explained. The dashed line represents the actions that are actually completed. The area between the dashed line and the solid black line denotes to what extent the REPM model is inapplicable to the project being evaluated (called *model lag*), the area between the solid black line and the gray line represents the area of possible improvement of the RE process.

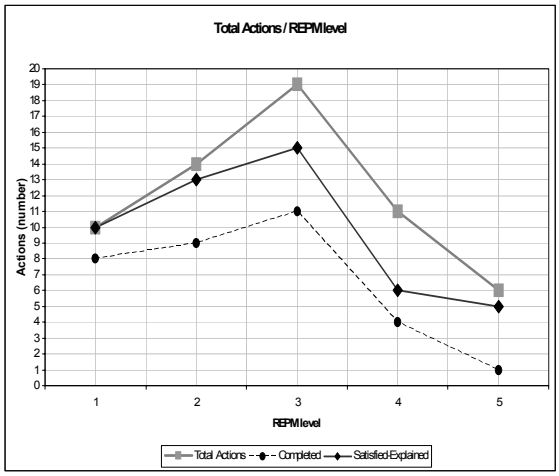



**Figure 1. Example of result diagram**

The tables and graphs are interpreted as follows. Starting with the first REPM level, if all actions are completed or satisfied-explained, i.e. the solid black line overlays the solid gray line, this level of maturity is achieved. This would mean, if no more REPM levels are achieved, that the company has a consistent and complete requirements engineering process of a low maturity level. This is then repeated for each of the REPM levels. Note that all lower REPM levels

must also be completed before a certain REPM level is achieved.

In Table 2 and Figure 1, for example, we see that REPM level one is achieved and only one action more is required to achieve REPM level two, whereas four more are required to achieve level three. Strictly speaking, this project would be considered to be on REPM level 1, but as only one action is necessary to get it up to level two this is the level that we think this company should aim for in a first step. This would ensure a consistent requirements engineering process that is fairly basic but may be sufficient for this company's needs.

### 2.3. Validity evaluation

In this section we discuss the threats to this investigation. We base this on the discussion of validity and threats to research projects presented in Wohlin et al. [13].

#### 2.3.1. Conclusion validity

Four cases are inadequate for statistically sound generalization purposes. However, this is not the main intention of this study. The main intention of this study is to serve as a pilot project to see whether it is possible to assess the state of requirements engineering practice using the REPM model, and to possibly provide some early results as to what areas within requirements engineering are subject to most problems.

The checklist used for the case studies (structured interviews) was validated through preliminary testing and proofreading by several independent parties, this to avoid factors like poor question wording and erroneous formulation.

Each case study interview was conducted without any break. Thus the answers were not influenced by internal discussions about the questions during e.g. coffee breaks.

The sampling technique used, i.e. convenience sampling, can pose a threat to the validity of the investigation. The companies selected are not necessarily representative of the general population of software development companies, on the other hand there is no evidence that they are not.

#### 2.3.2. Internal validity

Prior to the interviews the subjects had access to the REPM model so that they could acquaint themselves with the model (see Section 2.2.1) In addition the REPM evaluation checklist was explained before it was used. These steps were taken to avoid problems with maturity, i.e. that the first questions would act like a learning process and thus be answered with a different presupposition than the remaining questions. The preparation described was executed in an identical manner for each of the four case studies. We did not encounter any problems with this due to the fact that the subjects being interviewed had similar backgrounds, i.e. experience in the field of requirements engineering and software development.

#### 2.3.3. Construct validity

The REPM model itself can be a threat to the investigation. If the model is unsatisfactory when it comes to content and/or structure it follows that the result of each case study (and thus the investigation) may be threatened to the same extent. Another important point is the usage of the model by the investigators and by the interview subjects. Using the REPM model as a tool in a manner not consistent with the initial intent may diminish the result obtained through the usage.

To reduce these risks the REPM model and the checklist were validated before the case studies were conducted. The validation was two-fold. First the model was scrutinized by the creators and an independent expert from academia. The second round of validation was conducted by an independent expert from industry. During the validation process the models structure and contents was modified when appropriate.

## 3. Operation

Having prepared the model, the checklist and a way to present the results, the next step consists of contacting companies and actually conducting the study. In this section we describe this, starting with the preparations and continuing with experiences from the actual execution.

### 3.1. Preparation

A copy of the REPM Model was sent to the interview subjects in advance. This to give all people involved a chance to study the model and its constituents in order to prepare them for the work during the project evaluations and to note any questions they wished to address.

In addition to this the participating companies were asked to choose a project to be evaluated and instructed that the project should be of a customer-developer type, i.e. not an internal development project but featuring an external customer. This due to the fact that the REPM model at this stage of its development is adapted primarily for these types of projects. In addition we asked the person(s) mainly responsible for the requirements engineering process for the projects selected to be present as the interview subjects at the project evaluation session.

The interviews in Ireland were preceded by personal contact and emails to prepare for the evaluation.

### 3.2. Execution

At the initial stage of the interview the basic layout of the evaluation was explained. Information about how the Project Evaluation Checklist version 1.4 (can be obtained from the authors) is structured, and how it is connected to the REPM model was provided. In addition we asked the people evaluating the project to "think-aloud" when answering the questions. An important thing we clarified was the concept of satisfied-explained actions, i.e. that the model may not suit all projects and that this concept was introduced to compensate for that fact.

All interviews were made on-site and in person. We find in-person interviews beneficial in several ways. It is often possible to extract more information as well as avoiding misunderstandings. However there can also be negative aspects like the interviewer influencing the interviewees [14] [15]. By strictly adhering to the pre-planned structure of the interview and only deviating when further explanations were necessary, we believe that the risk that this has occurred is low.

Each case study interview was estimated to take between 1.5 to 2 hours. This was based on an earlier test interview conducted using a software engineering post graduate student. The time spent on the interviews mirrored our estimation.

During the interviews we were somewhat surprised by the tendency for the subjects to engage in discussions related to the topic of requirements engineering in general and more specific topics as the questions reflected certain actions. We tried to steer the interview towards completing the checklist but did not categorically refuse to discuss matters. More often than not the discussions helped to clarify different points, and often the discussions were more or less one sided, i.e. the interview subjects discussed matters to clarify their own trail of thought to themselves before answering a question. We only got involved in the discussion when necessary, trying not to influence the answers by adding information not already present in the model and/or in the checklist. We took this decision to ensure that each interview subject received the same amount of information, so as not to change the conditions of any one interview in comparison with another.

### 3.3. Data validation

During the interview the subjects discussed their answers aloud as the questions were answered. This helped us catch misunderstandings, and if necessary we clarified the relevant questions in an effort to elicit a relevant answer to the question at hand.

All of the answers are included in this investigation. Our goal is to ascertain the status of the requirements engineering process in companies today, therefore all answers are relevant. Four case studies, and thus four interviews, were conducted. They are all presented in the study, no case studies were disqualified due to reasons such as lack of conformity.

## 4. Analysis and interpretation

In this section we present the results from the case studies (each case study is represented by one project). We do this in four tables, one for each project/case study. As mentioned earlier, it is easier to analyse the data if the results are presented as graphs in the same form as the examples in Figure 1. Due to lack of space, we are unfortunately not able to do this here, and instead refer you to [11], where these graphs are available.

We refer to the different projects as project alpha, beta, gamma and delta respectively. Projects gamma and delta are from what we consider small companies.

### 4.1. Presentation of results

In this section we present the results from the case studies (each case study is represented by one project). We do this in four tables, one for each project/case study. We refer to the different projects as project alpha, beta, gamma and delta respectively. Projects gamma and delta are from what we consider small companies.

In Table 3 we see the data from project alpha, and in Table 4, 5 and 6 we see the data from project beta, gamma and delta, respectively.

These tables are read as follows. There are four groups of data, each containing three columns. These columns list the total number of actions on each REPM level, the number of actions completed and the last column lists the number of actions satisfied-explained for the REPM level. The first group of three columns is a summary of the following three groups, which are the actions within the MPAs Elicitation, Analysis and Negotiation and Management, respectively.

We see that no project fulfils even REPM level 1. However, if we count those where only one or two actions are missing, we see that project alpha is close to REPM level 1, project beta close to level 2, project gamma close to level 1, and project delta close to level 2. We also see that project beta has potential for being on level 5, and that the others probably would benefit from completing the remaining actions for their respective REPM level and aim for REPM level 3. Level 3 represents a requirement engineering process that is fairly advanced and not just the bare basics, while still being streamlined and suitable for many software organizations.

**Table 3. Project alpha**

| | Action Summary | | | Elicitation | | | Analysis & Negotiation | | | Management | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REPM level | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained |
| 1 | 10 | 8 | 0 | 4 | 3 | 0 | 1 | 0 | 0 | 5 | 5 | 0 |
| 2 | 14 | 8 | 2 | 3 | 1 | 1 | 1 | 1 | 0 | 10 | 6 | 1 |
| 3 | 19 | 12 | 2 | 3 | 2 | 1 | 6 | 4 | 0 | 10 | 7 | 0 |
| 4 | 11 | 6 | 1 | 3 | 1 | 1 | 4 | 2 | 0 | 4 | 3 | 0 |
| 5 | 6 | 2 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 4 | 2 | 0 |

**Table 4. Project beta**

| | Action Summary | | | Elicitation | | | Analysis & Negotiation | | | Management | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REPM level | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained |
| 1 | 10 | 8 | 1 | 4 | 4 | 0 | 1 | 1 | 0 | 5 | 3 | 1 |
| 2 | 14 | 10 | 2 | 3 | 2 | 1 | 1 | 1 | 0 | 10 | 7 | 1 |
| 3 | 19 | 15 | 1 | 3 | 3 | 0 | 6 | 5 | 1 | 10 | 7 | 0 |
| 4 | 11 | 9 | 1 | 3 | 2 | 1 | 4 | 3 | 0 | 4 | 4 | 0 |
| 5 | 6 | 4 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 4 | 2 | 0 |

**Table 5. Project gamma**

| | Action Summary | | | Elicitation | | | Analysis & Negotiation | | | Management | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REPM level | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained |
| 1 | 10 | 8 | 0 | 4 | 3 | 0 | 1 | 0 | 0 | 5 | 5 | 0 |
| 2 | 14 | 7 | 0 | 3 | 2 | 0 | 1 | 1 | 0 | 10 | 4 | 0 |
| 3 | 19 | 13 | 1 | 3 | 3 | 0 | 6 | 3 | 0 | 10 | 7 | 1 |
| 4 | 11 | 3 | 2 | 3 | 0 | 1 | 4 | 2 | 0 | 4 | 1 | 1 |
| 5 | 6 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 4 | 1 | 0 |

**Table 6. Project delta**

| | Action Summary | | | Elicitation | | | Analysis & Negotiation | | | Management | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REPM level | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained | Total Actions | Completed | Satisfied-Explained |
| 1 | 10 | 9 | 0 | 4 | 4 | 0 | 1 | 0 | 0 | 5 | 5 | 0 |
| 2 | 14 | 9 | 4 | 3 | 2 | 1 | 1 | 1 | 0 | 10 | 6 | 3 |
| 3 | 19 | 11 | 4 | 3 | 3 | 0 | 6 | 1 | 3 | 10 | 7 | 1 |
| 4 | 11 | 5 | 3 | 3 | 3 | 0 | 4 | 0 | 1 | 4 | 2 | 2 |
| 5 | 6 | 3 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 4 | 2 | 0 |

## 4.2. Analysis of Results

First, we always recommend a homogenous REPM level across the three MPAs. The reason for this being mainly dependencies. If a project's requirements engineering process resides on REPM level five when it comes to the MPA of *Requirements Elicitation* but only on level two in the other two MPAs this would not be a consistent and coherent requirements engineering process. Moreover, in many cases actions under one MPA are dependent on other actions being completed under another MPA.

In this respect, none of the projects succeed. The process area where all of the projects fail is that of *Requirements Management*. There are several reasons for this. Requirements Management is the largest MPA, i.e. housing the largest number of actions, and thus there is more to complete for this area. Moreover, this area contains actions which are notoriously difficult to accomplish in an efficient way, such as traceability and change policies. We are also quite stringent in how we think a requirements document should be structured and the individual requirements written which is reflected in the REPM model under requirements management where the actions for requirements documentation are included.

Assuming that all organizations should strive for at least REPM level 3 there are several actions that two or more projects need to complete. These are:

- In *Elicitation*, the action *In-house Scenario Creation (REPM level 1)*. This action is concerned with creating scenarios for elicitation purposes but to only consult the developers in the projects. As there are other actions where stakeholders are consulted and none of the projects fail to do this, the companies may think that if stakeholders are consulted it is not necessary to consult the developers.
- In *Elicitation*, the action *Research Stakeholders (REPM level 2)*. Who the stakeholders are is researched during this action. This can help identify groups that are affected by the requirement, and thus the basis for gathering information about a requirement is wider if the action is performed. That this action is not done may stem from a trust that the customer provides information regarding all stakeholder groups as this is in the best interest of the product and hence the customer. The development companies may forget who it is in the customer-developer relationship that has experience with requirements engineering.
- In *Analysis and Negotiation*, the actions *Analysis through Checklists (REPM level 1)* and *Interaction Matrices (REPM level 3)*. Analysis through checklists refers to having standardized checklists that are used to ensure certain standardized qualities of each requirement. Interaction matrices are constructs that can help catch requirements dependencies, conflicts or other interactions between requirements. These are very simple tools that can help catch problems with

requirements at an early stage, and we are surprised that this is not done in all companies.

- In *Analysis and Negotiation*, the action *Risk Assessment – Selected (REPM level 3)*. This action is part of a group of actions, of which one, but only one, must be completed. The remaining actions in this group are on REPM level 4, so this is the least possible amount of risk management that should be done with respect to requirements analysis and negotiation. What it means is simply that certain requirements are, based on intuition, identified as being extra prone to risk and then a risk assessment is performed for these requirements.
- In *Management*, the action *Quantitative Requirements Description (REPM level 2)*. With this action we mean that all requirements – especially quality requirements – should be specified such that they can be measured. Quality requirements are a notoriously difficult subject that is still the subject of much research. It is thus not surprising that the companies reflect the amount of uncertainty in the research community regarding quality requirements and their quantifiability.
- In *Management*, the actions *Requirements Review (REPM level 3)* and *User Manual Draft (REPM level 2)*. These two actions are concerned with validating the requirements. The second of these two actions suggests that an initial user manual is written based on the requirements, which serves as an informal walkthrough of the requirements. That this action is not completed in two of the companies may simply be that they have not thought of it.
- In *Management*, the action *Document Usage Description (REPM level 2)*. Different user groups often use the requirements document in different ways. A document usage description is a manual aimed at aiding different users of the document, helping them use and navigate the document. This may often be forgotten as it seems obvious to those writing the document how to read it.
- In *Management*, the action *Backward-to Traceability (REPM level 2)*. This action links the design and implementation back to the requirements. That this is not done may be a result of waterfall-inspired development methods. If previous development steps such as requirements engineering is never re-visited, what is the point in tracing the requirements backwards. Moreover, many companies may not update the requirements specification if the design and implementation deviates from what is specified.

### 4.3. Summary and Interpretation of results

Only 9 of the 43 actions up to REPM level 3 are not completed by two or more projects. However, many of the uncompleted actions are quite severe, and we are surprised that they are not done in all companies. For example, risk assessment seems to be a neglected area, and interactions between requirements do not appear to be mapped, which of course can cause severe problems if there are in fact conflicting or volatile requirements. Also, we are surprised that companies still spend time and effort on stating requirements without providing a measurement, as there is no way of telling when the requirement is fulfilled if this is not done. As mentioned quality requirements, where this is most often missed and required the most, is a subject with much research focus and methods are still needed for quantifying these quality aspects of software systems.

The size of a company does not seem to have a direct correlation to the maturity of the requirements engineering process to the extent believed initially. If we look at project alpha and beta approximately 68% (for project alpha) and approximately 85% (for project beta) of all the actions are fulfilled. The numbers for gamma and delta (the smaller companies) are 60% and 81%, respectively. At an initial glance the numbers seem very much comparable, even if the larger companies have a small advantage. To see the whole picture however one has to take the amount of actions deemed satisfied-explained under consideration. In project alpha, beta and gamma the number of actions in this category seem to be a fairly constant 8%, whereas in the case of delta the number of satisfied-explained actions is 20%. It is vital to understand whether the latter figure is the result of model inapplicability (See Section 2.2.1) or not, in order to find out whether project delta is representative as a small company project or not.

If the number has another reason than model inapplicability this may indicate a distinct line between medium sized and smaller companies when it comes to requirements engineering. The projects in this study were chosen with the same criteria in mind, and thus should be generally compatible when it comes to model applicability. A widened study, involving more companies is necessary to understand whether this distinction between medium sized and smaller companies exist.

The MPA of *Requirements Management* is generally the one needing most improvements, i.e. the MPA with most actions not completed. There are several reasons for this. As mentioned Requirements Management is the largest MPA, i.e. housing the largest number of actions, and thus there is more to be completed. Another reason may be that the actions under this particular MPA are fairly advanced, i.e. on

a higher REPM level relative to the total number of actions. An example of this is that there are more actions under the MPA of *Requirements Management* on REPM level 5 than there are under the other MPAs in total.

## 5. Conclusions

The main purpose of the investigation presented above was to (1) to give an idea of the problem scope pertaining to requirements engineering practices in industry, and (2) test a method for quickly ascertaining the status of requirements engineering in companies. This was done through the design of the REPM model, which in turn was used for the investigation.

Firstly, if we look at results gathered from the four cases they may of may not be generalizable to a larger set of companies, i.e. if replications of this study show similar results, there are several ways to react to this, depending on what audience one belongs to.

For researchers, the question seems to be to find effective and attractive methods for risk assessment and for requirements management. For development companies, the reaction should be to first assess whether the very simple actions are done, as we believe this would vastly enhance the quality of the resulting products and, above all, reduce the risks involved.

For companies participating in an evaluation such as the ones presented above, the results should be analyzed to decide whether or not there is an indication of a problem. The second step is then to study the evaluation, examine the improvement suggestions and the conclusions drawn for inaccuracies and/or neglected parts. It is also important to scrutinize the actions deemed to be under the category of Satisfied-Explained to ensure that they are put there for the right reason. After the evaluation review a plan should be constructed based on the improvement suggestions and improvement consequences. This plan should state what measures should be taken. One way proceed is to order a more comprehensive and exhaustive examination of the requirements engineering process using the results from the REPM model evaluation as a first insight to where the problems may reside. Another way to go is to use the REPM results purely as a basis for deciding on a more thorough investigation. In the latter case the results from the two investigations could be validated through a subsequent comparison.

Secondly, if we look at the REPM model itself and the use of it in the four cases there are two main parts, namely the gathering of the data using the Project Evaluation Checklist (see section 0), and the evaluation and interpretations of the results (see section 4).

The main point in constructing the REPM model was to get a fast, easy and cost effective evaluation of a requirements engineering process. During the industrial application of the model the gathering of the data took no more than eight person-hours per project. The subsequent analysis of the data took approximately 30 to 40 person-hours per project. This gave us at most 48 person-hours in cost for the REPM evaluation of a project, which can be considered to be fast.

In this paper a fairly detailed description was presented of how the industry cases (projects) were evaluated. During the gathering of data the actions were deemed as belonging to one of three categories, i.e. Completed, Uncompleted or Satisfied-Explained (see section 0). The diagram example presented in section 2.2.3 gives information of how result diagrams could be constructed to give an overview of the process (as well as parts of the process), and how the data could be evaluated and interpreted is presented in sections 4.2 and 4.3. One could go so far as to claim that the collection of the data is fairly easy using the Project Evaluation Checklist. The analysis and interpretation of the results on the other hand does demand some expertise in the area of requirements engineering, although the complete REPM model offers some insights into the area.

Whether usage of the REPM model is cost effective or not is of course dependent on how accurate and exhaustive the gathered results are. As mentioned before emphasis was put on speed and ease, not exhaustiveness. If we look at the results from the case studies quite a few indications point to areas of possible improvements. However this does not give any information about things potentially missed. In our experience the REPM model provides an indication of problem areas that should be scrutinized further. An REPM evaluation could be used to develop a plan for what steps to take in order to improve a requirements engineering process, though it is important to realize that the REPM model is lightweight, and an evaluation taking 48 person-hours is bound to overlook issues.

It is also important to notice that the REPM evaluation is project based. If a company has a "generic" (typical) project to evaluate one instance may suffice. On the other hand most companies may need to evaluate more than one project in order to get an accurate (and more exhaustive) picture of their RE process, this is especially true for companies without a standardized and repeatable process.

As the results from the cases were presented to each of the companies the general view was that some of the results gathered were already more of less known, but for the most part the REPM evaluation results were seen as valuable insights in the respective company's process offering an indication that further study was warranted.

### 5.1. Future work

This study serves as a pilot study as far as ascertaining (1) the problem scope pertaining to requirements engineering practices in industry, which is the reason why only four companies have participated. In order to gain real understanding of what is done and what is not done in industry today, an extended study involving a larger set of companies needs to be performed. We also need to further understand the relation between what is considered an inadequate requirements specification and the connection between this and the maturity of the requirements engineering process at large.

We encourage others to design and execute similar investigations on their own, as there is a clear and present need for knowledge about the status of requirements engineering in industry today.

(2) If we look at the REPM model there is a need for further evaluation, refinement and validation. There is also a need to validate results gathered with the REPM model by using another approach (other model/method). This would provide further information on of how accurate and cost effective the REPM model is, as well as how the REPM model can be improved upon.

(3) A study of how the results from a REPM evaluation can be used in further and more detailed RE process evaluation is also warranted. I.e. how to use the REPM results obtained to take the next step, either towards further evaluation or maybe even a the creation of an adequate improvement plan.

## 6. References


[1] C. Clegg et al, *The performance of information technology and the role of human and organizational factors*, OASIG Study, University of Sheffield, UK, 1996.

[2] M. Jirotka and J. Goguen, *Requirements Engineering – Social and Technical Issues*, Academic Press, London, UK, 1994.

[3] I. Sommerville and P. Sawyer, *Requirements Engineering – A Good Practice Guide*, John Wiley & Sons, Chichester, UK, 2000.

[4] J. Van Buren and D.A. Cook, *Experiences in the Adoption of Requirements Engineering Technologies*, in CROSSTALK - The Journal of Defense Software Engineering (11)12, pp. 3-10, 1998.

[5] G. Kotonya and I. Sommerville, *Requirements Engineering – Processes and Techniques*, John Wiley & Sons, Chichester, UK, 1998.

[6] M. C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading Massachusetts, USA, 1995.

[7] *The TickIT Guide – Using ISO 9001:2000 for Software Quality Management System*, *Construction, Certification and Continual Improvement*, Issue 5.0, 2001, http://www.tickit.org/ Last checked: 2002-10-01.

[8] P. Sawyer, I. Sommerville, S. Viller, *Capturing the benefits of requirements engineering*, in IEEE Software 16(2), pp. 78-85, 1999.

[9] *CMMI*® Product Development Team, *CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing Version 1.1* (CMMI-SE/SW/IPPD/SS, V1.1), *Staged Representation.* Technical Report CMU/SEI-2002-TR-012.

[10] http://www.sqi.gu.edu.au/spice/, March 2003.

[11] T.B. Gorschek, K. Tejle, *A Method for Assessing Requirements Engineering Process Maturity in Software Projects*, Master's Thesis MCS-2002:2, Blekinge Institute of Technology, Ronneby, Sweden, 2002.

[12]http://www.comp.lancs.ac.uk/computing/research/cseg/projects/reaims/index.html , May 2003.

[13] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering – An Introduction*, Kluwer Academic Publishers, Boston, USA, 2000.

[14] S. Körner, *Statistisk Slutledning (eng:"Statistical Deduction),* Stundentlitteratur, Lund, Sweden, 1985.

[15] C. Robson, *Real World Research*, Blackwell Publishers Ltd., Oxford, UK, 1995.

[16] N. Malhotra, *Marketing Research, An Applied Orientation, 2nd Edition*, Prentice Hall, Upper Saddle River NJ, USA, 1996.

# From Process Model to Problem Frame – A Position Paper

Karl Cox
*CSE, University of New South Wales,*
*National ICT Australia*
*karlc@cse.unsw.edu.au*

Keith Phalp
*ESERG,*
*Bournemouth University, UK*
*kphalp@bmth.ac.uk*

## Abstract

*Jackson's Problem Frame (PF) approach presumes that some knowledge of the application domain and context has been gathered so that a Problem Frame can be determined. However, the identification of aspects of the problem, and hence, its appropriate 'framing' is recognized as a difficult task. One way to help describe the problem context is through process modelling. Once contextual information has been elicited, and explicitly described, an understanding of what problems need to be solved will emerge. However, this use of process models to inform requirements is often rather ad-hoc. Hence, this position paper proposes guidance for directly deriving Problem Frames from business process models. The paper presents an outline method for PF derivation, and argues why this may be useful to the developer. Finally, the authors discuss the issues involved in attempting to derive a more formal mapping between Problem Frames and business process models.*

## 1. Introduction

In recent years many software developers have produced models of client business processes [1] as an up-stream software development phase [2]. However, although it is generally agreed that such process models are valuable in informing requirements, the exact nature of how the process model maps to subsequent (requirements) phases is less clear.

Some authors have suggested what might be termed 'process approaches' [3] to development methods, but these tend to adopt particular design tactics, where the process model replaces more 'popular' design notations. Others have attempted to examine how process models might map to existing approaches, for example, mapping process models to formal approaches [4] or more latterly, to use cases [5]. Although there is merit in these approaches, one of the problems is that in methodological terms they are implementation dependent. That is, they assume a particular design approach, whether process driven or more conventional (such as the UML) [6].

However, it would be particularly useful if process models could be used to help partition and inform requirements, without assuming a particular subsequent approach to design. This leads on to the idea of combination with Problem Frames [7]. Indeed, one of the premises of the PF approach is that the proper 'framing' of the problem should suggest appropriate notations both for requirements capture and design [8]. In addition, it is also clear that whilst simple single frame problems may often be correctly identified, the framing of real-world problems is often far from trivial [9].

Therefore, in this paper we attempt to show how process models might be used to inform the derivation of Problem Frames. This would then allow process knowledge to be used within requirements phases, and would aid the, non-trivial, process of 'framing' problems. As an exemplar notation, we use Role Activity Diagrams (RAD) [10] a well-regarded process modelling notation.

### 1.1. Related work on problem frames

Related work on Problem Frames (PF) has focussed on identifying what techniques are most useful to eliciting and documenting requirements and specifications once the PF is known [8, 11], and in attempting a formalization of the PFs [12]. Current research is exploring the role PFs have with aspects of software architecture [13]. These works view the PF as already determined and present ways to help subsequent development. Sikkel et al. [14] propose a variant on the PF. They present a decision tree to help determine what kind of business solution a company might need, such as whether to opt for a COTS product or to bolt on new functionality to the current system. With regard to process modelling and problem frames, there is, to our knowledge, no research currently being conducted.

## 2. From process models to context diagrams: a good starting place?

The step from process models to context diagrams is not new [15]. Indeed, to map from a Role Activity Diagram (RAD) to Jackson's variant of the traditional context diagram is straightforward. Table 1 shows the components of both diagrams and how they map.

**Table 1. Mapping RAD to context diagram**

| RAD | Jackson Context Diagram |
|---|---|
| Role | Domain of Interest / Machine |
| Interaction | Interface |
| Action | - |

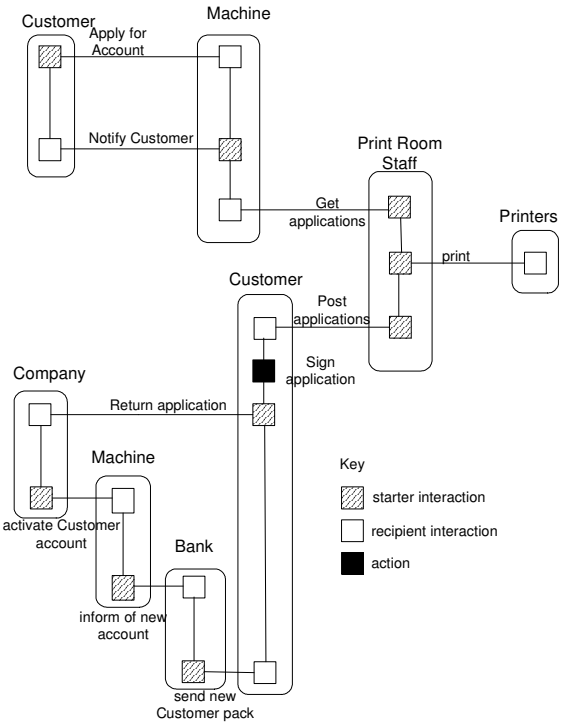



**Figure 1. Example role activity diagram**

As an example, figure 1 describes a RAD of a simplified process of applying for an online share trading account. This is mapped to a context diagram (figure 2).

Essentially the diagrams (figs. 1 and 2) are the same. In fact, it can be conjectured that there is a loss of information if we describe by context diagram alone. There is no explicit representation of the internal actions of the domains that are vital to the success of the business. In figure 1, actions within roles are made apparent (by black squares) – the Customer role action 'sign application'. What is also required is a textual description of each domain (not detailed in this paper).

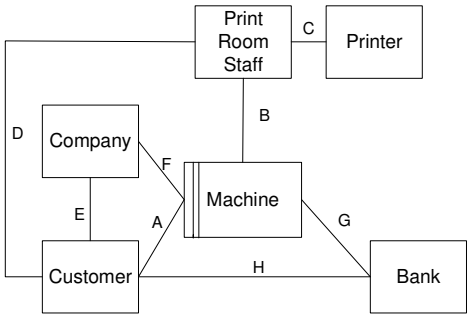



**Figure 2. Context diagram**

The interfaces between the domains can be made explicit and are described in table 2. For example, for interface A, CU!{…} means that the Customer domain is responsible for the interaction with the Machine domain.

**Table 2. Interfaces on the context diagram**

| Interface | Description |
|---|---|
| A | CU! {apply}<br>MA!{notification} |
| B | PRS!{retrieve application} |
| C | PRS!{print application} |
| D | PRS!{post application} |
| E | CU!{return application} |
| F | CO!{activate account} |
| G | MA!{new account details} |
| H | BA!{welcome} |

This indicates which domain is responsible for what, that is, what role they play in the process. The next step ought to be to consider how to determine the PFs. But there is a problem here.

## 3. Problems mapping to problem frames?

The context diagram, as derived from the process model, does not explicitly show the information the problem frames might need. For example, if we have a Workpiece frame, where in the context diagram or the process model is there a design domain (other than the machine)? The process model does not necessarily describe what type of problems there might be – just the way that the business works for this particular scenario.

As such, it is not clear whether we are describing fundamental problem frames or decomposed 'process-

oriented' problem frames. Hence, there is a risk of simply following the process through onto subsequent frames without consideration of the 'big picture'. That is, bypassing the fundamental problem frames for the finer details of transforming a process model into a set of 'process frames'.

## 4. The frames

What, then, can be derived from a process model that will determine the problem frames? Figure 1 shows the Customer creating an online trading account. Thus, this is a Workpiece problem frame (figure 3).

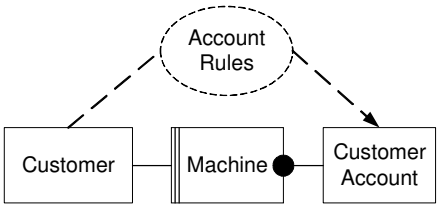



**Figure 3. Workpiece frame**

It can be seen that the Customer Account domain in figure 3 is not apparent in either the context diagram or the RAD, though it is a fundamental (design) domain in this problem. This shows that the mapping to the context diagram from the RAD, though easy, does not necessarily provide all of the information for the problem frame. (The black dot indicates the Customer Account domain is found within the Machine itself – it is a design domain.) However, we can elicit this domain by further exploring the nature of the account creation activity. This can be achieved, for instance, by decomposing the RAD further, revealing the details of the interaction. We can also examine the interfaces within the context diagram. Validating the account creation process with the necessary stakeholders will verify that the Customer Account domain is right – and of its legal status.

### 4.1. Further potential problem frames

There are at least two other frames identified through further analysis of the problem domain (not shown): the Commanded Behaviour frame allows the Customer to manipulate their Customer Account online – transfer funds, buy and sell stocks and shares. The third frame would be an Information Frame. The Customer can check the current stock prices on the Web Application.

These three core frames might need to be decomposed further. For instance, how does the Web Application show the stock prices? Perhaps a Connection frame is required here.

## 5. Outline of a mapping

Process models do not necessarily convey the information required to determine PFs, even when mapped into context diagrams, because domains key to the success of the PF approach are not always apparent, particularly if the missing domains are design domains – such as in a Workpiece. This makes the step from a process to a PF view more complicated. We thus propose initial guidelines to assist in this task. The guidelines are rudimentary and are based upon our experiences thus far. We will formalise them as our research continues. Table 3 describes the steps in this (iterative) process.

**Table 3. RAD to problem frame**

| Step | Action |
|---|---|
| 1 | Describe Role Activity Diagram |
| 2 | Identify outcomes of interactions |
| 3 | Identify potential domains from outcomes |
| 4 | Identify potential rules that govern interactions |
| 5 | Identify problem frames |

The first step is to describe a process model (in our case a Role Activity Diagram (RAD)). We note that companies might have existing process models in other notations, but choose, for now, to limit our guidance to RADs.

Step two identifies the outcomes of interactions between roles. In the above example, an outcome of the 'apply for account' interaction is the creation of a new customer account.

As step three indicates, this outcome is then considered as a potentially new domain. Each is asked:

- Is the outcome something that will be used, altered or referred to a number of times from different perspectives? In other words, a domain of interest. That is, it is not simply a transient outcome. (The Customer Account will be manipulated or referred to through its lifetime by the Customer, the Bank, and the Print Room Staff in different scenarios.)
- We use Bray's domain taxonomy to determine its type [11]. Is the domain a design domain? Inert? (We can say that the Customer Account is something that will be created and held within the machine and will not change its state independently.) Other questions are: is the domain static (not changeable with time in any way), reactive (predictable), completely controllable (programmable), partially predictable (biddable) or entirely uncontrollable (autonomous)?

Step four explores what rules are in place to control interactions. For instance, when the Customer applies for the account, they have to enter required financial information, such as current bank account details. The financial credit status of the Customer, we discover, is electronically checked by connecting to a credit agency. Legal requirements also govern the application procedure and these have to be discovered. The machine then steps the Customer through a precisely defined application procedure.

Step five then identifies the PF. For example, once the RAD is described, the Customer Account has been identified as an outcome of the interactions, and the legal and financial requirements (the rules) are determined, we can state: We have an inert, design domain (Customer Account) – created by the Customer and considered a legal document (legal / financial rules). We, therefore, have a Workpiece problem frame.

## 6. Discussion

This position paper outlines a way to derive (appropriate) Problem Frames from process models. The method is illustrated by describing the derivation of a Workpiece frame from a Role Activity Diagram. It is shown that although traditional mapping from business processes to context diagrams might be viable, as an intermediary step towards a problem frame, such a mapping has potential pitfalls because important design domains are often missed. Therefore, we can bypass this step and consider the problem frames direct from the process model. Key to eliciting further domains, vital to the identification of the problem frames, is exploring the interactions between roles for outcomes (potential domains) and rules (potential requirements or contraints governing use or control of the domains).

### 6.1. Further work and potential issues

Our goal is to provide a complete, formalised set of guidelines to help determine problem frames from process models. However, there are some 'mapping' issues to be addressed.

Is it necessary for a complete mapping to be produced? We are not saying that Role Activity Diagrams and PFs are isomorphic, in the way UML sequence and collaboration diagrams are. For example, it is likely that in moving from the RAD to the PF, some information is lost. When changes are made to the requirements some of these may impact the business model, but (if they do not concern the interfaces between the domains or the rules governing the frame) the frames may be unaltered. Hence, it may be necessary to consider a multiple mapping among business model, problem frame and requirements. Indeed, similar issues have been described among process models, use cases and class diagrams [5].

This also brings into question whether a direct mapping is most beneficial or whether it may be necessary to use an intermediate notation. Again lessons may be drawn from process modelling where notations, such as POSD, have been used in this manner [2].

Finally, we note, that although well regarded, the existing PFs are seen as a starting point, and that certain contexts may yet suggest the need for further frames.

## 7. References


[1] P. Henderson, "Software Processes are Business Processes Too", *Third International Conference on the Software Process*, IEEE Comp. Soc. Press, Reston, Virginia, USA, Oct 1994.

[2] K.T. Phalp, "The CAP Framework for Business Process Modelling", *Information and Software Technology*, 40 (13) 1998, pp. 731-744.

[3] Warboys, B, Kawalek, P, Robertson, I. and M Greenwood, *Business Information Systems*, McGraw Hill, 1999.

[4] G. Abeysinghe and K.T. Phalp, "Combining Process Modelling Methods", *Information and Software Technology*, vol. 39, num. 2, 1997, pp. 107-124.

[5] K.T. Phalp and K. Cox, "Guiding Use Case Driven Requirements and Analysis", *7th Int. Conf. on Object-Oriented Information Systems*, Springer, LNCS, Calgary, August 27th-29th 2001, pp.329-332.

[6] Jacobson, I., Booch, G., and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[7] Jackson, M., *Problem Frames*, Addison-Welsey, 2001.

[8] Kovitz, B., *Practical Software Requirements*, Manning, 1999.

[9] K. Phalp, and K. Cox, "Picking the Right Problem Frame - An Empirical Study", *Empirical Software Engineering Journal*, 2000, 5(3), pp. 215-228.

[10] Ould, M., *Business Processes*, Wiley, Chichester, 1995.

[11] Bray, I., *An Introduction to Requirements Engineering*, Addison-Wesley, 2002.

[12] D. Bjorner, S. Koussoube, R. Noussi, and G. Satchok, "Michael Jackson's Problem Frames: Towards Methodological Principles of Selecting and Applying Formal Software Development Techniques and Tools", *1st IEEE Int Conf on Formal Engineering Methods*, IEEE Comp Soc Press, Hiroshima, Japan, 12-14 November, pp. 263-270.

[13] J. Hall, M. Jackson, R. Laney, B. Nuseibeh, and L. Rapanotti, "Relating Software Requirements and Architectures using Problem Frames", *RE'02*, IEEE Computer Society Press, Essen, Germany, Sept 2002, pp. 137-144.

[14] K. Sikkel, R. Wieringa, and R. Engmann, "A Case Base for Requirements Engineering: Problem Categories and Solution Techniques", *REFSQ'2000*, Stockholm, Sweden, 5-6 June 2000.

[15] Britton, C. and J. Doake, *Software System Development: a gentle introduction*, McGraw-Hill, 1993.

# A Requirements-based Framework for the Analysis of Socio-technical System Behaviour

Jon G. Hall
Open University, Milton Keynes (UK)

Andrés Silva
Universidad Politécnica de Madrid (Spain)

## Abstract

*Requirements Engineering's theoretical and practical developments typically look forward to the future (i.e. a system to be built). Under certain conditions, however, they can also be used for the analysis of problems related to actual systems in operation. Building on the Jackson/Zave reference model [2] for requirements and specifications, this paper presents a framework useful for the prevention, analysis and communication of designer and operator errors and, importantly, their subtle interactions, so typical in complex socio-technical systems.*

## 1. Introduction

There are three system 'dimensions' that come into play in the analysis of socio-technical systems: these are system structure, system behaviour and human-computer interaction. Many techniques for the analysis and prevention of accidents in socio-technical systems focus on only one or two of these dimensions but not on all three at the same time. However, it is well known that accidents combine all three [1, 5].

In this paper we propose an analytical framework for socio-technical systems. The framework is based upon a dynamic view of the reference model (RM) for requirements and specifications [2]. The formal approach of the RM, complemented with the semi-formal approach of Jackson's Problem Frames [3], provides a clear distinction between descriptions of the world (or environment of the system) $W$, the requirements $R$ and the specifications $S$ of a solution machine. This specification $S$ contains descriptions expressed exclusively in terms of shared phenomena between the machine and its environment. For a machine to satisfy the requirements, $W \wedge S \models R$ must hold.

With the purpose of including the analysis of system-operator interactions, our framework also contains elements that resemble those of Norman's *Mental Models* [6]. However, Norman concentrates on the design of the system in non-formal terms; our location in mission- (including safety-)critical systems requires reasoning capability above the *ad hoc*. As with Norman, we work with three different viewpoints: two mental (corresponding to design and operating stakeholders), one physical (the system in operation). Those stakeholders can hold discrepant views of $W, S, R$ [8] that, through their interactions, can lead to undesirable physical situations. Our intention is to place stakeholders within the same spatio-temporal framework to allow proper analysis of them and the physical system individually and severally. Although we admit design and operating *teams*, throughout this paper we use the singular, for consistency.

## 2. Designer and operator viewpoints

Our framework is summarised in Figure 1 which should, initially, be seen as three concentric squares. We explain, first, the middle square, corresponding to the actual physical execution of the system. According to the reference model, the system "moves" the environment from a state described by $W$ at time $t$ (abbreviated to $W^t$ in the figure) to a state[1] at $t+1$ ($W^{t+1}$, which, ideally, satisfies requirements $R$), as indicated by the mapping $W^t \rightarrow W^{t+1}$. This step is decomposed into three: *(i)* a "sensor"

[1] We assign unit time to the change for simplicity only, it could, in fact, be any delta.

step: the state of the world is input to the software ($W^t \rightarrow S^t$), *(ii)* a software step: the software produces output from its input ($S^t \rightarrow S^{t+1}$) and *(iii)* the "actuator" step: the software output effects some change in the world ($S^{t+1} \rightarrow W^{t+1}$). The other squares correspond to the views of our identified stakeholder: outermost the operator's view (subscript $o$); innermost the designer's view (subscript $d$). In this way, we enable the systematic forwards or backwards analysis of any discrepancies between the operator's and designer's views of a step (or sequence of steps) and/or the actual step(s) that took place.

A *delta expression* (shown in the in the figure as a decorated $\Delta$, i.e. $\Delta_o^{SW}$) represents the gap between the operator's view of a step and the step itself. So, for instance, $\Delta_o^{SW}$ represents the operator's view of an "actuator" step against reality. Discrepancies in a step can either be in the step itself, or derived from discrepancies between actual states and designed or perceived ones. This second class of discrepancies we label with decorated $X$s.

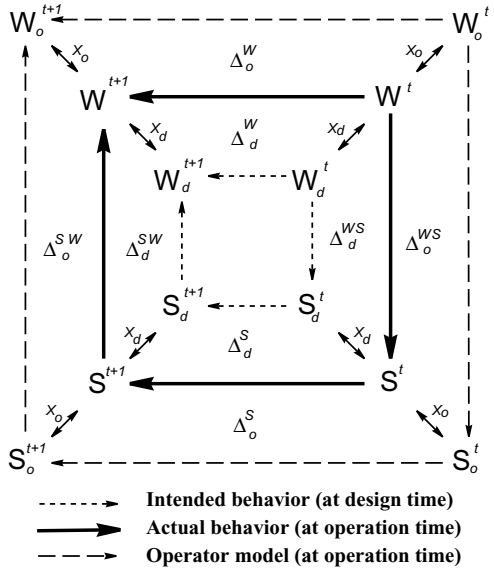



**Figure 1. Resume of the framework**

One of the immediate advantages that our framework provides is its taxonomic capability. For example, from the delta expressions, *operator errors* can be classified into one of:

$\Delta_o^W$ the operator misconstrues an environmental step;

$\Delta_o^{SW}$ the operator misconstrues an actuator step;

$\Delta_o^S$ the operator misconstrues the sensor/actuator linkage;

$\Delta_o^{WS}$ the operator misconstrues an sensor step.

This is a more granular and detailed view of Norman's mental model (mis)constructions [6]. Also, similar taxonomies exist for misconstrued states (the $X$'s) and for design errors (the $\Delta_d$'s), not shown here for reasons of space.

## 3. Case study: the chemical reactor

This section shows, through a real-world example, how our framework may be used in the analysis of the chemical reactor explosion [4]. The requirements for a chemical reactor, a schematic for which appears in Figure 2, included the clause that when a component in the plant reported a problem, the software should send a warning message to the operators and stop executing. On one bad day, the operator sent an order to the software to open the catalyst (with the aim of increasing the output of the reactor). The program was instructed to, first, open the catalyst and, second, open the flow of cooling water, to regulate the reaction. An accident occurred when a component gearbox reported low oil levels to the software, just after opening the reactor, causing the software to stop. As a consequence, opening the cooling water was never performed. The temperature of the reactor increased resulting in an explosion.

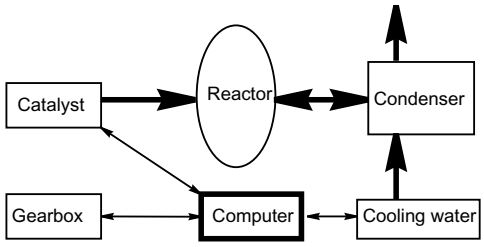



**Figure 2. Chemical reactor schematic**

For our analysis, we will use a formal approach, related to individual traces of behaviours in, respectively, the real world (i.e., the actual behaviour), the designer's view and the operator's view. However, less formal approaches could also be used under our

framework (for example, based on problem frame concerns[3]).

In the simplest of safety and liveness terms, the situation $\{catalyser\_open, \neg water\_open\}$ should never occur. Figure 3 represents, step by step, the sequence of events. Analysis goes backwards in time, from top to bottom. The three columns represent the designer's view, the actual events and the operator's view of those events. States are shown in angle brackets, where shared phenomena are under the line.

There is a $X_d$ difference, shown at the top of the figure, between the (actual) hazardous state and the state envisioned by the designer. If we trace back in time the sequence of events we find that, previously, in the $S^{t+1}$ state, there are some differences but there are no differences in the previous one ($S^t$). So it is in this $S^t \rightarrow S^{t+1}$ step where a $\Delta_d^S$ actually happened (i.e., the software did not achieve its intended action, even when sensors, actuators and the operator behave correctly). Analyzing this step, what we find here is a design decision that proved to be wrong: the atomicity of the operations of opening the catalyser and the water when, actually, the water was never opened, as the second column shows.

Please note that our framework would admit an alternative analysis if the designers view ignored the possibility of *unit_needs_service* in $W^t$. In this case, a difference would appear in the $W^t$ stage, indicating that an expectation $X_d$ of the designer was wrong, but at the end, the conclusion would be the same: the program has been poorly designed.

The actual behaviour of the world, machine and operator is expressed by the relation $\models$ on the $W$, $S$ and $R$ rules, where

| | |
|---|---|
| $R$ | `if` $(catalyser\_open)$ `then` $water\_open$ |
| | `if` $(unit\_needs\_service)$ `then` $(warn\_operator; STOP)$ |
| | `if` $(req\_catalyser\_open)$ `then` $catalyser\_open$ |
| $S$ | `if` $(openc_{sen})$ `then` $(openc_{act}; openw_{act})$ |
| | `if` $(req\_service_{sen})$ `then` $(warn\_operator_{act}; STOP)$ |
| $W$ | `if` $(openc_{act})$ `then` $catalyser\_open$ |
| | `if` $(warn\_operator_{act})$ `then` $warn\_operator$ |
| | `if` $(openw_{act})$ `then` $water\_open$ |
| | `if` $(unit\_needs\_service)$ `then` $req\_service_{sen}$ |
| | `if` $(request\_catalyser\_open)$ `then` $openc_{sen}$ |

This model satisfies the RM in the sense that $W \wedge S \models R$ holds[2]. On the other hand, the designed behaviour ($W_d, R_d$ and $S_d$) differs from this model. Although $W_d = W$ and $R_d = R$, $S$ changes to:

| | |
|---|---|
| $S_d$ | `if` $(openc_{sen})$ `then` $[openc_{act}; openw_{act}]$ |
| | `if` $(req\_service_{sen})$ `then` $(warn\_operator_{act}; STOP)$ |

where $[a; b]$ indicates that $a$ followed by $b$ should be atomic, i.e., that no event should come between their occurrence. Similarly, rules and relation $\models_o$ could be provided for the operator; however, due to the informality of the operator (it is a *biddable domain* in Jackson's terms[3]) we prefer to analyse it informally as follows:

Referring back to Figure 3, from the point of view of the operator the first departure from actual behaviour happens in $W^t$: the operator ignores that, at the same time he is requesting the catalyzer to open, the gearbox signals a request for service. The operator continues, under the assumption that there is no abnormal behaviour, and is therefore never aware of the hazard, as can be seen in the $W^{t+1}$ row. Although it is not "the solution", this suggests that a proper feedback mechanism for the gearbox request for service is missing.

Although we have been able to present only a simple analysis, we claim that our framework has the potential to raise awareness about many real design and operator errors as well as suggesting solutions by which they can be avoided. To support this claim, consider for instance, the long sequence of $\neg water\_open$ facts without the operator being aware of it, even after his request was made. The analyst could ask why this happens, if it has some importance or not, and how can be avoided in future designs. Also, an analysis of mixed causes (intermingled design *and* operator errors) can be done, like when even subtle design errors lead to wrong information in a display that lead the operator into taking a bad decision [5].

## 4. Conclusions and Further Work

We have presented a framework into which many aspects of socio-technical systems fit, and within which their interrelationships become clear. With its strong foundations in the reference model, from

[2]The reader can check this by chasing round Figure 1 the sequence of events.

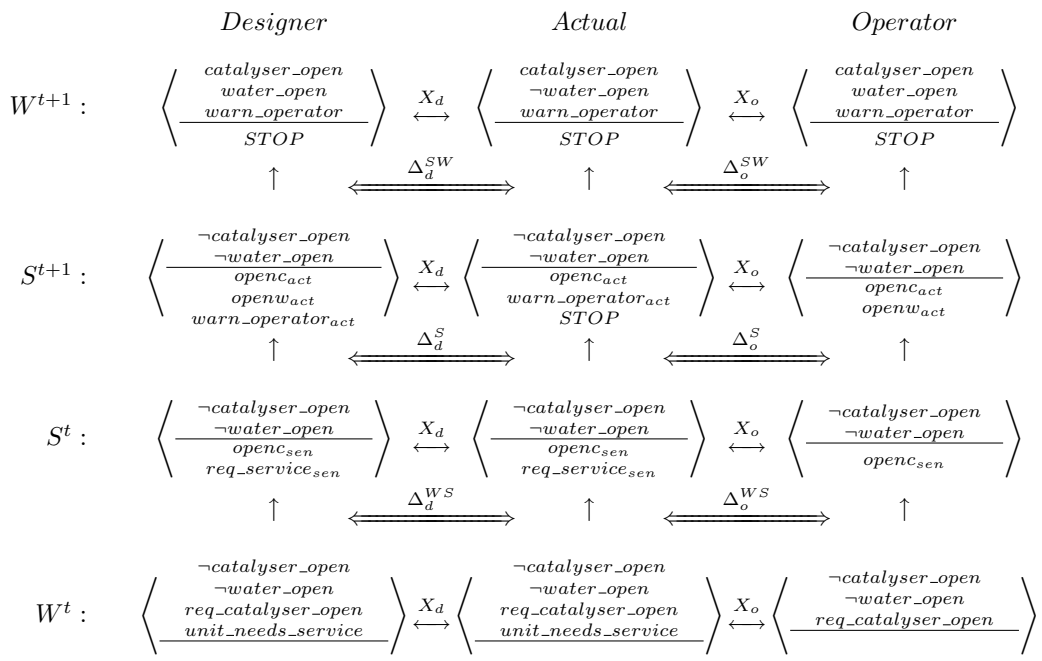



**Figure 3. Analysis of the chemical reactor accident**

a safety engineering point of view, we aim in further papers to show how widely used techniques (fault tree analysis (FTA), HAZOP and Event Tress (ET), for instance) can be embedded within our framework. For instance, our Framework provides the means to structure FTA analysis as, in FTA, the backward chaining sequence from the root of the tree (the hazard) can be guided by stepping backwards in the cycle shown in Figure 1. For each step, the operator and designer view can be interleaved to study their interactions, improving former proposals like [1]. In HAZOP, on the other hand, our framework provides a general structure for HAZOP meetings [7]. In this way, HAZOP keywords can be applied to each small step in the cycle, both from a designer point of view and/or from an operator point of view. A similar argument can be provided with regard to ET or other techniques. In this way, our framework provides a coherent view that underlies hazard analysis techniques, also integrating the interaction between operator and design errors.

This framework provides the basis of our future work, whose mission is to bring a complete and coherent view of the misbehaviours in socio-technical systems. Another approach we are exploring is the potential use of our framework for storing and distributing accident-related information.

## References


[1] C.F. Fan and Chen W.H. Accident sequence analysis of human-computer interface design. *Reliability Engineering and System Safety*, (67):29–40, 2000.

[2] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, 2000.

[3] M. Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[4] T. Kletz. *Computer Control and Human Error*. Institution of Chemical Engineers, 1995.

[5] N. Leveson, L. D. Pinnel, Sandys S.D., S. Koga, and J.D. Reese. Analyzing software specifications for mode confusion potential. In *1st ACM SIGSOFT Symp. on the Foundations of Software Engineering*, Dec. 1993.

[6] D.A. Norman Cognitive Engineering. In D.A. Norman, S.W. Draper, *User Centred System Design*. LEA Associates, New Jersey, 1986.

[7] F. Redmill, M. Chudleigh, and J. Catmur. *System Safety: HAZOP and Software HAZOP*. Wiley, 1999.

[8] A. Silva. Requirements, domain and specifications: A viewpoint-based approach to requirements engineering. In *International Conference on Software Engineering 2002 (ICSE 2002)*, 2002.

# The Simulator; Another, Elementary Problem Frame?

Ian K Bray
*Bournemouth University*
*ibray@bournemouth.ac.uk*

Karl Cox
*CSE, University of New South Wales,*
*National ICT Australia*
*karlc@cse.unsw.edu.au*

## Abstract

*Problem framing is recognised as an important new technique for describing the problem context in order to identify and then apply the most appropriate modelling and description techniques to developing different kinds of software system. Jackson has proposed five distinct problem frames and variants on them. However, Jackson makes no claim that this is an exhaustive list. In order to better model certain problem domains, an addition to the published set of elementary problem frames is proposed. We present the Simulator problem frame and describe through example why the Simulator is not solely a variant of the other frames and should be considered a problem frame in its own right.*

## 1. Introduction

Since being proposed by Michael Jackson [1], problem frames have been recognised as having great significance for the development of software systems. As David Garlan observes [5], "Understanding and using problem frames will likely become an essential skill of all good software system designers".

The approach is grounded upon fitting problem frames to the given problem domain; the closer the fit, the greater the advantage that may accrue.

Jackson originally identified five elementary problem frames; workpiece, control, information, connection and transformation. As has been illustrated in several works (for example, [2], [3]) more complex problems can, generally, be fitted to composite frames composed of two or more interacting, elementary frames. Whilst Jackson explicitly made no claim to have identified all the elementary frames, as far as is known, to date, no others have been added. However, to better fit a particular class of problem, an additional frame is now proposed.

## 2. Problem frames and Requirements practice

Methodologists often suggest that general methods are better than specific and the "one size fits all" notion is superficially attractive. However, it is flawed; as Jackson [1] puts it, "the value of a method is inversely proportional to its generality". Use cases, for instance, might well prove useless for describing the functionality of a simulacrum.

Nonetheless, enormous effort gone into trying to wrap one "standard" approach around all problems and the uptake of general methods is widespread. Meanwhile, little effort has gone into assessing the effectiveness of applying specific problem frames to particular problems and there is a currently lack of published evidence. In the only empirical study conducted on problem frames thus far, Phalp and Cox [8] show that it is relatively straightforward to determine single problem frames but less easy to determine the frames (and their respective weightings) for multi-frame problems. There are, as far as we are aware, no published industrial case studies, of the application of problem frames. However, given a free choice of methods, many of our students adopt this approach in their final individual and group projects. (Indeed, one such project sowed the seed for this paper.)

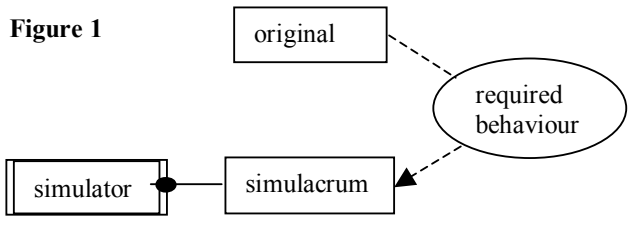
## 3. The simulator problem

This class of problem is characterized by the construction of an artifact (a realised domain) which exhibits behaviour that, to a required degree, mimics the inherent behaviour of some dynamic, real world original.

The simulation bears some resemblance to an external reality but this is established only through the requirements; there is no communication (no phenomena are shared) between the original and the simulation during operation. Even so, the original should not be ignored, it is central to the problem and investigation and description of the original would be a vital element of the requirements engineering.

The problem frame may be pictured as in figure 1.

**Figure 1**



The original is a "real world" domain. It is suggested that the original must be dynamic (or it would have no behaviour to simulate) but that is the only constraint[1]. In particular, it may be controllable or autonomous. The required behaviour will, clearly, be based upon the inherent behaviour of the original but (as illustrated in the following examples) it is likely that there will be approximations or simplifications.

The simulacrum is the realized domain which behaves in accordance with the required behaviour. It is an embodiment of the selected event responses and exists only within the simulator, the configured machine that brings about the desired behaviour of the simulacrum.

The simulator frame is likely to be used in conjunction with other frames. In particular, although it could be trivial, an information system may well be required in order to view the state of the simulacrum. (This is akin to the simple workpiece frame requiring an information system to provide feedback [5], page 98.) It is also possible for the inherent behaviour of the simulacrum to be controlled by an associated control frame. However, these associated frames are optional; the simulator can stand alone and this prohibits its classification as a partial frame.

## 4. Examples

Consider a simulation of a planetary system. The original has certain properties (mass and orbital period of bodies etc.) which can be detailed in a problem domain description. The requirements will state with what degree of fidelity the simulation should mimic the original. There will inevitably be some inexactitude and one might, for example, elect to treat the bodies as point masses and constrain movement to a single plane.

Once implemented, the simulation should comply with the requirements. However, during its operation, the model does not communicate (share phenomena) with the original and at any time, the state of the model may be very different from that of the original.

A second example concerns the simulation of ocean waves. The problem domain description would detail such matters as the propagation and decay of actual waves. The requirement would specify how closely the simulation must mimic these characteristics and the implementation should comply with those requirements.

The simulation could be used to make predictions about maximum wave heights, to provide test conditions for (simulations of) boats or simply as a piece of art. In none of these cases, however, does the simulation communicate (share phenomena) with the original or represent any actually occurring state of the original.

Thousands of examples of simulation problems appear in the literature, these include:

- simulation of chemical etching
- simulation of buildings (for earthquake research)
- simulation of vehicles and drivers (as part of a driver training environment)
- simulation of electronic circuits
- simulation of brewery fermentation (as a test rig for a new brewery controller)

## 5. Differences from other frames

The simulator shares certain characteristics with other elementary frames but has particular features which justify its separate classification. In part, the differences hinge upon the distinction between inherent behaviour and controlled behaviour. The inherent behaviour of a domain is a given. It is not open to negotiation but may be described more or less correctly. A complete description of the inherent behaviour of a domain encompasses all its possible behaviours. Bjorner et al. [4] define this in terms of a basis function which determines the range of its observable variables.

The simulator seeks to approximate the inherent behaviour of the original. If it is possible and desirable to impose a controlled behaviour over the inherent behaviour, this would be the subject of a separate (control) frame.

Jackson [5], page 339, offers a ranking of problem frames in terms of their "depth in the world outside the computer". This ranking runs from the transformation frame (the shallowest) to the control frame (the deepest) and may be viewed in terms of the degree of binding between the machine and the real world. Within this scheme it is suggested that the simulator problem fits between the workpiece and the information frame; the behaviour of the simulacrum is based upon some real world original (which, therefore, requires study) but during operation, there will be no communication between the simulator and the original.

## 6. Why isn't it a workpiece problem?

Both workpiece and simulator contain a realized domain (artifact). In the workpiece case, the realized domain is necessarily inert and it changes subject only to the whim of an editor. The simulacrum can be self modifying and is constrained by requirements that establish, to a greater or lesser degree, some mimicry of an external reality.

[1] There may be some argument as to whether a model of a static domain (for example, a road in a traffic simulation) constitutes a simulation in its own right.

An inert domain may be simulated but, in the absence of any stimuli, will do nothing. It is hard to imagine the circumstances under which the simulation of an inert domain would, of itself, be worthwhile.

An inert domain simulation could, however, have a role as a partial frame. It would embody the event responses that would allow some other machine to exert control over the simulacrum. This approximates to the decomposition of the workpiece problem into two partial frames; a simulator and an "editor".

However, there remains a difference in that, with the simulator, there does exist an external original which, through the requirements, imposes constraints upon the behaviour.

Furthermore, the simulator has a far greater range; spontaneous and, even, autonomous domains may be simulated. Even in the absence of any external stimuli, such domains may "freewheel" and continually exhibit behaviour. The bodies in the planetary simulation, for example, continue in their orbits without the need for any external stimulation and, in a brewery simulation, a vat of wort will continue to ferment if left alone.

## 7. Why isn't it an information problem?

An information system provides information about some reality. In order to do this, the information system will frequently contain a model of that reality (what Jackson refers to as the Dynamic Model, Partial Frame [6]). However, in that case, the correspondence between the reality and the model must be maintained (within acceptable limits) during the operation of the information system.

This is not the case with a simulator. At any given time, the state of the simulacrum need have no correspondence with the state of the original. It follows that the simulator frame diagram shows no connection between the original and the simulator; there is no communication, no shared phenomena. For example, a vehicle in a traffic simulation may crash but this certainly does not mean that any original has crashed.

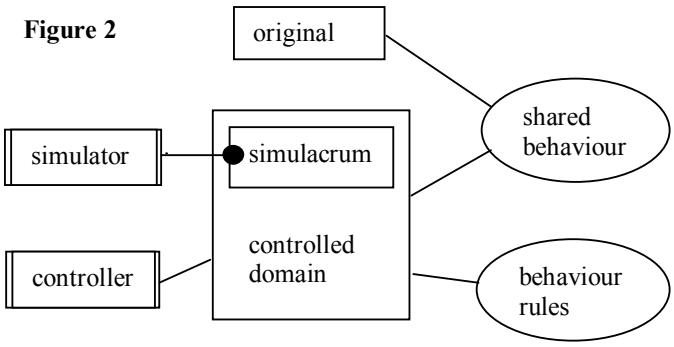
In practical terms, correspondence between the simulacrum and the original is determined only by the requirements; there is no updating of correspondence at run-time. That said, a simulation could form a partial frame in support of an information system *if* its purpose were to help derive reports about the actual state of the original *and* if there were some mechanism for ensuring an ongoing correspondence with the original. But this need not be the case.

## 8. Why isn't it a control problem?

The control problem is characterised by the machine exerting control over some external reality.

In the case of the simulator, the external reality (the original) is not controlled (not even connected). There could, of course, be an associated control system for which the simulacrum *is the reality*. This could be depicted as below (figure 2), but the simulator and the controller are separate systems. This situation commonly occurs where a simulator is built in order to provide a test environment for a control system (without any risk of accident cause by defective control of the original).

**Figure 2**




However, a telling fact is that it is possible to simulate an autonomous domain (such as a weather system) which, by definition, cannot be controlled.

Even where control is possible, it is not *necessary* to have any planned control. Simulators for two or more systems could be interconnected such that each provides stimulation for the other and, without any requirements for controlled behaviour, they could be left to interact and their emergent behaviour can simply be observed. Certain simulations of evolving systems fit this pattern.

## 9. Is the Simulator a lexical domain in disguise?

Jackson [5] describes a lexical domain as a "physical representation of data - that is, of symbolic phenomena" (p84). The lexical domain combines a mix of causal and symbolic phenomena. Because of this necessity for causal phenomena and its need to change the state of data in the domain of interest, we can say that a lexical domain interacts with the problem entity it is representing. However, the simulator does not interact with its original entity in any way. It does not change the state of its real world representation. It only mimics it or simulates behaviours beyond the capability of its original counterpart.

Jackson also presents another point of view that a lexical domain does not need to be causal, but that it is just "a structure of symbolic phenomena" (p84). For the Simulator frame, the lexical domain could be the combination of the simulacrum and the simulator machine. This lexical domain would then represent the original real world domain. However, since the original

domain is, necessarily, dynamic (e.g. controllable or autonomous) and lexical domains are often static representations of data, we argue that the simulator has particular qualities that distinguish it from a typical lexical domain. With the advent of machine learning/genetic algorithms and AI, there is no reason to suppose that the simulator will never go beyond the capabilities and behaviours of the original domain (perhaps to explore ways in which to enhance the original). For instance, in the simulation of fault tolerance of house safety structures under extreme conditions such as an earthquake or a tornado. To await an actuall tornado or earthquake to assess the effectiveness of the building materials or architecture would be impracticable.

## 10. The fantasy problem frame

A further possibility is that, within a simulation, the original is imaginary. This situation occurs with certain games, computer animations and the like.

An imaginary object, for example, a dragon, is endowed with certain behaviour and a machine is constructed that exhibits that behaviour. As before, it is quite possible that there will also be a connected system that provokes the simulation (with or without the aim of evoking a certain desired behaviour).

## 11. Conclusions and further work

Any development that allows closer fitting of problem frames to problem domains should prove advantageous and the proposed new frame appears to offer the possibility of achieving a better fit for certain types of problem.

Development of case studies will allow further investigation. In particular, the nature of the various domains can be more precisely determined and checklists (along the lines of those provided in [3], page 82 and in [7] page 103) for the investigation and documentation of the problem can be developed.

We are currently (through our students) exploring the efficacy of the problem frame approach and are in the process of beginning new industrial liaisons both in the UK and Australia where we plan to evaluate the problem frames in a variety of different industries. With the rapid growth of simulation modelling, for instance as software process simulation and modelling (e.g. the ProSim workshops [9]), through leisure activities [10] to safety critical assessment, for example, a simulation of docking a Space Shuttle to a space station [11], it is absolutely imperative to be able capture all the necessary domain elements of the problem. For simulation problems, the simulation frame is a definitive place to start.

## References


[1] M. A. Jackson, “Requirements & Specifications: A Lexicon of Software Practice, Principles and Prejudices”, Addison Wesley, 1995.

[2] D. Jackson and M. A. Jackson, “Problem Decomposition for Re-use”; *Software Engineering Journal*, Volume 11, Number 1 pages 19-30, January 1996.

[3] B. Kovitz, “Practical Software Requirements; A Manual of Content and Style”, Manning, 1999.

[4] D. Bjorner et al., “Michael Jackson’s Problem Frames: Towards Methodological Principles of Selecting and Applying Formal Software Development Techniques and Tools”; *ICFEM'97: Intl. Conf. on ``Formal Engineering Methods'', Hiroshima, Japan*, pages 263-271, Los Alamitos, CA, USA, 12-14 November 1997

[5] M. A. Jackson, “Problem Frames: Analyzing and Structuring Software Development Problems”, Addison Wesley, 2001.

[6] M. A. Jackson, “Problem Analysis Using Small Problem Frames”, *South African Computer Journal*, Volume 22, pages 47-60, March 1999

[7] I. K. Bray, “An Introduction to Requirements Engineering”, Addison Wesley, 2002.

[8] K. Phalp, and K. Cox, "Picking the Right Problem Frame - An Empirical Study", *Empirical Software Engineering Journal*, 2000, 5(3), pp. 215-228.

[9] ProSim - Software Process Simulation and Modelling International Workshop Series, http://www.prosim.pdx.edu/prosim2003/index.htm

[10] *The Sims* computer games series, Electronic Arts Games.

[11] NASA, Smart Systems Research Lab, http://ssrl.arc.nasa.gov/techsim.html

# A Reuse-Based Approach to Determining Security Requirements

Guttorm Sindre[1], Donald G. Firesmith[2], Andreas L. Opdahl[3]

[1] *Dept Computer & Info. Science, Norwegian U. Science & Technology*
*(on leave at Dept MSIS, Univ. Auckland, New Zealand)*
[2] *Software Engineering Institute*
[3] *Dept of Information Science, U. of Bergen, Norway*
guttors@idi.ntnu.no, donald_firesmith@hotmail.com, andreas@ifi.uib.no

## Abstract

*The paper proposes a reuse-based approach to determining security requirements. Development for reuse involves identifying security threats and associated security requirements during application development and abstracting them into a repository of generic threats and requirements. Development with reuse involves identifying security assets, setting security goals for each asset, identifying threats to each goal, analysing risks and determining security requirements, based on reuse of generic threats and requirements from the repository. Advantages of the proposed approach include building and managing security knowledge through the shared repository, assuring the quality of security work by reuse, avoiding over-specification and premature design decisions by reuse at the generic level and focussing on security early in the requirements stage of development.*

## 1. Introduction

*Use cases* [1-3] have become popular for eliciting requirements [4, 5]. Many groups of stakeholders turn out to be more comfortable with descriptions of operational activity paths than with declarative specifications of software requirements [6]. As use cases specifically address what users can do with the system, they are most relevant for functional requirements. But lately the application of use cases has also been investigated in connection with security and safety requirements, in the form of *misuse cases* [7-13], a.k.a. *abuse cases* [12, 14].

Misuse cases describe interactions that cause harm to the system or its stakeholders and can be used as an informal front-end to more formal security requirements engineering. A closely related topic of research is that of *security use cases* [15]. Like misuse cases these describe interaction sequences where harm is attempted, but unlike misuse cases, the system ends up preventing or at least mitigating the damage.

In spite of the growing research interest in misuse cases, and promising early applications [10, 11, 13], the approach has yet to be put into large-scale industrial use. Many software development organizations tend to put little focus on security requirements, even if these are increasing in importance [16]. Partly, this may be due to a lacking understanding of security requirements. Indeed, even when they attempt to write security requirements, many developers tend instead to describe design solutions in terms of protection mechanisms, rather than making declarative statements about the degree of protection required [17]. Another reason for neglecting security requirements may be a perceived shortage of time in projects with narrow deadlines. For instance, case studies [18, 19] showed that security requirements were poorly addressed in several e-commerce projects.

To make misuse case analysis more appealing to practitioners *reuse* may be essential – as security requirements could then be specified more rapidly. As pointed out already in the 80's, reuse of requirements could lead to significant savings in development time and cost [20]. Although requirements reuse has attracted some research attention since then, methods suggested from academia have failed to demonstrate practicality or scalability [21] – perhaps with the exception of types of development particularly suited for reuse, such as product family development [22, 23] or ERP systems implementation [24].

The purpose of this paper is to provide a reuse-based methodology for misuse case analysis and the subsequent specification of security requirements. There are two key processes in reuse-oriented development [25, 26]:

- Development for reuse, where reusable artifacts are developed and made available for future

reuse, for instance in a repository / library that facilitates easy retrieval.

- Development with reuse, where end-user applications are developed, partly by reusing artifacts created by the "for" process.

There are interconnections between these two processes. Development with reuse can discover weaknesses of existing components in the reuse repository, or inspire new ideas for reusable components. Development for reuse can steer development with reuse if you are in a position to choose between alternative projects, picking the one where you have the greatest potential for reuse. The rest of this paper is structured as follows: Section 2 deals with development for reuse, discussing what kinds of artifacts should be developed, how the reusability of these artifacts should be ensured, and how the artifacts should be packaged in a repository for future reuse. Section 3 then addresses development with reuse, discussing how to identify candidates for reuse and then adapt them to the specific application. Section 4 discusses related work, and section 5 makes a concluding discussion.

## 2. Development *for* Reuse

Reuse of systems development artifacts may improve the quality of development processes and products and may reduce development costs if each artifact is reused at least 3–4 times (because it is more expensive to develop something reusable than something which will be used only once [27].) To ensure repeated reuse of security threats and requirements, we must find good answers to the following questions:

1. Which development artifacts should be stored in the repository for reuse?
2. How should the repository be organized to best support reuse?

### 2.1 The reusable development artifacts

As mentioned in the Introduction, applications are likely to face the same kinds of threats and have similar categories of required security even if they have different functional requirements. The challenge for reusability will be to describe threats and requirements on a sufficiently generic level, so that detailed differences between applications (e.g., in functionality, architecture) do not hamper the possibility for reuse. On the threats and requirements level, we suggest these types of reusable artifacts:

- *Generic threats*, described independently of particular application domains. Here we will only look at threats described as misuse cases, but other forms of representation could also be envisioned.
- *Generic security requirements*, again described independently of the particular application domain. These can be represented as security use cases or "system shall" requirements.
- *Application-specific threats* and *requirements*. Apart from including application-specific terminology, these can be described by misuse cases and security use cases (and/or "system shall" requirements) respectively, much similar to the generic varieties.

In addition to this, there could have been links further on to design level specifications, test cases etc., integrating reuse efforts across more phases, but this is not explored in our work so far.

For an example of a generic threat, Table 1 shows a generic misuse case that represents the threat of *spoofing*, i.e., a misuser gaining access to the system by pretending to be a legitimate user. This is a highly reusable misuse case, covering many different spoofing attacks. It does not matter if authentication is done by username+password, card+PIN, fingerprint scan, voice recognition, human to human recognition of individuals or something else. The interaction sequence is inspired by *essential misuse cases* [2], which focus on the users' intentions rather than concrete actions. In [2] the main motivation for this is to simplify the interaction and avoid premature design decisions, but avoidance of premature design will also increase the reusability of the description.

**Table 1:** A generic misuse case

| **Generic Misuse Case:** Spoof User Access | |
|---|---|
| **Summary:** The misuser successfully makes the system (physical / human / computerized) believe he is a legitimate user, thus gaining access to a restricted system / service / resource / building. | |
| **Preconditions:**<br>1) The misuser has a legitimate user's valid means to identify and authenticate ***OR***<br>**2)** The misuser has invalid means to identify and authenticate, but so similar to valid means that the system is unable to distinguish (even if operating at its normal capabilities) ***OR***<br>**3)** The system is corrupted to accept means of identification and authentication that would normally have been rejected. The misuser may previously have performed misuse case "Tamper with system" to corrupt the system. | |
| **Misuser interactions** | **System interactions** |
| Request access / service | |
| | Request identification and authentication |
| Misidentify and misauthenticate | |
| | Grant access / provide service |
| **Postconditions:**<br>1) The misuser can do anything the legitimate user could have done within one access session ***AND***<br>2) In the system's log (if any), it will appear that the system was accessed by the legitimate user. | |

For an example of a generic requirement, Table 2 shows one path of the security use case "Access Control", more specifically the one requiring the system to reject misusers with valid means of identification but invalid means of authentication.

**Table 2:** One path of a generic security use case

| **Generic Security Use Case:** Access Control | | |
|---|---|---|
| **Path name:** Reject invalid authentication | | |
| **Preconditions:**<br>Misuser has valid means of user identification but invalid means of user authentication. | | |
| **Misuser Interactions** | **System Requirements** | |
| | **System Interactions** | **System Actions** |
| | Request user identity and authentication. | |
| Provide valid user id but invalid authentication. | | |
| | Reject misuser by cancelling transaction. | Attempt identification, authentication & authorization. |
| **Postconditions:**<br>1) Misuser has valid means of user identification but invalid means of user authentication ***AND***<br>2) Misuser not authenticated, not granted access ***AND***<br>3) Access control failure registered. | | |

Just like the generic misuse cases, this generic security use case is highly reusable – it makes no design assumptions, and neither does it presuppose any particular application domain. Access control is a feature wanted in a wide range of applications. The above use case could be a representative requirement for accessing an ATM, an internet entertainment service, or a missile control system.

This example may seem ridiculously simple – which it also is. But remember that this is just one of several paths of the security use case, and that the total response to the threat would encompass more than just one security use case. For the particular example threat "Spoof user access" the repository might contain:

- The security use case "Access Control", with several paths (more examples are shown in [15]).
- Other security use cases or normal use cases describing security related functions, e.g., "Cancel means of authentication")
- Requirements described by other means, e.g., "system shall" requirements or *mitigation points* in misuse cases [8].

Several alternative means of representation will be necessary here, as security use cases do have some limitations:

- They are easy to express when the threat is mitigated during the attempt (in the interaction path), not so easy for mitigations relating to the preconditions or postconditions of the threat.
- The are most easy to express for "absolute" requirements. But in many situations a 100% secure solution is impossible or infeasible. It can be noticed that it does not make sense to include a path in the Access Control use case for the situation where the misuser has valid means of identification *and* authentication, since then – to the system – the misuser *is* the legitimate user, and the system cannot be required to do anything else than it normally does, namely granting access.

Examples of other requirements that might be suggested as a response to the spoofing threat:

- "The means of authentication should have a stealability index value of [value]"[1].
- "Upon issue, the user shall sign a contract obliging him to keep the means of authentication safe from misuse, and to report potential compromise within [time limit]"
- A use case "Cancel Means of Authentication", to be applied when users report possible theft or loss of their means.
- "The [user] shall be limited to [maximum action] per [session / time period / …]"

The latter example, with 3 [ ] brackets, may seem so vague that one might wonder whether it has any utility in a reuse repository. But it does serve to remind the stakeholders of a certain mitigation option that could otherwise easily be forgotten. An example of an instantiation will be shown in the next section.

## 2.2 The organisation of the repository

A meta-model showing Threats and Security Requirements and the links between them is given in Figure 1. Threats are what misusers try to achieve, causing harm to the system, and security requirements describe the extent to which the system shall be able to mitigate those threats. Key to understanding the diagram are the two classes on the way from Threat to Security Requirement, namely Threat-Requirement Relationship and Security Requirement Bundle. To start with the latter, this is a set of requirements that pull together in mitigating the same threat. It is often interesting to look at

[1] This assumes the definition of a (yet non-existing) stealability index for means of authentication, similar to what exists for cars. Clearly, less precise statements like "The means of authentication shall be difficult to steal" are not useful as requirements.

such bundles rather than just individual requirements, because:

- A security requirement bundle is a bigger and more effective unit of reuse. To the extent that one requirement is a good unit of reuse, it is still possible to define a bundle consisting only of that requirement.
- In many cases, single security requirements provide little or no protection unless accompanied by other requirements. For instance, as observed in [17] identification requirements are seldom of much value alone – in most cases they must be accompanied by authentication requirements.

Indeed, it can be observed that a security use case is in itself a requirement bundle. The example in Table 2 already contains two requirements – a) that the system shall reject access to users without valid means of authentication, and b) that failed access attempts shall be registered. And this is only one path of the bigger use case, so in total it would encompass several requirements.

The Threat-Security link objects will most commonly represent "mitigate" relationships, i.e., that a certain requirement bundle (e.g., the security use case "Access Control") mitigates a threat (e.g., the misuse case "Spoof User Access"). Another possible relationship is "aggravate", i.e., the choice of a requirements bundle may actually increase the risk for a threat. An example is that a bundle of Access Control requirements might increase the risk for Denial of Service (DoS) threats. A classical example is the suspension of console login for a certain user after three failed login attempts – a misuser could then deny access for that user simply by making those three failed login attempts with that username. In general, any requirement that the system should suspend access if sensing an attempted attack might be utilized for DoS purposes.

The Requirement-Requirement Relationship is used to register relationships between requirements, e.g., that they may be overlapping or in conflict. The aggregation from Threat to Threat Specification enables one threat to have several parallel representations in terms of format or language. For instance, the same misuse case could be written both in English, French and Norwegian, or in the same language but with different templates, or there could also be other representations than misuse cases, for instance in more formal languages (not investigated in this paper). The upper right part of the diagram shows an analogous modeling of the requirements side.

The lower left part shows that a Threat can either be a Generic Threat or an Application-Specific Threat, and one Generic Threat may have many Application-Specific instantiations. For instance, the "Spoof User Access" threat of Table 1 may be instantiated to cover illegitimate access to an ATM, a building, or an internet entertainment service. The lower right part of the figure shows that the requirements side is structured accordingly.

Basing a reuse repository on the above meta-model, the following two advantages are achieved:

- Security requirements may be searchable *via threats* that they are meant to mitigate, rather than having to search for requirements "directly". The "direct" alternative is less useful here – to know what to look for the developer must have a pretty clear picture of the requirement already, which reduces the gain from reuse.
- Security requirements can be packaged in bundles that give meaningful protection against commonly seen threats. In most cases this should

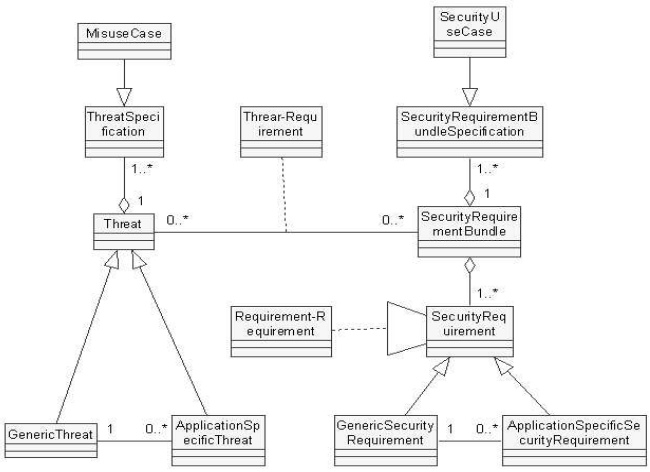



**Figure 1:** Meta-model for repository

be more effective than reusing requirements one by one and then assembling them in meaningful bundles on a project-to-project basis.

Having observed these advantages we now turn to development with reuse.

## 3. Development *with* reuse

Figure 2 shoes a UML Activity Diagram that outlines our suggested approach to development with reuse. The steps are as follows:

1. ***Identify critical and/or vulnerable assets:*** Here one must identify all the critical and/or *vulnerable assets* in the enterprise. A vulnerable asset is either information or materials that the enterprise possesses, locations that the enterprise controls or activites that the enterprise performs.[2] The "and/or" should be noticed specifically. It is interesting to look at assets that are critical but not vulnerable because a) further scrutiny may reveal that they were only believed not to be vulnerable, and b) their vulnerability might increase in the future. It is also interesting to look at assets that are vulnerable but not critical, at least if they are of the kind that misusers may use as stepping stones to launch attacks on more critical resources. For example, a server that holds no critical information and runs no critical services might still be used as a zombie in an attack against other computing resources, perhaps also in other companies, causing badwill or even liability to the organization. Starting the security analysis with a focus on assets ensures that the final security requirements are anchored in the protection of materials, information, locations and activities that are of value to the enterprise.
2. ***Determine security goals for each asset:*** For each critical and/or vulnerable asset identified in step 1., select the appropriate security goals for the asset. A security goal is specified in terms of (1) *who* are the potential misusers, (2) the *type* of security breaches the asset is vulnerable to and (3) the security *level* necessary for that type of breach. For example, the potential misusers may be Internet script kiddies, business competitors or disgruntled employees. Examples of security types are violations of, e.g., secrecy or integrity, and several of the security threat classifications in the literature can be used in this step. A possible taxonomy of security breaches is proposed in [17]. The security level to be achieved is specified as a probability that the assets will be kept safe from the particular type of breach from the particular type of misuser. Establishing security goals for all the critical and/or vulnerable assets ensures that the eventual security requirements are derived based on thoroughly identified types of misuers and of security breaches. Also, well-defined security goals are a prerequisite for identifying threats. (If you have no goals, there are no threats either. Even "being killed" is only a threat if you consider "staying alive" as a goal and "life" as an asset.)

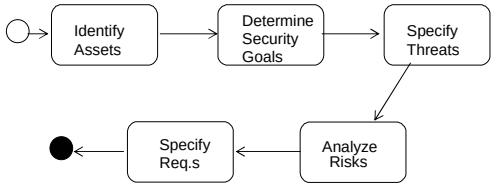


Figure 2: Development for reuse, process

3. ***Identify threats to each asset:*** For each security goal identified in step 2, find all the threats that can prevent the goal from being achieved or maintained. This is where the repository is used for the first time. First, find misuse cases in the repository that involve the right types of misusers as specified by the goal and, then, select those misuse cases that threaten the right type of security breach. Finally, assess whether the misuse case poses a threat that is relevant given the security level specified by the goal. For example, misuse cases that involve the breaking of cryptographic codes may be a relevant threat to the security and integrity of banks or military installations with extremely high security levels, but not to the security and integrity of student information in a university information system. In addition to using the repository, it is of course necessary to look for threats that are not directly implied by the determined security goals, because some security goals may indeed have been forgotten.
4. ***Analyze risk for each threat:*** In its most detailed form, the specification of threats must include the risk of the various threats, i.e., the estimated likelihood of occurrence and cost of the damage if the threat occurs. Whereas the description of threats is highly reusable, risks must normally be determined from application to application. For example, although both an Internet entertainment service and a missile control system face the

[2] The most important assets of enterprises, the knowledge and skills of its workers, is not directly important in an ICT security context, as they are only vulnerable indirectly, through misuse of the other, more tangible assets.

threat of spoofing, the associated risks may be quite different.

5. ***Determine requirements:*** For each identified threat, and taking its risk into account, determine requirements to mitigate the threat. The repository is used again here. For each threat retrieved from the repository, one or more associated bundles of security requirements may be found. For threats not retrieved from the repository, appropriate security requirements must be determined and specified by other means. Even when threats are retrieved from the repository, additional bundles of security requirements that mitigate the threat may be found by other means. Different levels of mitigation will be needed for different threats, and requirements workers must select requirements bundles that together produce the necessary levels of mitigation for all threats.

When the process is completed, there should be satisfactory requirements specified for all threats, and threats should have been investigated for the security goals of all assets.

In this paper we do not discuss the first two steps any further (although one might envision some reuse even in those steps, for instance by means of asset checklists), neither do we discuss step 4. It is however necessary to show these steps to illustrate the context in which the reuse of step 3 and 5 takes place. The activity diagram of Figure 3 shows the decomposition of the threat specification (step 3). Three possible ways are suggested to identify threats:

- Top down Threat search means that you start from the identified assets and security goals and then try to search the repository for threats relevant to such assets / security goals. This would be best supported if there were attributes pointing to relevant types of assets or security goals in the Threat class, or alternatively there could be separate classes for asset types and goal types which the threats were then associated with.
- Bottom-up threat search, i.e., starting by looking at what you have in repository (without regard for the determined security goals) and then considering whether different threats described there are relevant to your application. This might seem a less systematic approach than the top-down alternative, and if the repository is big it might cause a lot of wasted time looking at irrelevant threats. However, it might be a valuable corrective to a strict top-down development in that it gives an extra check that no threats have been overlooked. As security goals may have been overlooked in the previous stage (or assets before that), a strict top-down approach gives no guarantee that all threats will be discovered.
- Threat brainstorming. This is the option for threats which cannot be found in the repository (whether mandated from determined security goals or not). But of course, whenever a threat has been suggested by brainstorming, one should check to make sure it is indeed not covered by the repository.

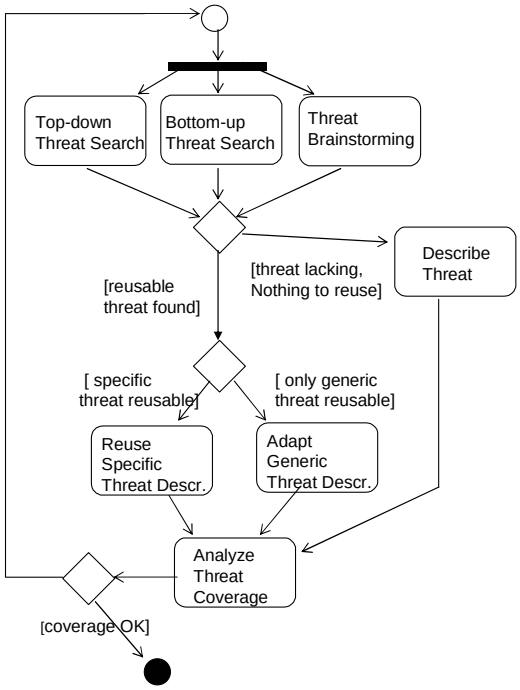



**Figure 3:** Decomposition of "Specify Threats"

Whatever method a threat has been identified by, one of two situations may occur:

- The repository contains no description that can be reused for this threat. In rare cases this could happen even for threats discovered through the bottom-up approach, i.e., browsing through the repository the developers come upon a threat that is indeed relevant to their application, but the description in the repository is not reusable enough.
- The repository contains a description that can be reused for this threat. In this case there are two new alternatives: Either there is only a generic threat description that can be reused, this must then be adapted to an application specific instantiation. Or there is already a fitting application-specific variety in the repository, then

this can possibly be reused as-is, saving even more work for the developers.

As an example, imagine that the repository contained the threat "Spoof User Access" of Table 1, and that this was retrieved and found relevant in the project at hand – to develop a new ATM system. Then, the generic misuse case could be adapted to the application specific misuse case shown in Table 3 – the only phrases that would have to be rewritten would be the underlined ones. However, if there had also been a previous development project for an ATM system by means of the repository, it might well be that there already was such an application-specific misuse case. Then this could be reused directly.

**Table 3:** Application-specific misuse case

| **Misuse Case Name:** Spoof Customer at ATM | |
|---|---|
| **Summary:** The misuser successfully makes the ATM believe he is a legitimate user. The misuser is thus granted access to the ATM's customer services. | |
| **Preconditions:**<br>1) The misuser has a legitimate user's valid means of identification and authentication ***OR***<br>2) The misuser has invalid means of identification and authentication, but so similar to valid means that the ATM is unable to distinguish ***OR***<br>3) The ATM system is corrupted, accepting means of identification and authentication that would normally have been rejected. | |
| **Misuser interactions** | **System interactions** |
| Request access | |
| | Request identification and authentication |
| Misidentify and misauthenticate | |
| | Grant access |
| **Postconditions:**<br>1) The misuser can use all the customer services available to the spoofed legitimate user ***AND***<br>2) In the system's log (if any), it will appear that the ATM was accessed by the legitimate user. | |

Moving on to step 5, the decomposed activity diagram for this can be found in Figure 4. When it comes to requirements, the chance for reuse should be considerable if a threat was reused – then one can follow the repository's links to one or more requirement bundles for that threat. If the threat had to be specified from scratch, there are no directly corresponding requirements in the repository, so the chance for reuse is much smaller. Yet, it could pay off at least to browse briefly for requirements related to similar threats, if any.

Either way, it may happen that no requirement bundles are found satisfactory for reuse, or there may be potential for reuse. Here the situation is quite similar to the reuse of threats: It might be that reuse is only possible with adaptation from the generic level, but one might also be lucky enough to be able to reuse something from the specific level, as is. If we take ATM systems as a concrete example, the Access Control path shown in Table 2 can be utilized almost directly at the specific level, possibly

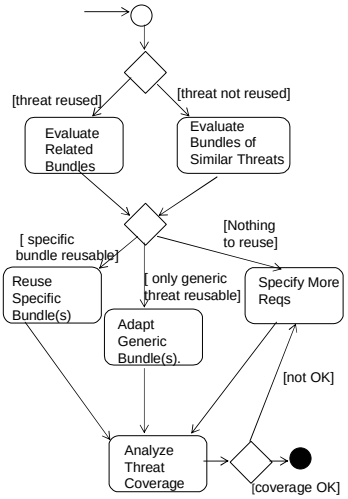



**Figure 4:** Decomposition of "Specify Req.s"

only with a slight name change to "ATM Access Control". As there is no point in showing the same example twice, we instead show an authorization example. An instantiation of the generic requirement *"The [user] shall be limited to [maximum action] per [session / time period / …]"* could be "The ATM customer shall be limited to withdrawing maximum 1000 USD of cash per week" – not preventing spoofing but at least reducing the damage for the cases when it does occur. It would also be possible to express this as a path of a security use case, and although the "shall" requirement is probably simpler and better to use in this case, we show it for illustration in Table 4.

**Table 4:** A specific security use case

| **Security Use Case:** ATM Authorization | |
|---|---|
| **Path name:** Reject withdrawal beyond weekly limit | |
| **Preconditions:**<br>1) Misuser has gained access to ATM customer services, e.g., by a successful "Spoof User Access".<br>2) The account has a weekly cash withdrawal limit of USD 1000, of which Y < 1000 has currently been withdrawn.<br>3) Account balance B > 1000 - Y | |
| **Misuser Interactions** | **System Interactions** |
| Request to withdraw Z1 > X - Y | |
| | Deny withdrawal as exceeding weekly limit |
| Request to withdraw Z2 <= X - Y | |
| | Accept withdrawal, dispense cash |
| **Postconditions:**<br>1) The misuser will max. have been able to withdraw X.<br>2) New Account Balance B is old B – Z2 | |

When threats have been analyzed, determining the level of security needed towards various threats, follow the repository links from the threats side to the requirements side, to look at alternative requirements to mitigate the relevant threats – and choose those most appropriate to the needed security levels.

The chosen generic security requirements should then be adapted to application specific ones. In some cases hardly any rewriting is needed, in other examples it may be necessary to change some terms to application specific ones, and to quantify requirements where the generic ones only indicate the possibility to quantify, e.g., changing <time limit> with an actual time limit or X% with a number.

## 4. Related Work

Reuse of requirements has been investigated by researchers for quite a time, e.g., reusing fragments of domain knowledge through inheritance [28], reuse by analogy of structure [29], semantic matching [30], or by clustering of specification diagrams [31]. Our approach differs from these in its specific focus on security requirements (contrary to, e.g., functional requirements), and the investigation of one particular form of representation as a vehicle for reuse (misuse cases).

Reuse of use cases or scenarios has been investigated by, e.g., [32, 33], through use case patterns and retrieval based on some concept of similarity. Our suggested approach also differs from these in its particular focus on security requirements. Also, the ideas concerning retrieval seem to be different. With use cases / scenarios (typically expressing functional requirements), the idea seems to be that the developer knows to some extent what he is looking for (e.g., being able to partially describe a use case), and then the system will suggest something similar). Our idea with reuse based on misuse cases, however, is that the reuser might not know what he is looking for. Indeed the highest benefits of reuse might be in cases where the developer, browsing the library, becomes aware of a threat to the system that he had no idea of beforehand (and would thus have overlooked). Hence, one can envision the repository structure being used more in a checklist manner, looking at all the various threat categories and considering whether they are relevant for the application to be developed, rather than specifying something vaguely and then searching for it. This means that the reuser's interaction with the repository will be more dominated by taxonomy-supported browsing than by massive automated searches.

A work that deals with reuse of security requirements – therefore being particularly close in topic to ours – is the SIREN approach by Toval et al. [34]. This approach suggests a repository of security requirements initially populated by using MAGERIT, the Spanish public administration risk analysis and management method conforming to ISO15408 (the Common Criteria Framework [35]). Here, it suggests a process with the following 4 steps: i) identify assets, ii) identify vulnerabilities (threats to assets), iii) analyze risks, iv) choose countermeasures. This is quite similar to our process for development with reuse of Figure 2: steps (i) are identical, our step (iii-iv) are similar to their (ii-iii). There are two differences, though:

- We suggest a step 2 of identifying security *goals* for each asset before going on to threats. Our argument for this is that the definition of security goals should precede threats.
- Our step 5 is the identification of *requirements*, while their parallel step 4 talks about *countermeasures*. As argued in [17], the premature specification of design in terms of countermeasures is unlucky – one should first try to express pure requirements (e.g., what level of protection is needed, rather than how to achieve that protection in terms of architectural mechanisms). The examples in [34] do indeed indicate some design tendencies in the suggested requirements (e.g., passwords, firewalls).

In addition to the 4 steps from MAGERIT, SIREN also suggests a larger scale process, based on a Spiral model (but concentrating on requirements engineering, not the entire development). Yet this process is much wider than what is addressed in our paper, the SIREN process deals with all tasks concerning the requirements specification – selection from the repository, elicitation, negotiation, specification, validation. We look more narrowly and in more detail only at the activities directly related to reuse.

The suggested method addresses many things not addressed in our work, e.g., organizing assets in 5 levels, and defining a requirements document hierarchy with 5 different documents (different kinds of requirements specifications and test plans). These are not contradictory with our approach, suggesting that they could supplement each other. When it comes to the SIREN repository structure, requirements can be structured according to *domains* and *profiles* – the former reflecting functional application areas, the latter opening for a possible structuring according to non-functional aspects (e.g., a profile for information systems security). Requirements can be parameterized or non-parameterized. The latter can be reused directly, whereas the former must have, e.g., some values filled in. This does not exactly parallel our difference between generic and application-specific – rather, both parameterized and non-parameterized requirements are on the same level in that respect.

Moreover, SIREN focuses on requirement lists, while we focus on (mis)use cases, but this is clearly a surface difference, as there is clearly nothing in the SIREN approach that excludes the inclusion of use cases in the repository (and neither would requirement lists be excluded from our repository). The requirements in the SIREN repository can be coupled to (or retrieved via) the aforementioned document structure and the MAGERIT asset hierarchy. This is different from our approach, where a) requirements are navigated from threats, not assets, and b) the threat specifications are seen as the principal artifacts of reuse, possibly together with corresponding requirements.

A distinguishing property of our suggestion is thus that we suggest to reuse specifications of both requirements and threats, which are closely linked. Apart from the fact that goals and threats are opposites, their relationship to requirements are quite the same (e.g., that there can be many different choices of requirements to address the same goal or threat). Hence, our approach is also related to goal-oriented approaches. For instance, the *obstacles* discussed in [36] (in connection with the KAOS method) could be likened to our threats, and [19] (in connection with the GBRAM approach) discusses the linking of security/privacy policies and requirements. Also relevant is the work on trust in i* [37], by which threats can also be modeled. Rather than contradicting these, our particular work again looks more narrowly at reuse issues with one particular form of representation (misuse cases).

## 5. Discussion and Conclusions

The paper has proposed a reuse-based approach to determining security requirements. The main weakness is that our suggested reuse approach has not been tried out in practice – for which a tool would have to be developed the repository populated with threats and requirements to be reused. Yet we contend that the contribution in this paper at least is a good starting point for such demonstrations of practicality, with its suggested models for the repository and reuse process. Development *for* reuse involved identifying security threats and associated security requirements. This can either be done as domain analysis or during application development, and should yield a repository of threats and related requirements. Threats can for instance be expressed as misuse cases and requirements as security use cases. Development *with* reuse involved identifying security assets, setting security goals for each asset, identifying threats to each goal, analyzing risks and determining security requirements, based on reuse of generic threats and requirements from the repository. Advantages of the proposed approach include building and managing security knowledge through the shared repository, assuring the quality of security work by reuse, avoiding over-specification and premature design decisions by reuse at the generic level and focusing on security early in the requirements stage of development. The proposed approach may also save time in the early development phases and produce more complete requirements, as the repository may prevent developers from forgetting important threats or requirements. The generic security requirements show the developers what level their description should be at whereas, otherwise, it would be tempting to jump directly from threats to design mechanisms (or even to mechanisms directly, without completely understanding the threats).

The main difference from related work is a specific focus on the reuse of threats *and* security requirements, both described in terms of use cases. On the other hand, this paper fails to address many issues that are addressed by related work, hence integration with other approaches is an interesting topic for further work.

Work on reuse-based determination of security requirements is still in its early stages, and industrial case studies are called for. To better support development *for* reuse, further work is needed on how to link misuse cases in the repository to relevant security goals, to better prepare for development *with* reuse. The repository should be implemented in a tool and integrated with CASE tools. For example, the tool should support abstraction of application specific threats and security requirements into generic ones. The tool should also enforce a common taxonomy and terminology, e.g., for types of misusers and security breaches, in order to increase search efficiency.

To better support development *with* reuse, further work is needed on method guidance for specifying security goals, in particular on how to best classify security threats. Heuristics for setting security levels would also be helpful. Of course, the tool should support searching for threats according to misuser and type of security breach, both exactly and approximately.

Comparing the present proposal to goal- and agent-oriented approaches to security requirements work is another path for further work. As emphasized also in previous publications, misuse case analysis has never intended to be a full-fledged development approach in its own right, rather the idea is that it must be integrated with other approaches.

## References

[1] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Boston: Addison-Wesley, 1992.

[2] L. L. Constantine and L. A. D. Lockwood, *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*: ACM Press, 1999.
[3] A. Cockburn, *Writing Effective Use Cases*. Boston: Addison-Wesley, 2001.
[4] J. Rumbaugh, "Getting Started: Using use cases to capture requirements," *Journal of Object-Oriented Programming*, pp. 8-23, 1994.
[5] D. Kulak and E. Guiney, *Use Cases: Requirements in Context*: ACM Press, 2000.
[6] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenario Usage in System Development: A Report on Current Practice," *IEEE Software*, vol. 15, pp. 34-45, 1998.
[7] G. Sindre and A. L. Opdahl, "Eliciting Security Requirements by Misuse Cases," presented at TOOLS Pacific 2000, Sydney, 2000.
[8] G. Sindre and A. L. Opdahl, "Templates for Misuse Cases," presented at REFSQ'2001, Interlaken, 2001.
[9] G. Sindre, A. L. Opdahl, and G. F. Breivik, "Generalization/Specialization as a Structuring Mechanism for Misuse Cases," presented at 2nd Symposium on Requirements Engineering for Information Security, Raleigh, NC, 2002.
[10] I. F. Alexander, "Initial Industrial Experience of Misuse Cases in Trade-Off Analysis," presented at RE'02, Essen, 2002.
[11] I. F. Alexander, "Misuse Cases, Use Cases with Hostile Intent," *IEEE Software*, vol. 20, pp. 58-66, 2003.
[12] J. McDermott, "Abuse-Case-Based Assurance Arguments," presented at 17th Annual Computer Security Applications Conference (ACSAC'01), 2001.
[13] I. F. Alexander, "Modelling the Interplay of Conflicting Goals with Use and Misuse Cases," presented at 8th International Workshop on Requirements Engineering: Foundation for Software Quality, Essen, Germany, 2002.
[14] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," presented at 15th Annual Computer Security Applications Conference (ACSAC'99), 1999.
[15] D. Firesmith, "Security Use Cases," *Journal of Object Technology*, vol. 2, pp. 53-64, 2003.
[16] R. Crook, D. Ince, L. Lin, and B. Nuseibeh, "Security Requirements Engineering: When Anti-Requirements Hit the Fan," presented at IEEE International Requirements Engineering Conference (RE'02), Essen, Germany, 2002.
[17] D. Firesmith, "Engineering Security Requirements," *Journal of Object Technology*, vol. 2, pp. 53-68, 2003.
[18] A. I. Anton, R. A. Carter, A. Dagnino, J. H. Dempster, and D. F. Siege, "Deriving Goals from a Use Case Based Requirements Specification," *Requirements Engineering Journal*, vol. 6, pp. 63-73, 2001.
[19] A. I. Anton and J. B. Earp, "Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems," presented at 1st ACM Workshop on Security and Privacy in E-Commerce, 2000.
[20] T. Biggerstaff and C. Richter, "Reusability Framework, Assessment and Directions," *IEEE Software*, vol. 4, pp. 41-49, 1987.
[21] A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective," presented at ICSE'2000, Limerick, Ireland, 2000.
[22] M. Mannion, B. Keepence, H. Kaindl, and J. Wheadon, "Reusing Single System Requirements from Application Family Requirements," presented at ICSE'99, Los Angeles, CA, 1999.
[23] R. R. Lutz, "Towards Safe Reuse of Product Family Specifications," presented at SSR'99, Los Angeles, CA, 1999.
[24] M. Daneva, "Measuring Reuse of SAP Requirements: a Model-based Approach," presented at SSR'99, Los Angeles, CA, 1999.
[25] E.-A. Karlsson, "Software Reuse: A Holistic Approach," in *Wiley Series in Software Based Systems*: John Wiley & Sons, 1995.
[26] G. Sindre, R. Conradi, and E.-A. Karlsson, "The REBOOT Approach to Software Reuse," *Journal of Systems and Software*, vol. 30, pp. 201-212, 1995.
[27] W. Tracz, "Software Reuse Myths," *ACM SIGSOFT Software Engineering Notes*, vol. 13, pp. 17-21, 1988.
[28] H. Reubenstein and R. Waters, "The Requirements Apprentice: Automated assistance for requirements acquisition," *IEEE Software*, vol. 17, pp. 226-240, 1991.
[29] N. A. M. Maiden and A. G. Sutcliffe, "Exploiting Reusable Specifications through Analogy," *Communications of the ACM*, vol. 35, pp. 55-64, 1992.
[30] P. Massonet and A. van Lamsweerde, "Analogical Reuse of Requirements Frameworks," presented at 3rd International Conference on Requirements Engineering, Washington DC, 1997.
[31] O. Lopez, M. A. Laguna, and F. J. Garcia, "Reuse-based Analysis and Clustering of Requirements Diagrams," presented at 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02), Essen, Germany, 2002.
[32] W. Lam, "Scenario reuse: A technique for complementing scenario-based requirements engineering approaches," presented at 4th Asia Pacific Software Engineering and International Computer Science Conference (APSEC'97 / ICSC'97), Hong Kong, 1997.
[33] H. G. Woo and W. N. Robinson, "Reuse of Scenario Specifications Using an Automated Relational Learner: A Lightweight Approach," presented at IEEE Joint Conference on Requirements Engineering (RE'02), Essen, Germany, 2002.
[34] A. Toval, J. Nicolas, B. Moros, and F. Garcia, "Requirements Reuse for Improving Information Systems Security: A Practitioner's Approach," *Requirements Engineering Journal*, vol. 6, pp. 205-219, 2002.
[35] CCIMB, "Common Criteria for Information Technology Security Evaluation," Common Criteria Implementation Board, Technical Report CCIMB-99-031, August 1999.
[36] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 26, pp. 978-1005, 2000.
[37] E. Yu and L. Liu, "Modelling Trust in the i* Strategic Actors Framework," presented at 3rd Workshop on Deception, Fraud and Trust in Agent Societies, Barcelona, 2000.

# A Framework for Modeling Privacy Requirements in Role Engineering

Qingfeng He and Annie I. Antón
*Department of Computer Science*
*North Carolina State University*
*Raleigh, NC 27695-8207, USA*
*{qhe2, aianton}@eos.ncsu.edu*

## Abstract

*Privacy protection is important in many industries, such as healthcare and finance. Capturing and modeling privacy requirements in the early stages of system development is essential to provide high assurance of privacy protection to both stakeholders and consumers. This paper presents a framework for modeling privacy requirements in the role engineering process. Role engineering entails defining roles and permissions as well as assigning the permissions to the roles. Role engineering is the first step to implement a Role-Based Access Control (RBAC) system and essentially a Requirements Engineering (RE) process. The framework includes a data model and a goal-driven role engineering process. It seeks to bridge the gap between high-level privacy requirements and low-level access control policies by modeling privacy requirements as the contexts and obligations of RBAC entities and relationships. A healthcare example is illustrated with the framework.*

## 1. Introduction

As the Internet and e-commerce have prospered, privacy has become of increasing concern to consumers, developers, and legislators. Legislative acts, e.g. Health Insurance Portability and Accountability Act (HIPAA) for healthcare [HIP96] and Gramm Leach Bliley Act (GLBA) for financial institutions [GLB01], require these industries to ensure consumer data's security and privacy. Companies and organizations protect consumer privacy in various ways, including publishing a privacy policy on their websites, enabling a P3P [P3P02] compliant privacy policy, incorporating a privacy seal program (e.g. Truste, BBBOnline), etc. However, these approaches cannot truly safeguard consumers because they do not address how personal data is actually handled after it is collected [AER02, AEP01, GHS00]. Companies' and organizations' actual practices might intentionally or unintentionally violate the privacy policies they published on their websites. Privacy violations are increasingly disclosed over the Internet, TV, newspaper and other medias, such as the famous Toysmart [Toy00] and Eli Lilly [Eli02] cases.

Privacy protection can only be achieved by enforcing privacy policies within an organization's online and offline data processing systems. Most organizations have one or more privacy policies posted on their websites. Due to separation of duties in an organization, privacy policies are usually defined as high-level natural language descriptions by an organization's privacy group, chaired by the Chief Privacy Officer (CPO). High-level natural language privacy policy descriptions are difficult to enforce directly via access control. Similarly, security polices are usually defined by another group of people in the organization, chaired by the System Security Officer (SSO). However, privacy requirements are often not reflected in the design and implementation of security policies. Thus, there exists a gap between security and privacy protection that is exacerbated by conflict of interests between stakeholders, system developers, and consumers. Researchers contend security and privacy requirements should be considered during initial system design [AE01, AEP01, AEC02]. Thus, modeling security and privacy requirements in the early stages of system development is essential for security and privacy enforcement.

Role-Based Access Control (RBAC) [SCF96, FSG01] has received increasing attention because it offers many additional benefits compared with traditional Discretionary and Mandatory Access Controls (DAC and MAC) [AS00]. RBAC is considered as a promising alternative to traditional MAC and DAC models [OSM00], especially in the healthcare domain. "It is generally accepted that RBAC is more suited to healthcare than other access control mechanisms to meet the requirements for the security of healthcare information" [ZAC02]. The Privacy-Aware RBAC (PARBAC) model enforces privacy policies in an organization [He03a], but it lacks a mechanism for mapping privacy requirements into the PARBAC model.

Role engineering for RBAC is the process of defining roles, permissions, role hierarchies, constraints and assigning the permissions to the roles [Coy96]. It is the first step to implement an RBAC system and essentially

an RE process. Before a system can realize all the benefits of RBAC, the role engineering activities must occur, yielding a complete specification.

Security requirements are modeled in the role engineering process. For example, the well-known separation of duties security requirement is modeled by defining exclusive roles; least privilege security requirement is modeled by assigning each role a minimum set of permissions to perform each task. However, privacy requirements are not addressed in role engineering. For example, purpose binding, i.e. data collected for one purpose should not be used for another purpose without user consent, is an important privacy requirement. To date the security and RE literature does not address purpose elicitation and modeling in role engineering. Another issue regarding to privacy protection is user privacy preferences modeling and the integration of these preferences with access control authorizations. A mechanism is needed to model privacy requirements and user privacy preferences in a systematic way so that privacy policies can be enforced in the software system.

This paper presents a goal-driven framework for modeling privacy requirements in the role engineering process. We model privacy requirements as contexts and constraints of permissions and roles using goal-based RE techniques. These contexts and constraints serve as a basis for defining access control policies. The proposed framework seeks to bridge the gap between high-level privacy requirements and low-level access control policies in the early stages of system development and provide a basis for enforcing privacy requirements with RBAC.

The rest of this paper is organized as follows. Section 2 provides a summary of related work. Section 3 describes privacy protection elements modeling. In Section 4, the framework for modeling privacy requirements is described. Then in Section 5, a healthcare example is illustrated with the framework. Finally, a summary of the paper is given in Section 6. The limitations of the framework and future work are also discussed in this section.

## 2. Related work

This section provides an overview of relevant work in role engineering, goal-driven requirements engineering, and privacy policies and requirements.

### 2.1. Role engineering for RBAC

There exist several role engineering approaches, the first of which applies scenarios. Neumann and Strembeck proposed a scenario-driven approach for engineering functional roles in RBAC [NS02]. In this approach, each task is depicted using a collection of scenarios and each scenario is decomposed into a set of steps. Because each step is associated with a particular access operation, each scenario is linked to a set of permissions. The work is limited in that it is only effective to derive functional roles. Fernandez and Hawkins suggested determining the needed rights for roles from use cases [FH97].

Crook et al. proposed an analytical role modeling framework to derive roles from organizational structures [CIN02]. Although this provides a way to derive roles, not all roles can be derived from organizational structures. The method is not general and does not address role constraints. Epstein proposed a layered model for engineering role-permission assignment by introducing three intermediaries between roles and permissions: jobs, workpatterns, and tasks [Eps02, ES01]. Epstein's approach provides an effective way to assign permissions to roles and aggregate permissions into roles. Roeckle et al. proposed a process-oriented approach for role finding to implement role-base security administration [RSW00]. Their approach provides a method to find roles but does not address how to find permissions and how to assign permissions to roles.

Unfortunately, neither of these approaches [Eps02, ES01, FH97, RSW00] considers constraints and role hierarchies. Epstein and Sandhu's UML based approach documents components of an RBAC model in UML syntax [ES99]. This approach can assist the role engineering process but it does not provide a method for deriving roles. Kern et al. proposed an iterative-incremental life-cycle model of a role in the context of enterprise security management [KKS02]. The role life-cycle concept is very important for security administration; however, this approach fails to support the derivation of roles and permissions. Schimpf argued role engineering is a critical success factor for enterprise security administration [Sch00]. He proposed to organize a role engineering project and follow a clearly defined life-cycle model for roles.

In conclusion, the above-discussed approaches focus on different aspects of role engineering. Each work has its own strengths and weaknesses. None of these approaches addresses privacy requirements.

### 2.2. Goal-driven requirements engineering

Goal-driven RE employs goals to elicit, specify, analyze, and validate requirements. Kavakli identified seven major goal-oriented methods in RE [Kav02]. A complete overview of goal-driven RE techniques is beyond the scope of this paper. Herein we only discuss goal-scenario combination approaches. A more complete overview of goal-driven RE approaches can be found in [Lam01, Kav02].

Goals and scenarios have complementary characteristics [Lam01]. Goals are usually abstract and declarative. They are high-level objectives of the business, organization or system. Scenarios are concrete, narrative, and procedural. They describe real situations using examples and illustrations. Hence combining goals and scenarios is an effective way to elicit and validate requirements. Goals are operationalized through scenarios and refined into requirements [AMP94]. Similarly, scenarios can be used to help discover goals [AP98].

The GBRAM uses goal hierarchies to organize requirements as scenarios, goal obstacles, and constraints [Ant96]. Others also organize scenarios hierarchically according to goals and goal obstacles [Coc97]. Rolland et al. proposed a bidirectional goal-scenario coupling approach between goal discovery and scenario authoring [RSA98]. Kaindl proposed a systematic design process based on a model combining scenarios with goals and functions [Kai00]. In the combined model, "purpose" serves as a link between functions and goals: a system's aggregated functions have some purposes and these purposes match the (sub)goals of the users. Purpose has also been integrated with scenarios to model tasks in one of Kaindl's early works [Kai95]. This paper herein builds upon this notion of purpose.

### 2.3. Privacy policies and requirements

Two major privacy protection principles are the OECD guidelines for data protection [OEC80] and the FTC Fair Information Practice (FIP) Principles [FIP98]. The OECD guidelines define eight privacy principles: collection limitation, data quality, purpose specification, use limitation, security safeguards, openness, individual participation, and accountability. The OECD principles intend to protect personal data privacy while pursuing free information flow between different organizations and different countries. The five FIP principles (notice/awareness, choice/consent, security/integrity, access/participation, and enforcement/redress) are less complete than the OECD guidelines. Both the OECD and FIP principles provide the general privacy requirements with which organizations should comply. Several industries have additional legislative acts (e.g. HIPAA and GLBA) regulating their data practices.

Based on these general privacy principles and acts, each organization defines its own privacy policies. These policies are the major privacy requirements that an organization should enforce in their data processing systems. For example, when websites collect information from customers, they need to inform customers for what purpose the data is being collected, who the data recipient is, how long the data will be kept, and how the data will be used, etc. (notice/awareness principle in FIP). They should also provide opt-in/opt-out choices for customers or obtain customer consent on how to use the collected data (choice/consent principle). The actual data operations of companies and organizations should be consistent with user consented privacy policies (enforcement/redress principle).

Fischer-Hubner summarized four privacy aspects that a system should protect: confidentiality of personal data, integrity of personal data, purpose binding of accesses to personal data, and necessity of personal data processing (i.e. the collection and processing of data shall only be allowed if it is necessary for completing appropriate tasks) [Fis01]. Confidentiality and integrity have been the focus of the security community for a long time. The principle of necessity can be enforced with task-based authorization models, such as the Workflow Authorization Model (WAM) [Fis01]. However, purpose binding is not addressed in traditional security models.

Similarities and differences between policies and requirements are identified in [AEP01]. Antón and Earp have proposed strategies to employ scenario management and goal-driven requirements analysis methods for specifying security and privacy policy for secure electronic commerce systems [AE01]. Antón et al. have also applied goal-based requirements analysis to align software requirements with security and privacy policies [AEC02]. A privacy requirements taxonomy for websites has been presented in [AE03] by using goal-mining techniques on privacy policies. In this taxonomy, privacy requirements are classified as either privacy protection goals or privacy vulnerabilities. This paper builds upon these specification techniques to better support modeling of privacy requirements in role engineering. All sample privacy policies given in this paper are privacy goals identified from 23 websites' privacy policies in Antón et al.'s goal-mining exercises [AE03].

## 3. Privacy elements modeling

High-level privacy policies and requirements that are specified with natural language must be formalized into authorization rules before they can be technically enforced. Therefore, it is necessary to identify privacy protection elements in the role engineering process.

A typical access control rule is expressed as a tuple $<s, o, op>$, such that a subject $s$ can access an object $o$ on operation $op$ [DD82]. A subject could be a user or a program agent. In an RBAC policy, this rule is expressed in another way: $<u, r, p>$ [SCF96]. A user $u$ can only access an object, if he/she is assigned a role $r$, and if the role is assigned certain permission $p$, which is allowed to access the object. A permission is usually represented as the combination of some operations on an object. Although the form is different, the basic elements of an RBAC rule are still subjects, objects, and operations.

These three elements, however, are insufficient to represent a privacy authorization rule. For instance, purpose binding is an important privacy requirement as we discussed in Section 2.3, but purpose is not reflected in the <*s*, *o*, *op*> tuple. In addition to the above three basic authorization elements (subjects, objects, and operations), three other privacy elements (purposes, conditions, and obligations) are identified in a privacy authorization rule [KS02]. Our framework builds upon these privacy protection elements as we now discuss.

## 3.1. Purposes

Purpose is a standard entity in most privacy policies as recognized in P3P [P3P02]. To enforce purpose binding privacy requirements, two kinds of purpose are identified: consumer data purpose and business purpose. Consumer data purpose is consented by a consumer and recorded by a data collector and expresses how the corresponding collected data can be used. Business purpose is the actual purpose for a business task that involves certain consumer data accesses or operations.

**3.1.1. Data purposes.** Customer consented data purposes are usually high-level and the number of such purposes is limited. According to the official P3P1.0 Specification [P3P02] released by the World Wide Web Consortium (W3C) on 16 April 2002, there are only 12 purposes[1] defined in P3P1.0. Table 1 shows these 12 purposes.

**Table 1. Purposes defined in P3P1.0**

| Purpose Name | Description |
|---|---|
| current | Completion and Support of Activity For Which Data Was Provided |
| admin | Web Site and System Administration |
| develop | Research and Development |
| tailoring | One-time Tailoring |
| pseudo-analysis | Pseudonymous Analysis |
| pseudo-decision | Pseudonymous Decision |
| individual-analysis | Individual Analysis |
| individual-decision | Individual Decision |
| contact | Contacting Visitors for Marketing of Services or Products |
| historical | Historical Preservation |
| telemarketing | Telephone Marketing |
| other-purpose | Other Uses |

**3.1.2. Business purposes.** Business purposes are defined in each organization according to its business process. They may be defined more specifically than data purposes. For example, the contact purpose may be divided into three categories: phone/fax contact, postal contact, and email contact. However, no matter how business purposes are defined, they must be connected with data purposes. We now introduce a purpose hierarchy to support this.

**3.1.3. Purpose hierarchy.** The relation between purposes can be modeled with a purpose hierarchy. The purpose relation is a partial ordered relation. A partial order is a reflexive, transitive, and antisymmetric relation. Partial ordered relations support complex purpose hierarchies, such as tree, inverted tree, and lattice structures. We employ the use of a purpose hierarchy to map high-level data purposes to low-level business purposes. If an operation is allowed for a given purpose, it is also allowed for all sub-purposes. Figure 1 illustrates a sample hierarchy for the marketing purpose. In this example, email marketing, postal marketing, and phone/fax marketing are sub-purposes of both direct marketing and third-party marketing.

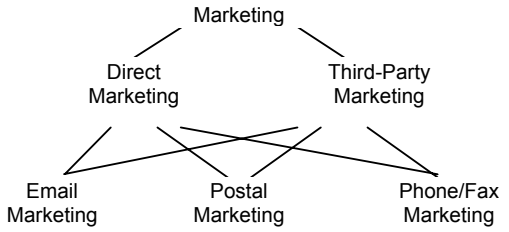



**Figure 1. Purpose hierarchy for marketing**

Purpose hierarchy allows unambiguous purpose lookup from business purposes to data purposes. The following is an example of an ambiguous purpose lookup. If a customer consents to have his personal information used only for email marketing purpose, the access decision of an operation (i.e. whether the data access request is granted or denied) with the purpose of direct marketing cannot be determined. This is because email marketing belongs to both the direct marketing and third-party marketing purposes. The system cannot determine its exact parent purpose.

The above problem can be solved by placing restrictions on the purpose hierarchy. We only allow business purposes to be mapped to the lowest level of the purpose hierarchy. The purpose for an operation must be defined as specifically as possible. In this way, data purposes are either in the same level as business purposes or in a higher level. This ensures there are no ambiguous purpose lookups from business purposes to data purposes.

## 3.2. Conditions

A privacy policy may express additional conditions that must be satisfied before a data access request can be granted. For example, one FIP principle is choice/consent, which means the data collector should provide opt-in/opt-out choices for consumers to allow

[1] There is some inconsistency in P3P1.0 specification. In the P3P1.0 XML DTD Definition (Non-Normative), two other purposes are defined: customization and profiling, which are not defined in XML Schema Definition (Normative).

them to decide how their personal information can be used. In the following sample privacy goal extracted from our goal library [AE03], *$G_{18}$: OPT-OUT from receiving emails from our company*, the access to customer data (e.g. email addresses) must be qualified by the condition Customer.EmailService.Optout = FALSE. In another example, *$G_6$: PREVENT disclosing PII (Personally Identifiable Information) without consent*, "obtaining consent" is a condition that must be satisfied if an organization wants to disclose PII.

Conditions are not solely for privacy protection. In security enforcement, conditions are usually modeled as authorization constraints [RZF01].

### 3.3. Obligations

Obligations are actions that must be carried out if a request to access data is granted. For example, in goal, *$G_{49}$: REQUIRE affiliates to destroy customer data after service are completed*, "destroy customer data" is an obligation for affiliates.

In current website privacy policies, obligations are seldom stated. We have reexamined the 171 privacy requirements taxonomy goals identified from 23 websites' privacy policies during the goal-mining exercises [AE03]. The above example is the only one we identified that involves obligations out of 171 privacy goals.

Obligation-based security policies can be enforced if they can be completely resolved within an atomic execution [RZF01]. However, with respect to the obligations in privacy policies, they are usually not an immediate action as the previous sample policy has shown. In most cases, it is a task or an action that should be executed in the future. Therefore, monitoring and auditing the execution of privacy obligations might be sufficient for obligation enforcement [BJW02].

## 4. The framework for modeling privacy requirements in role engineering

This section presents the goal-driven framework for modeling privacy requirements in role engineering. The framework includes a context-based data model and a goal-driven role engineering process. The data model expresses how the privacy elements can be modeled in RBAC. The goal-driven role engineering process addresses how privacy elements modeling can be achieved in the role engineering process.

### 4.1. A context-based data model

The data model models three privacy elements (purposes, conditions, and obligations) as attributes of roles, permissions, and objects, which we name contexts. Figure 2 depicts the data model architecture. We now discuss how these three elements are modeled in our framework.

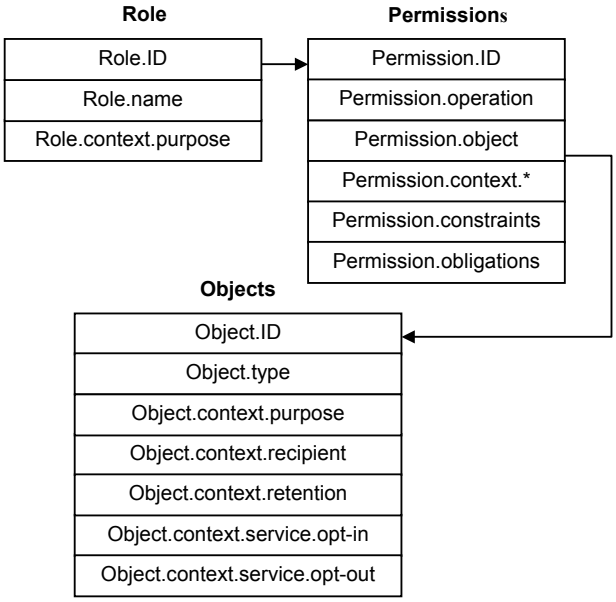



**Figure 2. A context-based data model**

Business purposes are identified in the role finding/definition process of role engineering. They are mapped as an attribute of roles, which we name *Role.context.purpose*. When a role is derived from a business process or an organization structure, some purposes are implicitly embodied. It is the job of role engineering to elicit and explicitly define these purposes associated with a role. For example, *system administrator* role implies that the purpose of this role is *administration*. From a more accurate and more specific aspect, business purposes not only depend on the role, but also depend on the operation the role intends to perform and the context under which the operation is performed. However, provided that business purposes are usually high-level and the number is limited, as described in Section 3.1, it is acceptable to associate business purposes with roles. In an RBAC model with role hierarchies, the super-role automatically inherits all the purposes associated with its sub-roles. This is different from the purpose relationship in the object model, in which a subtype object inherits all the purposes associated with its supertype object. This is not inconsistent because the purposes associated with roles are business purposes while the purposes associated with objects are data purposes.

Data purposes and other privacy preferences, such as the recipient of data, the retention period of data, etc., are modeled as object attributes in our data model. This work is more appropriate for data management than for role

engineering. In this paper, we assume that data are organized into the specified structure. In our framework, object attributes are operands of permission constraints, as we will discuss now.

The conditions of an operation specified in a privacy policy are modeled as permission constraints. Permission constraints are Boolean expressions. The operands of these expressions are attributes of roles, permissions, and objects. The operators of these expressions include standard comparison (i.e. <, >, =, <=, >=, and !=) and logical operators (i.e. Boolean AND, OR, and NOT). To extend the constraint for purpose comparison, we informally define another type of operator for purpose comparison: <<.

**Definition:** Given two purposes *p1* and *p2*, we claim purpose *p2* contains *p1* (or purpose *p1* belongs to *p2*) if and only if *p2* is on the path from the root of the purpose hierarchy down to *p1* or *p2* is the same as *p1*, which is represented as *p1* << *p2*.

Based on the above definition, the permission constraint to enforce purpose binding is

*Role.context.purpose* << *Object.context.purpose*

The obligations of an operation are modeled as permission obligations that should be executed afterwards. As we discussed in Section 3.3, obligations in privacy policies are usually not immediate actions, and they are not enforced by the reference monitor. In our framework, we record such obligations so that the reference monitor can send these obligations to another module (e.g. an obligation execution module) for future execution and monitoring.

The proposed context-based data model is inspired from [KKC02], in which Kumar et al. extends RBAC by introducing the notions of role context and context filters. Kumar et al. employs user context and object context to construct a context filter for a role, which is named role context. However, this approach is not suitable for modeling purposes because business purposes are not associated with users or objects. This approach does not consider the context of roles and permissions. Our data model assimilates the basic idea from [KKC02] but goes beyond that in scope. We also take role context and permission context into account. For example, in addition to purpose, a role may have other attributes, e.g. *Role.context.lifetime* defines the life period of a role. This enables our framework to provide fined-gained, context-based access control. Context-based access control not only takes into account the person attempting to access the data and the type of data being accessed, but also the context of the transaction in which the access attempt is made. This is an additional advantage of our data model. The topic related to context-based access control is beyond the scope of this paper.

### 4.2. A goal-driven role engineering process

We propose a goal-driven role engineering process to demonstrate how the privacy contexts in the above data model can be elicited and modeled. We now discuss the main steps of this process as shown in Figure 3.

The process is comprised of two phases: Role-Permission Analysis (RPA) and Role-Permission Refinement (RPR). These two phases are represented using dotted lines in Figure 3. During the RPA phase, we apply goal- and scenario-oriented requirements analysis techniques to analyzing business process and business tasks. The output of this phase is a collection of role candidates and permission candidates, as well as the corresponding role and permission contexts.

There are several possible input sources: (1) business process description, (2) policy statement (including legislative acts), and (3) requirements specification. The RPA phase starts by identifying tasks. Usually a task is performed to achieve some goals. For example, "schedule meeting" is a task in a meeting scheduler system. The goal to perform this task is to schedule a meeting.

After identifying the task domain, one or more scenarios are authored to model the task details. Every scenario contains a sequence of events, each of which

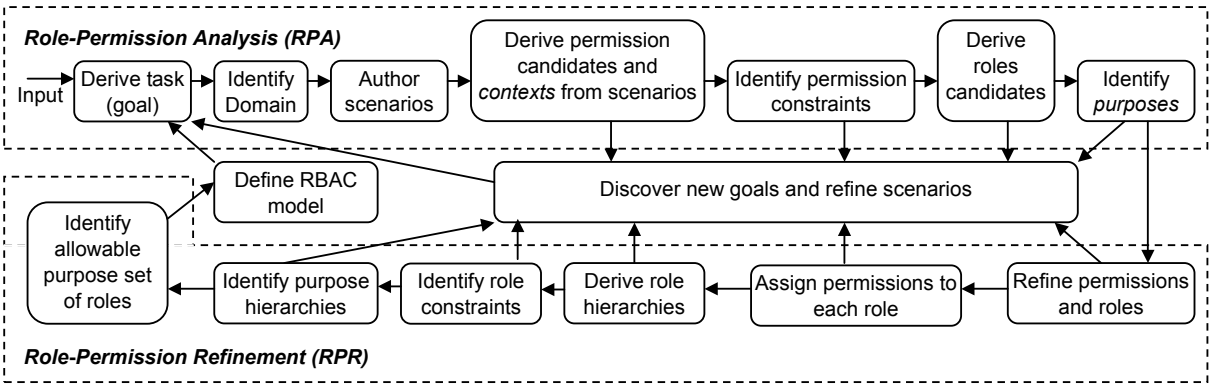



**Figure 3. A goal-driven role engineering process for RBAC**

may be modeled as an RBAC permission. Permission candidates are then identified. The object and operation are the most important elements of a permission. The next step is to identify permission contexts, the attributes of the permission, and permission constraints, the conditions that must be qualified to execute the permission.

After the permission identification step, role candidates can be identified from the actors of events. A set of permission candidates is associated with each role candidate. When a role is identified, the purpose is also identified and associated with this role.

The RPA phase continues until all module tasks have been identified. At this stage, we have a collection of role candidates and permission candidates, as well as the corresponding role contexts and permission contexts. These outputs are needed for the RPR phase.

It is very possible that the RPA phase does not generate a perfect role and permission set. The roles and permissions identified at this time are probably ambiguous and redundant. They must be refined in the RPR phase according to other factors, such as organization structure, policy statement, etc. As a result of role refinement, role hierarchy is defined and appropriate permissions are assigned to the roles. Finally, after all the purposes are identified, purpose hierarchies are defined and a role's allowable purpose set is identified. The RBAC model is defined thereof.

Although requirements analysis and role engineering analysis are interleaved in the above description, actual practices may not have to follow the exact sequence in Figure 3. Some requirements engineers may find it comfortable to complete requirements analysis first and then conduct role engineering analysis. Our example analysis in Section 5 adopts this scheme.

This process is convenient for modeling privacy requirements because it is easy to model the context of goals and permissions with goal- and scenario-based requirements analysis. A scenario's preconditions express possible permission constraints. The postconditions are possible obligations. The goal identified in this process is the possible purpose of the task and the possible purpose associated with a role. However, the RPR phase does not depend on the goal- and scenario-based requirements analysis. Other heuristics must be provided to facilitate role/permission refinement and the definition of role hierarchies.

The process shown in Figure 3 is simplified from a more complete life-cycle goal-driven role engineering process, which we are currently developing [He03b].

## 5. A healthcare example

This section presents an example analysis of a HIPAA scenario using our Scenario Management and Requirements Tool (SMaRT) [SMaRT03]. SMaRT is a web-based tool that supports scenario- and goal-based requirements analysis. It has been successfully applied in several case studies [AA03]. Because SMaRT does not currently support role engineering analysis, the derivation of RBAC elements was documented using a spreadsheet. We plan to extend SMaRT to support the proposed goal-driven role engineering process.

Consider the healthcare scenario below that is readily available in [HIP03]:

*A patient, Mr. Stalwart, is brought to a hospital's Emergency Department (ED). He is unresponsive with a gunshot wound (GW) to the abdomen. Upon his arrival, Dr. Goodcare examines the patient, and begins resuscitative efforts.*

*First, the ward secretary (WS) registers Mr. Stalwart into the ED system. According to HIPAA security regulations, four security and privacy requirements apply to this task:*

- *The secretary needs to have been trained in privacy and security.*
- *The hospital must document this training.*
- *The ward secretary needs to have been authenticated by the system, and his/her authority to perform the registration task confirmed (RBAC).*
- *The system should maintain an audit trail of information viewed and modified.*

The result of our scenario analysis is shown in Figure 4. The elements that appear above the line in Figure 4 correspond to the RE activities whereas the elements that appear below the line correspond to the role engineering activities. We now walk through the goal-driven role engineering process with the scenario.

We first conduct the goal-based requirements analysis process. From the task description, we identify the task domain is *ED Patient Info Management*, and the goal of this task is to *register patient into the ED system*. Then we author a scenario to model the task. To model a complex task, more than one scenario may be needed. A sequence of events is elicited to illustrate the scenario. An event includes an actor and an action. A collection of actors and actions are then identified. The preconditions are identified by asking what conditions must be satisfied to perform this task. The postconditions are identified by asking what are the results of the task, and what are the obligations if the task is performed. The information about the registration process may be obtained via interview with stakeholders or from existing job description manuals.

Based on the requirements analysis, we can then conduct the role engineering analysis. First, we map the actions to permission candidates and identify permission constraints from preconditions. We also identify permission obligations from postconditions, if there are any. After that, we identify role candidates and the

purposes of the task. We associate this purpose with the role and model it as a role context. The roles are then associated with appropriate permissions. These are the major steps in the RPA phase.

[**Goal**] Register patient into the ED system
[**Domain**] ED Patient Info Management
[**Scenario**] Ward secretary registers patient into the ED system
[**Actors**] Ward secretary
System
[**Actions**] Invoke patient registration procedure
Request PHI (Protected Health Information)
Enter PHI
Submit PHI
Save PHI
Confirm PHI saved
Generate audit trail
[**Events**] Ward secretary invokes patient registration procedure
System requests PHI
Ward secretary enters PHI
Ward secretary submits PHI
System saves PHI
System confirms PHI saved
System generates audit trail
[**Preconditions**] Ward secretary authenticated
Ward secretary trained in privacy and security
Hospital security and privacy training process documented
[**Postconditions**] Registration audit trail generated
Patient registered in the ED system

[**Permissions**] P1: can invoke patient registration procedure
P2: can enter PHI
P3: can submit PHI
P4: can request PHI
P5: can save PHI
P6: can confirm PHI saved
P7: can generate audit trail
[**Permission Context**] No permission context identified
[**Permission Constraints**] user.training = T AND user.training.documenting = T
[**Permission obligations**] No permission obligations identified
[**Roles**] Ward Secretary (WS)
System (S)
[**Role Context**] WS.purpose = patient registration
[**Role Permission Assignment**] WS (P1, P2, P3)
S (P4, P5, P6, P7)
[**Allowable Purpose Set**] APS (WS) = {patient registration}

**Figure 4. A healthcare example**

Because we are only analyzing a single task, this example does not have a collection of roles/permissions nor does it include a role hierarchy, role constraints or purpose hierarchy. Hence, the RPR phase is outside the scope of this example. However, we have specified *patient registration* as one of the allowable purposes for role *WS*. Although we have only elaborated one scenario, other plausible scenarios would typically be identified and elaborated as well. For example, *Dr. Goodcare requests patient record* and *Ward Secretary updates patient status*.

Because the system is an agent that performs some tasks, we also model System as a role in the example. Generally speaking, we only model the permissions and roles from a user's perspective. The system's permissions are built into the implementation program. Note that the derived permissions may depend on the implementation. If the system is designed so that whoever can invoke the patient registration procedure has full control of everything in the procedure, then the three permissions assigned to role *WS* can be merged into one: *can invoke patient registration procedure*.

The above example analysis is only a proof-of-concept evaluation of the framework. We are validating the approach in the specification of Transnational Digital Government (TDG) project for Belize and Dominican Republic [Cav03]. This study will allow us to evaluate the effectiveness, scalability as well as suitability of our framework for integration with other RE methodologies.

## 6. Conclusions and future work

Privacy enforcement is important for many commercial software systems. Modeling privacy requirements in the early stages of system development is essential for privacy enforcement and ensuring quality in software systems used in environments that pose risks of loss as a consequence. This paper presents a framework for modeling privacy requirements in role engineering. Basic privacy requirements such as purpose binding can be modeled as permission constraints. Privacy preferences, such as opt-in/opt-out choices, data recipient, etc., can also be modeled using the context-based data model. The framework provides a basis for enforcing privacy requirements with RBAC.

Our framework demonstrates that RE can bridge the gap between high-level privacy requirements and low-level access control policies. Requirements engineers can elicit and model privacy requirements as RBAC entity contexts and constraints by analyzing business processes and privacy policies using the goal-driven role engineering process. Privacy officers can then define privacy authorization rules based upon the context-based data model. These rules are similar to the access control rules derived from security policies and they are enforced via RBAC.

Our framework also demonstrates that RE can bridge the gap between competing stakeholders' security and privacy requirements, i.e., companies' privacy practices may be in conflict with user preferences. The approach presented in this paper allows both perspectives to be modeled (e.g. business purposes and data purposes) and tradeoffs to be analyzed.

Our role-engineering process is a top-down approach; we derive roles and permissions based on business process analysis. Industry experiences report role analysis should ideally be a mixed bottom-up and top-down

approach [Sch00, KKS02]. Our framework can be used with other bottom-up approaches to achieve best result.

Although our work is preliminary, early validation in the TDG project [Cav03] suggests that we will be able to address some of the following limitations in the future.

One limitation of the goal-driven role engineering process is that it is only effective in deriving functional roles/permissions in RBAC. Unfortunately, goals and scenarios are difficult to derive permissions that result from the chosen technology instead of functionality, for example, internal web server functions for a web-based application [NS02].

Our framework can model purpose binding but cannot directly model another privacy requirement, the principle of necessity. The principle of necessity can be enforced by RBAC if each task is granted a minimum set of permissions and users are allowed to perform one current task at the same time [Fis01]. Therefore, it is possible to support this requirement with our context-based data model by expressing tasks as permission context. We plan to support this in the future.

Recall in our example four HIPAA security and privacy requirements were identified from a policy statement. However, our framework does not address how to extract corresponding security and privacy requirements from existing legislative acts and organizational policies. We plan to develop techniques to elicit such requirements and associate them with the tasks we are modeling. Modal-Action Logic (MAL) [GF91] is one promising technique that we are exploring.

The goal-driven role engineering process described in Section 5 is high-level. Only the RPA phase is elaborated in this paper. We are developing detailed heuristics to elicit and refine roles, permissions, and role hierarchies. We also plan to integrate the proposed role engineering process into SMaRT to provide tool support.

## Acknowledgements

This work is partially funded by NSF ITR Grant #0113792. The authors wish to thank Dr. Peng Ning for discussions concerning security and privacy protection and William Stufflebeam for helpful comments.

## References

[AA03] T. A. Alspaugh and A. I. Antón. Contrasting Use Case, Goal, and Scenario Analysis of the Euronet System, *Submitted to the 11th IEEE International Requirements Engineering Conference (RE'03)*, 2003.

[AE01] A. I. Antón and J. B. Earp. Strategies for Developing Policies and Requirements for Secure Electronic Commerce Systems, In *E-Commerce Security and Privacy*, edited by A. K. Ghosh, Kluwer Academic Publishers, pp. 29-46, 2001.

[AE03] A. I. Antón and J. B. Earp. A Requirements Taxonomy to Reduce Website Privacy Vulnerabilities, To Appear: *Requirements Engineering Journal*, Springer-Verlag, 2003.

[AEP01] A. I. Antón, J. B. Earp, C. Potts, and T. A. Alspaugh. The role of Privacy and Privacy Values in Requirements Engineering, *IEEE 5th International Symposium on Requirements Engineering (RE'01)*, pp. 138-145, 2001.

[AEC02] A. I. Antón, J. B. Earp and R. A. Carter. Aligning Software Requirements with Security and Privacy Policies, *Proc. of International Workshop on Requirements Engineering for Software Quality (REFSQ'02),* 2002.

[AER02] A. I. Antón, J. B. Earp and A. Reese. Analyzing Web Site Privacy Requirements Using a Privacy Goal Taxonomy, *Proc. of the 10th Anniversary IEEE Joint Requirements Engineering Conference (RE'02)*, Sep. 2002.

[AMP94] A. I. Anón, W. M. McCracken and C. Potts. Goal Decomposition and Scenario Analysis in Business Process Reengineering, *Proc. of the 6th International Conference on Advanced Information Systems Engineering (CAiSE''94)*, Utrecht, The Netherlands, pp. 94-104, 6-10 June 1994.

[Ant96] A. I. Antón. Goal-Based Requirements Analysis, *Proc. of the 2nd IEEE International Conference on Requirements Engineering (RE'96)*, pp. 136-144, April 1996.

[AP98] A. I. Antón and C. Potts. The Use of Goals to Surface Requirements for Evolving Systems, *Proc. of the 1998 International Conference on Software Engineering (ICSE'98)*, pp. 157-166, Kyoto, Japan, ACM, April 1998.

[AS00] G.-J. Ahn and R. Sandhu. Role-Based Authorization Constraints Specification, *ACM Transaction on Information and Systems Security*, Vol. 3 (4), pp. 207-226, Nov. 2000.

[BJW02] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation Monitoring in Policy Management. *Proc. of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, IEEE, 2002.

[Cav03] V. Cavalli-Sforza et al. Enabling Transnational Collection, Notification, and Sharing of Information, Accepted, to appear in *Proc. of the 2003 National Conference on Digital Government Research*, May 2003.

[CIN02] R. Crook, D. Ince, and B. Nuseibeh. Towards an Analytical Role Modelling Framework for Security Requirements, *Proc. of the 8th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02)*, Essen, Germany, 2002.

[Coc97] A. Cockburn. Structuring Use Cases with Goals, *Journal of Object-Oriented Programming*, Vol. 10 (5), pp. 56-62, 1997.

[Coy96] E. J. Coyne. Role Engineering, *Proc. of the 1st ACM Workshop on Role-Based Access Control (RBAC'96)*, Gaithersburg, MD, 1996.

[DD82] D. E. Denning and P. J. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[Eli02] *Eli Lilly Settles FTC Charges Concerning Security Breach*, Federal Trade Commission, http://www.ftc.gov/opa/2002/01/elililly.htm, January 2002.

[Eps02] P. A. Epstein. Engineering of Role/Permission Assignments, *Ph.D. Dissertation*, School of Information Technology and Engineering, George Mason University, Fairfax, VA, 2002.

[ES01] P. Epstein and R. Sandhu. Engineering of Role/Permission Assignments, *Proc. of the 17th Annual*

*Computer Security Applications Conference (ACSAC 2001)*, pp. 127-136, IEEE, 2001.

[ES99] P. Epstein and R. Sandhu. Towards A UML Based Approach to Role Engineering, *Proc. of the 4th ACM Workshop on Role-Based Access Control (RBAC'99)*, pp. 135-143, 1999.

[FH97] E. B. Fernandez and J. C. Hawkins. Determining Role Rights from Use Cases, *Proc. of the 2nd ACM Workshop on Role-Based Access Control (RBAC'97)*, pp. 121-125, 1997.

[FIP98] Fair Information Practice Principles, Privacy Online: A Report to Congress (Part III), FTC, http://www.ftc.gov/reports/privacy3/fairinfo.htm, June 1998.

[Fis01] S. Fischer-Hübner. IT-Security and Privacy, *Lecture Notes in Computer Science 1958 (LNCS 1958)*, Springer-Verlag, 2001.

[FSG01] D. F. Ferraiolo, R. Sandhu, S. Gavrila, et al. Proposed NIST Standard for Role-Based Access Control, *ACM Transactions on Information and System Security*, Vol. 4 (3), pp. 224-274, August 2001.

[GF91] S. J. Goldsack and A. C. W. Finkelstein. Requirements Engineering for Real-Time Systems, *Software Engineering Journal,* Vol. 6 (3), pp. 101-115, May 1991.

[GHS00] J. Goldman, Z. Hudson and R. Smith. *Privacy: Report on the Privacy Policies and Practices of Health Web Sites*, California HealthCare Foundation, January 2000 http://www.chcf.org/topics/view.cfm?itemID=12497.

[GLB01] *Gramm-Leach-Bliley Act: Financial Privacy and Pretexting*, Federal Trade Commission, http://www.ftc.gov/privacy/glbact/index.html.

[He03a] Q. He. Privacy Enforcement with an Extended Role-Based Access Control Model, *NCSU Computer Science Technical Report*, TR-2003-09, 2003.

[He03b] Q. He. A Goal-driven Role Engineering Process for Privacy-Aware RBAC Systems, *Submitted to the 11th IEEE International Requirements Engineering Conference (RE'03) Doctoral Symposium*, 2003.

[HIP96] *The 1996 Health Insurance Portability and Accountability Act (HIPAA)*, HEP-C ALERT, http://www.hep-c-alert.org/links/hippa.html.

[HIP03] *Case Study - How HIPAA affects a patient visit*, NPower NY, http://www.npowerny.org/case+study+6.pdf, 2003.

[Kai00] H. Kaindl. A Design Process Based on a Model Combining Scenarios with Goals and Functions, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 30 (5), pp. 537-551, Sep. 2000.

[Kai95] H. Kaindl. An Integration of Scenarios with their Purposes in Task Modeling, *Proc. of the 1995 Symposium on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS'95)*, pp. 227-235, ACM, Aug. 1995.

[Kav02] E. Kavakli. Goal-Oriented Requirements Engineering: A Unifying Framework, *Requirement Engineering Journal*, Vol. 6 (4), pp. 237-251, 2002.

[KKC02] A. Kumar, N. Karnik, and G. Chafle. Context Sensitivity in Role-based Access Control, *ACM SIGOPS Operating Systems Review*, pp. 53-66, July, 2002.

[KKS02] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett. Observations on the Role Life-Cycle in the Context of Enterprise Security Management, *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pp. 43-51, 2002.

[KS02] G. Karjoth and M. Schunter. A Privacy Policy Model for Enterprises, *Proc. of the 15th IEEE Computer Security Foundations Workshop*, pp. 271-281, IEEE, 2002.

[Lam01] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour, *Proc. of the 5th International Symposium on Requirements Engineering (RE'01)*, pp. 249-262, IEEE, 2001.

[NS02] G. Neumann and M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles, *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pp. 33-42, 2002.

[OEC80] *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*. Organization of Economic Cooperation and Development (OECD), 1980, http://www1.oecd.org/publications/e-book/9302011E.PDF.

[OSM00] S. Osborn, R. Sandhu and Q. Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies, *ACM Transactions on Information and System Security*, Vol. 3 (2), pp. 85-106, May 2000.

[P3P02] *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, The World Wide Web Consortium, April 16, 2002, http://www.w3.org/p3p/.

[RSA98] C. Rolland, C. Souveyet, and C. B. Achour. Guiding goal modeling using scenarios, *IEEE Transactions on Software Engineering*, Vol. 24 (12), pp. 1055-1071, 1998.

[RSW00] H. Roeckle, G. Schimpf, and R. Weidinger. Process-Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a Large Industrial Organization, *Proc. of the 5th ACM Workshop on Role-Based Access Control (RBAC'00)*, pp. 103-110, 2000.

[RZF01] C. N. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An Access Control Language for Security Policies with Complex Constraints, *Proc. of Network and Distributed System Security Symposium (NDSS'01)*, pp. 89-107, 2001.

[SCF96] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. Role-Based Access Control Models, *IEEE Computer*, Vol. 29 (2), pp. 38–47, Feb. 1996.

[Sch00] G. Schimpf. Role-Engineering Critical Success Factors for Enterprise Security Administration, *Proc. of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, 2000.

[SMaRT03] Scenario Management and Requirements Tool. http://tigger.csc.ncsu.edu/~smart/

[Toy00] *FTC says Toysmart violated child Net privacy law*, Federal Trade Commission, http://www.ftc.gov/opa/2000/07/toysmart.htm, 2000.

[ZAC02] L. Zhang, G.-J. Ahn, B.-T. Chu. A Role-Based Delegation Framework for Healthcare Information Systems, *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pp. 125-134, 2002.