Fraunhofer Institut
Experimentelles
Software Engineering

User Documentation Based Product Line Modeling

Authors:
Isabel John
Jörg Dörr
Klaus Schmid

This work is a deliverable of the Eureka Σ!2023 Programme, ITEA (ip00004, Project CAFÉ) Ergebnis Bericht Nr. E 2.1 "Erstellen von Produktfamilien-anforderungen unter Berücksich-tigung individueller Systembe-schreibungen", for the subpro-ject CAFÉ (funded by BMBF).

IESE-Report No. 004.04/E
Version 1.0
January 15, 2003

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

This report describes an approach for extraction of product line requirements based on existing user documentation. The approach we describe in this report supports capturing of the information found in user documentation of legacy systems, e.g., user manuals, and the specification of this information in product line models, using, e.g., Use Cases. The main goal of the approach is expert load reduction; as we found out during our product line technology activities at IESE that the domain experts are a serious bottleneck during the product line scoping and modeling activities.

The approach is based on a variability modeling approach that allows integrating arbitrary modeling formalisms to capture the extracted requirements in a product line model. We propose a conceptual model describing the transition from user documentation to product line artifacts describing common and variable elements of a product line model. The approach uses extraction patterns that allow an easy identification of text elements in user documents. The pattern are then used to create a significant part of the requirements specification and product line model, respectively and we present an extraction process that guides the user in using the pattern. In this report we present the approach itself and show the application of our approach for three case studies, one from the embedded/ mobile phone domain, two from the information system domain. The case studies have shown that the approach is applicable to information systems and embedded systems and that it produces relatively complete and correct results. With one of our case studies that was performed in real industrial settings we could not only show expert load reduction when applying the approach but also an overall time saving when applying the approach for product line scoping. A predecessor of this report with a different focus can be found in [JD03b].

Keywords: Product Line Engineering; Requirements Engineering; User manuals; Documentation; Product Lines; Product Families; Commonality; Variability; Use Cases; Features; Extraction; Elicitation, Café, Families, ITEA

Table of Contents

List of Tables

List of Figures

1 Introduction

During the last decade reuse has become a key factor for improving software development efficiency and many different methods have been developed. Product line engineering can be described as an approach 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.

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.

Product Line Engineering includes the construction of a reusable asset set. With Product Line Engineering commonalities and variabilities of products in a domain are captured, modelled and implemented in parallel and stored in an asset base. 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 and scoping, elicitation, analysis, modeling and realization of the commonalities and variabilities between the planned products. So, all tasks that are known from single system engineering have to be done but in a more sophisticated way by thinking about several products and realizing solutions in a generic way.

1.1 Problem description

When starting a product line in an organization there normally are legacy systems which have been built in the product line domain. Legacy systems of the organization can be old systems which already have been completed or systems currently under development and not integrated into the product line yet. The integration of those existing systems into a planned and to be built product line can happen on different levels:

- Analysis and Integration of Code
  Some functions might be directly reused. Through reengineering techniques as, e.g. described in [DeB96] not directly reusable code can be integrated into the product line.
- Analysis and Integration of existing modules or components
  It might be possible to identify components which encapsulate parts of the old systems which can be integrated into the common basis of the product line [ABM00]. Other components might cover functionality, which is specific to only one system and so can be integrated during instantiation of this system. These components can be used like COTS components for the new systems.

- Analysis and Integration of the architecture
  To capture the architectures of the legacy systems, architecture recovery techniques (like [BGW+99]) can be used. In other cases reuse of the old architecture might be possible.
- Analysis and Integration of Requirements
  Functional and non-functional requirements of the old systems can be reused for domain modeling if they exist explicitly.
- Reuse and integration of Knowledge and expertise
  Knowledge or expertise which exists in the organization and is owned by the stakeholders which might be identified in the context of the development organization can be used for the transition of the legacy systems and for modeling and development of the product line.

Each one of these levels has meaningful information for the to be built or already existing product line. There are several approaches for domain analysis and domain modeling You can find an overview on Domain Analysis Methods like FODA [KCH+90], ODM [ODM96] or Commonality Analysis within FAST [WL99] in several surveys like [Ara93] or [DS98]. But in most of these approaches, the integration of legacy systems into the domain analysis phase is not described in depth. ODM [ODM96] is an example where software systems are described as a source of legacy knowledge, which can be integrated into the product line by reverse engineering. But there is lack of systematic support for integration of documentation assets into the lifecycle phases of a product line.

In the case of existing requirements and existing knowledge, the information about the legacy system, which exists on that level, has been recorded in documents. These documents and the information from the documents should be considered when doing a domain analysis for a product line. Here we focus on analysis and integration of requirements and reuse of knowledge in the form of legacy documents. In the context of the introduction of a product line approach in a software development organization the documents produced in former and ongoing software development projects in the domain should be integrated into the engineering process. Through analyzing the documents and integrating their information into the product line, knowledge and requirements can be reused for domain modeling like code can be reused by transformation through reengineering.

Legacy assets should not only be used for reengineering and documentation but also for (Re-)modeling a product line. There is information which is useful for domain modeling in legacy documents and in order to increase completeness During the lifecycle of a software system other documentation assets like customer contracts, any kinds of correspondence concerning the system, notes etc. are produced which are largely unstructured. In many cases there is also other paper based or electronically available material describing a domain, like web sites, books, technical papers etc. that can also be a rich source of domain information. This material most of the time is not produced by the development organization itself and so might lack the product focus the documents produced within the development organization have.

Product Line Engineering is an approach that aims at exploiting reuse potential between products developed in an organization by identifying the commonalities between the products and systematizing the variabilities. In Product Line Engineering,

variabilities have to be identified, modeled, stored, resolved, instantiated and changed.This requires a comprehensive approach to the management of variability that can be applied throughout the various life-cycle stages, their artifacts, and their accompanying notations in an universal manner. Moreover, in order to enable a smooth transition to product line development for an organization that so far did only perform single system development, it is necessary to keep as much of the existing notations and approaches in place as possible. For this reason, we developed a customizable approach to variability management that can be used as a full life-cycle approach [SJ04] but is also applicable for Product line modeling [SJ03] . This allows to practically apply the approach in a wide range of industrial settings. This is particularly motivated by our industrial projects using the PuLSE-approach [GKS+01] [KMS+00], as there we needed an approach that enables us to homogeneously manage variability, independent of the specific notation. So an extraction approach for product lines has to be accompanied by explicit variability management in order to integrate and manage the extracted artifacts at the right stage and at the right place into the product line model.

1.2 Goal of the approach

Until now, the information needed to build a product line model is elicited interactively with high expert involvement [DS98] [KMS+00]. As domain experts have a high workload and are often unavailable this high expert involvement is a risk for the successful introduction of a product line engineering approach in an organization. In this report we present an approach for extraction that overcomes the following problems:

- Domain experts have a high workload and are hardly available so we need to relieve the experts by eliciting of product line related information from documents
- There is a lack of guidance on how to integrate legacy information found in documents into product line models
- There is no variability management approach that is general enough to integrate all kinds of artifacts into a product line model.
- 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
- The principal process is highly interactive.

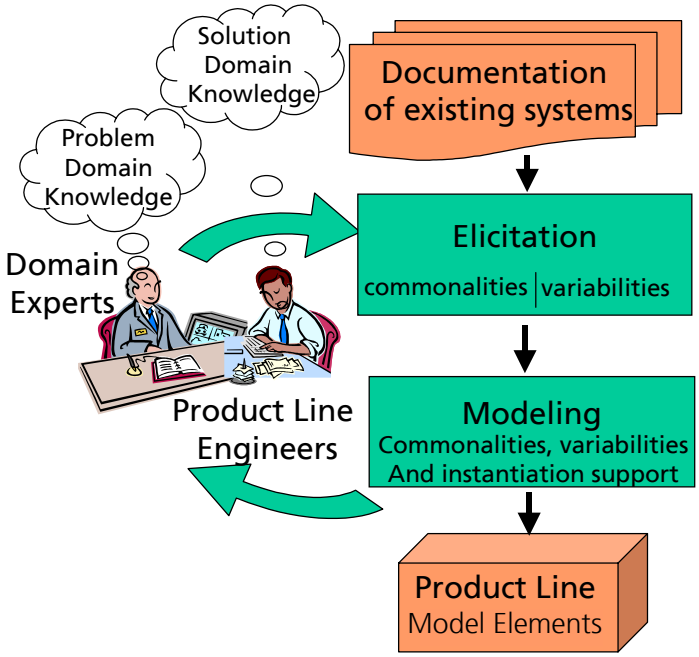


Figure 1 The principal elicitation and modeling process

Figure 1 shows how the modeling process normally looks like. Domain Experts with Problem Domain Knowledge and Product Line Engineers with solution domain knowledge sit together and find out commonalities and variabilities. The documentation of existing systems is an additional source for this elicitation. Then the requirements on the product line are modeled, further Information is elicited and clarified etc. The results are different product line model elements that are collected in a domain model or product line model.

1.3 Elements of the Approach

The elements of the approach that we describe in this report have the following coaction:

Our approach is embedded in product line engineering and produces artifacts that are used for product line modeling. In order to give a common understanding on those fields and on the terms that we use in the following we give a short introduction on product line engineering and product line modeling in section 2.

Product Line Modeling can be done in different ways. For an extraction approach that is independent on the notation it is not so important what is captured (the notation can be adapted) but how model elements are captured. For Product Line Modeling and Product Line Engineering in general, variability is a very important concern. In order to have a basis for the integration of variability information from documentation into the product line model we describe our variability management approach for product line modeling in section 3.

Each useful and scientific approach should be relying on the things that have been done in the past. In order to produce an approach that is on one hand state of the art and practice but does on the other hand not reinvent the wheel the related literature must be searched for similar and helpful approaches. For the problem that we want to solve with our approach "how can we systematically extract information that can be directly used for product line modeling" there is no single research field that can solve the problem but a collection of different research fields like Requirements Engineering, Product Line Engineering, Information Retrieval, Knowledge Engineering and others. We restricted the areas according to the needs of our approach and describe the outcome of the survey on related literature in section 4.1. From this survey we did not find the one approach that fulfils all our requirements but we found some useful techniques from other approaches that we could integrate in our approach (c.f. section 4.2). The possible input and output artifacts that restrict the application of our approach to certain information items are described in sections 4.3 and 4.4

The approach we developed follows the general domain analysis approach but gives support for elicitation and therefore relieves experts. The basis of the approach and of the pattern is a metamodel that we describe in section 4.5. from the metamodel we derived extraction pattern (described in section 5)that transform elements from one part of the metamodel to another part.

The extraction approach itself consists of three steps, preparation, analysis and selection that are described in section 6.It was applied in several case studies, three of them are described in sections 7, 8and 9.

1.4 Research Approach

The research approach we chose is a combination of action research [RB01], a research paradigm that uses an iterative process, which alternates between action and critical reflection, and experimentation [BS86].

First, we analyzed the related literature on requirements engineering, extraction and domain analysis for useful techniques in the context of requirements extraction for product lines. Then we followed the action research paradigm and applied those techniques and some new techniques in two initial explorative case studies. We built a first version of our approach and a conceptual model underlying our approach in parallel in short improvement cycles when performing the case studies. After stabilizing this first version we went over to the experimental phase. Currently we are also performing industrial case studies to validate the approach. We expect to further improve the approach when we have applied further case studies and conducted and evaluated an experiment. In the last phase we will finalize the approach and conduct a final experiment to further sustain our research hypotheses.

2 Product Line Engineering

In this section we describe the area where our approach can be applied, which is product line engineering and product line modeling.

2.1 Product Line Engineering

Product line engineering stands on a middle ground between specific assets (single system development, one time use) and general assets (general use libraries or components, applicable anywhere; generator based approaches). The idea is to define a product line, which captures the intended scope of reuse. The products included in a product line are determined to have sufficiently common characteristics to make it more efficient to study the commonalities and variabilities once for all products of the product line and to build reusable assets, than to study and build all the products separately. Product line engineering approaches define how to leverage the commonalities and create reusable assets that increase the efficiency of developing the products.

Product line engineering 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 [Bay99a], Fast [Wei99] and the SEI Product Line Practice Initiative [Cle01]. 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 (cf. Figure 2).

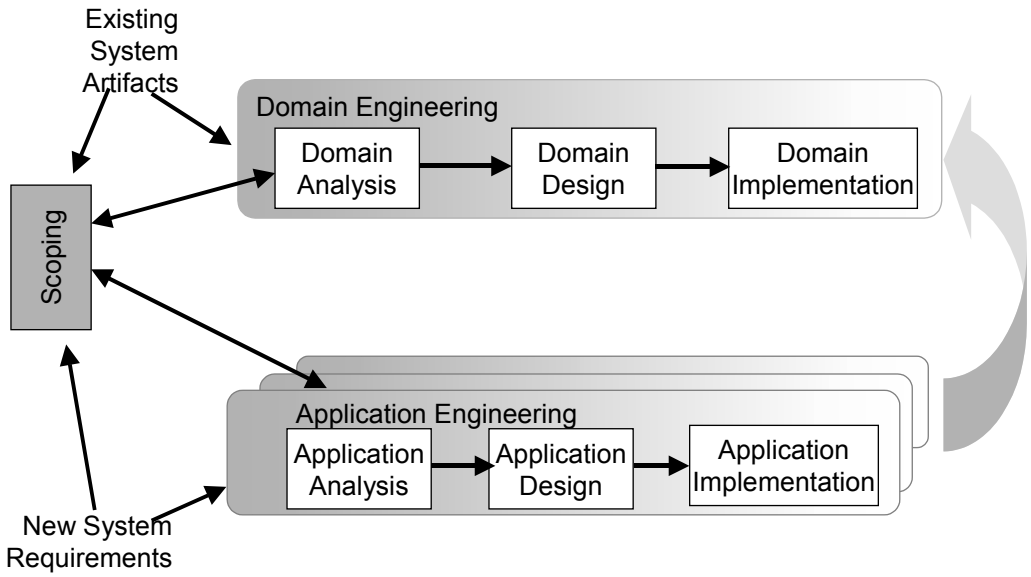


Figure 2 Product Line Engineering

One benefit of the product line approach is the idea that all members of the product line are based on a single set of assets. Thereby, the maintenance effort is reduced to this single asset base. Hence, existing products must always follow the evolution of the asset base. This is done by instantiating the changed domain model while reusing the existing resolution of the domain decision model.

2.2 Product line modeling

Domain analysis or product modeling is requirements engineering for product lines [KS00] [CN01]. 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 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 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.

The special properties, which are specific to requirements engineering for product lines, are:

- 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 functional and non-functional requirements, the commonalities and variabilities between those products have to be captured and adequately modelled.
- Instantiation Support
  As several products are modelled in one domain model it must be clear, which requirement belongs to which product. In order to have an application specific view on the product the instantiation of the generic and variable model for several products has to be supported.
- Decision Modeling
  To get this instantiation support, the decisions that have be made have also to be captured in a separate model. This model collects and abstracts the information on which requirement is instantiated in which product and supports the instantiation.
- 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) are provided.
- Evolution
  Product Lines are a means to cope with evolution. With product lines evolution in space (the space of the planned products) is controlled. When doing domain analysis on a portfolio of planned products evolutionary aspects are integrated and the evolution within the product portfolio is captured through commonality and variability modeling.

There are many approaches for product line modelling, e.g., PuLSE CDA [BMW99] within PuLSE [BFK+99] or Synthesis [Ka92] Most of them use similar notations. Notations and artefacts that are state of the practice in product line modelling are features (see [KCH+90]) and Use Cases (see [Co01]). In addition to these specific artefacts, general textual artefacts like functional requirements, data requirements, project issues, etc. are in use. Our approach aims at collecting information from user documentation to use them in state of the practice product line models. Therefore, we put special attention to the aforementioned artefacts.

3 Product Line modeling with explicit variability management

Variability Management is a concern that arises in Product Line development throughout all lifecycle phases. It can actually be seen as the key feature that distinguishes product line development from other approaches to software development.

We have developed an approach for the systematic management of variability in the specification phase that we will describe here In this product line modelling (or domain analysis) phase, a model of the requirements of the product line is developed which expresses the variability required from the product line. Many different notations are in practical use for requirements engineering [DA99]. While especially text-based and use-case/UML notations are used in the product line context [GS02], it is desirable that an approach supporting the specification of product lines is open regarding the notation. This lead us to the idea for the approach presented here This approach aims to support the modelling of variability for arbitrary specification techniques.

Our approach can actually be extended into an approach, which is sufficient as a basis for variability management across the various lifecycle phases, this extension is described in [SJ04]. However, we will focus here on the specification phase and will provide case studies that substantiate our claim.

3.1 Overview of the approach

The key question of course is: why would one want to be independent of the specification technique? There are two fundamental reasons motivating such an approach:

The scientific reason: such a generic approach could be evaluated in an arbitrary set of contexts, thus facilitating the growth of a scientific body of knowledge about it. As the mechanism is applicable in different contexts and in different domains it can be used in a variety of situations and can therefore be validated much easier than an approach that is applicable only with a single specification technique.

The pragmatic reason: Fraunhofer IESE applies its technologies in many different companies, leading to the need for highly adaptable techniques. As we do technology transfer to companies with different organizational structures, in different sizes and in different domains the approaches we develop must be generic and adaptable to many different contexts.

Particularly the latter reason originally leads to the definition of the PuLSE-CDA approach [BMW00] as part of the PuLSE method [BFK+99]. This approach is a highly customizable domain analysis approach, which can be augmented with the variability management mechanisms described here.

The specific approach to variability management we propose consists of the following components:

- A decision model as a basis for characterizing the effects of variability.
- A range of primitives for describing the relation between variation points and the specific decisions (or group of decisions) on which their resolution depends.
- A common (maximal) set of variation types.
- An accompanying mapping of the variability types on the specific specification techniques to express the variation points.

Only the last point, the mapping, has to be adapted to the specific representation technique. The other three parts as well as the semantic interrelation among the four are independent of the specific representation approach. We will now briefly discuss these four elements.

3.2 The Decision Model

The decision model was initially devised in the context of the Synthesis approach for variability management [Ka92]. In the meantime, this technique has been widely applied both in research and industry [MY00] [MI02].

The specific kind of decision model we propose is different from other approaches in two ways:

It is more comprehensive in terms of the information it contains

It does not explicitly relate to the variation points, but rather it defines a decision variable, which is then only referenced at each specific variation point using the decision evaluation primitives.

Each of the decision variables that is defined in the decision model is in turn described by the following information:

- Name: The name of the defined decision variable; the name must be unique in the decision model
- Relevancy: The relevancy of a decision variable for an instantiation may depend on other decision variables, e.g. the decision variable describing the memory size is only valid if the decision variable describing the existence of memory is true. This can be made explicit by the relevancy information.
- Description: A textual description of the decision captured by the decision variable
- Range: The range of values that the decision variable can take on. This can be basically any of the typical data types used in programming languages. However, instead of a real or integer often only a range is important. Moreover, probably the most common type is the enumeration, as the relevant values are often domain dependent. Further, Boolean variables are quite common.
- Cardinality: As opposed to other approaches, we do not emphasize the difference between variables, which can only assume a single value, and variables that can assume sets of values during application engineering. Rather, we define a selection criterion, defining how many of the values of a decision variable can be assumed by it. This is represented by m–n, where m and n are integers and give the upper- and lower-bounds for the cardinality of the set representing the value of the decision variable in the context of a specific application. Thus, basically, all decision

variables get a set of values during application engineering. However, we use 1 as a short-hand notation for 1–1 and in this case we also write the value of the decision variable as a single value (without curly brackets) and treat it for the purpose of decision evaluation like a non-set value.

- Constraints: Constraints are used to describe interrelations among different decision variables. This is used to describe value restrictions imposed by the value of one variable onto another variable. We use this approach also to describe the required relationship, as this simply results in a special case in our framework. This constraint can of course also contain domain knowledge. Consider for example the following constraint: the value of the decision variable describing the memory size has to be > 16384 if the decision variable describing the existence of memory is true. This constraint at the same time represents the domain knowledge that in the product line the minimum memory size is 16KB.
- Binding times: A list of possible binding times when the decision can be bound. This can be sourcetime, compiletime, installation time, etc.. Additional binding times may exist, and can be product line specific. As opposed to the FODA work and many related approaches, we allow several binding times, meaning depending on the specific product the variability may be bound at any of these times. This technique was first introduced in ODM [ODM96] as "binding sites".

Depending on the specific context of our industrial projects, we sometimes used slight variations of this approach to decision modeling. However, regarding the information content, it was always a subset of this information (if only product line modelling had to be supported) [SBK+00].

Using this description of a decision variable, we can define a decision model simply as a set of decision variable definitions. For practical reasons we usually represent them as a table.

3.3 Decision Evaluation Primitives

The way we defined the decision model, it is completely independent of the variation points in a variability model. In order to relate a decision to a variation point we must explicitly describe it. This is done using the decision evaluation primitives together with the variation point representation as discussed below.

The reason why we do not directly relate the impact of a decision variable to the variation points is that the same decision may easily have many different forms of impact on the variation points. This allows us to decouple the decision itself from its impact on the product line model.

Our approach to decision evaluation is very similar to expression evaluation in existing programming languages, the main extension being that we may need to deal with set values.

The following list provides some examples of relations we use for decision evaluation:

- sub real subset
- subeq subset or equal
- # cardinality of a set
- in is element of a set
- -> logical implication
- <=> mutual implication (iff)

Using these primitives, logical expressions can be built that can be used to denote in which way a specific variability must be resolved. It is also possible to build value expressions. We will discuss this further in the following section.

3.4 Supported Variability Types

Many different variability types have been mentioned in literature: optionalities, alternatives, set-optionalities (a set of options may be selected), etc.

From our practical experience we deem the following variabilities to be the most relevant:

- optionality: a property either exists in a product or not
- alternative: two possible resolutions for the variability exist and for a specific product only one of them can be chosen
- multiple selection: several variabilities may be selected for inclusion in a product
- single selection: only a single variability out of a group of variabilities may be selected for inclusion in a product

- value reference: the value of the decision variable can be directly included in the product line model. (This, of course, only makes sense with decision variables that only assume a single variable in application engineering.)

The optionality and variability refer by nature to a logical expression as constructed using the decision evaluation primitives. Further the multiple and single selection refer to a value expression, as this is used to differentiate among the different possibilities. The value reference, finally, takes an arbitrary decision variable with a single value.

3.5 Representation-Specific Mapping of the Variation Points

As we discussed above, the decision model is basically representation independent. However, we need to represent the variation points in the domain model, which employs a specific specification technique. Therefore we need to map the different types of variabilities to the target specification technique.

As we will see in the next section the specific notation for the variation point may be graphical, textual based, or on any other basis. In order to simplify the adaptation process, we did so far always use the standard description approach for referring to values of decision variables described in Section 2.2.

The different variability types should be mapped in a homogenous manner to the specification language. For each variability type a unique mapping has to be found. This mapping has to take a form so that confusion with other legal expressions in the target specification language can be minimized.

The mapping is domain independent but it depends on the specification mechanism used. Only this mapping from elements of the decision model to the specification formalism has to be adapted when the approach is applied with a new specification formalism. If the specification formalism uses graphical models, the mapping can be done using extra graphical elements with the decision variables as attributes of these elements. If the UML or a similar modelling approach is used as specification formalism, the model elements can be extended (e.g. with stereotypes). If the specification formalism is text, markers for the different kinds of variabilities can be introduced into the textual description.

Once the mapping has been specified, modeling of the product line and capturing the requirements on the system can be performed almost as it was done before. The models are built with the already existing modeling elements together with the new elements that are needed for the mapping. Together with the models, a decision model has to be built. The decision variables that appear in the new model elements and in the decision model ensure the mapping of concepts between the model and the decision model and ensure traceability between the models and also across different development phases.

3.6 Discussion of the Approach

The approach outlined above is sufficient to describe all common forms of variabilities and dependencies among them. For example, alternatives that are mutually exclusive can be represented using an alternative or a single selection, which refer to a decision variable (in the case of the single selection, this is only sufficient if the decision variable can take on only a single value).

The required dependency can also be modelled, and it is actually modelled on the level we believe to be the most adequate: it is made explicit on the level of the decision model in the form of constraints on the possible values of the variable.

3.7 Experiences Using the Approach

The approach to variability management in product line modeling described above has already been applied in several cases, most notably two industrial applications, where one used a graphics-based approach, while the other uses as a text-based approach. We will now briefly discuss the implementation of our approach in these two vastly different contexts, as this nicely illustrates the different forms of mappings that are made.

3.7.1 Experiences with a Graphical Representation

We applied our approach to the variability management in the context of product line modelling in an environment, where a graphical notation was required. This notation was the basis for a business process notation (ARIS), which was in turn the basis for requirements definition for systems of the customer [SBK+99].

The ARIS notation, which provides the basis for this application of the approach focuses on business processes. The basic notational elements are shown in Figure 3. In this specific case additional elements had to be defined in order to represent also sys-

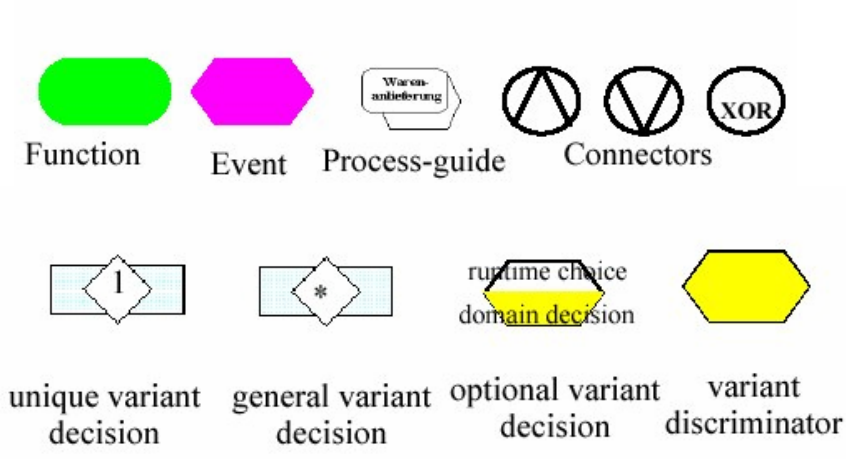


Figure 3 Basic (top row) and extended (bottom row) elements of the ARIS notation

tem internal information and control flows. Our approach focussed on augmenting this notation with additional variability elements that could be used both in business process as well as control flow modelling.

In order to describe selections the ARIS modeling notation uses two notational elements: the connector together with the event. The connector defines the form of selection, the event defines the different cases that can occur and under what circumstances each of these paths is taken. Figure 3 shows such an example business process with a selection. As defined by our approach the first three parts of our approach could be taken verbatim. We only made some minor pragmatic modifications:

The decision model had the same entries as defined above, with the exception of the binding times. There was no need to capture the binding time as this was always implicitly the modeling phase. Further there was one additional entry: the actual values for the various systems could also be defined as part of the decision description. This had pragmatic reasons, as in this case the number of decision variables was limited and especially the number of systems was small. The decision model was then simply written as a web page, as much documentation in this environment was kept in an intranet-based manner.

The decision evaluation primitives were used as described in section 3.3.

Regarding the different forms of variability, we decided to not support the value-reference, as we did not find a case where we would need this approach. Also the alternative is always described as a single selection.

Based on these decisions we defined the mapping of the basic variability types onto the representation mechanisms. In order to enable the users of the approach to clearly differentiate between the basic notation and any variability information, we defined completely different notational elements, which, however, fit into the overall approach. Figure 3 shows the different notational elements we introduced. We also adopted the differentiation between decision symbol (connector) and selectors for a specific flow (event), which is typical of ARIS.

When mapping the various variant discriminators it is key to keep in mind that we are using here a notation that imposes certain restrictions, for example, by removing some variation the overall flow may not fall apart. Thus, we can only remove certain (alternative) paths from the control flow. Based on these restrictions, we mapped the variability types optionality, multiple selection and single selection, we selected for representation, in the following manner:

- Optionality: This implies that a certain path may either be part of a system variant (an instantiation), or not. Thus, we need to attach two forms of information to it: the situation in which this path is part of the final model and if it is part of the final model, the (runtime) situation in which it is actually taken. The second part obviously corresponds to the event mechanism in the ARIS business modeling approach, while the former is the optionality-specific addition. We thus added the optional variant decision to the modelling language. As shown in Figure 4 it consists of a runtime decision and a domain decision part. The domain decision part in turn uses the decision evaluation primitives as described in Section 2.2 in order to describe whether the branch started with this decision should be part of the in-

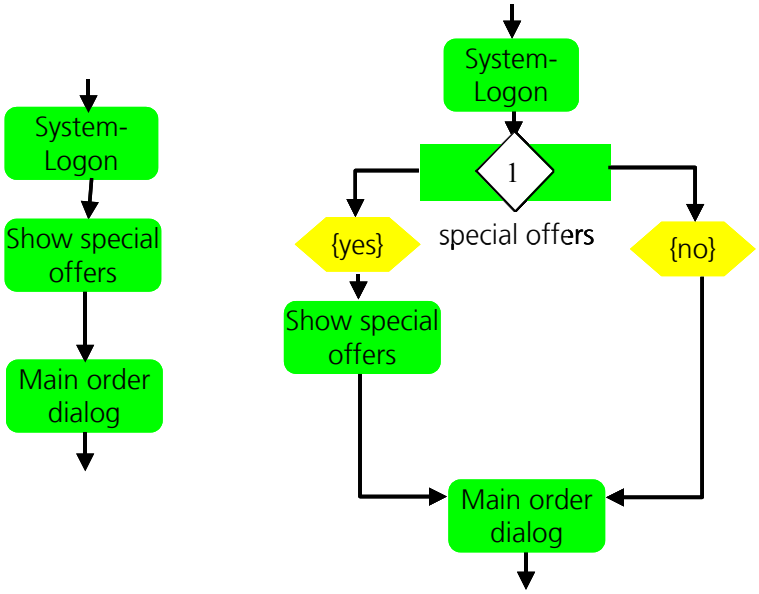


Figure 4 Simple (left) and extended (right) business process examples

stantiated model. The runtime decision part in turn is annotated using the ARIS-notation in order to describe what will happen in case this branch is selected.

- Single Selection: The single selection is mapped to the unique variant decision (cf. Figure 4) which works similar to a runtime decision in ARIS, with replacing connectors by the unique variant decision, and the events to the variant discriminator (cf. Figure 4). In this case we restricted the expression for selecting among the various paths to a decision variable, with the variant discriminators showing the different values. Note, that upon resolution of the variability none of the notational symbols for variability will remain in the instantiated flow.

- Multiple Selection: The multiple selection has been mapped in very much the same way as the single selection. The main reason for having both of them was clarity of the instantiation semantics. In a work flow (or control flow) representation like ARIS, a runtime decision must remain upon resolution of the specification variability in the case of a multiple selection. This is different from the single selection where all variability symbols are removed upon instantiation. Here, they are transformed into run-time variability (if more than one option is chosen).

This approach to modeling was used for modeling several systems in the domain of merchandising information systems and about ten e-commerce shops. We found this approach to be easily applicable to these systems. Especially in the e-commerce context it was also well accepted by the development personnel.

3.7.2 Experiences with a Text Based Representation

Our variability management approach has been applied in practice also with text-based requirements in an embedded systems company. A textual representation was chosen because the stakeholders in the domain were very familiar with textual repre-

sentations and not with other forms of requirements documents. They had also invested considerable effort into the improvement of their approach to textual requirements documentation.

In order to be able to model and manage variability, the existing mechanisms for writing textual requirements had to be extended into a product line modelling approach. According to our approach, only the mapping of the variability types onto the target representation formalism had to be adapted. However, to be complete, we will now briefly describe the specific realization of all four components of our approach.

The decision model as described in section 3.2was introduced. This was realized using an Excel-table. A sanitized version of such a table excerpt is shown in Figure 6.

We used the decision evaluation primitives shown in Section 3.3.

We did decide to not support the single selection, as it is a special case of the multiple selection. Moreover, so far most instances we found during our work in this domain were instances of the multiple selection anyway.

This shows that, as expected, we could transfer our concepts in a straight-forward manner to this domain. This leads to the most interesting part of the case studies: how was the mapping of the variation point types performed.

For this mapping the variability types onto the textual specification we decided to use textual constructs framed with "<<" ">>", as these are text fragments which did so far never occur in this domain.

Thus, we wrote optional variability in the following way:
`<<opt expr1 / text >>.`
Similarly, for alternative variability, we used the term:

```
<<alt  expr2   / value-1 /  text1
                       / value-2 /  text2 >>.
```

Here expr1 and expr2 are logical expression as discussed above. These expressions could be constructed using the primitives described in Section 3.3.
For multiple alternative variability we restricted the expression to a decision variable

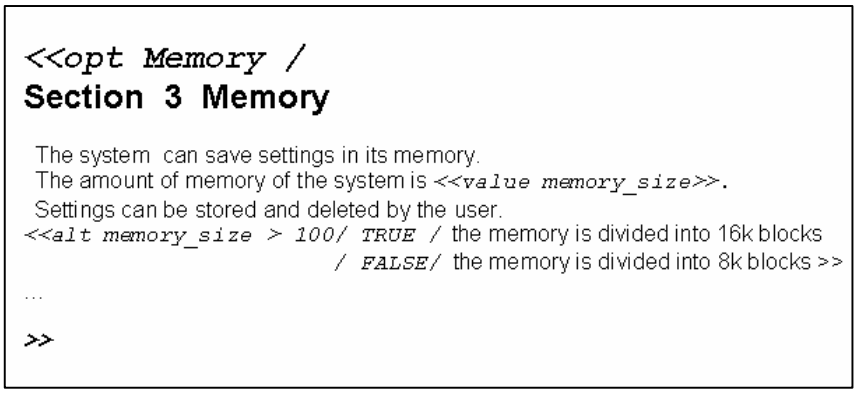


Figure 5 Example for the textual notation

Name	Relevance	Description	Range	Selection	Constraints	Binding times
Memory	System_Mem = TRUE	Does the system have memory?	TRUE/ FALSE	1		Compile time
Memory_ Size		The amount of memory the system has	0, 10, 100, 1000	1	Memory = TRUE => Memory_Size > 0	Installation; System initialisation
Time_Measurement		How is time measurement done?	Hardware, Software	1		Compile time

Figure 6 Example for the description of variants with decision variables

instead of a value expression and introduced the keyword mult:

```
<<mult    decision-variable / value-1 / text1
                            / value-2 / text2
                                        .....>>
```

Finally, for values the term `<<value decision-variable>>` was used.

Using this approach we described the product line model. Figure 5 shows a sanitized excerpt of such a product line model document, which includes optional, alternative, and value variability. The product line model consisted of a document for each of the subdomains we analyzed and an overall decision model in form of a table. As at least one decision variables was used in the text at each decision point there was a unique mapping from the text to the decision model. The sense of the variation point can be derived by interpreting the logical expressions built up from the decision evaluation primitives used in the expressions in the text and in the table with the decision model. Parts of the decision model is shown in Figure 6.

In this company, we identified so far during modeling about 50 decision variables and about 100 variation points had to be introduced into the documentation. We expect that once the product line model is complete, it will contain more than 100 decision variables and several hundred variation points. The resulting domain models went through inspection by the company and were well accepted by the development team. In particular the notation was considered to be well readable and the resulting models to be well understandable.

3.8 Applicability of the approach

We described here an approach to variability modelling for product line models. The development of this approach was driven from the need for an approach that can be easily applied in a wide range of practical contexts and in combination with many different specification techniques. Based on our experiences in applying this approach, we found that our approach to variability management is sufficiently expressive to support modeling variability for arbitrary specification techniques.

Moreover, we could already apply this approach as part of the PuLSE approach in different industrial contexts, demonstrating that it provides sufficient expressiveness for these situations. This variability management approach is the basis for bringing extracted documentation entities that conform to the metamodel described in section 4.5 into a product line model. The variability management approach helps in structuring the unstructured items extracted with the approach described in section 6 into a structured product liner model that is independent of the notation.

4 Basis of the extraction approach

In this section we describe the basis of the extraction approach. The work that was done in extracting information from software engineering related documents is investigated in section 4.1. In order to describe the related work systematically we developed a classification scheme and classified the related work into this scheme. The possible input and outputs of an extraction approach are classified in section4.3 and 4.4, Useful techniques that we found in literature are specified in section 4.2and the conceptual model that is the basis of our approach and was developed by ourselves in different projects is described in section 4.5

4.1 Classification of related work

The area of fields and techniques that can contribute to the elicitation of information that is useful for the product line modeling phase is rather large. The techniques can range from interview techniques over reading techniques to code based techniques from reengineering. In our survey we will only focus on textual information as the primary goal of our work is it to reduce expert load during the product line modeling phase and this can best be realized by analyzing existing textual information that is available (c.f. [Jo01]). Interview techniques can help for structured elicitation but they require a strong expert involvement and so do not help in reducing expert load.

The Subject of the analysis is:

All approaches that contribute to the extraction of common and variable information from existing textual software documentation that is relevant for product line modeling and requirements engineering.

As this is not a fixed area within software engineering, approaches to be analyzed come from requirements engineering, information retrieval, reengineering, product line engineering and other fields.

Although our focus is primarily in the area of product line engineering and product

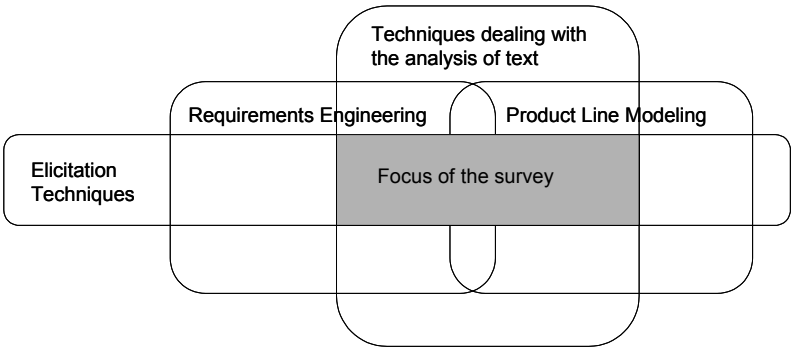


Figure 7 Focus of the related work

line modeling, approaches focusing on finding information for the requirements engineering phase is also integrated into the survey, as they contribute to the problem. Requirements engineering related approaches normally do not take variability into account but the can be helpful for extraction.

All in all, the focus of our survey looks as follows (c.f. Figure 7):
Techniques from requirements engineering and product line modeling (or domain analysis contribute to our survey if they deal with the topic of extraction and if they deal with the analysis of information that is available in textual form. So, the fields that we investigated are:

Product Line Engineering
As we want to elicit commonalities and variabilities for a product line it is important to know how variability and commonality is defined and handled during product line modeling.
Requirements Engineering
In Requirements Engineering, the Area of Requirements Elicitation is concerned with the collection of information relevant to a planned system from all relevant stakeholders. Documents can support the stakeholders there. In Requirements analysis there is work on restructuring and enhancing existing documents, which can also be applicable for legacy documentation. Work on requirements reuse
Artificial Intelligence
Knowledge Engineering, as it is strongly related to Requirements engineering is a field with useful techniques for eliciting knowledge from documents. Information Retrieval techniques can help in supporting the elicitation with (semi-) automatic techniques
Reengineering
The analysis of legacy information can be seen as a reengineering task. Although reengineering primarily deals with code there is also work on reengineering code comments or other documentations

4.1.1 Classification schema

In this survey we want to compare approaches, which can contribute to the problem of how to elicit textual information for product line models. As this is a new research problem, with a new focus, the approaches we consider come from different areas and are only comparable by looking at different dimensions of the approaches. Therefore a taxonomy is needed to make it possible to view the relation of the chosen approaches to the research question "how can I systematically elicit, and analyze textual legacy information for product line models?". In this section we will describe the taxonomy dimensions that were selected for the distinction of approaches

Taxonomy Motivation

According to the GQM Method, a study can be refined by setting goals specific to needs in terms of purpose, perspective and environment, by refining the goals into questions that are tractable and deduce metrics and data to be collected in order to answer the questions. The here-introduced taxonomy is the result of a GQM-Analysis,

which should support my research question and classify the related work. The Goal of the survey can be decomposed as follows:

Goal:

Object: (Process or product under study): All approaches that contribute to the extraction of common and variable information from existing textual software documentation that is relevant for product line modeling and requirements engineering"

Purpose (Motivation behind the measurement goal): characterization of the approaches and extraction of useful techniques

Focus (Quality attribute of the object under study): Usefulness of the approach/ techniques for product line engineering

Environment: (Context in which the analysis shall be performed) Applied Research in the field of knowledge recovery for product line modeling

So, the Goal can be summarized as follows:

"Analyze the approaches that contribute to the extraction of common and variable information from existing textual software documentation that is relevant for product line modeling and requirements engineering in order to characterize the approaches according to the usefulness of the approach and the techniques in the context of applied research in the field of knowledge recovery for product line modeling"

Questions

When applying GQM, Questions are defined which refine the Goal, are tractable and can lead to concrete metrics. These questions can be derived from the object, the environment and the focus. Several questions were identified and different taxonomy dimensions were derived from the questions as follows (the dimensions are described in Section 3.2 and Section 3.3:

Question1: Is the approach suitable for knowledge recovery for product line modeling?

Dimensions: Area, Legacy, Product Line Relation

Question2: Is the approach well-founded?

Dimensions: Metamodel, Tool-support

Question3: Does the approach contain useful techniques for analyzing textual requirements information?

Dimensions: Focus, Artefacts, Constructiveness

Primary Taxonomy Dimensions

From these questions the primary dimensions to classify approaches could be extracted: product line relation and constructiveness. As the goal of the survey is it to find information useful for product line modeling, the product line relation is the main classification dimension. Secondary, the constructiveness of the approach is also important for product line targets because purely selective and not constructive approaches cannot be used for the very constructive task of building a new domain or product line model in the context of technology transfer for industrial companies.

Product Line Relation

This taxonomy dimension describes whether the approach has a direct relation to product line engineering, which means that the approach addresses commonality and variability of products or common and variable information entities. Approaches with a strong product line relation are better suitable for the goal than those with a weak or no product line relation.

Values: not product line related; partially; commonality only; variability only; commonality and variability

Constructiveness

This taxonomy dimension describes if the approach is constructive, which means something is produced or modeled or if the approach only describes how to decompose existing information, elicit knowledge or analyze existing documentation. If the approach is described as a process, an indicator for constructiveness can be, whether there is an explicit output of the process. Approaches that are constructive are better for the goal than those that are less constructive because for the envisioned tasks, extraction of information from legacy systems in order to build a product line model, the construction of a new product line model has to be done.

Values: constructive (including producing; modeling); selecting; analyzing

Secondary taxonomy dimensions

The taxonomy dimensions described here are subordinate to the primary dimensions constructiveness and product line relation but are also useful for classifying existing approaches. These dimensions contribute to further classify existing approaches according to their usefulness for the overall goal of the survey but they are only secondary in a way that these dimensions do not give an estimate of the usefulness of the approach for the goal but give just a further classification of the approaches into subcategories. The secondary dimensions are area, metamodel, artefacts, operationalisation, focus, legacy, input and output

Area

This taxonomy dimension describes the research area (as described in Chapter 2) the approach is situated in

Possible Values (including the abbreviations for Table 1):

Requirements Engineering/RE (being the common super-area of the following three areas); Requirements Elicitation /RElicit; Requirements Analysis /RA; Requirements Reuse /RR; Domain Modeling /DM (including Product Line Modeling); Reverse Engineering / RevEng; Information Retrieval /IR; Knowledge Engineering /KE; Documentation Doc

Metamodel

This taxonomy dimension describes, the existence of an explicit underlying metamodel which describes basic elements of a documentation or how the to be constructed/analyzed models look like in general. With a metamodel it is in general possible to identify a match between the elements of a concrete documentation and the elements of a metamodel.

Possible values: yes; partially; no

Input Artefacts

This taxonomy dimension describes the artefacts that are analyzed or the basic documentation elements to be addressed by the approach. If the approach is described as a process, this is normally the input of the process. As the approaches analyzed are all related to software development, the artefacts addressed by the approach can be any textual artifact produced during the development lifecycle.

Possible values: Requirements; (User) Documentation; Code; Code Comments; Models, Other..

Output Artifacts

This taxonomy dimension describes the artefacts that are the result of applying the approach If the approach is described as a process, this is normally the output of the process. As the approaches analyzed are all related to software development, the artefacts addressed by the approach can be any textual artifact produced during the development lifecycle.

Possible values: Requirements; (User) Documentation; Code; Models, Other..

Representation

This taxonomy dimension describes the representation of the object or input artefacts of the approach. It specifies which objects will be analyzed and if the approach focuses solely on textual information or if other information sources (graphical models, expert interviews...) are considered also. Approaches, which do not analyze textual information, are out of focus of this survey.

Possible values: text only; text and others

Operationalisation

This taxonomy dimension describes if the approach is automatable/operationalizable and is described in a way that support by a tool seems to be possible

Possible values: Yes, No, Partially

Legacy

This taxonomy dimension describes the consideration of legacy information thus the integration of information from previously developed systems is integrated into a new development lifecycle or if the information analyzed will be integrated into the same lifecycle again. For Approaches that explicitly consider product line engineering it is also possible to address the instantiation of generic requirements built in domain engineering,

Possible values: legacy; same lifecycle, Domain Engineering -> Application Engineering

Level of abstraction

This taxonomy dimension describes the transition in level of abstraction the approach performs. The information to be analyzed can stay on the same level of abstraction, can get more abstract through the process or can get more concrete

Possible values: same, abstraction, concretization

4.1.2 Classification

In order to find related work we looked for approaches that can be found in the middle gray area described in Figure 7, the focus area of the survey. We identified the following approaches described in research papers and technical reports as related:

- [ACC+02] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo. Recovering Traceability Links between Code and Documentation
- [AG97] V.Ambriola, V. Gervasi. Processing Natural Language Requirements
- [AK02] I.Alexander and F. Kiedaisch. Toward Recyclable System Requirements
- [BK99] A. Becks, J. Köller. Automatically Structuring Textual Requirements Scenarios
- [BNT02] R.Biddle, J.Noble and E.Tempero. Supporting Reusable Use Cases
- [BR96] J. Burg and R. van de Riet. Analyzing informal requirements Specifications: A first Step towards conceptual modeling
- [CR00] J.Cybulsky and K. Reed. Requirements Classification and Reuse. Crossing Domain Boundaries

- [FFG+01] 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
- [FPF97] W.Frakes , Ruben Prieto-Diaz and C. Fox. DARE-COTS A Domain Analysis Support Tool
- [HvdH+99] J. Hoppenbrouwers, W. van den Heuvel, S. Hoppenbrouwers, H. Weigand and O. Troyer. The Grammalizer: A CASE Tool Based on Textual Analysis
- [LKM+02] T. Levendovszky, G. Karsai, A. Ledeczi, M. Maroti, A. Ledeczi and H. Charaf.: Model reuse with metamodel based-transformations
- [MB89] Y. Maarek and D. Berry. The Use of Lexical Affinities in Requirements Extraction
- [MK+99] M. Mannion, B. Keepence, H. Kaindl, J. Wheadon. Reusing Single System Requirements for Application Family Requirements
- [MKH+98] M.Mannion and B.Keepence and D.Harper. Using Viewpoints to define Domain Requirements
- [NRC+02] J. Natt och Dag, B. Regnell, P. Carlshamre, M. Andersson, J. Karlsson. A Feasibility Study of Automated Support for Similarity Analysis of Natural Language Requirements in Market-Driven Development
- [ODM96] Software Technology for Adaptable, Reliable Systems (STARS). Organization Domain Modeling (ODM) Guidebook
- [OLR01] S. Overmyer, B. Lavoie and O. Rambow. Conceptual Modeling through Linguistic Analysis Using LIDA
- [REG+00] P.Rayson, L. Emmet, R. Garside and P. Sawyer. The REVERE project: Experiments with the application of probabilistic NLP to systems engineering
- [RW99] M.Zopelari Roseti C.Werner. A Knowlede Acquisition Systematic within the domain Analysis Context
- [Sti00] E. Stierna Requirements Reuse in Support of the Aviation Mission Planning System Migration to the Joint Mission Planning System
- [TWJ97] B. Tschaitschian, C. Wenzel, and I. John. Tuning the quality of informal software requirements with KARAT
- [vKP+02] A. v. Knethen, B.Paech, F.Kiedaisch and F. Houdek. Systematic Requirements Recycling through Abstraction and Traceability.

When classifying the approaches according to our classification we get the results described in Table 1

	Product Line Relation	Construc tiveness	Area	Meta-model	Input Artefacts	Output Artefacts	Repre-sentation	Operatio-nalization	Legacy	Level of abs-traction
[ACC+02]	No	Select	RevE / IR	Par-tially	Doc+ Code	Links	Text+Co de	Yes	Same	Same
[AG97]	No	Analysis, select	RA	No	Req	Models	Text	Yes	Same	Abstr.
[AK02]	No	Select	RE, RR	No	Req	Models + text	Model + texts	No	Legacy	Same
[BK99]	No	Analysis	RE /RA	No	Doc	Req. maps	Text	Yes	Same lc	Abstr
[BNT02]	No	Constr	RR	Tax	Req + UC	Use Cases	Text	Yes	Same LC	Same
[BR96]	No	Analy-sis/select	RE	Par-tially	Req	Conc. models	Text	Yes	Same	Abstr.
[CR00]	Partially C	Analysis	RR	No	Req	Classified req.	Text	Yes	Legacy	Abstr.
[FFG+01]	No	Analysis	RE/ RA	Yes	Req	Req	Text	Yes	Same	Same
[FPF97]	C+v	Analysis	DA	No	Doc	Models	Text + code	Yes	??	Abstraction
[HvdH+99]	C	Constr	DA	Yes	Doc	models	Text	Yes	Legacy +same lc	Abstr.
[LKM+02]	No	Constr	Reuse	Yes	Models	Models	Text. models	??	Legacy	Abstraction
[MB89]	No	Analysis	RE	No	Req	objects	Text	Yes	Same lc	abstraction
[MK+99]	C+v	Select	RR, DA	Yes	Req.	Req.	Text	Yes	de->ae	Same
[MKH+98]	C+v	Analysis	DA/RR	No	Req	req	Text	No	Legacy	Same
[NRC+02]	No	Select	Rana	No	Req.	Req	Text	Yes	Same	Same
[ODM96]	C+v	Constr	DA	No	Systems	Domain models	Data	No	Legacy	Abstraction
[OLR01]	No	Constr	RE/ AI	No	Doc	models	Text	Yes	Legacy +same lc	Abstraction
[REG+00]	No	Analysis + select	RE/RR	No	Doc	Doc	Text	Yes	Legacy	Text -> text
[RW99]	C+v	Constr.	DA	No	Docs	Conc. models	Text+oth ers	No	Legacy	Abstr.
[Sti00]	C	Analysis + Constr	RR	No	Req		Text	Yes	Legacy	Same
[TWJ97]	No	Constr.	RE/ IR	Tax	Req	req	Text	Yes	Same lc	Same
[vKP+02]	No	Select	RR	Yes	Req	Req	Text	No	Legacy	Same

Table 1 Classification of related work

We analyzed the papers according to the classification dimensions. There is no approach that can create enough generality and is constructive enough to serve as an approach for the elicitation of product line knowledge. So, there is a need for an approach that is constructive and is applicable for product lines. At least there is a need to leverage the other approaches to be able to handle commonality and variability and to be constructive. In order to build a new approach that leverages the existing approaches we identified useful techniques from the existing approaches in Table1 . We will describe the techniques in section 4.2

4.2 Identification of techniques

We have identified techniques for eliciting knowledge that is relevant to a planned product line and can increase domain understanding. These techniques can be used for the extraction of commonalities and variabilities from legacy documents and to generate candidates for elements of a domain model, which can be proposed to the domain experts during domain analysis. For the extraction of commonalities and variabilities information retrieval techniques as described in [BR99] can be used and approaches concerning the elicitation of information from textual descriptions of single systems (e.g. [MB89], [TWJ97]) are currently being considered.

The following table shows examples for techniques and best practices that were identified. The techniques that were identified focus on their usefulness for textual information but are also applicable for other information (e.g. models). The techniques or practices are described with a short concise sentence. The application describes if possible the input and output of the technique and the goal that is achieved with the application of the technique (e.g. by analyzing only exemplars of systems the effort is reduced). Further information on the techniques can be found in the respective source document.

Technique/Practice	Application	Source
Find exemplars/exemplar system artifacts in the legacy documentation, in order to get a representative sample for the variability within the domain and between the products	Input: all documentation on legacy systems Output: descriptions of exemplars from the legacy documentation Goal: understandability, correctness, effort reduction	[ODM96]
Select a subset of known products or domains that is representative for the commonality and variability of all the products	Input. A domain with products in it Output: Some representative products and their description Goal: effort reduction	[ODM96]
For each system, record all available information sources like artifacts/workproducts, secondary data like books, practitioners and processes	Input: a domain Output: A list of all available and useful information sources Goal: completeness	[ODM96]
Treat legacy models separately as they contain direct taxonomic modeling information	Input: legacy system descriptions Output: models that can go directly into the domain model Goal: reuse, simplicity, understandability	[ODM96]
Translate or convert artifacts into common representation/notation	Input: legacy artifacts Output: normalized legacy artifacts Goal: comparability	[ODM96]
Free selection: browse an existing model and copy and paste a single requirements to the new model	Input: (application family) model Output: new model Goal: effort reduction	[MK99]
Identify matching requirements: Compare the requirements docs by select-	Input: requirements specs Output: similar requirements in the	[Sti00]

ing a base document and a match document and then iterating through each requirement in the base document, evaluation the degree of similarity of base requirement to match requirement	different specs Goal: finding similarity/variability	
Unmatched requirements where there is no similar requirement found in other documents can indicate functionality found only in one system or external interfaces	Input: requirements specs Output: candidates for variabilities or ext. interfaces Goal: finding variability	[Sti00]
Requirements from text can be analyzed by: - extracting requirements from initial text - decontextualize them by reformulation - recontextualize by explaining, classifying	Input: requirements specs Output: extracted requirements Goal: understandability without context	[TWJ99]
Prioritize requirements found in legacy documentation in order to determine their relevance for the domain model	Input: requirements specs Output: prioritized requirements Goal: focusing on the essentials	[TWJ99]
Integrate a view model and a requirements taxonomy in the analysis of legacy requirements and classify the requirements found according to the models	Input: requirements specs, view model and taxonomy Output: classified requirements Goal: increasing understandability and correctness	[TWJ99]
To assist analysis, enrich the analysis method by integrating automatic text analysis and information retrieval techniques. This will partially automate and actively support the analysis.	Input: - Output: - Goal: decreasing effort, increasing completeness and correctness	[TWJ99]
Reuse of requirements can be supported by - similarity /analogy analysis - case-based reasoning - generic modeling	Input: - Output: - Goal: supporting reuse of legacy requirements by state of the art analysis techniques	[AK02]
Use a conceptual system model in order to support identification and classification or legacy requirements	Input: requirements spec + conceptual model Output: classified requirements Goal: decreasing effort, increasing completeness and correctness	[vKP02]

The extraction approach was built, using these techniques. The techniques were used to specifically capture commonalities and variabilities of the different model elements needed for product line modeling described in the following section.

4.3 Identification of inputs

In the case of integrating requirements reusing and integrating knowledge about legacy systems, documents play an important role. The understanding of the domain as

far as it has been written down by the development organization in earlier projects can be found in documents. So those documents are a valuable source for domain understanding and domain analysis.

With documents or documentation assets we mean the representation of information concerning the legacy system. Typically during the lifecycle of a Software product many documents with different purposes and different target audience are produced.

Documents that are developed during system or software development are more or less structured. When developing a system, different kinds of documentation assets are produced and can be integrated into the product line as legacy documentation assets. Those assets can and should be integrated into the product line as legacy documentation assets. Those different documentation assets contain a different amount of and kind of information, which can be integrated into domain analysis:

- Requirements specifications
  A requirements specification can have many variations in size, structure and formality. It can range from a few pages of natural language requirements to formal models or from a list of customer requirements to a well-organized specification from several viewpoints. Requirements specifications should describe the system as a whole, should describe the features and parts of the system and should give functional and non-functional requirements concerning the systems in the domain. So requirements specifications are a good source of information for eliciting domain knowledge and increase domain understanding.
- Design Documents
  Design documents can as well as requirements specifications differ strongly in size and structure. If there are design documents they can range from a collection of boxes and lines to a detailed architecture description. Design Documents are also a good source of information but rather for the architecting phase than for the modeling phase of a product line because information is presented more from an architectural or design viewpoint. The information inside the documents is more from an architectural point of view than from a domain modeling perspective.
- Comments in Code
  Comments in code can also be seen as documentation assets and so as a source of information. They describe the implementation and concepts inside the implementation. The information inside the comments may be very useful when reengineering the code for a product line. Analyzing comments can allow direct conclusions from the not so structured comments to the highly structured code sequences. The perspective under which the comments are written though is more an implementation perspective than a domain modeling perspective.
- User Manuals
  User Manuals are often less structured and normally less formal than the other documentation assets. They provide an external or user view on the system and can also vary strongly in size. They can be expected to exist for almost any system, which has a certain amount of user interaction. User Manuals can therefore be a rich source for increasing domain understanding. They describe domain concepts from a user perspective and so can provide valuable input for domain analysis. Sometimes user manuals do not describe the system as it is but as it should be. But for use as a source for domain analysis this is rather an advantage than a disadvantage because the systems in the product line should be realized in a more ideal way

than the legacy systems. So information about the legacy system as it should be is a good input for product line modeling.

- Contracts, Correspondence and other material
  During the lifecycle of a software system other, mostly largely unstructured documentation assets like customer contracts, any kinds of correspondence concerning the system, notes etc. are produced which are largely unstructured. In many cases there is also other paper based or electronically available material describing a domain, like web sites, books, technical papers etc. that can also be a rich source of domain information. This material is most of the time not produced by the development organization itself and so might lack the product focus the documents produced within the development organization have.

All those documentation assets can serve as knowledge sources for the incremental introduction of product lines into the organization, which produced those legacy assets during engineering of several legacy systems. When introducing Product Lines with PuLSE [BFK+99] we encountered especially in SME's (c.f. [KMS+00]) that there are almost no requirements specifications or design document and if there are some, they are outdated. The only documentation assets that can be found often and are rather up to date are user manuals. In larger organizations, this situation is different. Due to defined processes, larger organizational units and more distribution of work, more documents can be found in larger organization. Companies following a defined development process have to develop certain documents to mark the end of certain process steps in their software development process. So there are many documents concerning legacy systems in the domain, which can serve as a valuable knowledge source.

As described above user manuals describe the legacy systems from a perspective, which is relevant for domain modeling. For document processing it is therefore reasonable to concentrate on the analysis of user manuals of all systems the development organization has built in the domain of the product line. According to [So01] there are different types of user documentation: a functional description, an installation document, an introductory manual, a reference manual and an administrator's manual. As we focus rather on the characteristics of the running system than on the installation, user documentation is for us at first hand the functional description, the introductory manual and the reference manual. Anyway, if installation of the system and the capabilities and features of installing the system should be investigated, the installation document or the administrator's guide can also be in focus

4.4 Identification of outputs

We expect the model elements of different modeling formalisms, which are concepts inside the application domain to be found as words, in statements or phrases, as sentences or as chapter headings in legacy documents. Not every phrase or chapter heading is a concept useful for domain analysis, so they have to be carefully selected and proposed to the domain experts for further and more concrete modeling.

It can be expected that not every kind of document is useful for every modeling formalism. We illustrate the expectations on the different modeling formalisms in the fol-

lowing table. In the columns the primitives of the modeling formalisms described in the last section can be found, In the rows you can find different document types-

	Foda Features	FAST Comm +Var	Variable Use-Cases	Variable OO-Models	Variable textual Requirements
Requirements Collections	X	X	X		X
Is-Analysis Documents		X		X	
Requirements Documents	X	X			X
Formal Requirements Specifications				X	
Glossary	X	X			X
Design Documents				X	
Rationale description					X
Contracts					X
Comments in Code				X	
User Manuals	X	X	X		X

Table 2 Classification of Outputs

4.5 Conceptual model

In this section, we shortly describe a conceptual model or metamodel that can be used for the extraction of common and variable requirements for a product line. This conceptual model was developed within the Empress Project [Empress] and is documented in an Empress deliverable [Emp03] and in some publications [JD03a] [JD03b] so we will just shortly describe the model here as far as it is needed to understand the extraction process described in section 5.

In general, a metamodel is "an information model for the information that can be expressed during modeling[Meta]. This conceptual extraction model consists of models describing the syntactic and semantic types of information found in user documents and requirements specifications.

The extraction model consists of four parts (see Figure 8):

- 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 product line 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, but do not need an explicit representation of variability.

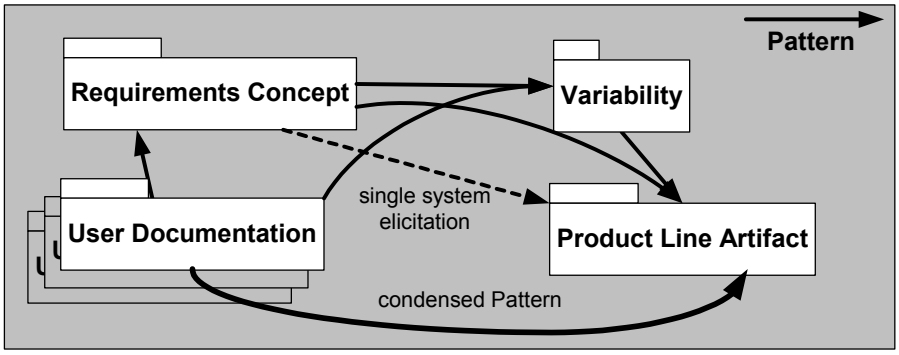


Figure 8 Overview of the model

The transition from one stage of the model to another stage is described by extraction pattern (specific rules-of-thumb or arguments derived from experience), see [JD03a] . It is also possible to directly transform requirements concepts into requirements artefacts without searching for variabilities (see arrow "condensed extraction patterns" in Figure 8).

4.5.1 User Documentation Model

Our user documentation model (see Figure 9) 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 attributes are not shown in the figure. This model describes the elements of a document on an adequate level for eliciting requirements concepts.

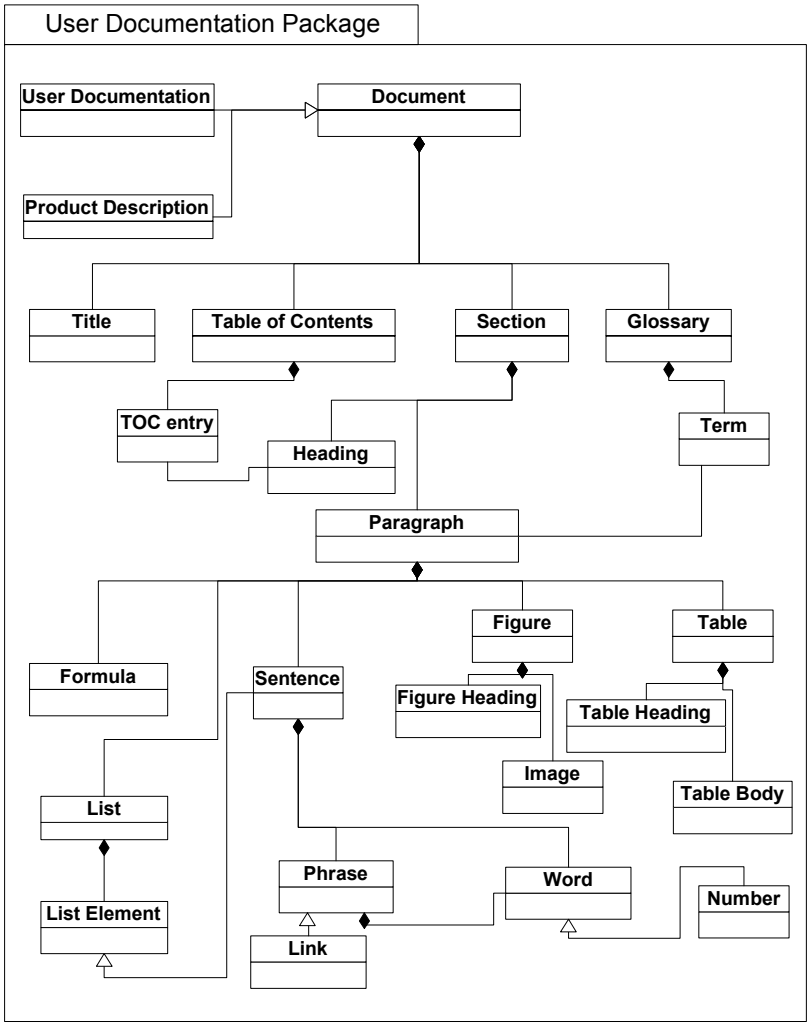


Figure 9 Model of User Documentation

4.5.2 Requirements Concept Model

The requirements concept model describes concepts that can be elicited from user documentation and that are normally realized or described by product line artifacts in requirements or product line 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.

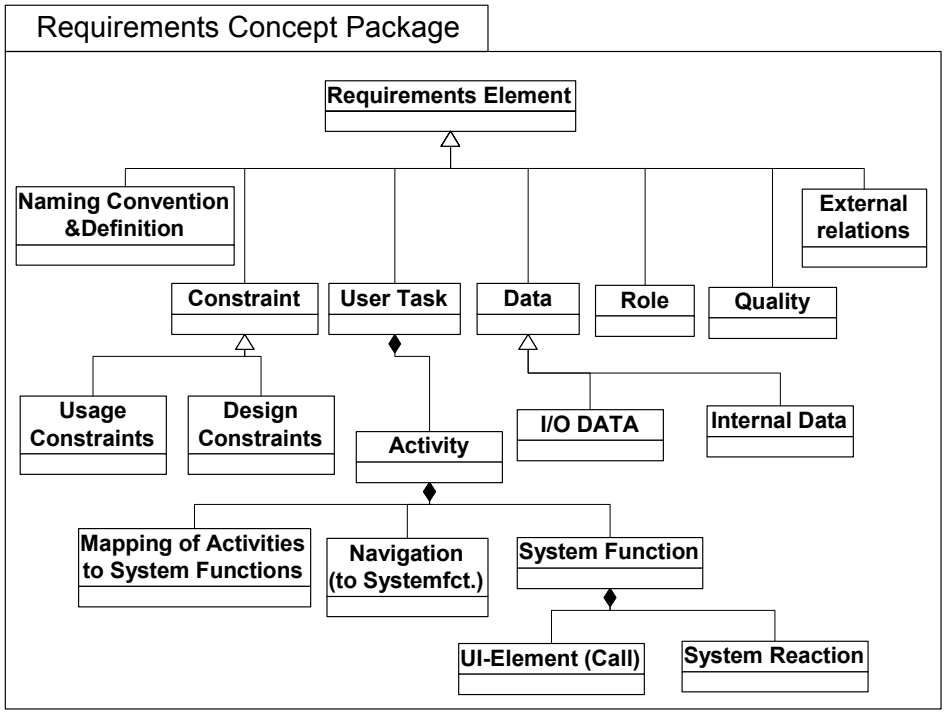


Figure 9 Requirements Concept model

4.5.3 Variability Model

A metamodel for modeling product lines has to support commonality and variability [SJ03]. In the variability model, the variation aspects are described. In order to find different variability elements, the requirements elements (from the requirements concept model) found in different user documentations are compared. The variability model is a product line specific model as it describes commonality and variability between different products. Variabilities can normally be found by comparing different documents. Figure 10 shows the elements of the variability model. The variability types supported here are more general than the ones described in section 3.4 . The metamodel comprises also alternative and otpionality but as it cannot be decided from scratch if elements that were found several times in different documents refer to a multiple selection, a single selection or to a value reference, these three variability types from the variability management approach are intograted into the one element "range". The descicions in the variability model package will later on be mapped to decisions in the decision model as described in 3.2.

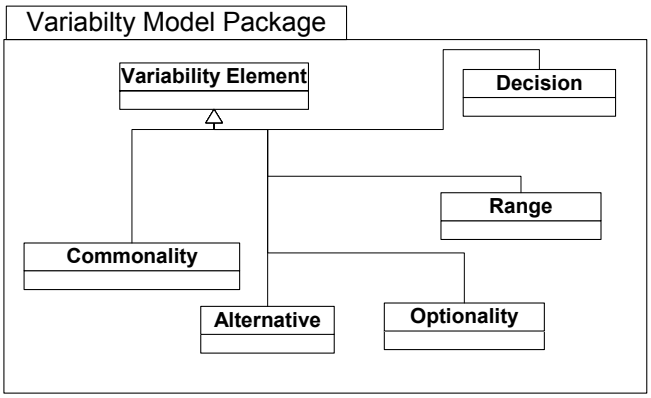


Figure 10 Variability Model

4.5.4 Product Line Artifact Model

The fourth package of our conceptual extraction model is the product line artifact model (see Figure 11). 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 product line 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 [Mu02] (see Figure 12 General structure of the meta model for product line information (from [Mut02])). The model described by Dirk Muthig is a model to describe generic product line assets. Product Line Assets can be product line artefacts, describing the product line itself and decision models describing the constraints on the product line artefacts. Figure 12 shows the general structure of the metamodel for product line information. The grey boxes indicate the two subpackages that were defined to improve the structure of this model. The first package, ProductLineArtifact contains the definition of product line artefacts. The second package DecisionModel depends on the first package and defines the concepts of a decision model with respect to the definition of product line artefacts.

In product line engineering, variability has to be made explicit in the product line artifacts. Different extension (e.g. to UML-Use Case diagrams or to textual Use Cases) 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.

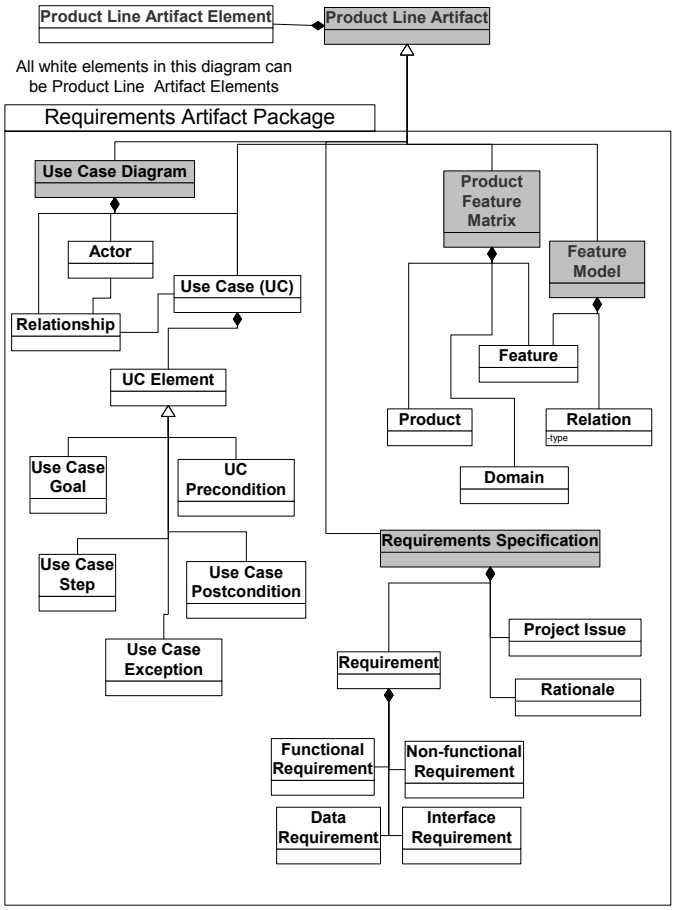


Figure 11 Requirements Artifact Model

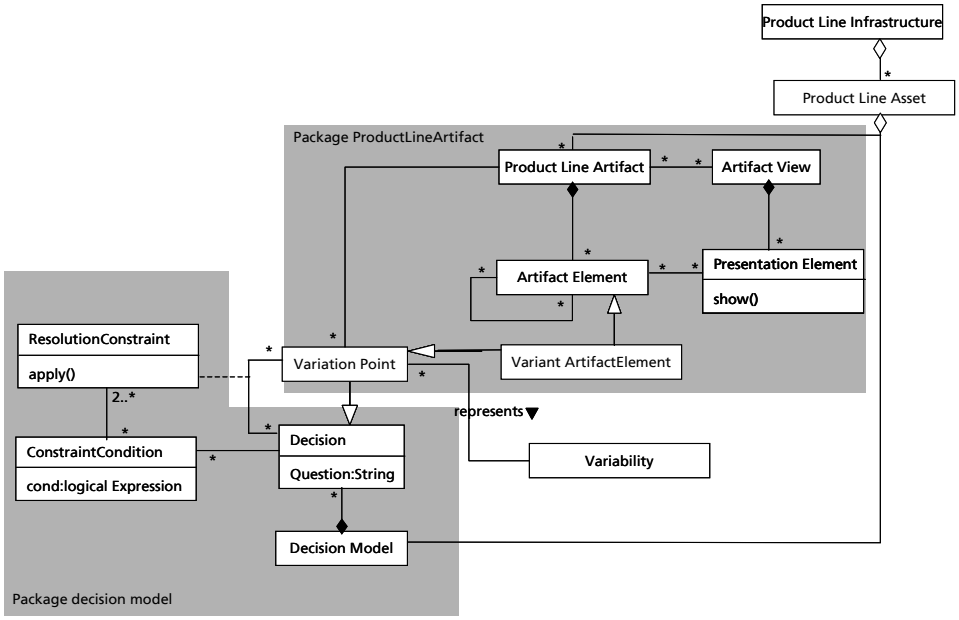


Figure 12 General structure of the meta model for product line information (from [Mut02])

As variability is encapsulated in the product line artifact and the product line artifact element (see Figure 12), the model can also be used for specifying single systems requirements. At the moment, we have specified different kinds of requirements notations: Use Cases, textual requirements specifications and the product line specific notations: product feature matrix [Sch03] and feature model [KCH+90]. Further requirements artefacts will be integrated into the product line artifact model. We added different representations here, as our general approach to product line modeling 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. 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 [Co01]) 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 [IEEE98]. A requirements specification is a textual document consisting of functional, non-functional and data requirements including project issues and rationales for the different requirements.

4.5.5 Using the Conceptual Model

We have defined extraction patterns for transitioning requirements concepts into product line artifacts (e.g., "a role is described as actor in a Use Case diagram") and extraction patterns that additionally include variability (c.f. Figure 8). An example of such an extraction pattern 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 extraction patterns. For users of the approach and the conceptual model those extraction patterns can be integrated

to condensed extraction patterns describing the transition from user documentation directly to product line artifacts (c.f. arrow "condensed extraction patterns" from user documentation to product line artifact in Figure 8). Between all four parts of the model, extraction patterns can be defined to describe how elements are typically converted from one part of the model to another. A heading from the user documentation model can be a user task in the requirements concept model and can then be physically represented as a feature. The transition rules for transitions between the different models will be further elaborated and published in the future.

In this section we described the general concepts of product line engineering and especially requirements engineering for product lines. The conceptual model forms the basis for describing requirements on a product line with Use Cases with commonalities and variabilities. With the help of transformations based on the conceptual model, elements from user documentation can be integrated into product line models, describing requirements on a product line.

5 Extraction Pattern

As described in Section 3, requirements elements can be found in document elements of different forms. To allow a structured extraction, we developed a number of extraction pattern for different document elements and different requirements elements that describe a transition between document elements and the other three parts of the conceptual model. In this section, we describe the template for the extraction patterns show some example pattern in this template form and give a short description of all pattern.

5.1 Extraction Pattern Template

The extraction pattern template is used to store the pattern and describes the applicability of each pattern. The template consists of the following elements:

Name
The name of the pattern and a unique number.

Short Description
A short description of what the pattern should elicit.

Input
The element that is converted into another element with the help of the extraction pattern. Input elements can be documentation model elements or requirements concept model elements (c.f arrows in Figure 8).

Output
The type of the element that is the result of the extraction pattern. Output elements can be requirements concept model elements, variability model elements or product line artefact model elements (c.f arrows in Figure 8).

Recall
The recall of a pattern describes its completeness. In Information Retrieval, the recall of a pattern is defined as the number of correct elements found by the extraction pattern divided by the number of correct output elements that can be found [BR99]. In the template the recall is given by a rough estimate of the correctness of the pattern. This estimate was determined experimentally (the determination is currently ongoing, so not for all pattern recall and precision exist) and corrected by the authors. Possible values are “--, - , o, +, ++” .

Precision
The precision of a pattern describes its correctness. In Information Retrieval, the precision of a pattern is defined as the number of correct elements found by the extraction pattern divided by the number of output elements found by the extraction pattern [BR99]. In the template the precision is given by a rough estimate of the correctness of the pattern. This estimate was determined experimentally (the determination is cur-

rently ongoing, so not for all pattern recall and precision exist) and corrected by the authors. Possible values are "--, - , o, +, ++".

Value/Relevance
The value/relevance of a pattern describes how valuable the results are for an expert. The results of a pattern are very valuable for an expert if the element extracted by the pattern is a correct element and is not apparent for everybody. So, pattern discovering unknown product line model elements that were not known to the domain expert before have a higher value than pattern that find elements that are apparent. The value is at the moment an estimate given by the authors based on literature (e.g. [Ka92], [ODM96]), as the pattern evolve and more case studies are available the values will hopefully get more exact. Possible values are "--, - , o, +, ++".

Transition
In the field transition the associated model of the input and output are given. Possible transitions are e.g. "Documentation -> Requirements Concept", "Documentation -> Variability" or "Requirements Concept -> Product Line".

Long Description
In the field long description a longer description of the pattern including background information or rationale can be given.

Example
In this field an example for elements elicited with the pattern can be given.

Related Pattern
This field gives numbers of pattern that are somehow related to the described pattern (e.g. that generate the same or similar output).

Table 3 shows an example for a filled pattern. An empty template and some other example pattern in the template are given in the appendix.

f	1 Heading-Feature
Short Description	Headings of sections or subsections typically contain features
Input	Heading
Output	Feature
Recall	+
Precision	++
Value/ Relevance	-
Transition	Documentation -> Product Line Artifact
Long Description	As features describe functionalities that are of importance for the user, they are found at prominent places in the UD.
Example	"Send SMS" as a heading of a mobile phone manual is a Feature of the mobile phone
Related Pattern	

Table 3 An example pattern template

5.2 List of Pattern

In this section we give a list of pattern. The long description/template of some pattern can be found in the appendix. The pattern are grouped by their output type (possible elements from the requirements concept model, the variability model or the product line artifact model). This pattern list is not complete; the list of relevant pattern is extended by and by:

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.

Use Cases

- 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
- Use case preconditions and goals can typically be found in the beginning of a chapter
- Use case preconditions can be found before or within the description of a Use Case
- 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

Functional Requirements: User Interface and E/A Information and system functions

- Phrases like "press", "hold", "hold down" , "press briefly", "select" , "key in" "scroll" etc. mark a dialogue with the user interface or navigation elements
- The following phrases give hints for E/A elements : "type in" " enter" , "transfer" …

Activities or system functions are all those elements marked as features that contain a verb.

Non functional requirements

- 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 Use Case document can be hint for a non-functional requirement (why was exactly this number chosen?).

Project issues and usage constraints

- Project issues can be found in the beginning of a chapter. Project issues are related to usage constraints

Text passages that do not fit into the textual flow, that describe facts that do not fit to the rest of the description or that use words from another domain or from another subdomain of the system can be hints for project issues.

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 Use Cases that are wholly optional.
- 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 Use Cases
- 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
- Menu items that are described only in some of the documents can be hints for optional or alternative functionality (Use Cases or parts of Use Cases).

6 The Extraction Process

Product line engineering includes the construction of a reusable asset set. 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.

Usually before starting to develop a product line, one or more systems in the domain were already built. These existing systems can be still in use or in evolution or can be old legacy systems. These systems can be distinguished by their functionality, their features, by the markets they address or by non-functional characteristics of the systems. The information from those systems is a valuable source for building the reusable assets. This information from existing systems can be found in the code, in architecture descriptions or in requirements specifications [Jo01]. All this information can be found in documents produced during the lifecycle of the existing system.

There are several disadvantages in the current product line modeling phase:

- Until now, the information needed to build a product line model is elicited interactively with high expert involvement. As domain experts have a high workload and are often unavailable this high expert involvement is a risk for the successful introduction of a product line engineering approach in an organization.
- There is a lack of guidance on how to integrate legacy information found in documents into product line models.
- Single system elicitation methods cannot be taken as is because multiple documentations have to be compared, commonalities and variabilities have to be elicited and additional concepts (e.g. abstractions, decisions) are needed.

6.1 Benefits

Systematically integrating documentation into product line models supports:

Integration and reuse of textual information
By integrating existing systems into product lines, commonalities and variabilities, the essence of product line engineering can be elicited from documents. By integrating textual information into a product line, not only code can be reused but all assets build during the previous lifecycles.

Feasibility of product line modeling
The feasibility of product line approaches and of product line modeling will be supported through these document-based techniques. A document-based technique can decrease the effort the domain experts have to spend with interviews and meetings and lead to a high expert load reduction. The basic information can be semi-

automatically elicited from documents and the experts can concentrate on planned innovative functionality.

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 old information, which was produced within the organization.

Better traceability from the product line to the existing systems
Traceability to the existing system can be established only with a systematic approach which sup-ports linking of legacy assets to the product line model built during domain analysis.

The process of analyzing a requirements document using information retrieval ideas 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.

There are different kinds of legacy documentation (e.g., requirements specs, user manuals, design documents). The documentation most useful to gather information for the requirements towards the product line is user documentation. This is due to the fact that user documentation describes the system from a users point of view and concentrates on the behavior of the system. When building a new product line, the approach to do so can either be **independent** (A company starts a product line without any predecessor products), **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) or **leveraged** (the company sets up a product line based on a product line that is already in place) [SM02]. Useful user documentation as an input for product line modeling can be found in the cases of project-integrating, reengineering-driven and leveraged product line engineering.

6.2 Outline of the Extraction Process

For controlled extraction we propose an approach here that 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 is called the PuLSE™[1]- CaVE-Approach (Commonality and Variability Extraction) that is integrated into the PuLSE-Framework for product line engineering [BFK+99], which is a customizable and flexible framework for all phases of product line engineering.

[1] PuLSE is a registered trademark of Fraunhofer IESE

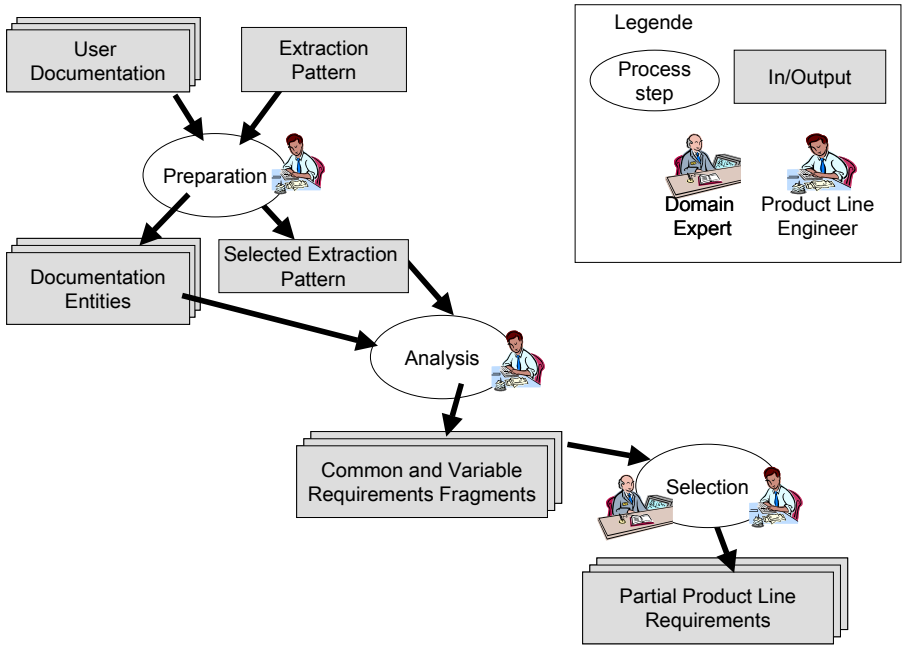


Figure 13 An outline of the extraction approach

CaVE is an approach enhanced with techniques for structured and controlled integration of user documentation of existing systems into the product line. With CaVE, common and variable features, Use Case elements, decisions and requirements can be elicited. We restrict our description here to the extraction of Use Case elements. As existing systems are the basis for this approach it can be seen as a reengineering approach for transferring user documentation into basic elements for product line Use Cases. The approach consists of the following phases (see Figure 13):

- Preparation
  The product line engineer prepares the user documentation and selects the appropriate extraction pattern
- Analysis
  The product line engineer analyses the documents with the selected extraction pattern and marks the elements found
- Selection, and change.
  The selected elements are put together to partial product line artifacts and presented to the expert who can change elements and add additional information

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. We will now describe the three steps in more detail.

6.3 Preparation

Preparation consists of the five sub steps **collection, selection, division, browsing** and **pattern selection**. During the collection step, the product line analyst collects all user documentations for the systems that should be integrated into the product line and of those systems that are related in order to have all information needed 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 attention and leads to lower performance [Wi84] the number of documents to read in parallel should be reduced to a minimum. So, if there are more than 3 systems, in the selecting phase, the product line analyst selects three documents that cover the variety of systems to compare (e.g. one documentation of a low-end system, one documentation of a high end system and one typical system) for a first search in the documents. The other documents can be used for validation. After selecting the three typical documentations, the product line analyst divides them into manageable and comparable parts. Experience showed that 3 to 10 pages (e.g. comparable subchapters) are a suitable size for the parts to compare. In the browsing step, for each of those manageable parts (or for a subset of those parts that includes typical subdomains) the product line analyst browses through them in order to decide the amount of variability in them. There are two alternatives:

- If the documents differ in more than one third of the text, the product line analyst shall process them one after another in the second step and choose the biggest document as the document to start with the analysis.
- If the documents differ in less one third of the text, the product line analyst shall compare the documents in parallel in the further steps.

The value of one third is a value we experienced to be suitable in the case studies.

6.4 Search

In the search step, the elements to be identified in the documents are marked and tagged.

With the help of the extraction pattern described in section 5, which are not complete but help in finding a relevant part of the features, Use Case elements, other requirements and variabilities, the user documents should be marked (e.g. with different colors for different Use Case elements and for variabilities).

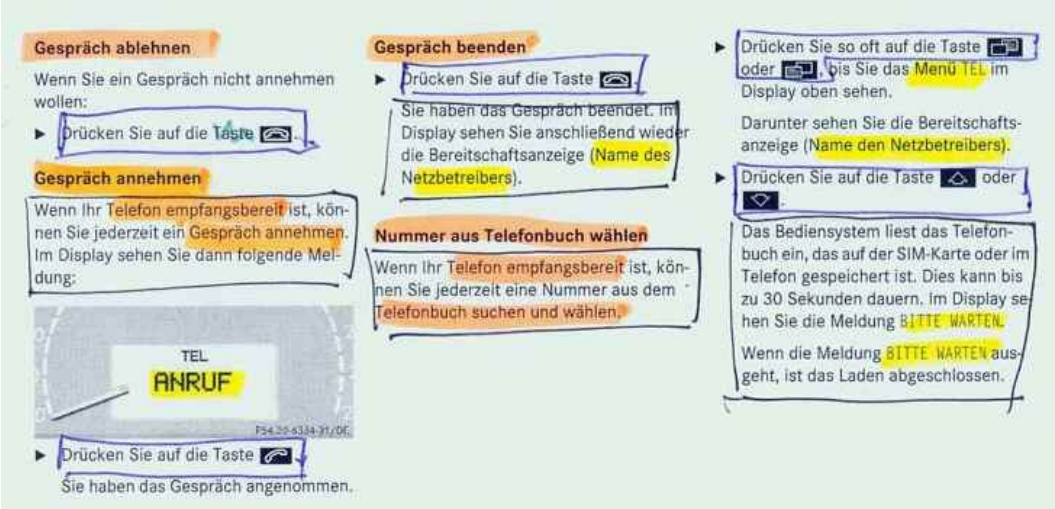


Figure 14 Example of marked user documentation

Figure 14 shows how the elements can be marked. The solid marked elements are potential features of the system, the boxes are Use Case elements (in this example Use Case steps and Use Case preconditions). There are two different ways to browse through the documents and mark the elements:

- pattern by pattern
  When manually following the approach for the first time, it might be useful to concentrate on one pattern or a group of pattern (e.g. all pattern eliciting features or all pattern having similar input elements) and browse through the document with those pattern in mind and mark all elements that can be extracted with this one or this group of pattern. Browsing through the documents is a bit time consuming but doing it this way makes sure that all elements are marked.
- integrated
  When following the process the second time or if there are not so many elements to be marked it is also possible to browse through the documents with all pattern at hand. This shortens the time to search for elements significantly but one must be sure to have all applicable pattern in mind.

Both ways have their advantages and disadvantages (completeness, time). So both ways of marking elements are possible.

The elements, which should be sized from one word to at most 5-6 lines, that were identified to be useful can be marked only in the document and presented to the expert but can also be extracted from the document and tagged with attributes containing the information needed for modeling the product line. Table 4 shows the attributes that can be captured.

Attribute	Values	Description
ID	e.g. 1…n or doc.number	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
Element Type	Product Line Artifacts	The elements of the schemata the text matches to
Var Type	Comm., opt, alt, para	The hypothesis for the variability type of the element (default is commonality)
Parent	ID	The element this element is part of
Relations	List of IDs	The IDs of other elements from the same or other documents that are possibly part of the same Use Case
Var relations	List of IDs	The IDs of other elements that contain alternatives or other parameters for this element

Table 4 Attributes for the elicited text items

With the help of this table, variability between documents can be condensed in the table and better compared.

6.5 Selection

In the last step selection, the extracted and tagged elements have to be validated and changed by a domain expert. E.g., not all text excerpts marked as possible Use Case elements really are elements of a Use Case, not all elements marked as optional in the user documentation will really be optional in the new product line. So a domain expert has to judge whether the extracted elements should be used for modeling the product line Use Cases, features, etc. or not. In this step the expert can change the "Use Case type" and "var type" in the tagged notation, extend or change the text of the element. The relations 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 documentations.

With the help of the extracted document elements and the tables that contain the condensed information and the variability between documents, product line artifacts like product line Use Cases including variability (as e.g described in [JM02] or in [HP03]) can be built. As the actual modeling is not part of the approach but only the extraction of candidates for models, we will not go into details for modeling here.

We successfully performed several case studies with user documentations from information systems and embedded systems with this approach. Two of the case studies are described in the subsequent sections.

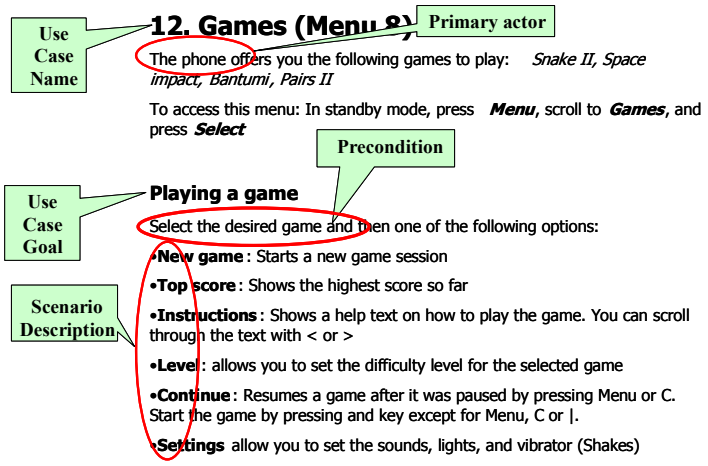


Figure 15 Correspondences between user manual parts and Use Case elements

6.6 Example Application: Modeling Use Cases

As an example, the approach described in Section 6 is applied on an industrial example to show its applicability: The approach is applied on the user manuals of mobile phones and the resulting information is modeled in the PLUC Notation [BFG+02], a notation that uses Use Cases that are extended with a mechanism to express variabilities The Case described here is the result of a cooperation of the authors and the Italian partners of the Café Project. The cooperation and its further results are described in [FGJ+03]. A similar but more systematic case study using a different notation is described in this report in section 7

We compare here existing user manuals of two mobile phones from the same product line that, from here in after, we indicate as P1 and P2. When the extraction process is applied to a single product, the effort can be concentrated on the identification of the correspondences between the parts of the manual and the Use Case elements. Starting from a fragment of a user manual of a mobile phone, represented in Figure 16, a possible correspondence with standard Use Case elements can be shown. When product families requirements have to be derived, starting from several documents, each related to a particular product of the product line, the process shown before is not in general sufficient. In this case all the commonalities and variabilities should be derived as the outcome of the extraction phase.

With reference to the approach described in the first part of section 6, after the collection, selection and division phases the function descriptions of the user manuals have been identified as being the basic documentation entities for the extraction process. In this case study the description of the GAMES functionality of the two phones P1 and P2 has been taken as example.

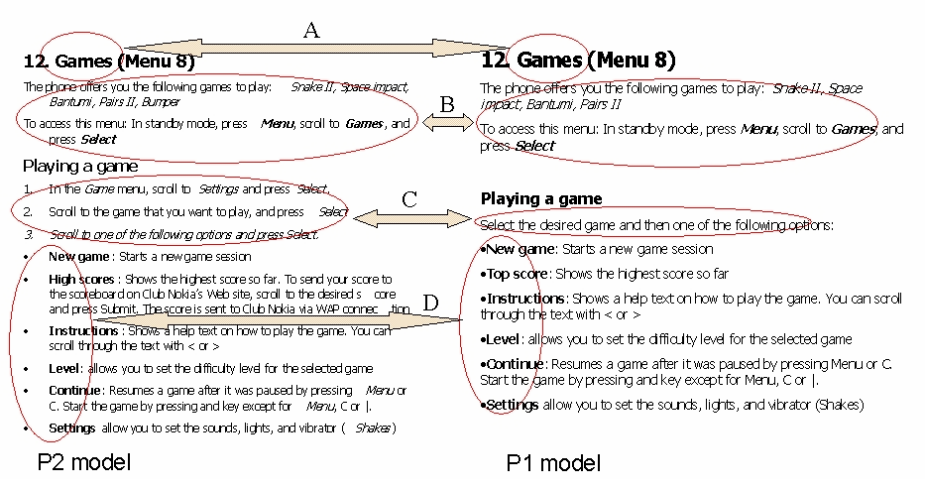


Figure 16 Correspondences between the two products

The browsing phase determines the amount of commonalities and variabilities between the two entities under analysis. It is then evident that the two products differ at least for the set of games provided to the user and for the presence of the WAP connection. Figure 16 shows how the parts of the user manuals of the mobile phones P1 and P2 related to the GAMES functionality can be put in relation after the preparation phase.

In the search phase the identified parts are analyzed and the Use Case elements are searched. The found commonalities and variabilities are extracted and tagged and then integrated into an intermediate document represented in Table 5 in a tabular format. Where each row of this table represents an elicited text item according to the

ID	Value	Doc.	Use Case Type	Parent	Var Type	Use Case relations	Var Relations
P1.1	The phone	P1	Actor	-	Comm.	UC Games	P2.1
P1.2	Games to play: SnakeII, Space impact, Bantumi, PairsII,	P1	Scenario Description	-	Comm	UC Games	P2.2
P1.4	Bumper	P1	Scenario Description	P1.2	Opt	UC Games	
P1.5	To access this menu: press menu, scroll to games, select	P1	Precondition	-	Comm	UC Games	P2.3
P1.6	In the game menu, scroll to settings and press select	P2	Scenario Description	-	Opt	UC Games	-
P2.1	The phone	P2	Actor		Comm.	UC Games	P1.1
P2.2	Games to play: SnakeII, Space impact, Bantumi, PairsII,	P2	Scenario Description	-	Comm	UC Games	P1.2
P2.3	To access this menu: press menu, scroll to games, select		Precondition	-	Comm	UC Games	1.3

Table 5 Text items found in the user manuals

scheme of Table 5

The outcome of this phase is an intermediate document, which is obtained from both the previous ones by tagging the common parts and the detected variabilities. The final outcome of the selection phase could be a first version of a Use Case expressed in the PLUC formalism.

Realistically, we do not have to expect that the variable parts to be tagged could be identified simply by a "difference" operation between the two docs. For example, the text elements put in correspondence by the double arrow in Figure 16 have the same meaning, but, in the case of the P1 model they seem suitable for being put in the Precondition element of the resulting Use Case, while for the P2 model they seem more suitable for being considered as belonging to the Scenario.

The resulting Use Case will depend on the decision on how to mark these text elements. Such a decision cannot be made in general but should be left to a domain expert in the selection phase of the approach. This can be made either as a case-by-case decision for each element or as a general parameter to select before applying the process. In Figure 17 a possible outcome in the PLUC format of the product line requirements related to the GAME functionality is shown.

Primary Actor: the user, the {[***V0***]} mobile phone (the system)
Goal: play a game on a {[***V0***]} mobile phone and record score
Preconditions: the function GAMES has been selected from the main MENU
Main Success Scenario:
- The system displays the list of the available games: SnakeII, Space impact, Bantumi, PairsII and {[**V1**] additional}
- The user select a game
- The system displays the logo of the selected game
- The user selects the difficulty level by following the {[***V2***] appropriate} procedure and press YES
- The system starts the game and plays it until it goes over
- The user records the score achieved and {[***V3***] possibly} send the score to Game Club via WAP
- The system displays the list of the {[***V1***] available} games
- The user presses NO

V0: alternative	1. P1 model 2. P2 model	
V1: optional	if *V0*=2 then Bumper	
V2: parametric	if ***V0***=1 then procedure-A:	- press Select - scroll to Settings and press YES - scroll to Difficulty Level and press YES - select the desired difficulty level, press YES
	else if ***V0***=2 then procedure-B:	- press Select - scroll to Level and press YES - select the desired difficulty level, press YES
V3: parametric	if ***V0***=1 then function not available else if ***V0***=2 then function available	

Figure 17 A use case derived from user manuals

7 "Mobile Phone" Case Study

In this section we describe the first case study we performed. The case study was to get experience with the approach and to come to a first set of applicable extraction pattern. We applied the approach described in section 6 to different user manuals in the area of software for a mobile phone. Our hypothesis when applying the approach were:

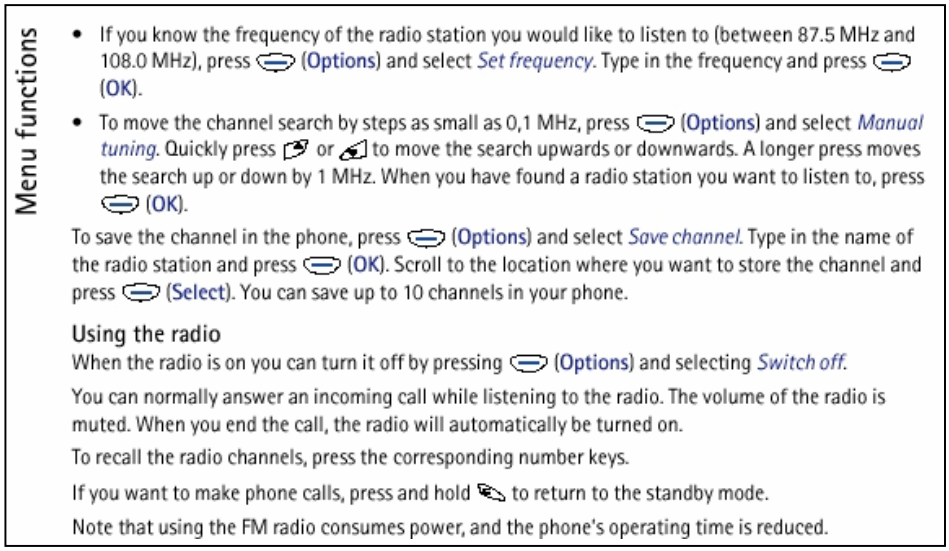
- We can identify product line artifact elements and extraction patterns according to our models
- Features and Use Cases can be extracted from documentation with the approach
- Variability can be found with the approach.

7.1 Description of the Case Study Material

The case study was performed with user manuals describing mobile phones X1, X2, X3 of a major player in the mobile phone industry. We concentrated on the radio sub-section of the user manuals. The following user documentations were analyzed during the case study:

- Phone feature lists of models X1, X2 and X3.
- User Guides of models X1, X2 and X3, section "radio".

The analyzed material ("radio" section) was about 3 English Din A4 pages for each phone. The Case Study input material is available on the vendors website. To give an impression, how the document looks like, Figure 18 shows an excerpt of the analyzed "radio" section.



Menu functions

- If you know the frequency of the radio station you would like to listen to (between 87.5 MHz and 108.0 MHz), press (Options) and select *Set frequency*. Type in the frequency and press (OK).
- To move the channel search by steps as small as 0,1 MHz, press (Options) and select *Manual tuning*. Quickly press or to move the search upwards or downwards. A longer press moves the search up or down by 1 MHz. When you have found a radio station you want to listen to, press (OK).

To save the channel in the phone, press (Options) and select *Save channel*. Type in the name of the radio station and press (OK). Scroll to the location where you want to store the channel and press (Select). You can save up to 10 channels in your phone.

Using the radio

When the radio is on you can turn it off by pressing (Options) and selecting *Switch off*.

You can normally answer an incoming call while listening to the radio. The volume of the radio is muted. When you end the call, the radio will automatically be turned on.

To recall the radio channels, press the corresponding number keys.

If you want to make phone calls, press and hold to return to the standby mode.

Note that using the FM radio consumes power, and the phone's operating time is reduced.

Figure 18 Example page of a mobile phone user manual

7.2 Conducting the Case Study

In a first step, we used the vendors user documents to validate our models described in Chapter 3: we identified and marked product line artifact elements according to the product line artifact model. In Section 6.4, there is an example screenshot of a marked user document from another case study. The marking of product line artifacts was done in a similar way for the mobile phone case study. In a second step, we found extraction patterns on how to map textual elements to product line artifacts like the extraction patterns described in Section 5.2. Based on this information, we built Use Cases from the elements (see Section 7.3) and compared them to real product line models from the phone vendor to see how complete the extracted information is.

To give a rough impression on how many elements of the various models were found in the used user documentation, the following table states the appearances for model X2. This information should not be treated as a complete and correct counting of elements. Rather, it shall give you an impression on the dimensions and distribution.

Number of appearances roughly (reference: X2, Radio-part):

Conceptual Model	Variability Concept Model	Product Line Artifact Model
Elements	Commonality: 10	Feature: 11
	Optionality: 2	Use Case: 2
	Alternative: 3	UC-Exception: 3
	Range: 2	UC-Description: 4
		UC-Precondition: 1
		UC-Postcondition: 0
		UC-Element (not further refined): 0
		Functional Req.: 15
		Data Req.: 4
		NFR: 6
		Project Issue: 3
		Interface Req.:16

It was interesting to see that we found several variabilities (2 optionalities, 3 alternatives and 2 ranges), even though only a small part of the user documentation was analysed. Furthermore, the user documentation in the mobile phone domain gives a lot of information about the interfaces. Please note that the number of features and functional requirements is not disjunctive, i.e., an element that was characterized to be a feature might also been characterized as functional requirement.

7.3 Validating the Case Study

From the product line artifact elements identified in the user documentation of the case study, three Use Cases were created. In the following, the three Use Cases are listed. All bold parts are parts of X1, all italic parts are parts of X3 and X2. Please note that in most cases, the user documentation stated pure user inputs and no system reactions. Therefore, only the inputs of the Use Cases listed in this section were directly elicited from the user documentation. The system reactions were implicit or easy to guess by common sense. Normally, this would lead to more load on the domain expert, as he has to check thoroughly if the system reactions are correct.

Use Case 1

Name: Auto Tuning X3, X2, X1

CHARACTERISTIC INFORMATION
Goal in Context: Tuning a radio channel
Scope: Mobile Phone
Level: detailed
Preconditions: **The radio is on**
Success End Condition:
Failed End Condition:
Primary Actor: Mobile User
Trigger:

MAIN SUCCESS SCENARIO (X1 and X3 are almost equal)
1.Press and hold the up or down button to start the channel search
2. Searching stops when a channel is found
3. To save the channel in the phone, press Options and select "Save channel".
4. Type in the name of the radio station and press OK.
5. Scroll to the location where you want to store the channel and press "Select".
6. You can save up to <<numchannels, 10, **20>> channels in your phone.**

EXTENSIONS
1a –6a You can normally answer an incoming call while listening to the radio.
The volume of the radio is muted. When you end the call, the radio will automatically be turned on.

SUB-VARIATIONS
3b To quickly save the channel in a location <<1 to 9 **???** >> press and hold the corresponding number key, then key in the name of the channel and press ok.

Figure 19 Mobile Phone Use Case 1

Use Case 2a
Name: Manual Tuning X3, X2, X1 (Frequency known)

CHARACTERISTIC INFORMATION
Goal in Context: Manual Tuning of a radio channel
Scope: Mobile Phones
Level: detailed
Preconditions: The radio is on. The Frequency is known.
Success End Condition:
Failed End Condition:
Primary Actor: Mobile User
Trigger:

MAIN SUCCESS SCENARIO
1.) Press Options
2.) The system shows a menu.
3.) Select SET FREQUENCY from the menu.
4.) The system shows a field to enter numbers
5.) Type in the frequency
6.) The system shows the frequency in the field
7.) Press Ok.

EXTENSIONS
1a –7a You can normally answer an incoming call while listening to the radio. The volume of the radio is muted. When you end the call, the radio will automatically be turned on.
1c-7c If you want to make phone calls, press and hold C to return to the standby mode.
1d-7d Applications using GPRS and HSCSD can interfere with the radio.

SUB-VARIATIONS
1b To quickly select set frequency press *+ when in the radio menu.
1b To quickly select manual tuning press # when in the radio menu.

NFRs /Issues
You can save up to <<numchannels, 10, 20**>> channels in your phone.**
Using the FM radio consumes power, an the phone's operating time is reduced.
The frequency range is limited from 87,5 MHz to 108 Hz.

Figure 20 Mobile Phone Use Case 2a

Use Case 2b
Name: Manual Tuning X3, X2, X1 (Frequency not known)

CHARACTERISTIC INFORMATION
Goal in Context: Manual Tuning of a radio channel
Scope: Mobile Phones
Level: detailed
Preconditions: The radio is on. The Frequency is not known.
Success End Condition:
Failed End Condition:
Primary Actor: Mobile User
Trigger:

MAIN SUCCESS SCENARIO
1.) Press Options
2.) The system shows a menu.
3.) Select Manual Tuning.
4.) The system shows a frequency.
User Interface (fast search yes/no)

<<ALT 1>>
5.) To move up or down in steps of <<stepsize, 0,1 / 1>> Mhz, <<duration, quickly / longly>> press the Up or Down key.
6.) The system shows the new frequency.

<<ALT 2>>
5.) To move up or down in steps of 0,1 Mhz, press the Up or Down key briefly. For a fast channelsearch, hold down the Up or Down key.
6.) The system shows the new frequency/ running frequency. The system stops the frequency if the Up or Down key is released.

7.) When the frequency is the frequency of the channel you would like to listen to, press ok.
8.) The system keeps the channel and switches to "normal mode".
9.) To save the channel in the phone, press Options and select "Save channel".
10.) Type in the name of the radio station and press OK.
11.) Scroll to the location where you want to store the channel and press "Select".

EXTENSIONS
1a –11a You can normally answer an incoming call while listening to the radio. The volume of the radio is muted. When you end the call, the radio will automatically be turned on.
1c-11c If you want to make phone calls, press and hold C to return to the standby mode.
1d-11d Applications using GPRS and HSCSD can interfere with the radio.

SUB-VARIATIONS
1b To quickly select set frequency press *+ when in the radio menu.
1b To quickly select manual tuning press # when in the radio menu.
9b To quickly save the channel in a location 1 to 9 press and hold the corresponding number key, then key in the name of the channel and press ok.

NFRs /Issues
You can save up to <<numchannels, 10, 20**>> channels in your phone.**
Using the FM radio consumes power, an the phone's operating time is reduced.
The frequency range is limited from 87,5 MHz to 108 Hz.

Figure 21 Mobile Phone Use Case 2b

During the case study it became obvious that our approach worked very well for identifying product line artifact elements. Starting from these elements, a first set of extraction patterns was derived. This set of extraction patterns was obviously not complete. But our intention is to build an experience-based set of extraction patterns that will grow and become more stable with each case study. Using our approach, we

were able to identify features and a lot of information for the Use Cases. More detailed lessons learned on the extraction of Use Cases can be found in Section 7.4.2. Another hypothesis we wanted to validate was that variability can be found with our approach. The table in Section 7.2 shows that our approach revealed the variability in the documents. To conclude, the first case study revealed good results forming a solid base for further case studies.

7.4 Lessons Learned

During the case study, we encountered several interesting aspects, some related to the classification of elements in user documentation, some related to Use Cases and the creation of the Use Cases.

7.4.1 Lessons Learned when Classifying

- It is difficult to separate data requirements from functional requirements.
- It is difficult to separate tasks, activities and system functions.
- Very often, there is mixed information in one row of the user documentation: E.g., "Wap over CSD". Wap is a commonality, CSD is an alternative (to GPRS).
- User documentation is different in the various domains. In this case study, the user documentation is similar to the menu structure of the phone. In other domains, it is more task-oriented.
- It is difficult to separate UI-elements from navigation elements.
- One can derive a functional requirement from each feature (The system shall include feature X).

7.4.2 Lessons Learned during Use Case Creation

- In most cases, the user documentation stated pure user inputs and no system reactions (compare table in Section 7.2). Therefore, only the inputs of the Use Cases listed in Section 7.3 were directly elicited from the user documentation.
- Comparing the elicited Use Cases with the vendors Use Cases in the product line model showed that the user documentation and the actual product line model complement each other:
  - the user documentation can reveal more information than the actual product line model (e.g., how to scroll during autotuning was only explained in the user documentation).
  - the actual product line model can reveal more information than the user documentation (e.g., the increment to increase or decrease the frequency (0,1 Mhz) was only given in the product line model).
- UCs created from user manuals are always on sea level (i.e., on a level visible for a user) or higher (see [Co01]).
- The precondition in the vendors Use Cases was the same information that we modelled in the Use Case precondition plus the Use Case goal.

8 "Civil Engineering" Case Study

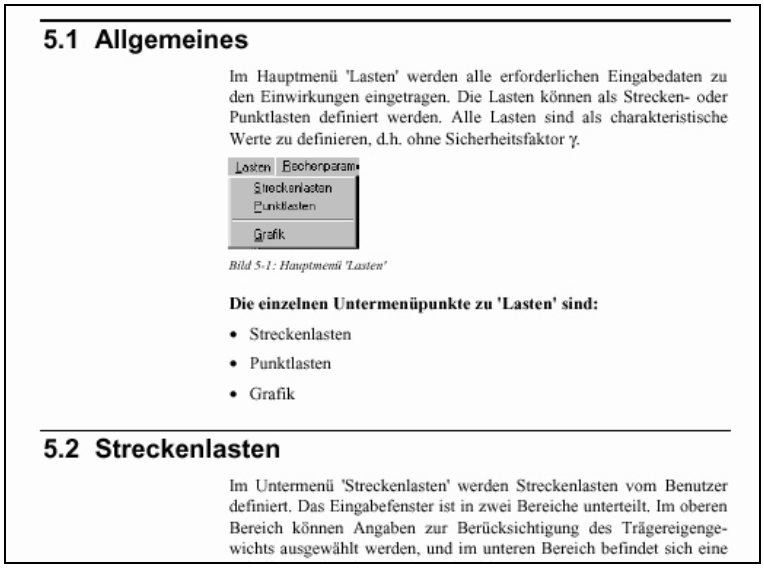
In this section we describe the second case study we performed. The case study was done to validate the first results we got with the first case study, to see if the pattern that we found during the first case study hold for different documentation and if the pattern are as applicable for information systems as for embedded systems. Furthermore, we applied the process that was only rudimentary existing in the first case study in a more fixed form by exactly following the steps we wrote down during the first case study. We applied the approach described in section 6 to different user documentations in the area of software for civil engineering. Our hypothesis when applying the approach were:

- The approach is applicable for pure information systems
- Features and Use Cases can be extracted from every documentation with the approach
- Variability can be found with the approach
- A product feature matrix (as an example product line artifact) can be built from the features we extracted from the documentation.

8.1 Description of the Case Study

The case study was performed with documents describing the systems of a small company producing civil engineering software for statistically calculating composite beams. The company develops software for static calculations used by civil engineers and architects in the domain of steel and composite (i.e., steel and concrete) construction. For projects where composite materials are used basically no software solutions exist that support the construction of whole buildings or major parts of these. The company provides separate products dealing with beams, columns, and slabs. The following german user documentations were analyzed during the case study (names changed because of anonymity):

- COL
  A software to calculate loads on columns
- SLAB
  A software to calculate slabs
- SIB
  A software to calculate loads on single beams
- BEAM
  A software to calculate composite beams.



5.1 Allgemeines

Im Hauptmenü 'Lasten' werden alle erforderlichen Eingabedaten zu den Einwirkungen eingetragen. Die Lasten können als Strecken- oder Punktlasten definiert werden. Alle Lasten sind als charakteristische Werte zu definieren, d.h. ohne Sicherheitsfaktor γ.

Lasten Rechenparam
Streckenlasten
Punktlasten
Grafik

Bild 5-1: Hauptmenü 'Lasten'

Die einzelnen Untermenüpunkte zu 'Lasten' sind:

- Streckenlasten
- Punktlasten
- Grafik

5.2 Streckenlasten

Im Untermenü 'Streckenlasten' werden Streckenlasten vom Benutzer definiert. Das Eingabefenster ist in zwei Bereiche unterteilt. Im oberen Bereich können Angaben zur Berücksichtigung des Trägereigengewichts ausgewählt werden, und im unteren Bereich befindet sich eine

Figure 22 Section "Lasten" of the BEM software, excerpt

The documentations are in german and were available as pdf.

8.2 Conducting the Case Study

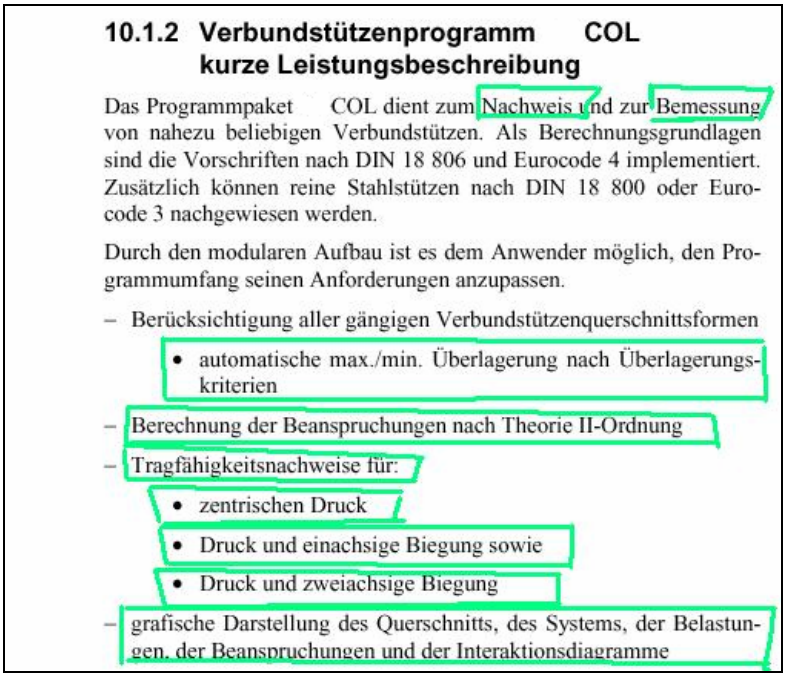
This case study was the second case study we performed, so we already had an initial version of the process of the approach and of the pattern. We followed the steps of the process (c.f. 6.2) in the following way:

Preparation

First, we browsed through the available documentation of the four products we wanted to compare in order to select a part of the documentation that is appropriate for our case study. This part should not be too big (5 – 20 pages each), it should be existent in all four of the documentations and it should contain variability in order to make it possible to write down variability between the products. The decision fell on the section "Lasten" (Loads) that was part of all four user manuals (as loads are a central concept of this software, they are present in all the manuals). In addition to the section "Lasten" (see Figure 22 for an example), a product description of all products of the company (including the four we analyzed) was used as a input. The section "Lasten" and the product description was printed out for further analysis. It was decided to analyze the documents in parallel as they were not too different. Altogether the descriptions of the 4 systems had 33 pages.

Analysis

In the analysis step, the initial pattern list was applied to the documentation. In order to do this, the four documents were read/compared in parallel and the pattern were applied, so when an element was identified by a pattern it was marked in the document and the element was compared to the other documentations. Figure 23 shows



10.1.2 Verbundstützenprogramm COL
kurze Leistungsbeschreibung

Das Programmpaket COL dient zum Nachweis und zur Bemessung von nahezu beliebigen Verbundstützen. Als Berechnungsgrundlagen sind die Vorschriften nach DIN 18 806 und Eurocode 4 implementiert. Zusätzlich können reine Stahlstützen nach DIN 18 800 oder Eurocode 3 nachgewiesen werden.

Durch den modularen Aufbau ist es dem Anwender möglich, den Programmumfang seinen Anforderungen anzupassen.

- Berücksichtigung aller gängigen Verbundstützenquerschnittsformen
  - automatische max./min. Überlagerung nach Überlagerungskriterien
- Berechnung der Beanspruchungen nach Theorie II-Ordnung
- Tragfähigkeitsnachweise für:
  - zentrischen Druck
  - Druck und einachsige Biegung sowie
  - Druck und zweiachsige Biegung
- grafische Darstellung des Querschnitts, des Systems, der Belastungen, der Beanspruchungen und der Interaktionsdiagramme

Figure 23 Example Features marked in the COL description

some example features that were marked in the document. Some were common features (could be found in all documentations) some were variable features.

The following elements were found by the analysis:

- 39 features (most of them common or optional features)
- 17 general functional requirements (most of them only in one product)
- 60 requirements on the navigation (user interface requirements).

Selection

As no real expert in composite calculation software was available, a real selection could not happen. A partial real requirements document from the company was available, but from a different subdomain and an uncomplete document, so the results could not be directly compared or integrated into a document.

8.3 Validating the Case Study

In order to validate the results we found by extracting requirements elements from the documentations, from the extracted items a product line model element was directly built, without adding or deleting elements. From the Features that were found in the documentation a Product Feature Matrix [Sch03] was built. A product feature matrix describes the characteristics of the product line by means of products, domains and features. Features are grouped into domains and the features that a product has are marked (or values are filled in). As only the subdomain "Lasten" of the products was analyzed all features of the product feature matrix are part of the domain "Lasten". Table 6 shows the product feature matrix that was built from the documenta-

tion. It shows more than 30 features of the four systems and a significant amount of variability between the systems.

Product-Feature Matrix	Col	Slab	Sib	Beam
Lastdefinitionen	X	X		
Lastgruppen	X			
vorgegebene Lastvektoren	X			
Lastkombinationen	X			
Interaktionsdiagramme	X			
Grafik	X	X	X	X
Kräfte	X			X
Momente				X
Biegemomente	X			
Gleichstreckenlasten	X			
Vorverformungen	X			
Schiefstellungen	X			
Einzelbeanspruchungen	X			
Normalkräfte in Staglängsrichtung	X			
Brand Einwirkungen	X			
Einfluß des Langzeitverhaltens	X			
Imperfektionen	X			
Eingaben für Stahlstützen	X			
Knickspannungslinie	X			
Eigengewicht		X	X	X
Beanspruchungen		X		
Einwirkungen		X		
Montagelasten		X	X	
Beanspruchungen auf Profilblech		X		
Beanspruchungen auf Verbunddecke		X		
automatische Lastgenerierung			X	
Streckenlasten			X	X
Punktlasten			X	X
Ausbaulasten			X	
Verkehrslasten			X	
Beanspruchungen auf Stahlträger ohne Unterstützung			X	X
Beanspruchungen auf Stahlträger mit Unterstützung			X	X
Beanspruchungen auf Verbundträger			X	X
Gleichstrecken und Trapezlasten				X

Table 6 Product Feature Matrix

Concerning our hypothesis we had the following (purely qualitative and partially subjective) results:

- **The approach is applicable for pure information systems**
  As we could reuse the pattern from the first case study and found elements with most of the pattern that were useful for the first case study, the approach was also

applicable in this case. Anyway, it is at this point in time not possible yet to conclude that the approach is equally applicable to information systems and embedded systems. There is currently an experiment going on that will hopefully validate this hypothesis with quantitative data.

- **Features and Use Cases can be extracted from every documentation with the approach**
  As we could not extract Use Case elements from the documentation of the second case study this hypothesis did not hold. After carefully analyzing the documentation we came to the result that the lack of found Use Case elements is not because of missing pattern but because the user documentations are not written in a "Use Case like" way. We found many Use Case elements in the mobile phone documentation because it described the interaction between the user and the mobile phone in a way like "the user should do this…the menu shows that". This interaction description was not at all present in the second user manual. If user interfaces were described only the general structure of menus and windows was described, not the interaction concerning the task of the user was described.
- **A product feature matrix (as an example product line artifact) can be built from the features we extracted from the documentation**.
  As we wanted to show different product line artifacts can be built with the approach we built a product feature matrix (see Table 6) in the second case study. The product feature matrix could be built with more than 30 features and a significant amount of variability could be captured in the matrix
- **Variability can be found with the approach**.
  As variability was identified with the features and captured in the product feature matrix this hypothesis was thoroughly valid.

8.4 Lessons Learned

Apart from the lessons from validating our hypotheses as described in section 8.3 and from the lessons learned from the first case study described in section 7.4 we made the following experience:

- As expected, new pattern could be found when performing this case study. With almost each application of the approach and the pattern, it is to be expected that new pattern will emerge because of the new structure of documents. There is no fixed form of user documentation; so all documentations differ in some aspects (structure, layout, wording etc.). But with each application of the approach and the finding of new pattern it has to be decided if the characteristics found are general characteristics, holding for many documentations or special characteristics that lead to a meaningful extraction of requirements elements or variability only in a very special context or even only in one kind of documentation.
- Not every documentation contains Use Case elements
  We were rather surprised to find out that we could not find Use Case elements in the documentation that we analyzed in the second case study. In all other documentations that we were analyzing or browsing through, Use Case like elements could be found. A reason for this lack of Use Cases could be that the documentation was not written by documentation specialists and so has no information about the interaction (as it should be). Another possible reason could be that it is more useful for embedded systems to describe user manuals in a Use Case like form as

there are always special interaction elements (buttons, screens)We would guess that in information systems the structure of the interaction is clear through the standard widgets of the operating system. But this hypothesis needs further validation

- You must understand a little bit of the domain to do an analysis (but less than without the pattern), especially for commonality and variability
  When analyzing the documentation it was hard to decide if some items are commonalities, alternatives or options. This was very hard to decide most of the time as we had not the domain knowledge of a real user or of a developer of a system. E.G. we could not decide if the terms " Lastdefinition" (Definition of Loads) and "Lasten" (Loads) as menu items mean the same thing in two different systems, so if this is a commonality, an option of one system or an alternative between the two systems.

9 "Roster Software" Case Study

In this section we describe the third case study we performed with the extraction approach. The case study was done to validate the results we got with the first two case studies in practice. This case study was performed in a real industrial setting, the results produced in the case study were evaluated by domain experts in the organization and further used to build a domain model. We applied the approach described in section 6 to different user documentations in the area of roster software. Our hypotheses when applying the approach were:

- A product feature matrix can be built from the features we extracted from the documentation.
- The CaVE Approach produces complete and correct results
- The CaVE approach reduces the expert load and is an efficient approach

9.1 Description of the Case Study

The case study was performed together with the Company SIEDA Gmbh, Kaiserslautern. The company "SIEDA – Systemhaus für intelligente EDV Anwendungen GmbH was started 1993 in Kaiserslautern and has a staff of 14 employees at the moment. The SIEDA GmbH is a specialist in the development of software for the intelligent solution of complex optimization problems in particular short-time manpower planning. Their main system "ORBIS Dienstplan" is a roster planning system with application emphasis in the public service sector. Contrary to most products available on the market, it is specialized in planning by multi-layer models (e.g. with services around the clock). For more intelligent planning constraint solving methods are used. Apart from the actual planning also the performed services are entered through the system and the data for the wage and salary statement can be computed with consideration of tariff regulation.

The larger context where this case study was applied was the project UseLine [JKS04], the case study is also described in [JD04]. The projects goal is it to integrate usability improvements of a software system and product line engineering activities. Within this project an additional product should be derived from the existing product line of roster software systems. The activities in this larger case study were performed in the Sieda Context between June 2003 and January 2004. Two Persons from IESE and two persons from Sieda were involved. The goal of Sieda in this case study was to derive a third product from their existing product line of rostering software systems, a so-called light product that should serve a low-end market. While deriving the product the overall usability of the system should be evaluated and improved in order to support these activities. Within this larger case study, the CaVE method was applied as a case study within, to find features from the existing products that can be used for the new product.

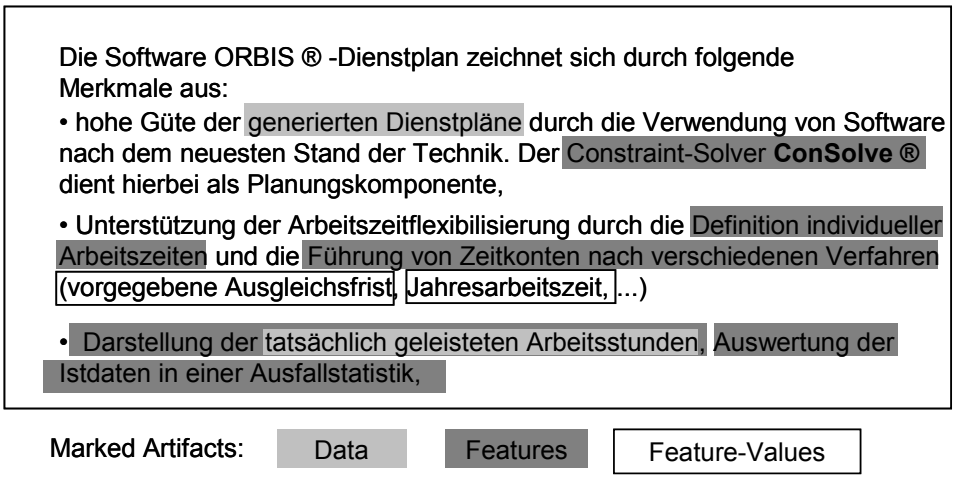


Figure 24 Artefacts in the user documentation of Sieda

9.2 Conducting the Case Study

The current product line of roster systems of the Sieda GmbH consists of two large products (one for hospitals, one for fire departments) that are customized into a larger number of customer specific products. A new, so called "light product" that should have less features than the other products should be derived from the existing product line. In order to systematize the derivation of the new features, the CaVE Approach should be applied to derive the common and variable features of the existing systems, so a product x feature matrix should be built with the help of the analysis of existing documentation.

Preparation

The documentation of the two main products of the product line "Orbis" and "Orbis Rettungsdienst" consisted of 28 pages each. As there were only two documentations and the amount of 2 * 28 pages is manageable for an analysis, the documents did not have to be split but could be analyzed in parallel and as a whole.

As the primary goal of the analysis activities was to find features, the pattern that extract features and their domains and subdomains were selected from the set of pattern for the extraction. Also, the pattern that should find variability were selected.

Analysis

The documentation was analyzed with the help of the selected pattern in parallel. The analysis with the Cave Pattern produced 116 features, 8 subdomains and 11 domains with domain descriptions. Those features were marked in the documentation (for an example see Figure 24) and after marking collected in a preliminary product feature matrix. We also produced domain descriptions that describe what a domain is with the help of the documentation. For a further description and examples for product feature matrix and domain descriptions see [Sch03] and [Sch01]. Those elements were controlled by the non-expert product line analysts in order to find duplicates and identify wrong elements. During this post-analysis two additional features and three sub-

domains were identified. So, as the final result of the analysis 118 features were found in the documentation. 8 domain descriptions were built and 11 subdomains were identified.

Selection

This preliminary product feature matrix (see Table 8) that contained the features of the Orbis and the Orbis Rettungsdienst Software that were found in the documentation of both systems was shown to the domain experts (the lead architect and the CEO of the company) in order to

- identify features that were extracted wrongly
- identify additional features that were not found by the pattern
- identify those features from the existing systems that should come into the new light product.
- Correct the domain descriptions

As a result of these activities, 5 of the existing features were identified as wrong or as duplicates, 17 additional features were found, so the final list consisted of 130 features. Later on, it was analyzed which of these features should be part of the Orbis light system (this step is not part of the CaVE Approach but can be seen as a modeling step). The domain descriptions did not have to be corrected.

9.3 Validating the Case Study

In order to validate the case study we counted the correct and wrong features, domains and subdomains in the different stages of the case study. Furthermore we tracked the time needed to complete the different stages of the questionnaire. With this quantitative data we can calculate the correctness and completeness of the approach for this case study. Furthermore we developed a questionnaire to get qualitative data on expert opinion on the approach and to get a qualitative measurements. In the following we describe how the hypotheses from section 9.1 were validated:

- **A product feature matrix can be built from the features we extracted from the documentation.**
  We found 128 Features in the documentation. Most of them were common features, only two features were found as variable feature due to the high similarity of the two product "Orbis" and "Orbis Rettungsdienst". We identified 8 domains and 11 subdomains with the help of our pattern and built a product feature matrix out of those features, domains and subdomains. So our first hypothesis that it is possible to build a product feature matrix holds.

- **The CaVE approach produces complete and correct results**

In order to get an overview on completeness and correctness of the results we counted the changes and additions of features, domains and subdomains during the case study. We came to the following results:

	# Features	# Domains	# Subdomains	Changes
Analysis Step	116	8	11	
Post Analysis	118	8	14	2 additional features and three domains identified
Selection	130	8	13	17 additional features identified; 5 wrong or duplicate features deleted; one sudomain deleted
Modeling	130	8	13	No features/domains added, variable features for Orbis light identified

Table 7 Number of identified features and domains

When computing the completeness and correctness of the results with this data we come to the following results:

The completeness of the approach can be calculated as the number of correct items found by the approach divided by the overall number of correct items (which is, in this case the number of items after the selection step because they were validated as correct by the domain experts):

Completeness (Features) = # correct features identified by CaVE / # correct features after selection = (116 – 5) / 130 = 85,3 %

Completeness (Domains) = # correct domains identified by CaVE / # correct domains after selection = 8 / 8 = 100 %

Completeness (subomains) = # right subdomains identified by CaVE / # right subdomains after selection = (11-1) / 13 = 76,9 %

The **average completeness** of the Cave Approach over features, domains and subdomains is **87,4 %** (Completeness of features, subdomains, domains / 3)

The Correctness of the approach can be calculated as the number of correct items found by the approach divided by the overall number of items found by the approach:

Correctness (Features) = # right features identified by CaVE / # all features identified by CaVE = (116 – 5) / 116 = 95,6 %

Correctness (Domains) = # right domains identified by CaVE / # all domains identified by CaVE = 8 / 8 = 100 %

Correctness (subomains) = # right subdomains identified by CaVE / # all subdomains identified by CaVE = (11-1) / 11 = 90,9 %

The **average correctness** of the Cave Approach over features, domains and subdomains is **95,5 %** (corectness of features, subdomains, domains / 3)

So, when analyzing this data it can be seen that the correctness of the results of the approach is very high (over 95 %), which means that the approach can find a very high amount of the data needed for modeling.
The completeness is with about 87 % not as high as the correctness. With a value of almost 90% the results of the CaVE analysis are quite trustworthy but also this data shows that it is essential to have a selection step where the domain experts identify errors and find additional elements. This is fleshed out by the fact that in the post analysis the non-experts could find additional features but could not identify wrong features or recognize duplicate features that had (slightly) different names.

- **The CaVE approach reduces the expert load and is an efficient approach**

In order to find out the value of the approach for the domain experts we developed a questionnaire and let the two experts fill out the questionnaire after the selection step.

As in this case study we only have two experts and so only two data points the results are of course not significant. But the results can give a trend in estimating if there is expert load reduction by using CaVE.

We asked the experts the question "Having such a documentation of the product line for me has a value of X" . Both experts answered 16 hours. One expert rated the domain and the feature description as equally valuable (8/8 hours), one expert rated the domain description as less valuable (1 hour domain description, 10 hours feature description).

Domain	Subdomain	Feature	Values	Product1	Product2	Product3
System-verwaltung	allgemein			X	X	
		Unterstützung verschiedener Diensttypen	Frühdienst, Tagesdienst, Spätdienst, Nachtdienst....	Alle	Alle	N
		Bereitschaftsdienste		X	X	
	Konfiguration			X	X	X
		Tarifgruppenbezug		X	N	X
		Freie Fehlzeiten		X	X	N
		….				

Table 8 Excerpt of the Product Feature Matrix

The times for the CaVE analysis were as follows:

15 min Preparation
170 min Analysis (including a first version of the Product Feature Matrix and first Domain Description)
95 min Post Analysis (elimination of duplicates, crosscheck, identification of new features, second version of matrix and domain description)

100 minutes Selection (Selection Workshop with three participants, one non-expert, two experts.

So the total time for the analysis was 15 + 170 + 95 + 100 * 3 = 580 minutes -> 9, 6 hours. Of these 9,6 hours, 3,3 hors were expert hours, 6,3 hours were non-expert hours.

That the value of the analysis to the experts (16 hours) was even higher than the time for the whole analysis (9,6 hours including expert and non-expert hours) shows that in this case there was a significant overall load reduction, not only an effort reduction. When we compare the 16 hours value of the analysis with the 3,3 hours the experts actually spent with validating the results we have an expert load reduction of 12,6 hours which is a reduction of 78,8 % (12,6 h / 16 h) compared to the expert value.

Concluding the validation we can say that the validation was very successful. The correctness and completeness of the results could be shown in this case as the results were validated by the domain experts and the expert load reduction was very high. As there is no fixed value of correctness and completeness above which one can say that the approach produces "good" results (c.f. discussions for Information retrieval approaches [BR99]) and the correctness and completeness of an extraction approach influence each other a completeness of 87% and a correctness of 95% for such an extraction approach can be seen as highly acceptable values. As expert load reduction is the main goal of the approach the high correctness of the approach is a very important issue. As the approach produces few "false positives", the expert time is not so much spent on deleting wrong results during the selection step.

It must be admitted though that there was not so much variability in the documentation so we could not really validate the variability specific aspects of the approach. But the situation that we had in this case study, having some similar products and deriving a next product that should have less functionality than the others showed out as ideal for the approach as all features for the new product could be found in existing documentation and so not so many new features had to be included in the selection or in the following modeling step.

As the lessons learned in this case do not differ much from the lessons in the other two case studies, we do not further address the issue of lessons learned here.

10 Conclusions

In this report we described an approach for extraction of basic requirements items from legacy user manuals that can be used for product line modeling. The basis of the approach is a variability management approach, a metamodel, different related work and a set of extraction pattern. With the approach different items (Features, Use Cases, functional and non-functional requirements) can be identified in user documentation by a non-expert and can later be approved and used for modeling by domain experts. The main advantage of this pattern based approach is therefore the expert load reduction.

We described in this report the general approach and three case studies where we applied the approach. Our general experiences are that the approach with its process steps and the pattern help in finding relevant elements that can be used for modeling. In one of our case studies we could build several variable Use Cases with the elements found with the approach that were comparable to real Use Cases used in the development of mobile phones. In the other case study we could build a product feature matrix, an artifact that is often used in product line scoping [Sch03]. The pattern that we identified, were captured in a template for pattern that holds all relevant information on the pattern. The pattern support the extraction process by giving concrete guidelines on how to identify items on a rather syntactical level, without having a deep domain understanding. At the moment there exist about 30 pattern, the list of pattern will surely be extended in the future. It cannot be expected to have a complete set of pattern sometimes, there will always be new kinds of elements to find and new relations to discover but it can be expected that a significant amount of product line model elements can be extracted from every user documentation with the help of the process described within the approach and the pattern, although documents significantly differ in layout, structure and content.

In the future, more case studies are planned. In order to quantitatively validate the approach, a controlled experiment has been conducted that has to be evaluated now. Additionally, a tool is planned to support the extraction process and to realize the pattern. Tool support could increase efficiency of processing and correctness of the results significantly for the techniques proposed and can relieve experts and product line engineers. With a tool, models can be generated semi-automatically and so traceability can be easily achieved. It is planned to develop an extraction tool, which integrates document-analysis and information retrieval techniques like indexing or morphology to support document-based modeling of the commonalities and variabilities of planned products in the domain of a product . This tool can then support stakeholders in the domain in identifying, eliciting and analyzing commonalities and variabilities in the domain, which are retrieved from the existing documents. The tool will support the proposed process by making suggestions for common and variable modeling elements (like features, partial use-cases, classes), which can then be selected and integrated into the product line model by the stakeholders and product line engineers.

Acknowledgements

We would like to thank Joachim Bayer, Michael Eisenbarth, Stefania Gnesi, Alessandro Fantechi, Ronny Kolb, Kirstin Kohler, Jens Knodel, Giuseppe Lami, Dirk Muthig and Barbara Paech for valuable comments on our work and for fruitful cooperation.

References

[ABM00] Colin Atkinson, Joachim Bayer, and Dirk Muthig. Component-Based Product Line Development: The KobrA Approach. In Patrick Donohoe, editor, Proceedings of the First Software Product Line Conference, pages 289--309. Kluwer Academic Publishers, August 2000.

[ACC+02] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo. Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering, Vol. 28, No. 10, October 2002.

[AG97] V.Ambriola, V. Gervasi. Processing Natural Language Requirements. In Proceedings of ASE 1997, pagg. 36-45. IEEE Press, 1997

[AK02] I. Alexander, F. Kiedaisch. Towards recyclable system requirements. In Proc. of the 9th IEEE International Conference on Engineering of Computer-Based Systems (ECBS 2002), pages 9-16, 2002.

[Ara93] G. Arango. Domain analysis methods. In W. Shaefer, R. Prieto-Diaz, and M. Matsumoto, editors, Software Reusability. Ellis Horwood, 1993

[BFG+02] A. Bertolino, A. Fantechi, S. Gnesi, G. Lami, A. Maccari, Use Case Description o f Requirements for Product Lines, Proc. of the International Workshop on Require ments Engineering for Product Lines - REPL02. Essen, Germany, Sep.9, 2002

[BK99] A. Becks, J. Köller. Automatically Structuring Textual Requirements Scenarios. Proc. of 14th IEEE Conf. on Automated Software Engineering, Cocoa Beach, Florida, 1999, pp. 271-274

[BMW99] J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In Proceedings of the First International Symposium on Generative and Component-Based Software Engineering (GCSE '99), Erfurt, Germany, September 1999

[BNT02] R.Biddle, J.Noble and E.Tempero. Supporting Reusable Use Cases. In Proceedings of the Seventh International Conference on Software Reuse, pages Springer LNCS pp. 210–226, 2002.

[BR99] R. Baeza-Yates and B. Ribeiro-Neto. Modern information retrieval. Addison-Wesley, 1999

[BR96] J. Burg and R. van de Riet. Analyzing informal requirements Specifications: A first Step towards conceptual modeling. In Proceedings of the Second International Workshop on Applications of Natural Language to Information Systems (NLDB'96), Amsterdam, 1996

[BS86] V. Basili, R. Selby, D. Hutchens, "Experimentation in Software Engineer-

ing," IEEE Transactions on Software Engineering, vol. SE-12, no.7, July 1986

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

[BGW+99] J. Bayer, J.-F Girard, M. Wuerthner, J.-M. DeBaud, and M. Apel. Transitioning legacy assets to a product line architecture. In Proceedings of the Seventh European Software Engineering Conference, Lecture Notes in Computer Science 1687, pages 446–463, Toulouse, France, September 1999

[CN01] Paul C. Clements and Linda Northrop. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, August 2001

[Co01] A. Cockburn. Writing Effective Use Cases. Addison Wesley, 2001.

[CR00] J.Cybulsky and K. Reed. Requirements Classification and Reuse. Crossing Domain Boundaries. In Proceedings of ICSR-6. Springer LNCS 1844,pp 190 210, 2000

[Da93] A. Davis. Software Requirements: Objects, Functions, and States. Prentice Hall PTR, 1993

[DeB96] J.-M. DeBaud. Lessons learned from a domain-based reengineeing effort. In Proceedings of the Working Conference on Reverse Engineering (WCRE '96), pages 217–226, November 1996

[DS98] J.-M. DeBaud and Klaus Schmid. A Practical Comparison of Major Domain Analysis Approaches - Towards a Customizable Domain Analysis Framework. In Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering (SEKE'98), June 1998.

[Empress] http://www.empress-itea.org Homepage of the Empress Project, last visited 22.12.

[Emp03] Deliverable 3.1.2 of the Empress Project "Framework for Requirements" December 2003

[FGJ+03] A.Fantechi, S.Gnesi, I.John, G.Lami, J.Dörr "Elicitation of Use Cases for Product Lines". International Workshop on Product Family Engineering 2003 (PFE5), Siena, Italy, November 4-6 2003

[FFG+01] 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

[FPF97] W.Frakes , Ruben Prieto-Diaz and C. Fox. DARE-COTS A Domain Analysis Support Tool. In Proceedings of the 17th International Conference of the Chilean Computer Science Society (SCCC '97), 1997.

[GKS+01] Christina Gacek, Peter Knauber, Klaus Schmid, and Paul Clements. Successful Software Product Line Development in a Small Organization: A Case Study. Technical Report IESE-Report No. 013.01/E, Fraunhofer Institute for Experimental Software Engineering (IESE), March 2001.

[GS02] Birgit Geppert and Klaus Schmid (Eds.). Proceedings of the International Workshop on Requirements Engineering for Product Lines, Sep. 2002.

[HP03] Günter Halmans, Klaus Pohl Communicating the Variability of a Software-Product Family to Customers. Journal of Software and Systems Modeling, Springer, 2003

[HvdH+99] J. Hoppenbrouwers, W. van den Heuvel, S. Hoppenbrouwers, H. Weigand and O. Troyer. The Grammalizer: A CASE Tool Based on Textual Analysis. Unpublished Paper, Submitted to Tools USA '99.

[IEEE98] EEE-Std 830-1998 IEEE Guide to Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, New York, 1998

[JD03a] I. John, J. Dörr Elicitation of Requirements from User Documentation. Proceedings of REFSQ '03, Klagenfurt, June 2003

[JD03b] I. John, J. Dörr Extracting Product Line Model Elements from User Documentation. IESE Report 112.03/E, Fraunhofer IESE, 2003

[JD04] I. John, J.Dörr. Extraktion von Produktfamilienanforderungen aus Benutzerdokumentation. In Produktlinienentwicklung in Deutschland. dpunkt, 2004, to appear

[JKS04] I.John, K.Kohler, M.Schmettow. Use Line: Process Description and Case Study. IESE Report, Fraunhofer IESE, 2004; to appear

[JM02] I. John and D. Muthig. Tailoring Use Cases for Product Line Modeling. In Proceedings of the International Workshop on Requirements Engineering for Product Lines (REPL'02), pages 26--32, September 2002

[Jo01] 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.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[Ka92] M. Kasunic. Synthesis: A Reuse-Based Software Development Methodology, Process Guide, Version 1.0. Technical report, Software Productivity

Consortium Services Corporation, October 1992

[KMS+00] P. Knauber, D. Muthig, K. Schmid, and T. Widen. Applying product line concepts in small and medium-sized companies. IEEE Software, Volume 17, Number 5, September/October 2000, pages 88–95, 2000

[KS00] Juha Kuusela and Juha Savolainen. Requirement Engineering for Product Families. In Proceedings of the 22nd International Conference on Software Engineering (ICSE), pages 61--69, Limerick, Ireland, June 2000. ACM

[La03] Soren Lauesen, "Task Descriptions as Functional Requirements", IEEE Software, March/April 03.

[LKM+02] T. Levendovszky, G. Karsai, A. Ledeczi, M. Maroti, A. Ledeczi and H. Charaf.: Model reuse with metamodel based-transformations. In Proceedings of the Seventh International Conference on Software Reuse, pages Springer LNCS pp. 153-165

[MB89] Y. S. Maarek and D. M. Berry. The use of lexical affinities in requirements extraction. In Proceedings of the Fifth Internationals Workshop on Software Specification and Design, Pittsburg PA, 1989, 1989.

[MK+99] M. Mannion, B. Keepence, H. Kaindl, J. Wheadon. Reusing Single System Requirements for Application Family Requirements. In Proc. of the 21st International Conference on Software Engineering (ICSE'99), pp 453-462, ACM, 1999

[MKH+98] M.Mannion and B.Keepence and D.Harper. Using Viewpoints to define Domain Requirements. In IEEE Software Vol. 15, No. 1, January/February 1998

[Meta] http://www.metamodel.com visited 13.6.2003

[MY00] A. Mili and S. M. Yacoub. A Comparative Analysis of Domain Engineering Methods: A Controlled Case Study. In P. Knauber and G. Succi, editors, Proceedings of the International Workshop on Software Product Lines: Economics, Architectures, and Implications, Limerick, Ireland, June 2000.

[Mu02] D. Muthig. A Lightweight Approach Facilitation the Incremental Transition Into Software Product Lines . PhD Theses in Experimental Software Engineering, Fraunhofer IRB, 2002

[NRC+02] J. Natt och Dag, B. Regnell, P. Carlshamre, M. Andersson, J. Karlsson. A Feasibility Study of Automated Support for Similarity Analysis of Natural Language Requirements in Market-Driven Development, Requirements Engineering Journal, 7(1), 2002.

[NSK00] U. Nikula, J. Sajaniemi, H. Kälviäinen. A State-of-the-Practice Survey on Requirements Engineering: In Proc. of INSPIRE V, The British Computer Society 2000.

[OLR01] S. Overmyer, B. Lavoie and O. Rambow. Conceptual Modeling through Linguistic Analysis Using LIDA. In Proceedings of 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Canada, 2001

[REG+00] P. Rayson, L. Emmet, R. Garside, P. Sawyer. The REVERE project. In Proc. of the International Conference on Applications of Natural Language to Information Systems (NLDB 2000), pp 288-300, Springer, 2000.

[RB01] P. Reason, H. Bradbury (eds.). Handbook of Action Research. Sage Publications, 2001

[RR99] S. Robertson, J. Robertson. Mastering the Requirements Process, Addison-Wesley, 1999

[RW99] M.Zopelari Roseti C.Werner. A Knowledge Acquisition Systematic within the domain Analysis Context. Second Ibero-American Workshop on Requirements Engineering, Buenos Aires, Argentina, September 1999.

[SBK+00] K. Schmid, U. Becker-Kornstaedt, P. Knauber, and F. Bernauer. Introducing a software modeling concept in a medium-sized company. In Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, 2000.

[Sch03] K Schmid. Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines. PhD Theses in Experimental Software Engineering, Fraunhofer IRB, 2003

[Sch00] K. Schmid. Product Line Mapping Report. IESE-Report 028.00/E, Fraunhofer IESE 2000.

[SJ04] Klaus Schmid and Isabel John. A Customizable Approach To Full-Life Cycle Variability Management, Accepted for Publication in Science of Computer Programming (Elsevier) - Special Issue on Software Variability Management 2004 , to appear

[SJ03] Klaus Schmid and Isabel John Generic Variability Management and its application to Product Line Modeling in Bosch, vanGurp (eds): Proceedings of the first Workshop on Software Variability Management, Groningen, 2003.

[SM02] K.Schmid and M. Verlage. The Economic Impact of Product Line Adoption and Evolution. IEEE Software, 19(4):50--57, JulyAugust 2002.

[ODM96] Software Technology for Adaptable, Reliable Systems (STARS). Organization Domain Modeling (ODM) Guidebook, Version 2.0, June 1996

[So01] I. Sommerville Software Engineering, Addison Wesley 2001

[Sti00] E. Stierna. Requirements Reuse in support of the aviation mission planning system migration to the joint mission planning system. Masters Thesis, Naval Postgraduate School, Monterey, California, 2000

[TWJ97] B. Tschaitschian, C. Wenzel, and I. John. Tuning the quality of in-formal software requirements with KARAT. In Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ'97), 1997.

[vKP+02] Antje von Knethen, Barara Paech, Friedemann Kiedaisch, and Frank Houdek. Systematic Requirements Recycling through Abstraction and Traceabil-ity. In RE'02, IEEE Joint International Requirements Engineering Conference, September 2002, Essen, Germany, 2002.

[WL99] D. Weiss, C. Lai. Software Product Line Engineering. Addison-Wesley, 1999

[Wi84] C. D. Wickens. Processing resources in attention. In R. Parasuraman & R. Davies (eds.), Varieties of attention (pp.63-101). New York, 1984, Academic Press.

[What] www.whatis.com ; http://whatis.techtarget.com/ definition/0,,sid9_gci211982,00.html visited 13.6.2003

Appendix

A Template for Elicitation Pattern

The Elicitation Pattern are described with a template. The template contains all information that is necessary to characterize each pattern, to clarify how the pattern is used and under which circumstances the pattern can be used.

Name	#	The Name of the Pattern
Short Description	A one sentence description of the pattern	
Input	The input model element	
Output	The output model element	
Recall	The average recall (the percentage of the total relevant elements retrieved by the pattern)	
Precision	The average precision (the percentage of relevant elements in relation to the number of total elements retrieved	
Value	The value or relevance this pattern has for the stakeholder "domain expert" (given as --,-, o, +, ++)	
Transition	The input and output level for the pattern in the conceptual model	
Long Description	A longer description of the pattern, including keywords	
Related Pattern	A list of other patterns this pattern is related to (e.g. composed of these pattern)	
Example	An example for an input element where the pattern holds and the elicited result	

B Further Examples

Name	1	Heading-Feature
Short Description	Headings of sections or subsections typically contain features	
Input	Heading	
Output	Feature	
Recall	+	
Precision	++	
Value/ Relevance	-	
Transition	Document element -> Product Line Artifact	
Long Description	As features describe functionalities that are of importance for the user, they are found at prominent places in the UD.	
Related Pattern		
Example	In a mobile phone UD "Sending an SMS" is a heading that describes a feature	

Name	2	Phrase-to-extension
Short Description	Phrases like "normally" "with the exception", "except" can mark Use Case extensions	
Input	Phrase	
Output	Use Case extension	
Recall	o	
Precision	++	
Value/ Relevance	This extraction pattern has a {low, average, high} value to the expert	
Transition	Document element -> Product Line Artifact	
Long Description	Exceptions are described in the UD as special/exceptional cases of normal usage	
Related Pattern		
Example		

Name	3	Sentence-constraint
Short Description	a highlighted sentence containing the phrase "normally" or "with the exception" can describe constraints	
Input	Sentence; phrase	
Output	Constraint	
Recall	o	
Precision	o	
Value	++	
Transition	UD -> RC	
Long Description		
Example	"The phone plays music with the exception of an emergency call"	

Name	3	Parameter-value
Short Description	If Sentences or Phrases are identical in different documents but include a different numerical value this can be a parametrical variability or alternative values	
Input	Sentence; phrase	
Output	Alternative	
Recall	-	
Precision	++	
Value	+	
Transition	UD -> RC	
Long Description	Parameters in the systems that have a different value could be realized differently in the software	
Example	"The phone can send SMS with at most 124 characters" <-> "The phone can send SMS with at most 136 characters"	

Document Information

Title:	User Documentation based Product Line Modeling
Date:	15. January 2004
Report:	IESE-004.04/E
Status:	Final
Distribution:	Public