Details of Formalized Relations in Feature Models Using OCL

Detlef Streitferdt Matthias Riebisch Ilka Philippow

Technische Universität Ilmenau
{detlef.streitferdt | matthias.riebisch | ilka.philippow}@.tu-ilmenau.de

Abstract

System families are a form of high level reuse of development assets in a specific problem domain, by making use of commonalities and variabilities. To represent assets belonging to the core of the family and assets belonging to variable parts, feature modeling is a widely used concept. Consistency checking in feature models is not yet addressed appropriately by current methods. This paper gives a brief overview of feature modeling and elaborates the problems of current approaches. Based on the applications of these approaches within an ongoing research project this paper proposes a formalized definition for feature modeling using the Object Constraint Language (OCL) and a set of associations and constraints to be used in the feature model. The relations between features in the feature model and features to external assets are examined and a way to formally handle these relations is presented as a result of a research project.

Keywords: Feature modeling, feature constraints, OCL, variability

1 Introduction

Software engineers are trying to fulfill a steady desire of the market for high quality software and short development cycles by reuse efforts, component technologies, and new programming paradigms. These solution ideas are used to satisfy the customer needs as fast as possible. Reuse is a very promising idea adapted out of many other engineering disciplines. The idea is easy, but many problems arise when software engineers try to apply the idea of reuse to software development.

The family approach bridges the gap between a fully custom-built single product and the mass production of software. By using a flexible system architecture, built out of a core used by all members of the family and several variable components, the application engineer can develop new applications within a short time period and with less resources compared to conventional software development. System families ensure a planned form of reuse within a specific problem domain.

Many systems are based and dependent on the flexible family architecture, which was itself developed based on the overall requirements for the family. Thus, requirements engineering activities have a high impact on all applications based on the family. Many of the current system family development methods are making use of feature modeling, initially described by Kyo Kang in FODA (Feature Oriented Domain Analysis) [1], to capture all the mandatory and optional 'parts' of the system. The feature model is a high level description of the system and understandable by customers. As depicted in Figure 1 the feature model located between the requirements model and the system design model.

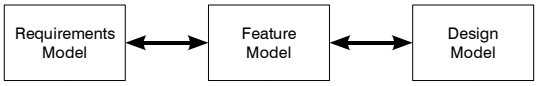

Figure 1: Features in Context

Within an ongoing research project we used feature modeling to capture system family related information and experienced problems using current feature modeling methods while keeping the model consistent. In section 2 a short introduction to the research project is given, to understand the examples in the next sections. Section 3 gives an overview of current feature modeling methods together with usage examples taken out of the project. The section closes with a summary of the problems with current feature modeling. Section 4 starts with a formal view onto feature modeling and introduces new associations and constraints, to enable constraint checking. Section 5 closes with an overall summary and an outlook containing open questions and future research directions for a lively discussion.

2 System Family Example: Digital Video Research Project

Within a group of students a digital video system was built making use of existing hardware and software components. The project goal is to develop a system family making use of adapted methods and thus, evaluating these methods. The students are taught to use the system family paradigm for the development of a software system within the context of a real-life example. The system is meant to capture and process digital TV shows. Since all of the system features including their internals need to be extracted, to form a system family, Linux, and thus, open source as the basic platform was chosen. In addition we are making use of the *vdr* project [2], the *video-streaming* project [3] and the *dvb* driver [4] for Linux. In Figure 2 a brief system overview is sketched, as a Unified Modeling Language (UML) [5] component diagram.

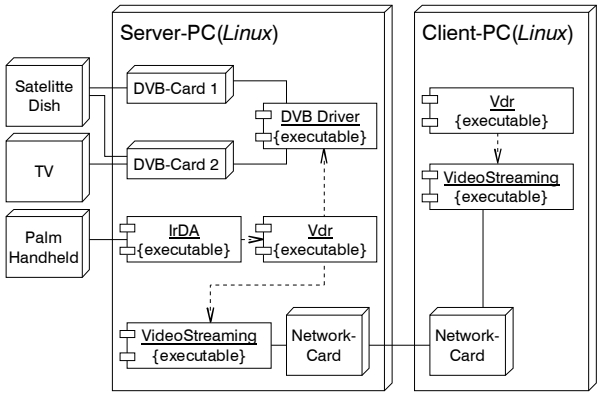

Figure 2: Digital Video System Overview

Within the requirements elicitation phase we evaluated commercial digital video systems, using brochures, web sites and interviews in local electronic stores. Together with a future product analysis we elaborated a list of more than 70 high level requirements. In this stage, features deducted out of requirements and elaborated in discussions were recorded within a simple list, forming the first step towards feature modeling.

3 State-of-the-Art

Feature modeling was first described in the FODA paper [1]. The goal of feature models is to describe a system according to its features, where a feature is defined as:

> "A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems"

This general definition was refined by later publications of Kang to address the needs of different stakeholders.

> "Features are any prominent and distinctive aspects or characteristics that are visible to various stakeholders (i.e., end-users, domain experts, developers, etc.)."

There are still many possible interpretations of this definition. A first little example is presented to explain the definition and to give a brief introduction to FODA.

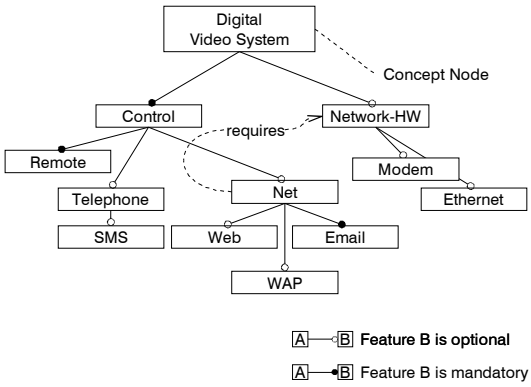

Figure 3: Simple Feature Diagram

As depicted in Figure 3, a feature diagram has a root node called *concept*, referring to the complete system. Hierarchically located below the concept node are all the features of the system. Features are marked either mandatory or optional. All mandatory features are part of all systems to be generated within this family. Thus, the set of mandatory features is forming the core of the system family.

In Figure 3 we have taken a subset out of all the features of the digital video project. At the top, the concept node refers to the system itself. Below the concept node the "Control" feature is modeled mandatory, since we need to control the video system in some way. At the level below the "Control" feature there is the mandatory "Remote" feature. The system will be equipped with a simple infrared remote control in any case. Thus, these two mandatory features belong to the core of the system family – they are present in every system. Optionally we can choose to control the system via the internet. By the time we choose the "Net" feature we automatically will have to choose the "Email" feature, which means, we are able to send an email to the video system to control the recording of a TV-show. All features of the system are arranged hierarchically and will be marked optional or mandatory to model the core and all variable parts of the system.

To build an application, based on the system family, a selection of features has to be made. The user can choose out of all optional features of the family. FODA defines two relations between features to support the consistency of the overall model and the correctness of a choice of features in particular. First, a relation called "requires" can be established to state the need for the selection of a spe-

cific feature in case another feature should have been chosen. In the above described example a user might want to choose the "Net" feature to control his system via emails. This features would definitely require the installation of network hardware in the system. Thus, the "Net" feature requires another hardware feature. Second, the opposite relation can be used to state the vital exclusion of a specific feature. Assume we have the choice between two graphics cards, but we only have a single graphics port. In case a user chooses to include one graphics card the other one has to be excluded. This is modeled with the "excludes" relationship.

Featured Reuse-driven Software Engineering Business (FeatuRSEB) [6] is a use-case based process for company wide system development, as a combination of RSEB [7] and FODA for addressing the system family specific issues. FeatuRSEB defined enhanced use-cases in the requirements engineering phase. With a variation point in use-cases FeatuRSEB models choices of different behavior. After the modeling phase the developer has to select the behavior needed to get to a valid instance of the model. Furthermore FeatuRSEB formulates first ideas to combine requirements with features. A «trace» association is used to address the requirements-feature relationship. Although, for a complete and consistent model a more precise definition of how to relate requirements to features is needed.

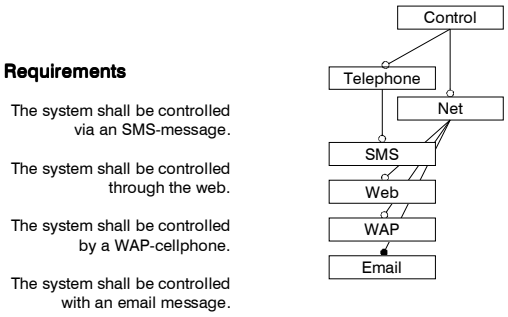

Figure 4: Relation Requirements to Features

In Figure 4 four requirements of the video project and the corresponding feature subtree are given. Is a simple one-to-one mapping with traces appropriate, or do we need more to express the relation of requirements to features? FeatuRSEB did not address these questions and did not define the «trace» relation formally.

The original FODA approach was refined by the Feature Oriented Reuse Method (FORM) [8] and was recently extended by Feature Oriented Product Line Software Engineering (FOPLE) [9]. Because of the general definition of the term feature, as stated at the beginning of the section, FOPLE separates features into four different feature categories.

- *Capability features* are distinct services, operations, or non-functional aspects. Features of this category are end-user visible and are selected by the customer to specify the desired system. (Examples: email control, on-demand videos, flicker-free view)
- *System environment features* address the hard- and software components used by the family. All the components of a system with their interfaces and protocols are part of this category. (Examples: DVB card, TV, handheld, MPEG-decoder)
- *Domain technology features* are domain specific technologies and problem solutions, used by domain experts. (Examples: DivX de-/encoder, slide show as Video-CD)
- *Implementation technique features* are general problem solutions, which may be used in different domains. (Examples: data compression, network protocols)

The categories are used to support different views of different stakeholders. There is still one single feature model holding all features, but FOPLE proposes layers corresponding to the categories to separate the features. All features are arranged into these four categories but still have relations to each other, crossing the layer boundaries. There is still some ambiguity when to use which of the different views and the the benefit of the views is not yet clear.

The "requires" and "excludes" relations of FODA are not mentioned in FOPLE. Instead, there are three new relations described. The "composition" relationship is used to hierarchically arrange all features. The "generalization" relation connects very general features with concrete ones. Finally, the "implementation" relation represents a connection between user-visible features and their implementation strategy, used in the specific domain. These relations are used in a yard inventory system, described in the FOPLE paper. This system is used to store and manage intermediate iron parts between two plants.

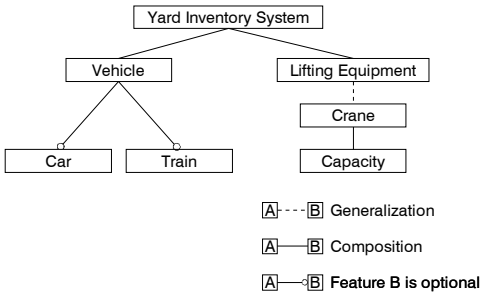

Figure 5: FOPLE Feature Example

Given the example of the FOPLE paper in Figure 5, “Car” and “Train” are optional features of “Vehicle”. “Crane” is a generalization of “Lifting Equipment”. What is the difference between a crane and a car in terms of the feature definition given at the beginning of the section. When do we need to choose the generalization and when the composition relationship? What happens to the consistency of the model if we choose to leave out both, the “Car” and “Train” feature? The described uncertainties within the model lead towards inconsistencies which in effect prevent the successful evaluation of the model.

Features of the Digital Video Project have been categorized into a user-visible part and a developer part, where the user-visible part would correspond to Kang's capability features and the developer part corresponds to the system environment features. We propose to use only the two described views. The views in mind, modeling of the system becomes clearer, since the software engineer knows who is going to use the model in the future and what the purpose of the model will be. The user-view addresses the needs of all stakeholders using the feature model to configure a system meeting their specific requirements. The developer-view is used to represent the details of the family required to further develop the system design. In addition, the features in this view are important to check the consistency of the feature model in the context of the system environment. To enable a customer to make a valid feature choice, the hierarchy of the model is the first important mechanism and associations between features are the second mechanism to uncover unwanted choices leading to unusable configurations.

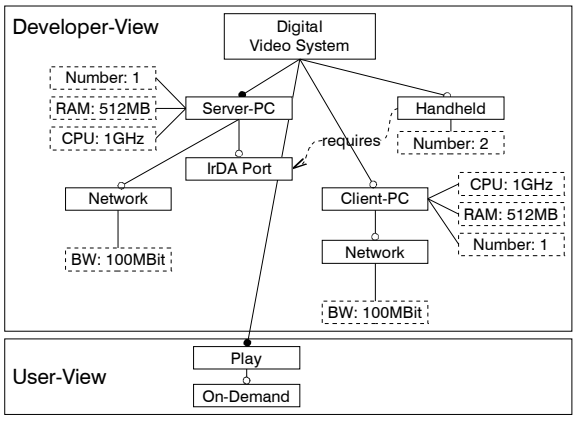

Figure 6: Feature Categories

In Figure 6, showing another subtree of the feature model of the digital video project, two subsets of features are shown. In the upper part, hardware components of the system are included. In the lower part, the on-demand server feature is shown. Without doubt the on-demand system with three clients would have specific requirements on the bandwidth of the network hardware to be functional. How can we express these technological constraints?

Czarnecki introduced the notion of sets of features [10]. Using the optional, mandatory and alternative criterion for features, it is possible to define subsets with constraints for the minimum and maximum number of features to be taken out of this set. This notation was ambiguous, what was fixed in an earlier paper [11] of our research group, proposing a new notation.

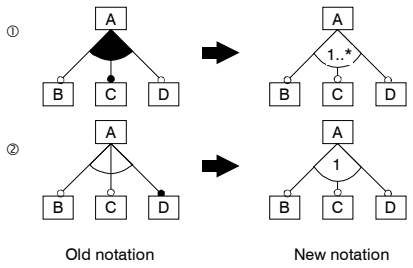

Figure 7: Multiplicities for Features

For a brief explanation take a look at Figure 7. On the left side we have the notation with the ambiguities. The intended meaning of the feature tree marked with ① is the choice of feature B, C or D. All features are related to each other by the logical OR-operator. What does the mandatory C feature mean? Do we have to choose C in any case? This would, without doubt, totally contradict to the logical OR. Within the feature tree marked with ② we have a similar situation. In this case the features are connected with the logical XOR. What does the mandatory D mean? Is it just the feature D, that we can select?

The new notation, shown on the right side of Figure 7, has absolutely clear semantics. Whenever we define a subset of features, we declare all of them optional. To define the possible selections of features out of the set, we use the multiplicity notation. The first number tells us how many of the features we have to choose at minimum, the second number tells us the maximum number of features we can choose. In addition, Czarnecki also introduced parameters for features, which are used to represent simple values. This concept was heavily used in the Digital Video Project. The notation was slightly changed, since it is hard to identify the parameter features in large models.

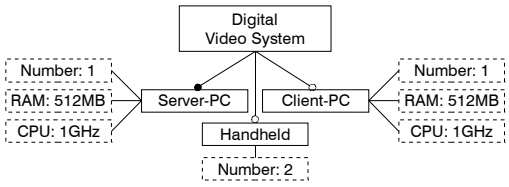

Figure 8: Parameter Features

Figure 8 shows some of the hardware features. In any case, the system has a server – the main computer system.

This feature has parameters representing the amount of RAM in the system and the CPU speed. The first parameter of the 'Server-PC' feature refers to the number of such server systems, which is set to one. The optional client system comes with the same parameters. In the middle there is an optional handheld device, which will be used as an advanced remote control via the IrDA port of the PC. In the above sketched example the complete system would have two handheld devices, what is correct as long as the customer has chosen to buy a client PC. There is an interdependency between "ServerPCs", "ClientPCs" and "Handhelds". For our system the number of handhelds is required to be the sum of the number of server and client PCs. The "Number" features in Figure 8 interrelate with each other, what currently cannot be expressed.

As described in [12], the use of artificial intelligence to describe feature interactions, as relations between features, is proposed. Unfortunately no detailed information about the exact notation is given. Other approaches as [13] are using colored Petri nets to model and manage feature interactions. This project is in an early stage and currently the method is capable of addressing the user interface of the system family.

3.1 Conclusion

The key idea of family development is the derivation of many applications out of family model. The feature model capturing commonalities and variabilities of the family is used as a bridge between the requirements model and the design model. Current research works in the feature modeling domain don't have a clear definition of the relations between features and don't supply the relations of the feature model to the other software engineering models of the family. We are hardly able to check the consistency of the feature model with the currently available modeling technologies and methods.

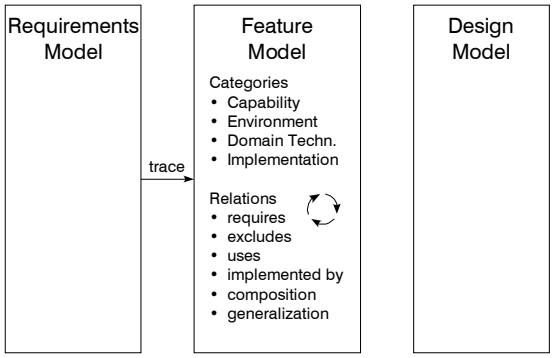

Figure 9: Overview of Current Feature Relations

The current situation is shown in Figure 9. There is a simple trace from the requirements model to the feature model and there are several relations within the feature model. Different categories help to elaborate features, but they cannot support the precise usage of relations for modeling and checking the consistency of the model. In addition, the more relation types we have, the more a developer gets confused, when to use which type.

The problems stated in the last section are boiling down to the following problem statements:

- Feature relations are defined in many flavors, but have no unified and formal background definition.
- Categories of relations are defined to help the developer elaborating features. A distinction between internal feature relations and relations to external development assets has to be defined.
- The definition of the term feature is still very vaguely defined.

In this paper we address the first two problem statements and propose a new way to look at feature relations and a way to describe feature relations, so they can be automatically evaluated and checked for consistency.

4 Defined Relations in Feature Models

To get an overview, feature relations need to be put into the software development context. As stated in the first section, features are located between the requirements model and the design model. They are describing the system family at a higher level than requirements and are thus an abstraction. On the one hand we wanted to simplify feature relations and on the other hand we would like them to be described in a way we can automatically check the feature model for consistency. As the result of the video project we propose a single and simple general relation described with a slightly adapted Object Constraint Language (OCL).

4.1 Feature Relations using OCL

OCL [14] is used to express constraints within UML models to get to a complete specification, since not all requirements can be described with UML models. OCL expressions are without any side effects since they cannot make changes to the model. OCL describes pre- and postconditions as well as invariants, what is stated in the corresponding context variable of the OCL expression.

To describe a constraint we need to define an OCL-construct with a fixed form, shown in CodeSection 1. Within the "context" we include the element affected by the constraint. The "inv" section contains an invariant, which needs to be satisfied all the time, otherwise the constraint is broken and the developer needs to check the corresponding model element. The "pre" and "post" sections contain pre- and postconditions for class methods.

```
context <elements>
inv: <OCL-Expr>
pre: <OCL-Expr>
post: <OCL-Expr>
```

CodeSection 1: Basic OCL construct

Since features are not part of the UML yet, we adapted the OCL to meet our needs for consistency checking in feature models. For class diagrams the OCL defines access functions to read the model elements. For feature models we need the same functionality, what is only possible to obtain with an extended version of the OCL. In the following paragraphs we call the adapted OCL, A-OCL. The new constructs are described in Table 1. For the simplicity of this paper we also use the info(<text>) construct, to give the user some informational text. In the following examples this construct is used as if it were part of A-OCL. In a real implementation it would rather be part of a scripting environment, which a developer can use to manipulate the model. OCL itself won't make any changes to the model.

construct	explanation
selected(<feature>)	True, if feature is selected.
value(<feature>,<type>)	Feature is a parameter and the return value is of the required OCL-type.

Table 1: Adapted OCL (A-OCL)

In addition to the new constructs we only use the "inv" section. Invariants are checked whenever one of the attached features is changed in any way. Invariant checking is continued for further involved constraints until no more features are reachable over constraint relations.

4.2 Feature Relations explained

One of the basic ideas of feature modeling is to supply a view onto the system for the customer, who is willing to buy a personalized system. The more features a customer chooses, the higher will the price of system be. With a higher price a company might even be able to rise their profit. A tradeoff between the fully loaded and very expensive system and the minimal system suitable for the customer has to be made. Within the video project we used "hints" to capture information, that is needed to support the customer while he is in the decision making process. As shown in Figure 10 the video system has several options of media it can play. If a customer should choose to have the option to watch slides with the system, a hint to the audio feature below "Edit" is very useful, since a slide show with background music or even a narrated slide show is more impressive than a normal one. This information can be either directly displayed, if the user should have an automatic product configurator or it will be shown to the sales representative, so he could point the customer to the additional feature. The corresponding A-OCL construct is shown in CodeSection 2.

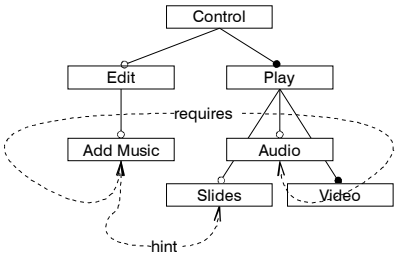

Figure 10: Features with a "hint" Relation

```
context Slides
inv hint:
    info("Check Add Music as well!")
```

CodeSection 2: "hint" OCL construct

For the parameter features we used a mathematical relation holding the formula to calculate the corresponding values. In Figure 11 the customer can choose to have server and client PCs and he can choose to control the system with handhelds. For our system we need to have as many handheld remote controls as we have PCs. Thus, the formula comes to: (number of handhelds) = (number of server PCs) + (number of client PCs). Within the relation this formula will be stored and the formula can be checked whenever the user changes the parameter features. The corresponding A-OCL construct is shown in CodeSection 3.

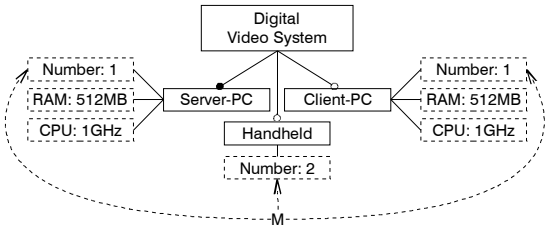

Figure 11: Features with a "mathematical " Relation

```
context Server-PC:Number,
        Client-PC:Number,
        Handheld:Number
inv mathematical:
     Handheld:Number =
     Server-PC:Number + Server-PC:Number
```

CodeSection 3: "mathematical" OCL construct

The same mathematical relation was used to model relation between parameter features and simple features in different views, shown in Figure 12. We have a relation between the number of client PCs and the network bandwidth of the server PC. The more client PCs a system has, the more bandwidth we need. Of course this can only be a

constraint if we want to have the "On-demand" feature for our system. Here the relation is expressed using a conditional expression to check whether the "On-Demand" feature is selected or not. The corresponding A-OCL construct is shown in CodeSection 4.

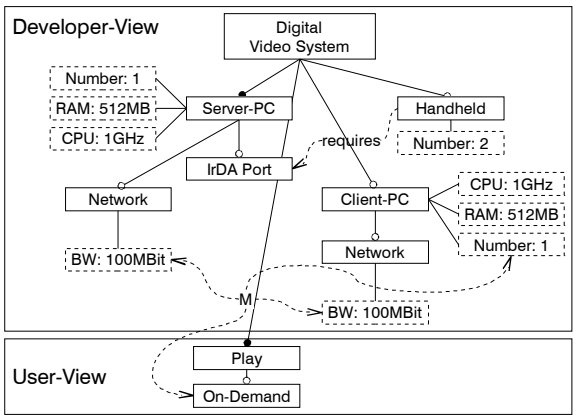

Figure 12: Relation Between Categories

```
context Network:BW,
        Client-PC:Number,
        On-Demand
inv mathematical:
     if selected(On-Demand) then
       Network:BW = Client-PC:Number * 20
     endif
```

CodeSection 4: "mathematical" OCL construct

The "requires" relationship, already defined in FODA is also expressed using A-OCL. Figure 13 shows the example, who's A-OCL expression for the "requires" relationship is shown in CodeSection 5.

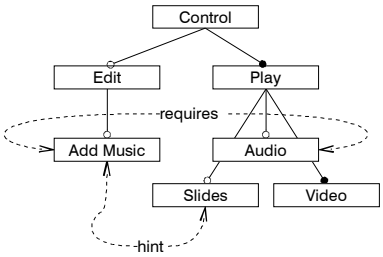

Figure 13: Features with a "requires" Relation

```
context 'Add Music',
         Audio
inv requires:
(if not(selected(Audio)
        implies selected('Add Music'))
 then
   info("Select 'Add Music'")
 endif)
and
(if not(selected('Add Music')
        implies selected(Audio)
 then
   info("Select Audio")
 endif)
```

CodeSection 5: "requires" OCL construct

```
context F1, F2
inv excludes:
   (selected(F_1) implies not selected(F_2))
   and
   (selected(F_2) implies not selected(F_1))
```

CodeSection 6: "excludes" OCL construct

In addition to the relations used in the video project we are also able to formulate A-OCL constraints for the "excludes", and "uses" relations in CodeSection 6 and CodeSection 7.

```
context F_1, F_2
post uses:
        (selected(F_1) implies selected(F_2))
        implies

context F_1, F_2
post uses:
        selected(F_2) implies selected(F_2)
```

CodeSection 7: "uses" OCL construct

4.3 Benefits of our approach

We propose only a single graphical notation for relations between features, which is easy to read and easy to understand. Relations are further specified by an A-OCL construct, which is a precise and formal way to describe feature relations.

The new "hint" relation of use for the marketing of the product. With this relation we are able to support the configurations process of the customer and we can point to useful and additional features. Thus, the customer is more satisfied and the company is able to increase their profit. The other new "mathematical" relation enables us to make use of parameter features. Up to now, parameter features could be defined, but their values or the selection of values could not be verified with rules given in the requirements model. Now it is possible to integrate these requirements into the feature model to get to a more complete and consistent model.

Together with the extension for multiplicities the feature model is simpler and thus easier to understand. The adapted OCL enables developers to define their own relations in case the predefined ones are not sufficient. Thus, consistency checking is possible and easy to obtain.

5 Summary and Outlook

In this paper we proposed a new way to express feature relations in feature models. We use an adapted form of OCL to formally describe the relations. With a single notation we can express all kinds of relations.

This position paper presented an ongoing research out of the domain of system family development. Engineering software artifacts with commonality and variability issues in mind results in a flexible design out of which several system variants can be generated. Based on existing approaches relations were presented to master the feature modeling task. The goal is to get to a constraint feature model as the interface between the requirements engineering and the design phase of system family development.

This research is embedded in a project, aiming towards a complete development method for system families. The work is currently validated within a university project.

Currently all data is represented as XML and constraints will also be integrated into this data model. For this, the XML constraint description in [15] will be evaluated and integrated into out model.

Future work will be put into the improvement of current prototypes and the integration of the adapted OCL into the prototypes for an automated consistency checking of constraint feature models. Furthermore, a seamless integration of the prototypes with other CASE-tools is needed.

References

[1] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, "Feature Oriented Design Analysis (FODA) Feasibility Study",CMU/SEI-90-TR-21 ESD-90-TR-222, 1990.

[2] Klaus Schmidinger, "vdr Projekt Homepage", http://www.cadsoft.de/people/kls/vdr, 2002.

[3] Simon Latapie et al., "Video Streaming Project", http://www.videolan.org/, 2002.

[4] Ralph Metzler, Marcus Metzler, "linuxTV.org", http://linuxtv.org/, 2002.

[5] OMG, "Unified Modeling Language Specification",OMG, 1999.

[6] Martin L. Griss, John Favaro, Massimo d' Alessandro, "Integrating Feature Modeling with RSEB",Hewlett Packard, 1998.

[7] Ivar Jacobson, Martin Griss, Patrik Jonsson, "Software Reuse: Architecture, Process and Organization for Business Success", Addison-Wesley, 1997

[8] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, Moonhang Huh, "FORM: A Feature-Oriented Reuse Method",Pohang University of Science and Technology, 1998.

[9] Kyo C. Kang, Kwanwoo Lee, Jaejoon Lee, "Feature Oriented Product Line Software Engineering: Principles and Guidelines",Pohang University of Science and Technology, 2002.

[10] Krzysztof Czarnecki, Ulrich Eisenecker, "Generative Programming: Methods, Tools, and Applications", Addison-Wesley, 2000

[11] Matthias Riebisch, Kai Böllert, Detlef Streitferdt, "Extending Feature Diagrams with UML Multiplicities", 6th Conference on Integrated Design & Process Tech IDPT 2002, 2002.

[12] Andreas Hein, John MacGregor, Steffen Thiel, "Configuring Software Product Line Features",Springer, 2001.

[13] Louise Lorentsen, Antti-Pekka Tuovinen, Jianli Xu, "Modelling Feature Interactions in Mobile Phones", Springer, 2001.

[14] Sinan Si Alhir, "UML in a Nutshell", O'Reilly, 1998

[15] Zisman A., Emmerich W., Finkelstein A., "Using XML to Build Consistency Rules for Distributed Specifications", International Workshop on Software Specification, 2000.