Configuring Embedded System Families Using Feature Models

Detlef Streitferdt, Periklis Sochos, Christian Heller, Ilka Philippow

Softwaresystems / Processinformatics
Technical University of Ilmenau
Helmholzplatz 1
98693 Ilmenau
{detlef.streitferdt | periklis.sochos | christian.heller | ilka.philippow}@tu-ilmenau.de

Abstract: A well planned reuse is a precondition to fulfil short release times and high quality demands for an embedded system development. System families have become an accepted method addressing exactly these problems. In this paper we present an extension of the Family-Oriented Requirements Engineering method out of [1] and [2] towards the development of embedded systems. We present a five step process that uses FORE to integrate models originating from the hardware, the hardware abstraction, and the application level. With our digital video example used in several student projects we show the applicability of the method. Further research efforts will be put into the completion of the method towards the design and implementation phase.

1 Introduction

The demand for short release times with a high product quality has lead to prefabricated components used within embedded system families to ensure a planned reuse. This has to be considered in all phases of the system family development. Especially the configuration of the final system with its hard- and software modules is a vital part of the development. Requirements originating from the underlying hardware, design decisions and technological constraints imply changes to the configuration of the family. The challenge of keeping the configuration information in a consistent state while developing the system family is only supported with an extended version of feature models.

In this paper we point out the importance of feature models for the task of configuring an embedded system family. We present an extension of our Family-Oriented Requirements Engineering (FORE) development method towards embedded systems. After an overview of the FORE method we use a digital video system as an example, to show and explain our new extension.

Within the state-of-the-art section of this paper we give a short introduction to feature models and their integration in our family-oriented development method. In the following second section we present the FORE extension using an example development cycle with our prototypical implementation of an embedded system.

2 State-of-the-Art

A way to describe and model variabilities of system families by means of features was initially described in FODA (Feature-Oriented Domain Analysis) [3] and further developed in [4], [5] and [6]. Features describe the system family for a future user, so that he can choose an own application based on the features of the family. Features should describe an important, distinguishable, user visible aspect, quality or characteristic of a software system or system. Based on own experience and analyses made by [4] and [6], features are very well suited for users or customers, to understand the system quickly and thus make a sound choice of their desired system based on the features offered.

Throughout this paper we will use a prototypically developed embedded system, a digital video system family. On top of a Linux system we used vdr [7] as extensible, plug-in based software platform. A small subset of the features of the digital video system family are shown in Figure 1. Features are hierarchically organized starting with the concept node at the root of the tree. All leaves with only mandatory markers up to the root of the tree belong to the core of the system family. In Figure 1 these features are "MPEG-2" and "Video". The other features form the variabilities of the family. In this example one could choose to buy the "MP3" feature to play audio files. With the "requires" and "excludes" constraints we can further limit the possible choices of a set of features in the tree. For a given feature we can state, that another feature is required or must be excluded, as this is modeled for the "Parental Control" feature, which simply locks the remote control and thus would be useless with the possibility of a direct control panel at the hardware system.

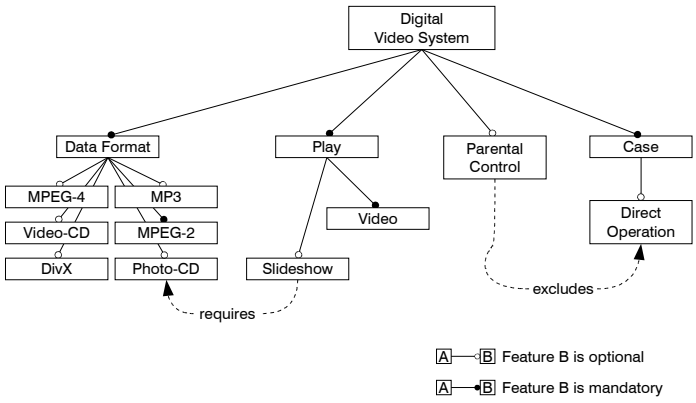

Figure 1: Feature Model of a Digital Video System Family

Out of all features a customer can choose a subset which will suffice his needs. In case the feature selection is valid it forms a new member of the system family, but the validity of a feature model can only be checked with a formalized model. In the next section such a formalized model will be shortly introduced.

2.1Extended Feature Models

Feature models have to be extended to reflect different kinds of relations between features. Such relations arise from technological hard- and software constraints. In the best case these constraints can be elicited in the requirements engineering phase and have to be propagated to the feature model. Other constraints are elaborated within the design phase or may originate from implementation specific details.

In [8] and [2] feature models are extended by constraints that can be processed in an automated way. Over 20 new and pre-defined constraints are available for a feature developer to be used with the corresponding data model. As shown in Figure 2, constraints can be defined between several features using our Feature Constraint Language (FCL) [8]. As an example, mathematical constraints "m_1" and "m_2" are shown. We process digital TV signals with our video system, referred to the "DVBCard" feature.

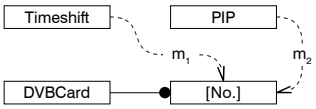

Figure 2: Extended Feature Models

As mandatory parameter feature the number of DVB cards is required. By the time the "Timeshift" feature is chosen, the constraint "m_1" has to be true. The "Timeshift" feature allows the user to interrupt current TV shows at any time. For the time of the interruption the systems stores the TV show on the hard disc. As soon as the user wants to continue watching the TV show, the stored data is presented. Because of hardware constrains we need at least two DVB cards to accomplish this task. With the first DVB card we receive the TV signal and store the data, with the second DVB card we decode and display the previously stored data stream. Thus, "m_2" in Figure 2 also requires at least two DVB cards, referred by the parameter feature attached to "DVBCard". The same constraint was defined for the "Picture In Picture" (PIP) feature. Here, the user can watch two TV channels at the same time while one channel is displayed as small picture inside the second channel, that is shown in full size. Again, at least two DVB cards are needed to watch two different channels at the same time.

Constraints are defined in a language similar to the Object Constraint Language (OCL) [9]. Thus, the feature model can be checked in an automated way for consistency and each feature set can be validated against the constraint rules. As a result, we can proceed in the development process with valid feature selections, that are variants of the family.

In [10] and [11] a Product Configuration Modeling Language (PCML) and the corresponding tools are described. With PCML configurable products can be modeled and single variants of this product can be derived. In contrast to PCML we use an extended feature model to hold the constraints limiting the number of variants in the family. Thus, we don't need a translation of features to PCML. The inference engine described in [11], supporting the users configuration task is not yet part of our approach, but will be considered in further research efforts.

An Eclipse [12] plug-in for feature modeling can be found at [13]. With this plug-in features can be modeled hierarchically together with constraints, that can be freely defined as Prolog statements. This feature modeling tool is a good choice for the inclusion in a tool chain, to establish and develop an embedded system family.

2.2Features Within Family-Oriented Development Methods

All system family development methods model the commonalities and variabilities of the family in question. Family-Oriented Abstraction, Specification, and Translation (FAST) [14] developed by Lucent Technologies uses scenarios with variability points, to distinguish between common and variable parts. In addition, a decision model guides through the creation process of an application out of the family. The goal of FAST is to develop an own language for each domain, with which family members will be "coded". Together with a tool chain for the language a FAST family would be complete. Variabilities are captured in textual documents and decision models. For customers without a sound technical background it is hard to understand the variabilities of the family. Here, the benefits of feature models are missing.

The component-based application development (KobrA) [15] offers three processes to develop a system family. Within the *Context Realisation* a business model of the family will be developed. The second phase, the *Framework Engineering*, leads to a reference architecture of the family to be used for all family members. In the third phase, KobrA components will be developed, so they can be used within the previously developed framework. KobrA uses a decision model to capture commonalities and variabilities of the family, just like FAST. Decisions will be resolved using typed parameters. Relations between decisions are part of the model but cannot be automatically processed. Again, non-technical customers will have a steep learning curve to understand the family with its benefits and the developers of the family need to put extra effort into handling and maintaining the important decision model.

System family development methods, like FOPLE [5] or FeatuRSEB [16], use features to model the commonalities and variabilities of the family. They are perfect to communicate with customers. As stated in the last section, feature models are the most intuitive way to model commonalities and variabilities. For the complete integration of feature models in a development method they need to be processed in an automated way. Current methods won't offer an automated processing of their feature models. Thus, we integrated the positive sides of the system family development methods with the power of extended feature models into a new method.

3 Developing Embedded System Families

Our own Family-Oriented Requirements Engineering (FORE) Method [8], [2] is composed of three main building blocks. The requirements engineering phase for the system family, the data model holding all development assets and the requirements engineering phase for an application derived out of the family.

Within the requirement engineering phase for the system family we start with a tailoring

step where the document model is set up (tailored), to reflect the company specific structure of specification, user and developer documents. Gathering domain knowledge is of course not only ordering and reading books. The complete development team and its management need to perform an in depth study of the domain in question. Enough time has to be planned for this task to avoid very expensive misunderstandings later on. Legacy systems and possible future systems are then analyzed to elicit the requirements and features per system. We end up with a requirements and feature list for each system. Within the synchronization step all requirements and features are re-organized into a requirements and feature model, while keeping their relation to the originating system, which becomes a variant in the FORE data model. The scoping step is important for the strategic planing of the future development steps. Out of the complete feature model a subset has to be chosen to form the first version of the family. Nevertheless all features will be considered for the derivation of the first architecture version. Thus, a roadmap for the future development of the family can be created, without risking an architectural drift. After a review the commonality / variability analysis improves the feature model with additional relations between features, that may even originate out of the next technology selection step. We will discuss this in the next section. Based on the feature model and the chosen technology a first version of the family architecture can now be derived.

The six building blocks of the data model, as shown in Figure 3, start with the requirements model. Requirements are kept as simple textual sections, organized hierarchically with freely defineable relations between requirements. These relations are also formulated using our FCL and thus, they can be automatically processed. The stakeholder model is a "*Who is Who*" for the current family development. Requirements and later on features will be related to the stakeholders who initially formulated them, to keep the original sources of information in the data model. As already stated above, the document model is a structural description of the documents to be created during the development of the family. Relations of data model elements to the corresponding parts in a documentation are modeled. This structure together with a style description is then used to automatically generate the developer and user documentation, as well as the specification document. Although generated documents outdate very fast, in fact they may be outdated by the time they have been printed, they have to be printed for legal reasons and for better readability, since some stakeholders don't like reading documents on a computer monitor. A glossary is kept to improve the quality of the documentation. Readers can exchange terms on the fly or they can even "ask" the system to explain terms they don't understand. References are provided within a database accessible over a network, to keep the overhead of referencing as small as possible while writing documents. Additionally, all data elements should be traced to the relevant references, what improves the readability and comprehension of the model. As in FAST and KobrA, FORE also includes the concept of a decision model. FORE uses the decision model to guide a customer through the feature selection process. For each feature the possible decisions and assigned questions are kept in the model, so the system is able to guide through the feature model with "question-answer" pairs. For successfully validated and implemented feature sets a variant is stored in the system. The key concept of the data model is the Extended Feature Model (EFM) and is located between the requirements model and the UML design models.

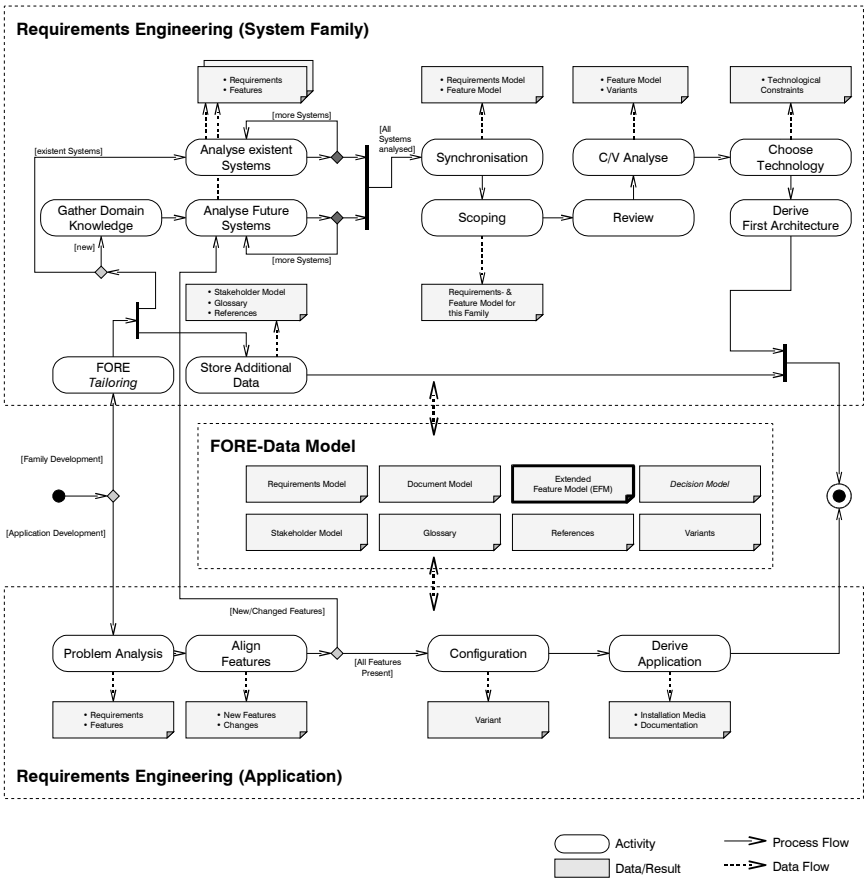

Figure 3: FORE Development Process

As depicted in Figure 4, all model elements can be related to each other. A set of requirements can relate to a feature out of the feature model and a feature can point to the relevant design elements. The relation of any element to any other element, namely traceability, is realized as own element in the FORE data model and is expressed in FCL. On the implementation side the model is realized as XML-Schema [17] and the relations are based on the XML Linking Language [18]. Together with the formal description of relations expressed in FCL an automated consistency checking and processing of the data model and the extended feature model is possible.

Finally, the requirements engineering phase for an application as member of the system family uses the assets developed and stored in the data model. While inside the problem analysis a customer will be questioned in a system family specific way. His requirements need to be “fit into” existing family features. At this stage a very important task is to describe the system family using the wording of the customer, so he understands what the family offers. It is important to note, that customer requirements can often be related to a configuration of features. As we will see in the next section, features relate to components, which are configurable plug-ins in our example. Thus, we can simply “adjust” the

components a feature relates to, to satisfy the customer needs. Within the alignment phase we check whether the customer raises requirements or features that are not present in the family. Based on our complete feature model we had (before we did the scoping), the strategic planing and financial issues, we decide whether the customer can be satisfied with an application out of our family or not. To configure an application all parameter features need to be set to correct values. These values will be checked using the constraints attached to the modeled relations between the parameter features. Finally, within the derivation step an application with the corresponding documentation will be created.

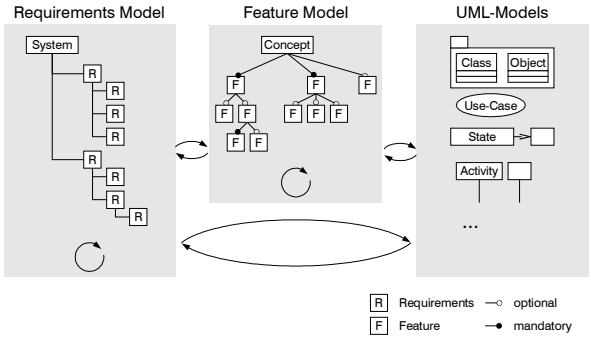

Figure 4: Integration of Feature Models

3.1Constraints of Embedded Systems within Feature Models

As an example we tested our method with an embedded digital video system. For this system the hardware and operating system requirements are strongly influenced the system family, particularly the feature model. To address these requirements specific to embedded systems the FORE method had to be applied in three iterations. Finally, we developed an embedded process extension for FORE.

1. Develop a family model for the domain. In our example we developed a model for the domain of digital video systems.
2. Develop a family model for the hardware components of the system family. We developed a feature model for the hardware of the digital video system, which consists of electronic as well as the mechanical parts.
3. Develop a family model for the operating system components. Since we use Linux we had to check all the relevant Kernel options to derive an optimized kernel for each member of our system family.
4. Interconnect these models to reflect all the relevant constraints between the features and the assigned hard- and software components.
5. Integrate selection criteria for equivalent components in the feature model.

Within the 1st step we developed a feature model based on freely available information about digital video systems and the software we used, SUSE Linux [19], vdr [7] and its plug-ins [20]. We elicited more than sixty top level requirements and developed more

than seventy features for a family model of general digital video systems.

For the 2nd step we checked the available hardware components, while keeping relevant requirements and features in mind. In Figure 5 a subset of this feature model is shown. In this example there are four complex relations that need to be checked whenever a feature choice is made. Relations “m1” and “m2” will be processed as soon as the type, size or weight of the system case is selected or changed. The size is given as length, height, width triple and the size ranges for our Desktop and Barebone cases are attached to the corresponding features. Should the user selectable size of the system contradict to the size of the selected case type, where at least one case type has to be selected, then “m2” will evaluate to false and the current feature selection would have do be adapted to satisfy “m2”. In an analogous way, the weight is checked by processing relation “m1”. Both relations, “m1” and “m2”, are internal relations since they connect features with each other. In contrast, “m3” is an external relation connecting a feature with two components of the design model. Some of the current hard drives offer internal registers with the actual temperature of the drive. In case the feature “Temp” is selected only hard drives with temperature control can be added to a new family member. In our example there are only two hard drives one with, the other one without temperature control. In case we could offer several drives with temperature control “m3” would be processed in the “Derive Application” step of FORE method. “m3” would then contain lists of hard drives sorted by different criteria, for example the price. The user will then be asked to give his opinion on the price of the hard disc. Bases on his decision the system can automatically choose a hard disc, we will discuss this in the next section. The last relation of this example is given by “m4”, which is external as well. Based on the desired amount of RAM a main board has to be chosen. And again, in case more than one main board would fit the constraint of the relation, a choice based on given criteria will be offered.

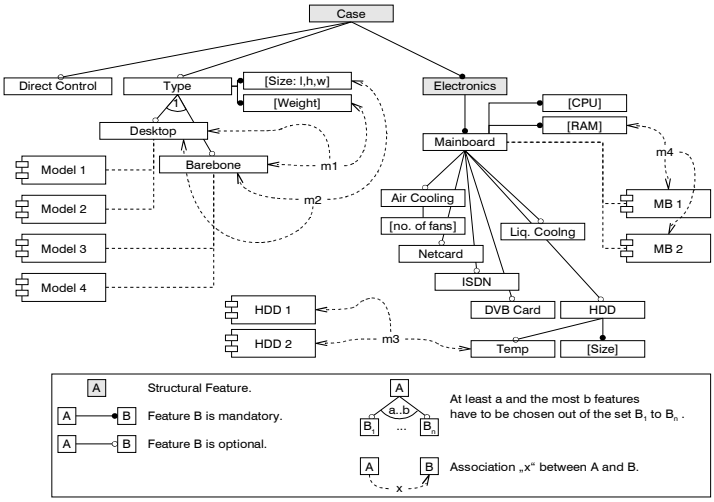

Figure 5: Feature Model for Hardware Components

The 3rd step is needed to complete our embedded family. We analyzed the kernel configuration possibilities and set up a kernel feature model. With this model the configuration for a kernel compilation can be set. Thus, an optimized kernel can be built for each application of the system family. The modular kernel concept of Linux allows us to build very small kernels and a set of modules. With this, the kernel size changes over time and occupies only the needed memory for the shortest possible time. Depicted in Figure 6 and starting from the left to the right, the example subset of the kernel feature model has the "Enhanced-BIOS" feature with which we ensure the proper operation of large hard discs and the installation of BIOS "wake-up" timers, to start the system even if it is turned off (but, of course it still has to be in "stand-by" mode!). The "Processor-type" features enables the use of specific processor extensions of the instructions set. As modeled, only one processor type can be selected. The power management of the CPU frequency differs between the processor types. Thus, the frequency features are internally related to the selected processor type. In general, none of the frequency power management features would have to be selected, see the 0..1 cardinality for the feature "CPU Freq.". In case something should be selected, exactly one feature can be selected and this feature automatically requires the correct processor type for the system. Finally, the ACPI features can be selected to get enhanced control over the fan speed and thus, the noise of the system. With the temperature sensor the system controls a shut-down in case of overheated components. Finally, the operation mode of the "shut-down" button for the system can be changed with the "Button" feature.

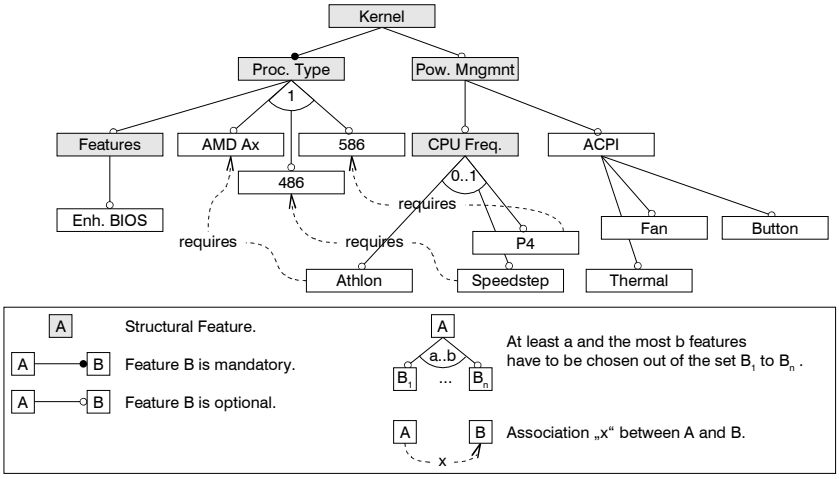

Figure 6: Kernel Feature Model

Within the 4th step the models developed above need to be interconnect to form a single embedded feature model. With this model we will be able to correctly derive a complete system. We are still in the requirements engineering phase and we are still modeling on a very abstract level. The key benefit is exactly this level of abstraction. With the models, we can reason about choices and requirements of the customer and we will get an answer about whether or not the customer can be satisfied with a given feature set. For a better

view in Figure 7 some of the relations and elements are left out, although they still belong to the model. The gray lines visualize the origin of the sub feature models.

The left part of the figure holds a small subset of the feature model originating from the 1st step, the feature model for the digital video system domain. The TV feature is for the bare watching capability of the system and the attached EPG feature offers an electronic program guide. For the recording feature the overall recording time has to be set as parameter. To schedule recording ahead of the actual TV show EPG+ is recommended. The "recommend" relation just points out interesting facts for the customer. Here, it may be useful for the customer to have EPG+ in case he plans to schedule recordings. He could choose a TV show using the extended EPG+ version and schedule this show with just a button.

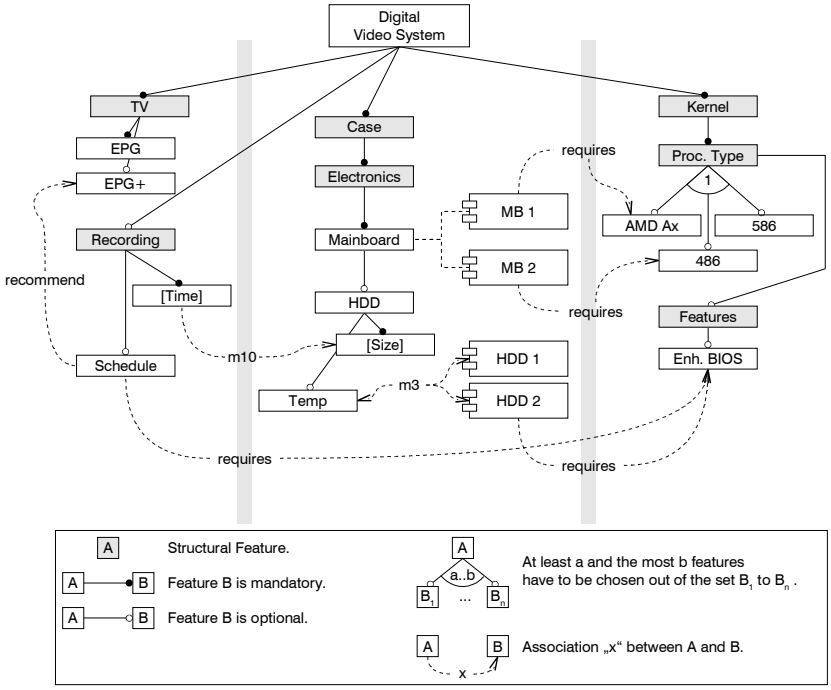

Figure 7: Complete Feature Model

The middle part of the figure originates from the 2nd step of the development method and the right part originates from the 3rd step. For this example four new "requires" relations and one mathematical relation had to be modeled. In case a customer would like to schedule TV show recordings, an enhanced BIOS is needed to turn on the machine to start recording. The enhanced BIOS is also needed for the second hard disc, due to its size. Old BIOSes would not recognize such large hard discs. The last two "requires" relations start at the main board components. Of course main boards are built for a specific processor type, what these two "requires" relations simply reflect.

Finally, there is the "m10" relation between the overall recording time and the size of the hard disc and is a simple mathematical relation. The more recording time a customer requests the more capacity a hard disc must have.

The 5th step is the elaboration of the criteria for the selection of equivalent components. We are using *Selector Components* to accomplish this task. These components contain the data relevant for the automated decision process. For the complete support of the component based decision process we developed an own component model extension for the FORE data model, described in [1].

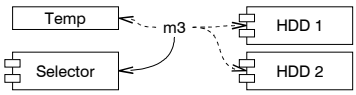

Figure 8: Selector Component

For the digital video example we connected the selector components to the relevant relations, which is "m3" in Figure 8. In our prototypical implementation we use Ant scripts [21] for the selector component. These scripts interactively question the customer within the Application Derivation phase using the corresponding criteria.

4 Summary and Outlook

In this paper a requirements engineering development method for embedded systems was presented. The method is based on FORE [8], [2] and was extended to be used with embedded systems. They key concept of FORE is the extended feature model, that is used to model the common and variable aspects of the family and at the same time it can be automatically validated.

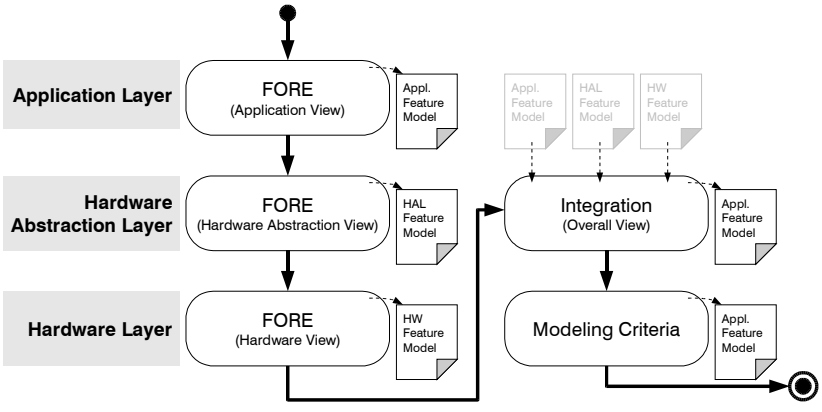

Figure 9: FORE Extension for Embedded System Development

The extension of the method presented in this paper is divided into five steps. Within the first three steps FORE will be used to elaborate and develop a feature model based on a specific view point of the underlying system. In the last two steps the previously developed models will be integrated into a single family model of the embedded system and the criteria for selecting equivalent components are then added.

The method extension shown in Figure 9 was tested with a digital video system. Given the results of several masters thesis in this area, the work is very promising and we plan to do further research in the embedded domain. Our goal is to extend our method towards a complete family development method by integrating design as well as implementation techniques.

References

1. Detlef Streitferdt, Christian Heller, Ilka Philippow: A component model for applications based on feature models. In Proceedings of the 2nd Groningen Workshop on Software Variability Management: Software Product Families and Populations, University of Groningen, (2004).
2. Detlef Streitferdt: Feature-Oriented Requirements Engineering, Dissertation at TU-Ilmenau, (2004).
3. Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson: Feature-Oriented Design Analysis (FODA) Feasibility Study. Report: CMU/SEI-90-TR-21 ESD-90-TR-222, www.sei.cmu.edu, (1990).
4. 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, www.postech.ac.kr/e/, (1998).
5. Kyo C. Kang, Kwanwoo Lee, Jaejoon Lee: Feature-Oriented Product Line Software Engineering: Principles and Guidelines. Pohang University of Science and Technology, www.postech.ac.kr/e/, (2002).
6. Krzysztof Czarnecki, Ulrich Eisenecker: Generative Programming: Methods, Tools, and Applications, Addison-Wesley, (2000).
7. Klaus Schmidinger: vdr Projekt Homepage, (2002), www.cadsoft.de/people/kls/vdr.
8. Detlef Streitferdt, Matthias Riebisch, Ilka Philippow: Details of Formalized Relations in Feature Models Using OCL. In Proceedings of the 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems, Huntsville Alabama, USA, (2003).
9. OMG: Unified Modeling Language Specification, (1999), www.omg.org.
10. M. Heiskala, A.Anderson, V.Huhtinen, J.Tiihonen, A.Martio: A Tool for Comparing Configurable Products. In Proc. of Workshop on Configuration of the 18th International Joint Conference on Artificial Iintelligence, (2003).
11. Asikainen, T., Männistö, T., Soininen, T.: Using a Configurator for Modelling and Configuring Software Product Lines Based on Feature Models. In Proceedings of Software Variability Management for Product Derivation - Towards Tool Support at International Workshop of SPLC, (2004).
12. Eclipse.org: Eclipse open extensible IDE, (2005), www.eclipse.org.
13. Danilo Beuche: Pure Variants - A Plug-In for Eclipse, (2005), www.pure-systems.com.
14. David M. Weiss, Chi Tau Robert Lai: Software Product-Line Engineering: A Family-Based Software Development Process, Addison-Wesley, (1999).
15. Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, Jörg Zettel: Component-Based Product Line Engineering with UML, Addison-Wesley, (2002).
16. Martin L. Griss, John Favaro, Massimo d' Alessandro: Integrating Feature Modeling with RSEB. Hewlett Packard, (1998).
17. W3C: Extensible Markup Language (XML) 1.0(Second Edition). , (2000).
18. W3C: XML Linking Language (XLink) Version 1.0. , (2000).
19. Novell: SUSE Linux Professional, (2005), www.novell.com/de-de/linux/suse/.
20. www.vdr-portal.de: VDR - Portal, (2005), www.vdr-portal.de.
21. The Apache Ant Project: Ant - A Java Build Tool, (2004), ant.apache.org.