SPLC 2008

Using Feature diagrams with Context Variability to model Multiple Product Lines for Software Supply Chains

Herman Hartmann, Tim Trew
NXP Semiconductors, Eindhoven, The Netherlands
herman.hartmann@nxp.com, tim.trew@nxp.com

Abstract

Feature modeling is a key concept in Product Line Engineering. Due to the large number of different features and relations in practice, feature models often become too complex to deal with. One of these complexities is caused by the need to support multiple product lines.

In this paper we introduce the concept of a Context Variability model, which contain the primary drivers for variation, e.g. different geographic regions. The context variability model constrains the feature model, which makes it possible to model multiple product lines supporting several dimensions in the context space.

We will discuss how this concept can support software supply chains. Here it can be used to facilitate the process of merging feature models during staged configuration.

Our initial experimental results indicate that the approach is intuitive and straightforward to use, and can be applied with commercially available variability management tools.

1. Introduction

In software product line engineering seeking the commonalities and managing the variability is one of the essences [1]. Feature modeling is considered as a means to describe the variability in a product. It is used for requirements engineering and for configuring a product. Feature modeling originates from the FODA method [2] and since then has been extended (e.g. [3,4,5]).

Feature models can become large, for instance in the automotive industry [6,7]. Adequate tools are important for an effective method deployment. Various variability management tools have found their way into the industry, many of them support feature modeling [8].

Reducing complexity in feature models has been a topic of research since their introduction. This complexity can be caused when different units of an organization are working together, amendments are made to feature models or because different levels of abstraction have to be maintained [9,10,11,12].

The need to support multiple product lines is described in [13]. A common example is that of two car types, a cabriolet and a station wagon [3,14,15]. A cabriolet cannot be equipped with a rear wiper, but for a station wagon a rear wiper is a mandatory feature. A cabriolet has a roof control, which can be manual or automatic. For a station wagon, a roof control is not available. These differences between the car types necessitate a separate feature model for each type. This causes redundancy and may lead to inconsistencies [14,15].

The reuse of software components between different product lines in the consumer electronics domain is described in [16]. In this domain, variability in requirements is driven by several different dimensions, e.g. different product types and different geographic regions [4,17].

In many industries Software Supply Chains will emerge to feed the demand for components [18]. Common components will become commodities, which are developed by specialist suppliers. Software supply chains could occur in a wide variety of domains.

A supply chain [19] is: “A network that starts with raw material, transforms them into intermediate goods, and then into final product delivered to customers. Each participant consumes logical and/or physical products from one or more upstream suppliers, adds value, usually by incorporating them into more complex products, and supplies the results to downstream consumer”.

For software this means that a company buys a set of components from (several) suppliers and integrates them into a product, possibly combined with its own software. For each of its customers, i.e. the next participants in the chain, a specialized version may be created. Each of the participants adds new functionality and makes configuration choices based on their role and responsibility in the supply chain.

Overview: The remainder of this paper is structured as follows: We will start with giving an overview of earlier work. We will then introduce the novel concept of a *Context Variability* model, which, combined with a Feature model, is able to model multiple product lines supporting several dimensions in the context space.

We will then describe how staged configurations can be applied to generate a specialized feature model. We will continue with describing how this concept facilitates the merging of feature models from different suppliers. We will then make a comparison with related work and we will end with a description of the results of a preliminary case study and describe topics for further research.

Although the semiconductor industry contains a huge variety of different features and provides a challenging case, we will use examples from the automotive industry to illustrate our approach. The example is described in such an order that it matches that of the players of a software supply chain.

2. Related work

2.1. Multiple product lines

Czarniecki and Eisenecker [3] suggest the use of "vertical" dependency rules between high level features and lower level features to deal with different feature models for different product types.

Buhne *et al.* [14] describe the problems related to the use of a single feature model and propose the use of a centralized variability model, which is later elaborated and described as the Orthogonal Variability Model [20]. In a later paper, Buhne *et al.* [21] discuss various approaches and describe a way to model different product lines as part of the Orthogonal Variability Model. In their approach a solution is presented for modeling variation across different product types (e.g. DVD, MP3 and Hard-disk-recorder).

Reiser and Weber [15] discuss several scenarios, including the use of a product tree with relations to a feature model. They came to the conclusion that this method is not scalable because it would lead to a lot of redundancy for large product families. Also the loss of the rationale for selection criteria is discussed. Reiser and Weber suggest the use of a product set to describe the constraints for different products and the tool support that is essential to make it practical.

2.2. Staged development and Software Supply Chains

Pohl *et al.* [20] described that there are different stakeholders that decide on the variability of a product and Czarniecki [5] used a similar notion of different parties and roles that eliminate variabilities.

Czarniecki *et al.* [4] describe that a process of specifying family members may be performed in several stages, where each stage eliminates certain configuration choices. A transformation process is described that supports the specialization for refinement of feature models in different stages of development. In [5] this approach is extended to support software supply chains. The need for merging feature models, due to software supply chains, is also addressed here. In this paper the use of multi-level configuration is described to represent feature descriptions at different levels and use this for different stages. They suggest using a level-0 feature model describing the different market segments or geographic regions as a high level requirement.

3. Feature diagrams for multiple product lines

3.1. Context Variability model

In this paper we will introduce the novel concept of *Context Variability*. Context variability is the variability of the environment in which a product resides.

A Context Variability Model consists of general classifiers of the context in which the product is used. A general classifier stands for a set of requirements or constraints for that context. The Context Variability model captures the commonality and variability of the context.

Examples: A general classifier of the context could be a geographic region. For a mobile phone the requirements for the European continent is to support for the 1800MHz band (a technical requirement) and SMS messaging (a commercial requirement). Both 1800MHz and SMS messaging are requirements that hold for *all* mobile phones in Europe. Another classifier of the context could be the type of user. Is it a type of user that likes all kinds of gadgets (also known as technology enthusiasts) or is it an elderly person that is reluctant to use new technology. Another example of classifier can be the type of product in which the software is used, e.g. a DVD-player, MP3 player or hard-disk-recorder.

The Context Variability can be represented with the commonly used FODA diagrams, although the context

classifiers are not features. It may include sub-"features" (meaning sub-classifiers), cardinalities and dependencies between classifiers.

For cardinality or group cardinality the minimum value is always 0 (e.g., [0..x]). In other words, a context variation point cannot be mandatory.

3.2. Multiple-Product-Line-Feature Model

The Context Variability Model is combined with a conventional feature model to create a new model, which will be called the *Multiple Product Line Feature Model*, or for short, the *MPL-Feature model*.

The feature model is used in the classical way, and describes the commonality and variability for a product range and the dependencies between different features.

By making this separation, a feature model contains the possible features, with dependencies that capture the (technical) dependencies between the features.

The dependencies related to the context are modeled as dependencies between the Context Variability model and the feature model, as shown in figure 1. These dependencies are captured in the same way as dependencies within a conventional feature model.

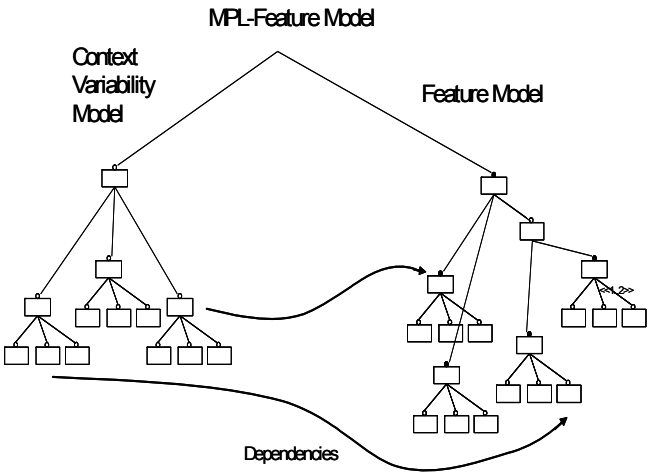

Fig. 1. MPL-Feature Diagram

The types of dependencies between the Context Variability model and the Feature Model include "requires" and "excludes", as is common with conventional Feature models. Besides these dependency relations we will introduce a new relation. This type of relation narrows the cardinality of either a feature or a group, as will be illustrated in section 3.3. Furthermore, each dependency can be annotated with a rationale for the constraint imposed by the context. It is important that the rationale is captured, e.g. to assess the impact of requirements changes. For instance a purely "commercial" rationale can always be changed, but is valuable for a software supplier to record. In contrast, it may not be possible to change a dependency with "technical" rationale, if it were a property of the domain, or it might have a major impact on the product line architecture.

Because of the size of the Feature Model for realistic product lines and the complexity of the constraints, it is therefore necessary to create a model, rather than relying on static diagrams, supported by appropriate tooling. Consequently, we will represent dependencies in textual form, and display filtered graphical views.

3.3. Example

To illustrate our concept we will use a fictitious example. Imagine a manufacturer of car infotainment systems, a tier-1 supplier in the automotive industry.

Features: The software of the infotainment system consists of several optional features for information and entertainment purposes. For instance: a navigation system, a radio, MP3 player, DVD player, interface to external MP3 player dock, a USB connector and memory card slot, Bluetooth connections, etc. Several standards are support such as various DVD-regions, broadcasting and communication protocols (FlexRay, ZigBee and Bluetooth). Figure 2 shows a subset of the infotainment system, using the FODA notation.

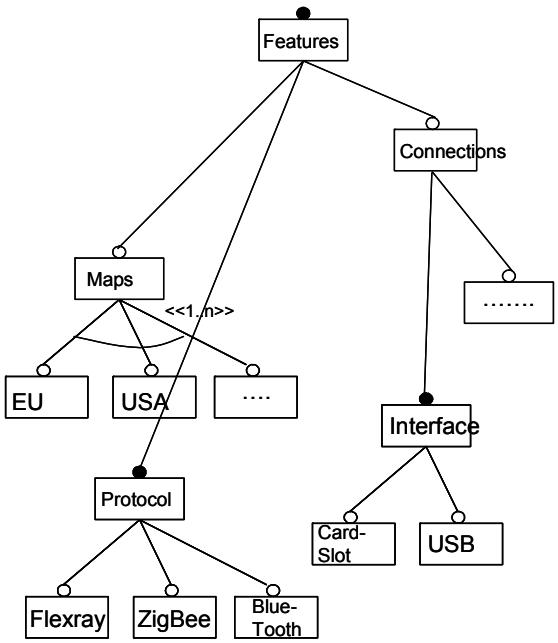

Fig. 2. Feature Diagram infotainment system

Context: The infotainment manufacturer creates products for different Car manufacturers: CarA, CarB, CarC and CarD, for different continents: USA, Asia and Europe, and has three product types: budget, mid-range, high-end. The budget, with a small feature set, fits into the space used for ordinary car radios. The mid-range type contains more features, is larger and therefore needs more cabin space. The high-end contains many possible features, is significantly larger then the mid-range, so needs space in the trunk.

A diagram of the Context Variability model is shown in figure 3.

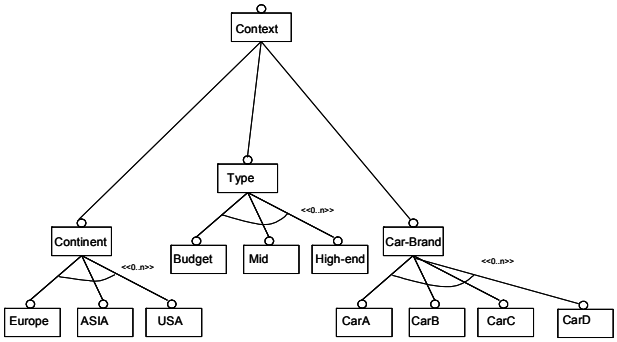

Fig. 3. Context Variability Diagram of the infotainment system

The dependencies between the context variability model and the feature model represent the specific requirement of each classifier. Some examples of dependencies and their rationales are presented below, using a textual notation.

European Maps are part of the Navigation system and in Europe DVDs with region 2 must be supported:

```
Continent.Europe <<requires>>
European Maps {Rationale: Obvious}
Continent.Europe <<requires>> DVD-
region 2 {Rationale: Standard}
```

Manufacturer CarA may not support a specific communication protocol:

```
Car-Brand.CarA <<excludes>> FlexRay
protocol
```

The budget (small) configuration can only support a USB interface *or* a Card Slot, not both, because the size of the front panel is too small.

```
Type.Budget <<sets group cardinality
of >> MP3 interface >> << to>> [0..1]
{Rationale: Technical; Too small size
of front}
```

Dependencies can also relate to different context classifiers: In the USA, CarA does not sell budget:

```
Continent.USA AND car type.CarA
<<exclude>> budget {Rationale:
Logistics}
```

Or become even more complex: In the Asian market, CarB only sells midrange without Memory-Card slot:

```
Car Type.CarB AND Continent.Asia AND
Type. Mid-range <<exclude>> Memory-
Card. {Rationale: Commercial}
```

Figure 4 shows a diagram of the MPL-Feature model with a graphical representation of the dependencies between the Context Model and the Feature Model (for reasons of clarity only a subset of the dependencies are shown).

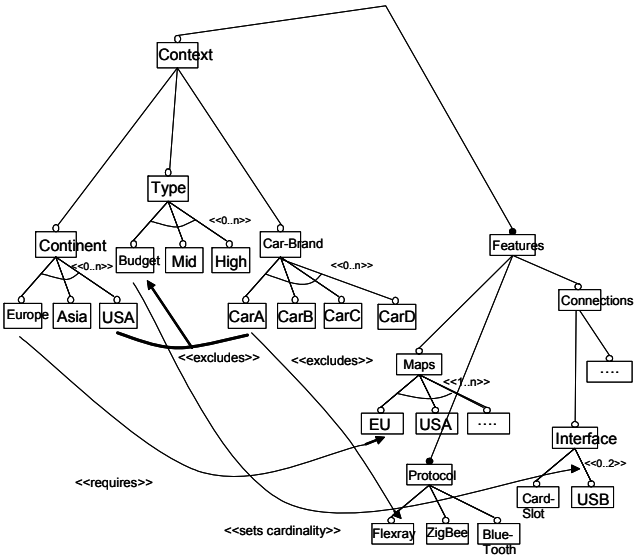

Fig. 4. MPL-Feature Diagram of the infotainment system

3.4. Extendibility and Scalability

In theory, all kinds of requirements can be captured. The car infotainment manufacturer could use this type of information, for instance, to exclude this combination from the test set or to analyze the commonality and variability in their product families.

However, some of the dependencies shown in the example of section 3.3 are more likely to be maintained and used by the car manufacturers, i.e. the downstream participant in the supply chain. It is their

"role" in the supply chain to capture this and use it to generate different car types with an integrated infotainment system. In other words, the dependency relations "belong" to the participant of the supply chain that is responsible for it.

In this way the MPL-feature models are reduced in size and complexity for each participant in the supply chain. This will be explained in section 4.

3.5. Sub-classifiers as part of the context variability model

A sub "feature" as part of the context variability model can be used to capture specific requirements for this sub classifier.

In our example we could use sub-classifiers for the continent Europe, such as The Netherlands, UK and Germany, which would capture specific requirements for these countries. In addition to this one could add sub-classifiers describing the use of the Euro as standard monetary unit, which holds for some countries of the EU.

The dependency rules that are part of the sub-classifiers, i.e. the children, "overrule" that of their parents. In this way it is avoided that one ends up with many context classifiers, each representing a small set of requirements.

All kinds of groupings and sub classifiers are possible to capture variability and commonality in the context space.

4. Context Variability in Software Supply Chains

4.1. Requirements from Software Supply Chains

In software supply chains, each of the participants creates software components that are specialized for their customers, i.e. the downstream participant. The customer integrates the components of the supplier into their own products and generates specialized products for their customers, i.e. the next downstream participant

The supplier has to be able to capture the requirements for each customer to be able to generate a specialized product for that customer. In order to do this efficiently, the software supplier has to create a product family, or a set of reusable artifacts, and has to maintain the commonality and diversity of the different contexts, such as the different product types and customers.

In this section we will describe how specialized feature models can be generated from the MPL-Feature model. We will apply the concepts of specialization and configuration that have been described by Czarnecki *et al.* [4, 5]. In the next section we will discuss the subject of merging feature models.

4.2. Generating a specialized feature model

The concept of specialization and configuration can be applied to the feature model as well as the Context Variability model.

A selection in the Context Variability model, followed by a transformation of the MPL-Feature model, leads to a more specialized MPL-Feature model with fewer options.

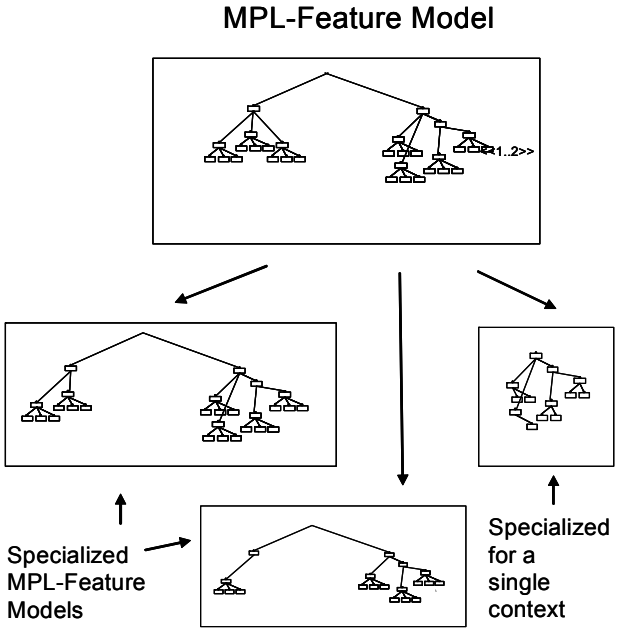

Fig. 5. Generating specialized Feature Models

A fully specialized Context Variability model results in a feature model for a single context i.e. that contains only those variation points which are available for that context. A partly configured Context Variability model results in an MPL-Feature model for a reduced set of product lines, with still one or more dimensions of freedom in the context space. See figure 5.

The decision on what and how far to specialize a feature model depends on what variability will be left to the customer. Some of the selections have to be made by the customer, like described in section 3.4. Some other choices will be for the supplier, such as for which customer the product will be and the dependencies related to the hardware that will be delivered to that customer.

4.3. Example

Figure 6 shows a diagram of the specialized feature model for budget product for CarA of the car infotainment system that is obtained by selecting the budget and the CarA classifiers in the Context Variability model, followed by a transformation. Because of the dependency relations, FlexRay is no longer part of the features and the MP3 Interface has a group cardinality of [0..1].

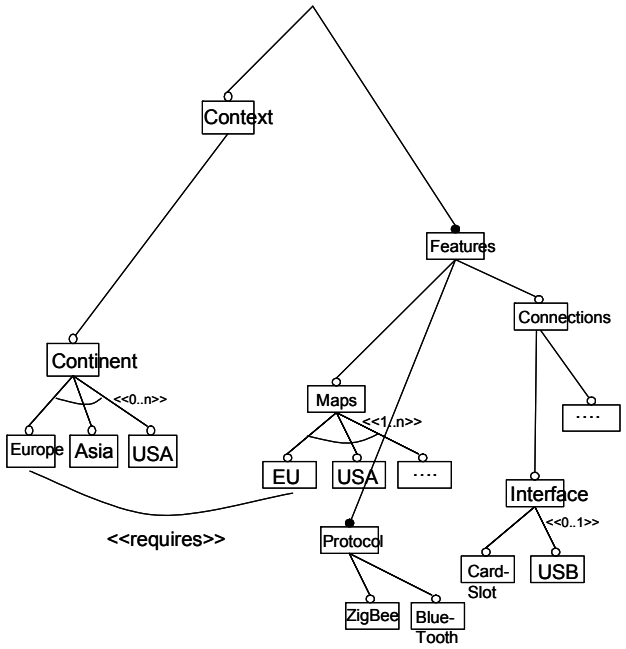

Fig. 6. MPL-Feature Diagram for CarA budget

This MPL-Feature model can be supplied to CarA, which can select the region and select specific features to be included.

4.4. Specializing group cardinality in the Context Variability model

As described in section 3.2 it is possible to use group cardinality in the Context Variability model. The following example shows the value of this in practice.

We could have added a group cardinality of [0..3] in our example for the different continents instead of the current [0..1]. We then could have configured Europe and Asia and thus would have obtained a specialized feature model that represents the features for the infotainment system that is suitable for both Europe and Asia.

4.5. Extensions through the development lifecycle

More branches can be added to the MPL-feature model during later development phases. For instance a branch that models the different hardware platforms on which the software runs, i.e. the *Product Line Architecture feature model*. Once a small set of platforms have been defined to satisfy the required product line variability, these will impose additional constraints on how the product line requirements can evolve without an expensive redesign. Furthermore, the requirements for specific manufacturers could be added to support specialization of the feature model.

5. Merging MPL-Feature models

5.1. Merging feature models

The next stage in a software supply chain involves the merging of feature models. These could be feature models from one or more suppliers, possibly with their own feature models. There are number of pitfalls that might occur when software components are integrated and feature models are merged.

Merging feature models is a general problem in software supply chains and not only for multiple product lines. In this section we will start with describing the specific issues related to MPL-Feature models and will discuss the issue of *overlapping* feature models, which is generic for software supply chains. We will discuss the following possible scenarios:

- Merging a single product line feature model into an MPL Feature model.
- Merging an MPL Feature model into another MPL –Feature model.
- Merging models from different suppliers.
- Merging overlapping feature models.

5.2. Merging a single product line feature model

Merging a single product line feature model into a MPL-feature model is a straightforward three step approach. First, the two feature models are merged as separate leaves, or one feature model becomes a leaf of the other. The second step is to add the dependencies between the features of the merged models. Finally, the dependencies between the context and the new leaf are added.

5.3. Merging two MPL-Feature models

When two MPL-Feature models are merged into one MPL-feature model, an extra step is necessary, which is the merging of the context variability models.

It could be that the classifiers of the two models represent different contexts. In this case, the obtained context variability model is simply a combination of both models.

In case the classifiers describe the same context, the dependency relation of one model has to be added to the other.

The classifiers could also be describing the same context but are grouped in different ways. For instance one participant uses NAFTA as geographical regions because of the broadcasting standards, and the other uses USA. For Mexico the default language of the user interface could be Spanish and for the USA it could be English. In this case one could choose to create a new classifier in which the two countries end up as sub-classifiers, as described in section 3.5. An alternative approach is to create two different contexts, capturing broadcasting standards and the other describing language preferences. In this way the two context classifiers describe more precisely what variability aspect is captured, but the disadvantage of this is that the amount of contexts is growing.

It may be necessary to use a combination of all three options to merge Context Variability Models.

5.4. Merging feature models from different suppliers.

Merging feature models from different suppliers, offering a similar feature set (e.g., components of two different infotainment solutions) a set of sub features can be used to represent the alternative choices. After the feature models have been merged, the dependencies related to the context variability can be added.

5.5. Example

The example given here uses the specialized feature model from the previous section, i.e. the budget car infotainment product for CarA, to be merged into the overall feature model of CarA. These are two MPL-Feature models of which the features do not overlap and the context classifiers, e.g. the different continents, describe the same context. CarA can still change the regional context and can configure the remaining features of the infotainment system.

After the feature models have been merged, the dependency relations, between the features of the car and that of the infotainment systems have to be added. An example could be the following: the car infotainment has a feature that the outside temperature can be displayed. This feature depends on the availability of an outdoor temperature sensor of the car itself.

After this step the context variability models are merged, meaning that the dependency relations related to the regions have to be combined. Finally there might be some new dependencies. For instance related to the type of the car in which the infotainment system is used e.g. the support of a memory card slot for certain car types or regions. Figure 7 shows a diagram of the merged MPL-Feature models.

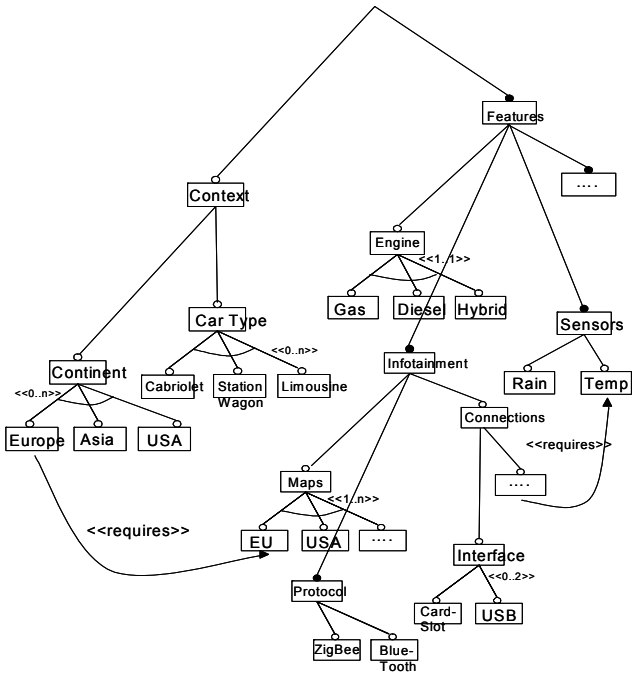

Fig. 7. MPL-Feature Diagram of CarA with merged infotainment system

5.6. Merging overlapping feature models

Two or more feature models could partly represent the same functionality. We will denote this as *overlapping* feature models. When features are overlapping it results in dependencies between the features. A feature, coming from one party cannot operate together with another feature, coming from another party, since they represent alternative implementations of the same functionality.

Example: One supplier may offer software components that play various kinds of audio and video files, such as MP3, WMA, MPEG4, BluRay and DivX.

Another supplier may offer software components that play various audio files and can also rip CDs. In this example part of the functionality is overlapping.

The easiest way to avoid this would be to ask the suppliers to create specialized versions that do not contain overlapping features. In practice one might be faced with commercial off the shelf software for which this is not possible. Furthermore it might be possible that different suppliers are used for different contexts, e.g. one supplier for the Asian market and others for the European market. Another solution is to split the two feature models and create separate leaves, which represent different functionality. The last solution is to add dependency relations where one feature (group) excludes the other.

In this section we have given an introduction to overlapping feature models, which is a generic problem for software supply chains. Merging feature models, especially when they are overlapping, requires a significant engineering activity. We believe that this can hardly be automated, but that the explicit representation of context assists the process of identifying and resolving overlap.

6. Comparison with related work

Several authors have addressed how to store the feature information for multiple product lines one model, thus avoiding the redundancy and inconsistencies due to changes, as discussed in [14,15]. We discuss the differences between the approaches in this section.

Reiser and Weber [15] also consider using of a product model in tree form with "includes" and "exclude" links to the feature model. They begin by exploring how constraints that depend on combinations of contexts could be represented by nesting contexts hierarchically in the product tree, so that only one leaf need be considered for each product. However, they identify the limitations of this approach and introduce the use of Product Sets, each of which is a subset of all the conceivable products. Product Sets are defined by the inclusion or exclusion of particular features from the final products. This cannot be applied in a software supply chain, where a supplier does not know all the configurations of the final product. In contrast, our approach only reasons about constraints on the features that we supply, maintaining them as logical expressions on the contexts. At successive integration stages, the constraints on features from different suppliers can be related. A variability management tool would combine the expressions to form composite constraints and create the appropriate graphical representations.

Our approach therefore achieves the aims of Reiser and Weber, in storing the dependencies and their rationales together, thereby ensuring that no new source of inconsistency is introduced, but in a form that can be employed in a supply chain.

Buhne *et al.* [21] describe an approach to model multiple product lines by using an extension of the OVM model [20]. In their approach only one dimension of the context can be modeled, for instance the product type (DVD, MP3, Hard-disk-recorder), or the geographical region. In our approach several dimensions of the context variability can be expressed within one model.

Czarnecki *et al.* [5] suggest using a level-0 feature model describing the different market segments or geographic regions as a high level requirement. However, they do not address how to integrate the constraints from combinations of these classifiers.

Buhne *et al.* [11] and Reiser and Weber [22] describe the use of multiple levels of feature models to deal with different abstraction levels. Their research aimed at dealing with large feature models in large organizations.

7. Preliminary experimental results

A number of preliminary experiments where executed within NXP Semiconductors to validate our approach. NXP is a tier-1 supplier. It delivers products to various customers such as manufacturers of Televisions, DVD-recorders, Mobile Phones and the automotive industry. Products consist of both Hardware (ICs) and software.

The customers integrate the hardware and software from NXP and other suppliers to create a product that is delivered to end-users. Part of the variability is configured by NXP and part by the customers. NXP buys software components, for instance operating systems that have to be configured for our products. Furthermore, within NXP there are organizational units that are specialized in creating reusable assets. These internal units are act as internal suppliers.

The following experiments where executed:

- Defining context classifiers and creating an MPL-Feature model.
- Generating specialized MPL-Feature models.
- Merging feature models.

Five persons from different organizational units from NXP where involved.

We used two commercial variability management tools that are based on conventional feature modeling. Creating an MPL-feature model, including a wide range of dependency relations, was straightforward and was well supported by the tools. There were several

ways to present the features and relations in different graphical and textual styles and provide consistency checking. One of the tools was able to visualize the result of a choice in the context model on the feature model. This proved to be very essential when making configuration choices in large models, when a feature selection is found to be constrained unexpectedly.

Neither of the tools we used could support the specialization of dependencies between features that specializes the cardinalities. However, there was no case where actual products needed this.

For staged configurations a prototype was available in one tool. With this prototype we could generate specialized MPL-feature models, both for internal as well as external customers.

For merging feature models it was possible to create a model in which the feature models where separate leaves. It was not possible to merge a feature model in such a way that one feature model is a leaf of another. For this, we used a manual process by copying and pasting parts of a feature model into a new feature model.

We have not executed experiments with merging overlapping features, although we have identified cases from NXP's practice in which this occurs.

Although the tools could not fully support this way of working at this moment, the approach was considered as intuitive and straightforward. It has therefore been decided to adopt this approach at NXP.

8. Discussion and further research

The definition of the context classifiers may not be easy in all situations. To differentiate between product types and customers is straightforward and seems to work for different domains. To define regions may not be so straightforward. For instance, in some countries traffic drives on the left side of the road: South Africa, the UK and Suriname. These three countries are located in different continents. It requires careful consideration to decide what aspect of the context is captured. We believe that requirements elicitation can benefit from this approach and may avoid problems like interoperability in consumer electronic product [23]. Further research is needed.

The dependency relation between a feature and the cardinality of another feature could be useful in conventional feature modeling as well. For instance the amount of processes that can run together depends on the amount of memory in the system.

9. Conclusions

The concept presented in this paper provides a solution for modeling multiple product lines by using the novel concept of Context Variability.

The Context Variability Model provides an intuitive way to capture the variability and commonality in the requirements that originate from different contexts, such as different product types, geographic regions and customers. With this concept it is possible to capture several dimensions of variability in the context space in one model.

The concept fits very well in staged development and supports software supply chains. Extensions to the approach are possible to support the development lifecycle, for instance by adding an extra branch that describes the hardware architecture.

Our preliminary experiments showed that the approach is straightforward to use. Because our approach is based on conventional feature modeling, many variability management tools give the support needed for our approach. This makes it a very attractive candidate to be applied in the industry. NXP Semiconductors has decided to adopt this approach.

10. Acknowledgements

The authors would like to thank Hans den Boer, Wouter Batelaan, Jean-Marie Trager, Somasundaram Venkatachalam and Aart Matsinger from NXP Semiconductors and Rob van Ommering from Philips for sharing experimental results, fruitful discussions and the challenging problems they have given us.

11. References

[1] Clements P., Northrop L.: *Software Product Lines: Practices and Patterns*, Addison-Wesley, Boston, MA 2001.

[2] Kang K., Cohen S., Hess J., Nowak W. Peterson S.: Feature-oriented domain analysis (FODA) feasibility study, *Technical Report CMU/SEI-90-TR-21,* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[3] Czarnecki, K, Eisenecker, U.W. *: Generative Programming: Methods, Tools, and Applications*, Addison-Wesley 2000.

[4] Czarnecki, K, Helsen, S, Eisenecker, U: "Staged Configuration using feature Models". *Software Product Lines, Lecture Notes in Computer Science*, Springer, 2004.

[5] Czarnecki, K, Helsen, S, Eisenecker, U: "Staged Configuration through specialization and Multi-Level

Configuration of Feature Models". *Software Process Improvement and Practice*, 10(2), 2005.
[6] Thiel S., Hein A.: "Modeling and Using Product Line Variability in Automotive Systems", *IEEE Software vol19*, no 4. 2002.
[7] Weber M. Weisbrod J.: "Requirements Engineering in Automotive Development – Experiences and Challenges", *IEEE Software.* vol. 20, no 1. 2002.
[8] Debbi O., Salinese C., Fanmuy G.: "Industry Survey of Product Lines Management Tools: Requirements, Qualities and open Issues". *15th IEEE International Requirements Engineering Conference*, 2007 IEEE.
[9] Ye, H. Liu, H.: "Approach to modeling feature variability and dependencies in software product lines". *IEEE proceedings Softw.*, vol152, June 2005.
[10] Reiser, M, Weber, M.: "Managing Highly Complex Product Families with Multi-Level Feature Trees", *14th IEEE International Requirements Engineering Conference* IEEE 2006.
[11] Buhne S. Halmans G., Pohl K., Weber M, Kleinwechter H., Wierczoch T.: "Defining Requirements at different levels of abstractions", *12th IEEE International Requirements Engineering* Conference 2004.
[12] van den Hamer, P., Lepoeter, K.: "Managing design data: the five dimensions of CAD frameworks, configuration management, and product data management". *Proceedings of the IEEE, Volume 84*, Issue 1, Jan 1996 pp: 42 – 56.
[13] M. Aoyama, K. Watanabe, Y. Nishio, and Y. Moriwaki: "Embracing Requirements Variety for e-Governments Based on Multiple Product-Lines Frameworks". *Proceedings of the 11th IEEE International Requirements Engineering Conference*, 2003.
[14] Buhne S., Lauenroth K., Pohl K.: "Why is it not Sufficient to Model Requirements Variability with Feature Models". *Proceedings of Workshop: Automotive Requirements Engineering* (AURE04), Nazan University, Nagoya, Japan, 2004, pp5-12.
[15] Reiser, M, Weber, M, "Using product sets to Define Complex Product Decisions", *SPLC* 2005 Springer.
[16] Ommering R. van: "Building Product Populations with Software Components", *Proceedings of 24th International Conference on Software Engineering*, Orlando, Florida, September 2002, pp255-265.
[17] Ommering R. van, Trew T.: "Building Product Populations with Software Components Symposium and the consequences for requirements" *on: Recycling Requirements for Systems and Product Families*. London, April 24th, 2002
[18] Greenfield, J and Short, K. *Software Factories: Assembling Applications with Patters, Models, Frameworks, and Tools,* 2004, Wiley, Indianapolis, IN.
[19] H. Lee, C. Billington: "Designing Products and Processes for Postponement". In S. Dasu and C. Eastman (eds) *Management of Design: Engineering and Management Perspectives*, pp105-122, Kluwer, 1994.
[20] Pohl, K, Bockle, G, van der Linden, F: *Software Product Line Engineering.* Springer 2005.
[21] Buhne, S, Lauenroth, K, Pohl, K: "Modeling requirements Variability across Product Lines", *Proceedings of 2005 13th IEEE International Conference on Requirements Engineering*.
[22] Reiser M., Weber M.: "Managing Highly Complex Product Families with Multi-Level Feature Trees", *14th IEEE International Requirements Engineering Conference (RE'06)* pp. 149-158
[23] Ouden, E. den: "Development of a Design Analysis Model for Consumer Complaints". *PhD. Thesis, Technical University of Eindhoven,* Eindhoven 2006.