Advanced Modularity for Building SPL Feature Models: a Model-Driven Approach

João Araújo, Miguel Goulão,
CITI/FCT,
Universidade Nova de Lisboa,
2829-516 Caparica, Portugal
joao.araujo@fct.unl.pt,
mgoul@fct.unl.pt

Ana Moreira, Inês Simão,
Vasco Amaral,
CITI/FCT,
Universidade Nova de Lisboa,
2829-516 Caparica, Portugal
amm@fct.unl.pt,
nessimao@gmail.com, vma@fct.unl.pt

Elisa Baniassad,
Australian National University,
Canberra ACT 0200, Australia
elisa.baniassad@anu.edu.au

ABSTRACT
Feature Models are commonly used to specify commonalities and variabilities in Software Product Lines (SPL). Our goal is to enhance feature modeling with traceability and improved support for crosscutting concerns. While traceability will show the features' requirement-origins, providing means to reason about their existence, crosscutting concerns will be handled through advanced modularity mechanisms (e.g. aspects), making the impact of changes to SPL models less difficult to understand and analyze. The result is Theme/SPL, a novel SPL requirements technique based on a concern-driven approach (Theme/Doc). Theme/SPL includes the proposal of a domain-specific language for specifying Theme/Doc models and uses model-driven development to generate automatically feature models from them. We show the applicability of the technique through a case study using a within-group design to evaluate the final results and tools developed.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – languages, methodologies.

General Terms
Design.

Keywords
Model-Driven Development, Software Product Lines, Advanced Modularity

1. INTRODUCTION
Software Product Lines (SPL) use feature modeling as a key technique for capturing commonalities and variabilities of a software product family [14].

We aim at using feature models for SPL requirements modeling and simultaneously offer both means to trace the requirements origin of features present in a particular product line, and improved early modularity support by identifying and modularizing concerns that otherwise would be scattered throughout the feature models.

SAC'13, March 18–22, 2013, Coimbra, Portugal.

Feature models [12] show a very specific perspective of a product line, but other perspectives are necessary to offer the rationale or origins for each feature. Requirements descriptions are the source to identify features, thus providing a basis for justifying their origins and rationale. Intuition and domain knowledge help developers predict which features will be present in a system, but these alone cannot provide system-specific scope of a feature. For example, in mobile phones SPL, a security feature is expected to be present, but the respective feature model provides no rationale for the inclusion of this, or any other, feature. Although this rationale might be present in the system's requirements, firstly, security is not necessarily restricted to one portion of the requirements, and secondly, it is intractable to exhaustively search through all the requirements to find the rationale for each feature in an unstructured fashion. Traceability mechanisms should be used, as they provide support to solve this problem.

The second issue addressed in this paper is to capture crosscutting concerns in an SPL that are scattered throughout its specifications, from requirements to the feature model. Going back to the mobile phones SPL example, let us consider a *check balance* service and a *security* feature that may be replicated in several components. To facilitate product derivation, each concern or feature should be encapsulated in separate modules. Aspect-oriented approaches [6][15] offer a step forward to improved or advanced modularization of crosscutting concerns from requirements to code.

We looked for an aspect-oriented approach that could be tailored to address the aforementioned problems at the early stages of SPL development. We chose Theme/Doc [6], an advanced separation of concerns method for requirements traceability that aims at identifying, modeling and composing crosscutting requirements through the identification of themes (i.e., concerns that encapsulate behavior) and their relationships. Theme/Doc offers traceability, by recording a link between each model element and its requirements origins. But Theme/Doc is not tailored for use with SPL, therefore missing model elements for managing variability. That is, although Theme/Doc is a good theoretical fit for our two problems, its adaptation to SPL development is not trivial, as its constructs need to be extended to express SPL concepts and properties.

The goal of this paper is to extend Theme/Doc by adopting Model-Driven Development (MDD) [18] techniques (metamodeling and model transformations). The result is

Theme/SPL, an approach supported by an Eclipse based tool defined and implemented using a Domain-Specific Language (DSL) [10]. Two metamodels were developed, one for Theme/SPL models and another for feature models. The transformations of Theme/SPL models into feature models are also defined using ATL [3] and supported by the tool. The approach was applied to several examples, and the resulting models were evaluated. We also evaluated the usability of the tool support.

This paper is structured in 9 sections. Section 2 summarizes the background of this work. Section 3 describes the new Theme/SPL approach. While Section 4 describes the requirements and theme elicitation activities, Section 5 provides heuristics to build a Theme/SPL model and Section 6 shows the required transformation rules. Section 7 completes the core of our contribution by presenting the evaluation results. Finally, Section 8 discusses some related work and Section 9 summarizes our conclusions and indicates future work.

2. BACKGROUND

SPL development has two main activities: domain engineering and application engineering [14]. In this paper we focus on the domain engineering activity at requirements level, where the commonalities and variabilities in product lines are captured using a feature model. A feature may be[7][12]: (i) mandatory (the feature must be present in all members of the product line), (ii) optional (the feature may or may not be present in a product of an SPL), (iii) inclusive-or (a feature that is composed of a set of features of which one or more is chosen), or (iv) alternative (a feature that is composed of a set of features of which exactly one is chosen). We use the cardinality-based feature model developed by Czarnecki et al. [7], which gives more semantics to the traditional feature model [12].

The Theme/Doc approach is defined for analysis based on the concept of *theme* [6]. Themes are obtained from a requirements list, which is the starting point of Theme/Doc. This list is analyzed and the main concerns – the themes – are identified, consisting of a verb plus an object. For example, requirements stating that mobile phones should allow making phone calls and sending SMS originate the themes "Make a call" and "Send an SMS". These themes encapsulate concerns that are related to specific system functionalities. There are two kinds of themes: *base themes*, which can be affected by aspectual behaviors that are specified by *crosscutting themes* (also known as *aspectual themes,* or just *aspects*). Also, Theme/Doc defines *entity* as an object that interacts with a theme.

Theme/Doc identifies themes in requirements documents and provides heuristics to identify which of the themes are crosscutting (the aspects), and which are not (the bases). Theme/SPL extends the Theme/Doc approach by including new heuristics to support SPL, adding the variability information to the Theme models, and then mapping Theme models to a corresponding feature model. The basic concept we use to bridge the gap between Theme and SPL is concern, which encapsulates a system's property. We map a theme to a feature as both are concern representations.

3. THEME/SPL APPROACH OVERVIEW

Figure 1 gives an overview of the process of the approach (modeled using a UML activity diagram). It builds Theme/Doc and feature models for Domain Engineering. The process is divided into three main activities:

1. **Elicit requirements and themes.** Examine the requirements documentation of the system to elicit requirements and themes. The deliverables of this stage are lists of requirements and themes (see examples in Section 4).
2. **Build Theme model.** Use requirements and themes to identify entities and to define the relationship among themes and between themes and entities, captured in the Theme model (see example in Figure 2). The 10 heuristics described in Section 5.1 support this activity. We finish with the stakeholders' validation of themes and requirements.
3. **Build feature model.** Automatically derive the feature model, where features are primarily identified and variability analyzed, using the Theme model defined in the previous activity. The Theme model is used as input for the ATL transformation rules defined in Section 6. Then, validate the feature model. The identification of aspectual themes may lead to a refinement of the feature model, which is finally validated (in this paper we will not focus on validation, but it must be carried out by the stakeholders in inspection sessions, for example).

This process is supported by a rigorous DSL for specifying Theme/SPL models as well as model transformations specified in ATL [3], to derive feature models from Theme/SPL models. The models obtained by applying this process can then be used to guide the remaining development activities. Concerning evolution, this is benefited as the modularization of crosscutting concerns obtained will make it easier the modifications as they will be localized. Throughout this paper, we use a small part of a mobile phone software product line as a running example. The example's aim is to develop software components to *make calls*, *put phone calls on hold*, *insert contacts in a contacts list*, *send and receive SMS and MMS*, *take pictures*, and *transfer data between two mobile phones.*

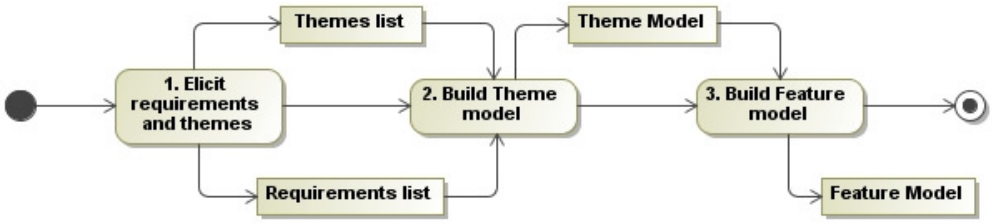

Figure 1. Theme/SPL Domain Engineering process.

4. REQUIREMENTS AND THEMES ELICITATION

The first step is to build a requirements list, using the typical inputs from clients, such as documentation and interviews. Table 1 shows a partial requirements list for the mobile phone SPL.

Table 1. Requirements List

Req.	Requirements description
R4	To make a call a user should search the receiver contact in contacts list and press the call key.
R5	To make a call the user needs to check the balance on the phone.
R6	If there is no balance in phone, display the message "Out of balance".
R7	If there is enough balance, the phone call is made.
R8	When the user is making a call, another call may be on hold, showing on the display the message "Call Waiting".
R10	The user, using the camera phone, takes a picture and sends it by MMS to another user.
R11	To send an MMS, the user must attach the photo to the message, write the text on the photo if s/he wants, and find the contact in the contact list
R12	To send an MMS the user needs to check the phone balance.
R13	If there is no balance in mobile phone, display a message "Without balance".
R14	If the mobile phone has balance, display a message "Multimedia message sent".
R15	When user receives an MMS a message "New message received" will be shown on the display.
R17	To send an SMS, the user will have to write the message and find the contact in the contacts list, since it can be sent to one or more recipients.
R19	To send an SMS the user needs to check the phone balance.
R20	If there is no balance the mobile phone displays the message "No balance".
R21	If there is balance, the mobile phone displays the message "Message sent".
R31	It allowed the user to transfer data between mobile phones for at least one of the mechanisms: Bluetooth, Infrared or USB.
R32	To make the data transfer the user has to activate the transfer mechanism, select the data type to send, photo or SMS, and select the file.

Based on this list we identify a (non-exhaustive) set of themes, following the theme elicitation process [6]: (T1) Make a call; (T2) Put phone call on hold; (T3) Receive MMS/SMS, (T4) send MMS/SMS; (T5) Take pictures; (T6) Transfer data. Basically, by analyzing the requirements, the main concerns (the themes) are extracted from them — they are based on the verbs (and objects) that can be abstracted from these requirements. So, each theme is related to one or more requirements. For example, a theme "Make a call" can be extracted from the requirements R4 and R5.

5. THEME/SPL MODEL CREATION

We propose ten heuristics designed to evaluate the requirements list (the input to this process) to produce the Theme model, which is then used to generate the feature model. These heuristics are applied to the requirements list. The requirements must follow a particular style, as described next. In this section, we discuss the heuristics and the DSL created to support the construction of the Theme model. We extend the Theme model with SPL concepts, defining the following relationships: alternative, obligatory and part-of. Also, each theme is mapped into a feature.

5.1 Heuristics for the theme model creation

H1. Identify the root. Find a requirement that says what the system is. Model parts (possibilities) that make up the root and each component will lead to a root theme. The link type is explained in heuristics H2 and H3.

H2. Identify optionality. If a requirement contains descriptions that indicate optionality, such as *"when X, Y may run"*, obtain the themes X and Y; Y is optional and is represented by a link with the stereotype «`alternative`». This is the case of the theme *"Make call"* (i.e., *X*) and *"Put call on hold"* (i.e., *Y*).

H3. Identify mandatory dependent features. If a requirement contains descriptions to indicate a mandatory dependency situation, such as *"when X, it has to have Y"*, *"when X, there must be Y"*, or *"X requires Y"*, if X runs, Y must be executed. This is represented by a link with the stereotype «`obligatory`» from X to Y.

H4. Identify themes aggregation. If there are requirements related to a theme that identifies other themes that compose it, the relationship is represented by a connection with the stereotype «`part_of`». Therefore, in the action view, the theme *"Take Picture"* is part of *"Send MMS"*.

H5. Identify generalized themes. If two or more themes are closely related, then a new theme is created so that it generalizes those themes. The themes *"Send SMS"* and *"Send MMS"* differ on the kind of information to be sent, so we create a new theme *"Send Message"*, whose *sub-themes* will be those which originated it.

H6. Identify OR alternatives. If a requirement contains expressions describing several alternatives such as *"selects at least one of several"*, then we have a theme called ThemeGroup (i.e., a theme that is decomposed into alternatives) that contains a minimum and maximum number of alternatives. The alternatives are called GroupedThemes, and they are linked to the ThemeGroup through a link with the stereotype «`alternative_or`». For example, for the ThemeGroup *"Transfer Data"* we can choose from 1 to 3 GroupedThemes (*"Bluetooth"*, *"Infrared"*, *"USB"*).

H7. Identify XOR alternatives. If a requirement contains expressions such as *"only one can be selected"* among several alternatives, then we have also a ThemeGroup, and the alternatives (GroupedThemes) and are linked to ThemeGroup with the stereotype «`alternative_xor`». For example, for the ThemeGroup *"Choose Payment Method"* we choose either the *"ATM"* or the *"bank's website"* GroupedTheme.

H8. Identify relationships between themes and entities. If in a requirement there is a theme related to an entity (i.e., objects in the Theme model), that entity will be *"part-of"* the theme.

H9. Identify aspectual themes. An aspectual theme is a theme that appears repeatedly in two or more requirements. After identifying it, a relationship from the aspectual theme to the base theme with the stereotype «`crosscutting`» is defined. In this example, an aspectual theme is *"Manage Contacts"* since it is needed to *"Make Call"* and to *"Send Message"*.

H10. Identify "requires" relationships. In a requirement, if there are themes or entities where one needs another, in expressions such as *"X using Y"* or *"X through Y"*, then there is a link between the two. This link from X to Y is decorated with the

stereotype «requires». For example, the theme *"Take picture"* requires the entity *"Camera"*, as described in the requirement R10 *"using a camera, take a picture"*, so there is a «requires» link from the theme *"Take picture"* to the entity *"Camera"*.

After applying these ten heuristics, we obtain a theme-relationship view model. The heuristic H1 gives the root, important to establish the traceability between the Theme model and the feature model. Variability is addressed explicitly in H2, H3, H6 and H7, resulting in an extension of original Theme/Doc approach with stereotypes that extend Theme with SPL concepts. H5 is important as an extensibility mechanism, to facilitate the introduction of new sub-themes. The *part-of* relationship covered by H4 and H8 and the *requires* relationship in H10 are also extensions of Theme/Doc. Here we obtain indirectly additional mandatory features. H9 is crucial to obtain a better modularized model, which will have similar impact on the generated feature model.

The advantages of these heuristics are threefold: (i) the obtained Theme model is closer to feature model, but not exactly the same, as we want to preserve some semantics of themes relationships that do not exist in the feature model; (ii) the Theme model justifies the features by tracing them back to the requirements list – the traceability is achieved as features are obtained from themes by transformation, and themes are originated from requirements; (iii) the aspectual themes identification and mapping into separate features promote an improved modularity of the feature model.

Figure 2 presents a resulting screen shot of a Theme model for Theme/SPL DSL editor tool after applying the heuristics. Features are represented as diamonds, and their relationships, as stereotyped arrows. In the figure, the root is called *MobileSPL*. Also, we see three compartments that group features related to data transfer, sending messages and receiving messages. It is also shown obligatory (e.g., between the root and the theme *Send Message*) and alternative (e.g., between the root and the theme *Transfer Data*) relationships between themes. Crosscutting relationships are also exemplified (e.g., from the theme *Manage Contacts* to the themes *Make Call* and *Send Message*). A *part-of* relationship is illustrated between the *Send SMS* and *Write Text* themes.

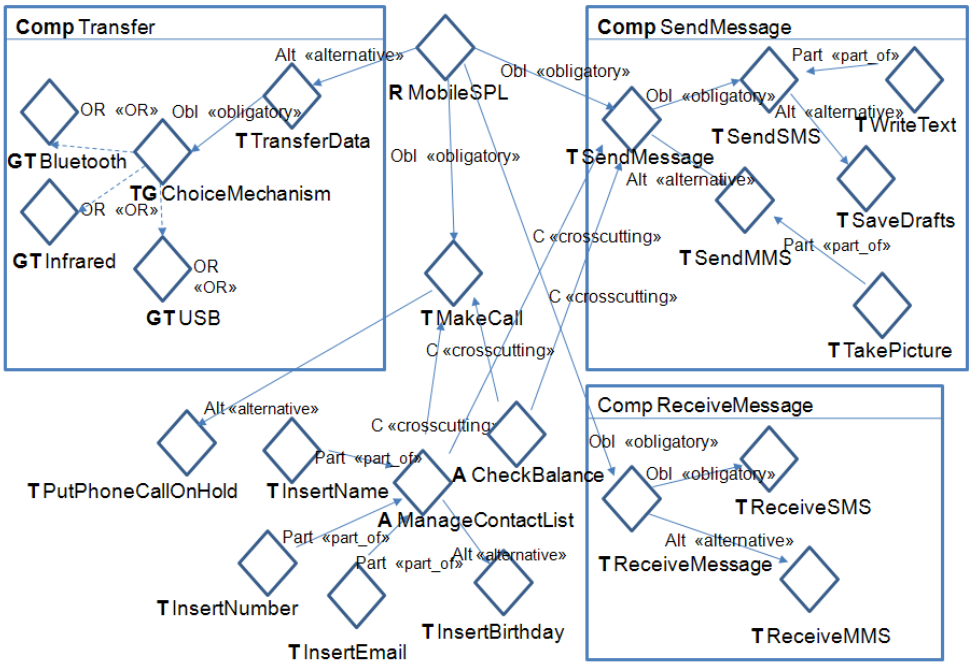

Fig. 2. Theme model for the mobile phone SPL

5.2 A DSL for Theme/SPL

We created a DSL for the Theme/SPL approach. We first defined a metamodel for Theme/SPL and another for the feature model (based on the metamodel described in [7]) using the models in the Ecore metamodeling language.

The Theme/SPL approach metamodel contains a root node called *ThemeApproach* which links the classes that represent nodes and links between nodes. Figure 3 illustrates the Ecore model, where the identified nodes are *RootTheme*, *Themes*, *Aspect*, *Entity*, *ThemeGroup* and *GroupedTheme*. The possible links are: *Obligatory*, *Alternative*, *Part_of*, *Crosscuting*, *Alternative_or, Alternative_xor*, *Require and Extend*.

The metamodel for the feature model contains a root node named *FeatureModel*. As in the Theme approach metamodel, there are classes that represent the *Nodes* and *Links* between nodes. We identified the nodes *RootFeature*, *Feature*, *FeatureGroup* and *GroupedFeature*, and the links *Optional*, *Mandatory*, *Alternative_or*, *Alternative_xor*, *Requires* and *Excludes*. The metamodel is not shown here for space reasons. Figure 4 shows a screen shot of a generated feature model.

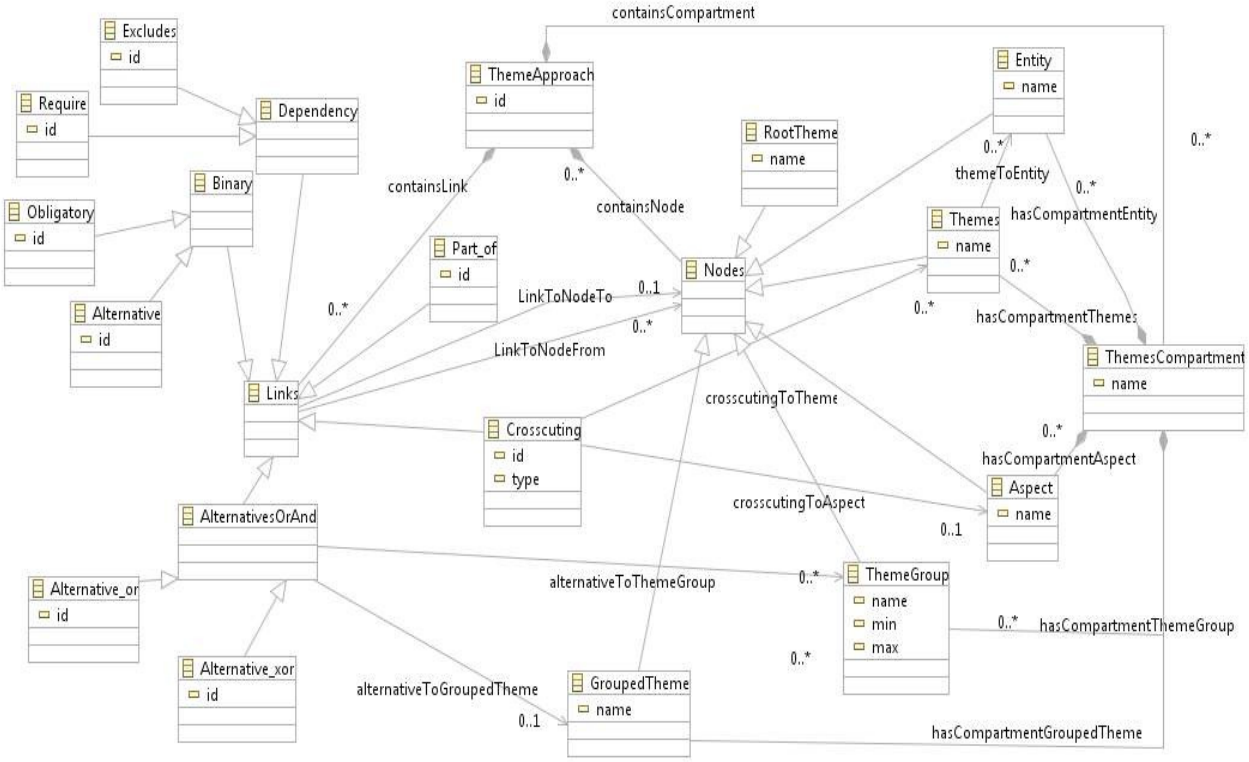

Figure 3. Ecore model for Theme/SPL approach.

6. FEATURE MODEL GENERATION

The mapping from the Theme/SPL model to a feature model is straightforward. Each theme becomes a feature and the relationships between the themes are directly mapped to the feature model. Each aspectual theme appears only once in the feature model. To specify and implement the transformation from the Theme/SPL model to feature model, we used the Ecore model for the Theme/SPL approach (Figure 3) as the metamodel of the transformation source. Then, through transformation rules, and using the Ecore model for feature model as a target metamodel, we generate the feature model that can be visualized and edited by the graphical editor (Figure 4).

In Table 3 we present part of the set of transformation rules implemented in ATL to transform automatically the Theme/SPL models into the feature models. The complete set of rules can be found at http://ctp.di.fct.unl.pt/~ja/ThemeSPLrules.pdf.

These rules are in accordance with the heuristics defined in Section 5.1. Table 3 shows the transformation rules to map themes to features, aspect to features, and for the "Part_of" relationship. The ATL code is presented on the left hand side of Table 3. On the right hand side of the table, we explain each rule informally. Figure 4 shows the resulting feature model after the application of the transformation rules to the Theme/SPL model shown in Figure 2. Note that some features have multiple parent features (e.g., Manage contact list); those are crosscutting features, that are modularized in a separate feature as a result of the Theme analysis.

Table 3. ATL transformation rules

ATL Transformation rule	Comments
rule ThemesToFeature { **from** p: Theme!Themes **to** out: FM!Feature (nameF <- p.name)}	In the `ThemesToFeature` rule (related to heuristic H4), for each instance `Themes` of the origin metamodel, an instance `Feature` of the destination metamodel is created, in which the name of the feature matches the name of the theme.
rule AspectToFeature { **from** p: Theme!Aspect **to** out: FM!Feature (nameF <- p.name)}	In the `AspectToFeature` rule (related to heuristic H9), for each `Aspect` instance of the origin metamodel, a `Feature` instance of the target metamodel is created, in which the name of the feature is the name of the aspect.
rule Part_of{ **from** p: Theme!Part_of **to** out:FM!Mandatory(sourceFeature <- p.LinkToNodeFrom, targetFeature <- p.LinkToNodeTo)}	In the `Part_of` rule (related to heuristic H3), for each instance `Part_of` of the source metamodel `source`, an instance `Mandatory` of the target metamodel is created, in which the source link, `source feature`, corresponds to the `LinkToNodeFrom` link, and target link, `target feature`, corresponds to the `LinkToNodeTo` link.

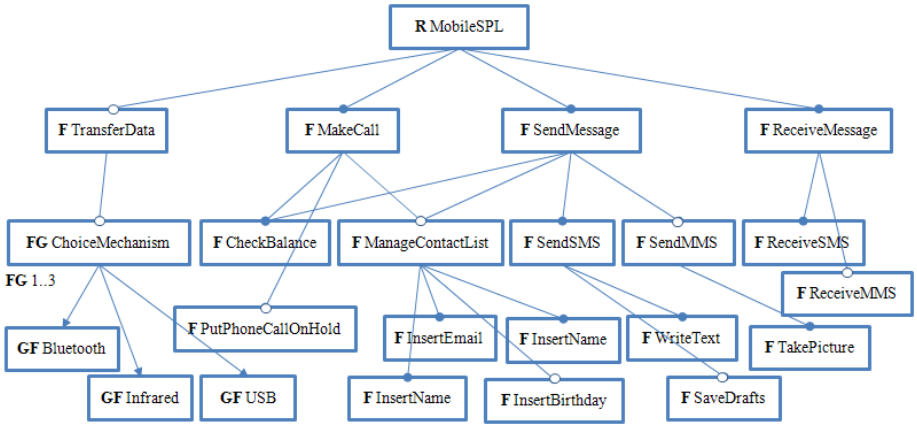

Figure 4. Feature model resulting from the transformation.

7. EVALUATION

Our evaluation strategy was twofold. On the one hand, we applied the Theme/SPL approach to case studies from different domains where the combination of traceability and crosscutting concerns support was considered very important. This allowed us to assess the feasibility of the approach. On the other hand, we also wanted to assess the utility and usability of our prototypal tool support to the Theme/SPL approach.

7.1 On the feasibility of the approach

The approach was applied successfully to the Smart Home case study, a real case study used in the European project AMPLE [1]. The Smart Home case study defines a SPL targeted to embedded systems, in the context of home automation. The rationale for using a SPL in this context was that the instantiation of a specific Smart Home product should be easy and cost effective. This required the base assets to be of good quality and easy to reuse, as well as sophisticated support for product derivation, as different homes require tailored configurations of the Smart Home. These requirements were the reason why technologies like MDD or AOSD were selected. When applying this case study to our approach, the generated feature model from the Theme/SPL models was very similar to the existing feature model that was built with AMPLE techniques which did not involve the use of DSLs and high level transformation languages, as we did here.

The approach was also successfully applied in the realm of case studies from the mobile phones domain (briefly described in this paper), and public health care.

7.2 On the feasibility of the approach

Having an effective tool support for the Theme/SPL is essential for its applicability. In this section, we describe a usability evaluation aimed at assessing the Theme/SPL tool support. Using Wohlin's experimental objectives template definition [21] we can briefly summarize the objectives of our evaluation:

Our goal was to analyze the Theme/SPL approach, for the purpose of characterizing its existing tool support, with respect to its usability and usefulness, from the point of view of software engineers engaged in building feature models, in the context of a case study conducted in an academic environment.

Usability and usefulness are too abstract to be assessed directly, so we can be break them into more specific goals. In particular, we consider the following: (G1) Ease of feature identification with Theme/SPL; (G2) Ease to build feature model using Theme/SPL; (G3) Extent to which the Theme/SPL approach introduces quality problems, i.e., leading to incorrect, incomplete, or unnecessarily complex models, when compared to the baseline; (G4) Extent to which the tool support for Theme/SPL was useful in increasing the efficiency and correctness in model building; (G5) Extent to which the tool support for Theme/SPL was easy to use.

The evaluation was performed using 10 graduate computer science students from our university as subjects. Participants were asked to use the Theme/SPL approach to partially model a SPL in the domain of public health monitoring. The participants' answers were made anonymous, to mitigate the risks of biases, such as evaluation apprehension and hypotheses guessing, by the participants. All participants had previous experience with aspect-oriented and feature modeling, but not with Theme/SPL. As such, participants' skills were comparable to those of junior software engineers being introduced to the Theme/SPL approach. Evidence collected elsewhere [16] suggests that the results obtained by students of a profile similar to our participants are close to those obtained by novice professionals.

Participants were given basic training in Theme/SPL. They were then provided with a Theme/SPL model in the realm of health monitoring systems. After studying the model, participants were asked to design a feature model for a health monitoring systems SPL, based on their understanding of the requirements of the system, complemented with the information within the Theme/SPL model. Finally, using the Theme/SPL model as input, the transformation was carried out from Theme/SPL to the tool-generated feature model. By the end of the modeling tasks, each participant had his "hand-made" feature model, and the corresponding tool-generated feature model. All participants were able to manually produce a high-quality feature model in a timeframe ranging from 15 to 20 minutes. The "hand-made" feature model was then used as a baseline by participants, while answering a survey concerning the research goals previously described in this section (G1..G5).

The questionnaire included the 5 questions (Q1..Q5) listed next, which were aimed at addressing each of the research goals (G1..G5), respectively. The questionnaire was designed using a 5-level Likert scale, where 1 stands for the worse scenario, and 5 for

the best scenario, from the Theme/SPL tool user point of view. A level 3 answer would correspond to an average answer, which is considered as indifferent. In other words, the "average" answer corresponds to not identifying significant benefits, or drawbacks in using the Theme/SPL approach, when compared to the baseline alternative. The questions were as follows: (Q1) How easy was it to identify features, using the Theme/SPL approach? (1 – Very difficult .. 5 – Very Easy); (Q2) How easy was it to build a feature model (including the relationships among the features)? (1 – Very difficult .. 5 – Very easy); (Q3) Does the tool generated model introduce quality problems, when compared to the baseline? (1 – A lot .. 5 – None); (Q4) How useful was the tool support? (1 – Not useful .. 5 – Very useful); (Q5) How difficult is it to use the tool? (1 – Very difficult .. 5 – Very easy).

Table 4 summarizes the results of the qualitative assessment of the tool support. The first 6 columns represent, from left to right, the question identification and the frequency of answers in each of our 5-level scale, for each question (L1 through L5). The sum of the answers is always 10, indicating that all participants answered to all the questions in the questionnaire. Overall, we can observe that there is a concentration of higher frequencies in the favorable evaluations to the tool support. Although it does not make sense to sum the frequencies of answers in each level, as we would be counting frequencies in different scales (as noted by the value explanations for each of the questions), note that none of the respondents chose to use the two lower levels, for any of the questions, and even level 3, which would indicate an indifference between using the approach described in this paper and the baseline was rarely used (twice, in Q1, and once, in Q4).

Table 4. Summary of tool support assessment

Question	L1	L2	L3	L4	L5	Chi-Square	df	Asymp. Sig.
Q1	0	0	2	5	3	9,000	4	0,061
Q2	0	0	0	4	6	16,000	4	***0,003***
Q3	0	0	0	0	10	40,000	4	***0,000***
Q4	0	0	1	4	5	11,000	4	**0,027**
Q5	0	0	0	5	5	15,000	4	***0,005***

If subjects were to answer randomly to the tool assessment questionnaire, we would have an equal probability of getting answers in each of the levels. We need to test whether the answers obtained in our questionnaires are significantly different from those we would obtain by chance. To test this hypothesis, we formulate a null (H_0) and an alternative (H_1) hypothesis as follows: (H_0) There is no significant difference between the expected (all answers were equally probable) and the observed distributions of answers; (H_1) There is a significant difference between the expected and the observed distributions of answers.

We use a Chi-Square test [20] to test the null hypothesis (H_0). The Chi-Square test is a statistical test used to determine if observed data deviate from those expected under a particular hypothesis (in this case, the random distribution of answers. The last three columns in Table 4 present the Chi-Square statistic, the number of degrees of freedom, and the asymptotic significance of the Chi-Square statistic (**values in bold are significant at the 0,05 level**, ***while values in italic bold are significant at the 0,01 level***). We can reject the null hypothesis for questions from Q2 to Q5 (4 out of 5 questions). Q2, Q3, and Q5 have a *p-value* < 0,01, while question Q4 can be rejected with a *p-value* < 0,05. The observed difference between the observed and expected values for Q1 is not statistically significant at the 0,05 level, so we cannot reject the null hypothesis for Q1. The overall results provide a very encouraging feedback from the participants. The answers to question Q2 indicate that participants found the approach to be helpful in feature model development, when compared to the baseline (the answers to Q1 seem to support this tendency, but are statistically inconclusive). The unanimous response concerning question Q3 points to a "doing no harm" property of the approach, in the sense that the feature model generation was not perceived as introducing quality problems. This is an important aspect in model generation, as the potential introduction of quality problems in the feature models would be a significant drawback to the approach: it is unclear whether what the effort to find and remove such problems would be, if problems were to be introduced. The answers to questions Q4 and Q5 convey a good overall judgment of the tool's usefulness and usability, respectively.

In any empirical assessment, we must always consider the potential threats to the validity of the results, not only to set the boundaries of applicability of those results, but also to identify opportunities for extending the work. A first category of threats we can identify concerns the selection of participants. The number of participants (10) is relatively small. Nevertheless, usability experts have observed that, in general, having as little as 5 usability testers allows identifying around 80% of the usability problems [13]. Nevertheless, the overall consensus level in several of the questions is encouraging.

To circumvent the threats concerning the participants' number and profile, an adequate future evolution of this work would be to replicate the assessment in a different setting, thus increasing the number of overall participants and their diversity of backgrounds. Ideally, this replication should be performed by a completely independent team, so that other potential threats such as hypothesis guessing by participants would be further mitigated. Another threat concerns the usage of modeling problems from a single product line in this assessment. Again, replicating this assessment with other product lines from different domains would add to the external validity of this assessment.

8. RELATED WORK

Weston et al. [19] present an approach that helps to construct feature models from a set of documents where requirements are expressed in natural language. This approach is more appropriate when requirements are expressed textually, as it is based on natural language processing, but not suitable to analyze models. Our approach relies on MDD, which is more suitable when models are used.

FeatureMapper [9] is an Eclipse tool that allows for mapping features to arbitrary modeling artifacts. These include UML2, domain-specific modeling languages, and textual languages. FeatureMapper relates features and model elements and derives product models by removing all model elements associated with features not selected for that product. Compared to ours, this does not identify features from requirements.

In the AHEAD approach [4], a program has many representations besides source code, including UML documents and performance models, where each representation is written in its own language or DSL. When a feature is added to a program, any or all of the program's representations may be updated. Again, this approach does not consider the identification of features from requirements.

Trujillo et al. [17] discuss an MDD approach to create a product line (illustrated with the *portlets* SPL, i.e. components of web portals) by composing features to create models, and then transforming these models into executables.

FeatureHouse [2] is a general approach to the composition of software artifacts written in different languages supported by a tool. Both approaches are MDD, but without dealing with feature identification from requirements.

Jayaraman et al. [11] present an approach based on UML to maintain the separation of features during the modeling, using a composition language based on graphs transformation. The language can be used to compose SPL models for a given set of features. However, this work does not provide heuristics to identify features.

Carton et al. [5] present a tool that integrates the Theme/UML with an MDD process. The approach transforms Theme/UML models into platform-specific models and code. This method is adequate for the design phase, but not thought for SPL development.

Groher and Volter [8] present a model-driven and AO approach for SPL. Features are separated in models and composed by composition techniques on model level. However, little attention is given to aspects and MDD at early requirements as we do.

9. CONCLUSIONS

In this paper we adapted an aspect-oriented requirements approach (Theme/Doc) to represent variability in domain engineering of software product lines. To achieve that, we enriched the Theme/Doc approach to capture variability by defining a set of heuristics and stereotypes. The advantages of these heuristics include: (i) the proposed Theme model is adequate to SPL development while preserving the semantics of themes relationships; (ii) the approach provides traceability from requirements to the features; (iii) the aspectual themes identification and mapping into separate features promote an improved modularity of the feature model.

The combination of Theme/SPL DSL and the transformation rules produce feature models more quickly without sacrificing quality. Our approach offers a rigorous, reliable (since everything is modeled explicitly), auditable and efficient way to produce feature models.

As future work we will apply the approach to other real case studies. The next step is to extend the whole framework for Theme/UML.

10. ACKNOWLEDGMENTS

The authors would like to thank the AMPLE project and CITI – PEst-OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012) – for the financial support for this work.

11. REFERENCES

[1] AMPLE Project. http://www.ample-project.net/. Last Access: August 2012

[2] Apel, S., Kästner, C., Lengauer, C. 2009. FeatureHouse: Language-Independent, Automated Software Composition, ICSE 2009, IEEE Computer Society.

[3] Atlas Transformation Language (ATL), User Guide. http://wiki.eclipse.org/ATL/.

[4] Batory, D. 2005. A Tutorial on Feature Oriented Programming and the AHEAD Tool Suite, GTTSE'05, LNCS 4143, Springer (2006) 3–35.31.

[5] Carton, A., Driver, C., Jackson A., Clarke, S.: Model-Driven Theme/UML, Transactions on AOSD, Special issue on MDE, vol. VI, LNCS 5560 (2009).

[6] Clarke, S., Baniassad, E. 2005. Aspect Oriented Analysis and Design, Addison-Wesley Professional.

[7] Czarnecki, K., Helsen, S., Eisenecker, U. 2004. Staged Configuration Using Feature Models, Software Product Lines. SPLC 2004, Boston MA, USA.

[8] Groher, I., Voelter, M. 2009. Aspect-Oriented Model-Driven Software Product Line Engineering. T. Aspect-Oriented Software Development VI 6: 111-152.

[9] Heidenreich, F., Kopcsek, J., Wende, C. 2008. FeatureMapper: Mapping Features to Models. ICSE'08, New York, NY, USA, ACM 943–944.

[10] Kelly, S., Tolvanen, J. 2008. Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Computer Society ().

[11] Jayaraman, P., Whittle, A.M., Elkhodary, Gomaa, H. 2007. Model Composition and Feature Interaction Detection in Product Lines Using Critical Pair Analysis, MoDELS'07, Springer.

[12] Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report: CMU/SEI-90-TR-021, USA.

[13] Nielsen, J. 1993. Usability Engineering, Morgan Kaufman.

[14] Pohl, K., Böckle, G., van der Linder, F. 2005. Software Product Line Engineering Foundations, Principles, and Techniques, Springer.

[15] Rashid, A., Moreira, A., Araújo, J. 2003 Modularisation and Composition of Aspectual Requirements, AOSD'03, ACM press.

[16] Runeson, P. 2003. Using Students as Experiment Subjects - An Analysis on Graduate and Freshmen Student Data, EASE'03, Staffordshire, UK.

[17] Trujillo, S., Batory, D., Díaz, O. 2007. Feature Oriented Model Driven Development: A Case Study for Portlets. ICSE'07: 44-53

[18] Voelter, M., Stahl, T.: "Model-Driven Software Development - Technology, Engineering, Management", Wiley (2006).

[19] Weston, N., Chitchyan, R., Rashid, A.: A Framework For Constructing Semantically Composable Feature Models from Natural Language Requirements, SPLC'09, USA (2009).

[20] Witte, R., Witte, J. 2010. Statistics, 9th edition, Wiley.

[21] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A. 1999. Experimentation in Software Engineering, Vol. 6, Kluwer, USA.