Model-Driven Generation of Context-Specific Feature Models

Thibaut Possompès*, Christophe Dony*, Marianne Huchard*, Chouki Tibermacine*
*LIRMM, CNRS and Montpellier 2 University
Montpellier, France
{possompes, dony, huchard, tibermacin}@lirmm.fr

Abstract—Software Product Lines (SPL) aim at deriving software architectures or systems from a software artifact base. Configuring the SPL to derive a new product is now usually done by selecting appropriate software features in a kind of models, called feature models. In some situations, a feature represents a software artifact associated to an element $e$ of a context the software product will manage. Such a feature and its associated software artifact may be cloned according to the number of occurrences of $e$ in the context. Hence, the feature model proposed to users for configuration has to be adapted in a new dedicated phase according to the context elements. We propose a model-driven engineering approach for transforming a generic feature model according to a context model that a derived software product will manage. More precisely this paper describes an original model transformation able to generate context specific feature models including duplicated features, and removing inappropriate features. Our transformation is validated on a smart building optimization software case study.

I. INTRODUCTION

Configuration options of a software product line (SPL) to generate a new product are nowadays commonly represented with a feature model [8]. A feature represents components or functionalities that an instance of the product line can offer. A feature model indicates which choices (features) are mandatory or optional in some conditions, and how one choice can impact another one (feature inter-dependencies and constraints). Software products (applications) are built from a SPL by selecting features from such a feature model. A set of selected features is commonly called a product configuration.

In various situations, some features are semantically associated to elements of a context generic model, that describes concepts, that can be present in the context in which (or for which) the application will be deployed. For example in the case of a smart building energy-optimization software, the feature “solar optimization” is semantically associated to the context generic model concept Solar-panel and should only be proposed in a configuration feature model if the building to be optimized (described by a model, instance of the context generic model) has some solar panels (instances of Solar-panel). Corollary, such a software could be configured to manage differently each occurrence of a given electronic appliance or physical infrastructure [2].

Hence, a product configuration depends, on the one hand, on the product features, and on the other hand, on the number of occurrences of the context concept instances. In a feature model, cloning features and identifying inappropriate ones is a long and error prone task because each context-specific feature must be checked regarding each context concept instance.

Various approaches have been proposed to configure a feature model according to a context [5], [7]. In this paper we propose an original solution to automatically generate an optimized feature model (called a context-specific feature model), that conforms to a given deployment context. A context-specific feature model only includes features that make sense in the context. The features are appropriately cloned depending on the number of context elements. Our model transformation algorithm uses as inputs a generic feature model (representative of a SPL global set of functionalities), a model of context concepts and an instance of the model of the context.

We validate our proposal in the context of the development of a smart building management system software product line (RIDER project [10]). The project aims at creating a global intelligent system to perform energy optimization.

Section II presents a motivating example from the RIDER project. Section III presents a global view of our approach. Section IV details the algorithms used for performing the adaptation of feature models. Section V discusses the related work, and Section VI concludes this paper and gives several perspectives for this work.

1The RIDER project (“Research for IT as a Driver of Energy efficiency”) – http://rider-project.com/ – is led by a consortium of several companies and R&D laboratories, including IBM and the LIRMM, interested in improving building energy efficiency.
II. MOTIVATING EXAMPLE – A SMART BUILDING CASE STUDY

A RIDER software product purpose is to enhance lighting, heating, ventilation and air conditioning usages to save energy in buildings. A RIDER product is made of interfaces with building management systems (BMS), of several optional modules to add further functions (physical simulation, optimization algorithms, visualization tools, etc.), and a component allowing to orchestrate input and output data. The data orchestration component purpose is to decide how to manage incoming data and energy optimization computation results. For example, the physical simulation module requires data related to spaces that are instrumented with temperature and humidity sensors. A RIDER software product uses a representation of the building it will drive. This representation is also called building information model (BIM). It is able to represent static (e.g., blueprints) as well as dynamic (e.g., sensor measures) information.

An instance of the building model is used as a cornerstone to leverage information from building managers and energy optimization experts [6], [9]. It can gather into a single model information such as 3D geometric data for visualization, electric, and Heating Ventilation and Air Conditioning (HVAC) blueprints, the various building components along with their size and physical properties for simulation purposes, but also cost and project management-related information.

Each additional module function is modeled in the RIDER feature model. Some of them are related to the elements of the BIM. For example, 3D information is required to provide 3D visualization features. If it is missing then 3D visualization features are not available. If the information is available on some parts of the building, the visualization features are available only on those parts. When configuring a new product, it is important to know which parts of the building will be properly optimized, and to know which new equipments must be added to allow these features to properly work. More generally, each feature requires to consider if it can be duplicated, which context elements determine how many times it can be duplicated, and which constraints must be satisfied by the context element to make the feature available.

Next section introduces our approach and describes the four models involved in this approach.

III. APPROACH OVERVIEW

Let us introduce in this section our terminology and abstract our approach. A generic feature model (FM) of a software product line application represents the features globally available in the application (called generic features). Each generic feature can have a semantic relation (depends on) with one or several context model concepts. A context-specific feature model (CSFM) represents features relevant to a given context model instance (called context-specific features). Each context-specific feature (CSF) of such a model relates to a generic feature of the FM and, if this generic feature depends on a context concept, to one context concept instance. Our purpose is to automatically generate a CSFM, made of all possible CSFs, by analyzing associations between features of a generic feature model with concepts of a context model (e.g., a building meta-model) and their instances (elements of a concrete building).

The obtained CSFM allows stakeholders to choose CSFs for creating a product configuration adapted to the environment.

Our approach integrates the four models shown in Figure 1. The context model (CM) describes the context information that a software product manages. It can be presented by creating a domain specific language (DSL) or a UML model. CM is a set of connected concepts. In our case study, this model is the building infrastructure model which contains concepts such as, Building, Storey, Zone, Space, Sensor. It is created with the help of context domain specialists. Some concepts are hierarchically related with a specific relation. For example, we have the hierarchical relation Building → Storey → Space → Sensor.

The generic feature model (FM) is, for a given CM, a multiplicity-based feature model which is based upon Czarnecki et al. [3] definition. We extended it to make it possible to associate a feature to a CM concept. It is composed of features (denoted by \( f \) ) organized in a tree. The root of this tree is denoted by \( r_f \). A feature associated with context concepts can be duplicated according to the instances of those concepts. In our case study, it is used to describe all the possible features of an energy optimization software. In our approach, this model is not directly used to configure a new product. This model must be first adapted to a specific context, which is described by the context model instance.
A feature can be associated to none, one or several concepts of the \textit{CM}. It means that the feature can be duplicated for each instance occurrence of an associated concept. A constraint, associated to a feature, can be used to determine which properties must have an instance to duplicate a feature. For example, when \textit{CM} is written in UML, constraints are in OCL. A given feature \( f \) can have a group \( g \) gathering its sub-features. A group is used to specify how many grouped features can be selected.

The context model instance (CMI) is an instance of the context model. It describes instances of the concepts of \textit{CM}. If the \textit{CM} is similar to a set of UML classes, the \textit{CMI} is similar to a set of UML instances. The elements of the \textit{CMI} model are called \textit{instances}. There is also an hierarchical relation between the instances of concepts hierarchically related. It describes the specific context that will be managed by a software product. In our case study, it consists in modeling a specific building that will be managed by a new energy optimization software product, instance of the software product line. It is created with the help of building owners and managers.

The \textit{CSFM} is a kind of feature model resulting from the adaptation of \textit{FM} to a given context \textit{CMI}. It is the set of context-specific features (CSF) that can be chosen to build a new product to be used in a given context. A CSF associates a feature \( f \), to an instance \( i \). \( i \) must be an instance of one of the concepts associated to \( f \). If \( f \) is not associated to a concept then \( i = \emptyset \). The CSF are organized as a tree whose root is denoted by \( r_\varphi \). A CSF \( \varphi \) can have a group \( g_\varphi \) gathering its sub-features. \( g_\varphi \) must be related to an existing group \( g \) of the feature \( f \), \( f \) being associated to \( \varphi \). In our case study, a realistic \textit{CMI}, of a building \( b \), can have hundreds of instances that have to be considered to create the \textit{CSFM} to configure the product for \( b \). Our approach proposes an algorithm to generate this model automatically.

The models \textit{CM}, \textit{FM}, and \textit{CMI} are provided as input of our adaptation process. The output of the process is a \textit{CSFM} to be filled to create a new product configuration. Groups and multiplicities are also adapted in the \textit{CSFM}. The constraints associated to concepts are checked after having generated the \textit{CSFM}. We do not detail constraints checking here due to space limitation. Figure 2 depicts excerpts of a \textit{CM}, \textit{CMI}, \textit{FM}, and a \textit{CSFM}. The features \textit{TempOptim}, \textit{ScheduleOptim} and \textit{PresenceOptim} are associated to the concept \textit{Space} (only one link is shown to simplify the diagram). The \textit{CSFM} generation algorithm duplicated the feature sub-tree whose root is \textit{TempOptim} two times. The duplicated CSF sub-trees are associated respectively to the instances \( s1 \) and \( s2 \).

The next section presents the algorithm generating \textit{CSFMs}.

IV. CONTEXT-SPECIFIC FEATURE MODEL GENERATION ALGORITHM

The \textit{CSFM} generation algorithm traverses the feature model in depth-first order. We consider that the models \textit{FM}, \textit{CMI}, \textit{CM}, and \textit{CSFM} (which is empty at the beginning) are global data common to all following algorithms.

Algorithm 1 initializes the \textit{CSFM} generation algorithm and returns the resulting \textit{CSFM}. It creates the root context specific feature and, for each sub-feature of the root feature of the feature model, calls the recursive procedure \texttt{FeatureTreeTraversal} to build the \textit{CSFM}.

**Algorithm 1:** Main procedure of the \textit{CSFM} generation algorithm

**Input:** The models \textit{CM}, \textit{CMI}, et \textit{FM}

**Result:** A \textit{CSFM} model built according to the \textit{CM}, \textit{CMI}, et \textit{FM} models

Initialize an empty \textit{CSFM}.

\( r_\varphi \) is the root CSF of \textit{CSFM}, it is associated to the root feature \( r_f \) of \textit{FM}.

\textbf{foreach} sub-feature \( f \) of the \textit{FM} root feature \( r_f \) \textbf{do}

\texttt{FeatureTreeTraversal} \( (f, r_\varphi) \)

\textbf{end}

The procedure \texttt{FeatureTreeTraversal} builds the \textit{CSFM} recursively. It requires two parameters: A feature \( f \) for which related CSFs will be created, a CSF \( \varphi_{\text{parent}} \) which will be the parent of the created CSFs.

The CSFs are created differently in three cases:
Procedure \text{featureTreeTraversal}(f, \varphi_{\text{parent}})

**Input:** A feature \( f \) from which the FM is traversed. A parent CSF \( \varphi_{\text{parent}} \), such that \( f \) parent feature is associated to \( \varphi_{\text{parent}} \).

**Result:** Updates the CSFM according to the CM, CMI, and FM models.

\[ \text{if } f \text{ is not associated to a concept and } \varphi_{\text{parent}} \text{ is not associated to an instance then} \]
\[ \varphi = \text{addSpecificFeature}(f, \varphi_{\text{parent}}, \emptyset) \]

\[ \text{foreach sub-feature } f' \text{ of } f \text{ do} \]
\[ \text{FeatureTreeTraversal}(f', \varphi) \]
\[ \text{end} \]

\[ \text{if the CSF parent } \varphi_{\text{parent}} \text{ is not associated to an instance and } f \text{ is associated to a concept then} \]
\[ \text{foreach instance } i \text{ that is an instance of the concept associated to } f \text{ do} \]
\[ \varphi = \text{addSpecificFeature}(f, \varphi_{\text{parent}}, i) \]
\[ \text{foreach sub-feature } f' \text{ of } f \text{ do} \]
\[ \text{FeatureTreeTraversal}(f', \varphi) \]
\[ \text{end} \]
\[ \text{end} \]

\[ \text{if the CSF } \varphi_{\text{parent}} \text{ is associated to an instance and } f \text{ is associated to a concept then} \]
\[ \text{foreach instance } i \text{ which is either the same instance that is associated to } \varphi_{\text{parent}} \text{ and that is an instance of the concept associated to } f, \text{ or an instance that is hierarchically below the instance associated to } \varphi_{\text{parent}} \text{ and that is an instance of a concept associated to } f \text{ do} \]
\[ \varphi = \text{addSpecificFeature}(f, \varphi_{\text{parent}}, i) \]
\[ \text{foreach sub-feature } f' \text{ of } f \text{ do} \]
\[ \text{FeatureTreeTraversal}(f', \varphi) \]
\[ \text{end} \]
\[ \text{end} \]

1) The evaluated feature \( f \) is not associated to any concept and the parent CSF \( \varphi_{\text{parent}} \) is not associated to an instance. Then, one CSF is created, and the procedure is called recursively for each sub-feature of \( f \).

2) The parent CSF \( \varphi_{\text{parent}} \) is not associated to an instance and the evaluated feature \( f \) is associated to a concept. Then, a CSF is created for each instance whose concept is associated to \( f \), and the procedure is called recursively for each instance and for each sub-feature of \( f \).

3) The parent CSF \( \varphi_{\text{parent}} \) is associated to an instance and the evaluated feature \( f \) is associated to a concept. A CSF is added with the same instance as \( \varphi_{\text{parent}} \) and for each context concept instance which is hierarchically below the instance associated to \( \varphi_{\text{parent}} \). The procedure is then called recursively for each sub-feature of \( f \) and for each instance hierarchically below \( \varphi_{\text{parent}} \) instance.

The function \text{addSpecificFeature} creates a new CSF in the CSFM. It requires three parameters: the feature \( f \) which will be referenced by the CSF, the parent CSF \( \varphi_{\text{parent}} \), and an instance that will be also referenced by the CSF. As seen before, a CSF references a feature, and either an instance or nothing. First, a new CSF \( \varphi \) is created. Its parent CSF is \( \varphi_{\text{parent}} \), and it is associated to \( f \) and \( i \).

The first condition of this function identifies a special case: when \( \varphi \) is duplicated according to instances and its parent is not. In this case, a new CSF group must be created in which all duplicated CSF related to the same feature must be included. This group does not exist in the FM, its purpose is to reflect the multiplicity of \( f \) and ensure that the right number of CSF can be selected. The group is not created when there is only one CSF. Instead, \( \varphi \) is included as an optional sub-CSF when \( f \) minimum multiplicity is equal to 0, or as a mandatory sub-CSF when \( f \) multiplicity is equal to 1. Otherwise, \( \varphi \) must be included in the CSFM with respect to the feature groups described in the FM. Hence, the procedure \text{addNewCSFtoGroup} is called.

The procedure \text{addNewCSFtoGroup} updates the CSFM by creating groups considering those existing in the FM. It takes two parameters: the new CSF \( \varphi \), and its parent \( \varphi_{\text{parent}} \). If the feature \( f \) associated to \( \varphi \) belongs to a feature group \( g \) then \( \varphi \) must also belong to a group \( g_{\varphi} \) related to \( g \). The group is created only when there is at least two CSF in it. Otherwise, the CSF is either mandatory or optional according to \( f \) multiplicity.

V. RELATED WORK

Formal semantics of feature models have been defined in \cite{13} for many different kinds of feature models. We chose a semantics based upon Czarnecki et al. cardinality-based feature models \cite{3} as described in \cite{11}. They created a staged configuration process \cite{4} in which they specialize the feature model to restrict the multiplicity of features. It is not applicable to our situation, because a CSFM is not a specialization of a generic feature model. Indeed, each context-specific feature adds information about the context concept instance it is associated with. Even if our case study do not require a staged-configuration process, it could be applied to our work. The generic feature model could be specialized into a refined generic feature model.
**Function** addSpecificFeature(f, ϕ_parent, i) : ϕ

**Input:** a feature f, a CSF ϕ_parent, and an instance i

**Output:** Updates the CSFM with a new CSF, and returns the new CSF. Creates a new CSF ϕ, sub-CSF of ϕ_parent, associated to the feature f and the instance i. Its multiplicity is the same as f.

If ϕ is associated to an instance, and ϕ_parent is not associated to an instance or a different one then

- If there is no existing CSF group in ϕ_parent in which there is other CSF associated to the same feature then
  - Adds ϕ as an optional sub-CSF of ϕ_parent if f minimum multiplicity is equal to 0, or a mandatory sub-CSF if f minimum multiplicity is equal to 1.

else if there is no existing CSF group in ϕ_parent in which there is other CSF associated to the same feature then

  Creates a CSF-group with the same multiplicity as f, and adds ϕ and ϕ’ to this group.

else

  Adds ϕ to the existing CSF group in which all CSF are associated to the same feature as ϕ.

end

else

  addNewCSFtoGroup(ϕ, ϕ_parent)
end

**Procedure** addNewCSFtoGroup(ϕ, ϕ_parent)

**Input:** The CSF ϕ and ϕ_parent such that ϕ must be a sub-CSF of ϕ_parent.

**Result:** Updates the CSFM to make ϕ a sub-CSF of ϕ_parent.

Let f being the feature associated to ϕ.

- If f belongs to a feature group g then
  - If there is a CSF group g_ϕ in ϕ_parent related to g then
    - Adds ϕ to the group g_ϕ.
  - Else if there is already a CSF ϕ’ whose feature belongs to the same group then
    - Creates a new CSF group related to the group g, and adds ϕ and ϕ’ to this group.
  - Else

Acher et al. [1] work in the context of self adaptive and dynamic systems. They are interested by run time adaptation while we are concerned by the design time adaptation. They bind a context model, modeled with a feature model, with a feature model describing the application. Changes applied in the context feature model are automatically reflected on the application configuration model thanks to ECA rules [13]. We propose to adapt the feature model rather than a configuration model. In our case, the context is a business model provided by some stakeholders. This model is also used by the application to describe the managed data and not specifically created for the product line specification.

Quinton et al. [12] derive software products in the context of applications for mobile phones. They consider a feature model for the application and a feature model for mobile devices. They configure and generate an application model using the application feature model, and then check if the model is consistent with a set of mobile devices represented by the mobile device feature model. We address a different problem: the business model (e.g., a building model) is imposed and we have to propose to the stakeholders a feature model adapted to the business model, in order to allow them to configure an application consistent with their environment.

**VI. CONCLUSION**

We presented in this paper an approach able to adapt generic feature models to a business context. It allows
us to produce a CSFM according to a context model instance representing the context in which the future product will run. Our approach allows to automatically determine whether each feature related to a context concept can be cloned in a given context by checking constraints against context concept instances. Hence, each generated product configuration is specific to the instance of the context model it has been made with. Then, it enables us to generate software product architectures and implementations specific to a context. This approach automates a process that would otherwise requires to compare hundreds of features with hundreds of context concept instances.

Our methodology has been designed in a generic way to be reused in different domains. The prototype has been implemented with UML models and UML profiles, for modeling the context model and the generic feature model. We developed a tool as an Eclipse RCP platform. The Eclipse RCP platform takes as input the XML files representing the business model and the feature model. They are generated by an XSLT transformation from the XMI versions of the UML models.

We validated our approach in the RIDER project on a building meta-model used to describe smart buildings. The building meta-model has been modeled as a UML model on which classes and associations were stereotyped to represent navigable elements. The feature model was built with our UML profile for feature models [11].

Next, we intend to create views on the CSFM to facilitate feature selection. They could show features related to a stakeholder concern, or allow choosing several features at the same time, e.g., all the clones of a feature. In future work, we want to enable the configuration of new products according to features existing in other products to facilitate their communication.

REFERENCES


