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 and constraints have to be respected. 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 [9]. 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`). As a 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 [6], [8]. 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[1] 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.

[1]The 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 [7], [5]. 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* $\rightarrow$ *Storey* $\rightarrow$ *Space* $\rightarrow$ *Sensor*.

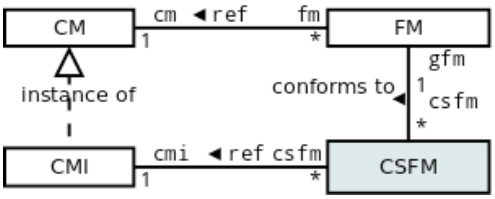

Figure 1. Approach overview

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. This association has a multiplicity. 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 and their multiplicity. 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.

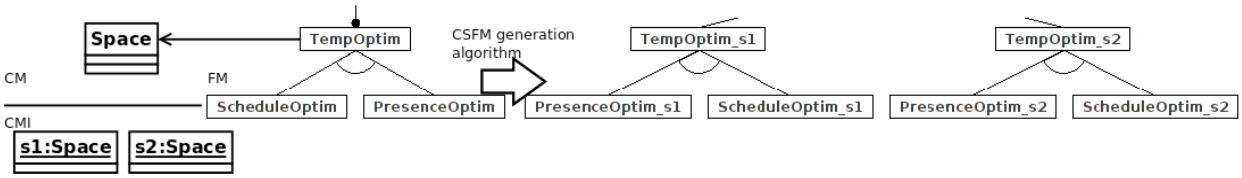

Figure 2. CSFM generation example

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 CM. It means that the feature can be duplicated for each instance 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 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 CM. If the CM is similar to a UML class diagram, the CMI is similar to a UML instance diagram. The elements of the CMI model are called *instances*. There is also an hierarchical relation between the instances of concepts hierarchically related. The CMI 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 *CSFM* is a kind of feature model resulting from the adaptation of FM to a given context 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 $i = \varnothing$. The CSF are organized as a tree whose root is denoted by r_φ. A CSF φ can have a group g_φ gathering its sub-features. g_φ must be related to an existing group g of the feature f, f being associated to φ. In our case study, a realistic CMI, of a building b, can have hundreds of instances that have to be considered to create the CSFM to configure the product for b. Our approach proposes an algorithm to generate this model automatically.

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

The next section presents the algorithm generating $CSFM$s.

IV. Context-Specific Feature Model Generation Algorithm

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

Algorithm 1 initializes the CSFM generation algorithm and returns the resulting 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 featureTreeTraversal to build the CSFM.

Algorithm 1: Main procedure of the CSFM generation algorithm
Input: The models *CM*, *CMI*, et *FM*
Result: A *CSFM* model built according to the *CM*, *CMI*, et *FM* models
Initialize an empty CSFM.
r_φ is the root CSF of CSFM, it is associated to the root feature r_f of FM.
foreach *sub-feature* f *of the FM root feature* r_f **do**
`featureTreeTraversal` (*f*, r_φ)
end

The procedure featureTreeTraversal builds the CSFM recursively. It requires two parameters: A feature f for

which related CSFs will be created, a CSF φ_{parent} which will be the parent of the created CSFs.

Procedure featureTreeTraversal(f, φ_{parent})

Input: A feature f from which the FM is traversed. A parent CSF φ_{parent}, such that f parent feature is associated to φ_{parent}.

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

if *f is not associated to a concept and φ_{parent} is not associated to an instance* **then**
  $\varphi =$ `addSpecificFeature(`$f, \varphi_{parent}, \varnothing$`)`
  foreach *sub-feature f' of f* **do**
    `featureTreeTraversal(`f', φ`)`
  end
if *the CSF parent φ_{parent} is not associated to an instance and f is associated to a concept* **then**
  foreach *instance i that is an instance of the concept associated to f* **do**
    $\varphi =$ `addSpecificFeature(`f, φ_{parent}, i`)`
    foreach *sub-feature f' of f* **do**
      `featureTreeTraversal(`f', φ`)`
    end
  end
if *the CSF φ_{parent} is associated to an instance and f is associated to a concept* **then**
  foreach *instance i which is either the same instance that is associated to φ_{parent} and that is an instance of the concept associated to f, or an instance that is hierarchically below the instance associated to φ_{parent} and that is an instance of a concept associated to f* **do**
    $\varphi =$ `addSpecificFeature(`f, φ_{parent}, i`)`
    foreach *sub-feature f' of f* **do**
      `featureTreeTraversal(`f', φ`)`
    end
  end

The CSFs are created differently in three cases:

1) The evaluated feature f is not associated to any concept and the parent CSF φ_{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 φ_{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 φ_{parent} is associated to an instance and the evaluated feature f is associated to a concept. A CSF is added either with the same instance as φ_{parent} or with each context concept instance which is hierarchically below the instance associated to φ_{parent}. The procedure is then called recursively for each sub-feature of f and for each instance hierarchically below φ_{parent} instance.

The function addSpecificFeature creates, and returns, a new CSF in the CSFM. It requires three parameters: the feature f which will be referenced by the CSF, the parent CSF φ_{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 φ is created. Its parent CSF is φ_{parent}, and it is associated to f and i. The lower bound of its multiplicity is equal to the maximum between the lower bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f. The upper bound of its multiplicity is equal to the minimum between the upper bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f.

Then, the procedure addNewCSFtoGroup is called to add the new CSF to a group if f belongs to a group in the FM. If the new CSF is associated to an instance, it also must belong to a group whose multiplicity is the same as the feature associated to the new CSF. This group does not exist in the FM. Its purpose is to transpose in the $CSFM$ the multiplicity of the feature f to guarantee that the number of CSF that can be chosen in a configuration respects f multiplicity.

The procedure addNewCSFtoGroup updates the CSFM by creating groups considering those existing in the FM. It takes two parameters: the new CSF φ, and its parent φ_{parent}. If the feature f associated to φ belongs to a feature group g then φ must also belong to a group g_φ 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 [13] for many different kinds of feature models. We chose a semantics based upon Czarnecki et al. cardinality-based feature models [3] as described in [14]. They created a staged configuration process [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

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. The lower bound of its multiplicity is equal to the maximum between the lower bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f. The upper bound of its multiplicity is equal to the minimum between the upper bounds of the multiplicity on the relationship between the concept whose i is the instance and f, and of the multiplicity on f. The procedure `addNewCSFtoGroup`($\varphi, \varphi_{parent}$) is called to add φ to a group is f belongs to a group in the FM.

if *φ is associated to an instance* **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 to the CSF group.
  else
    Creates a CSF-group with the same multiplicity as f, and adds φ to this group.
  end
end
return φ

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
    Creates a new CSF group related to the group g, and adds φ and φ' to this group.
  end
end

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 does 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 before generating the CSFM, and the CSFM could be specialized into a refined CSFM and configured through a staged configuration process.

There are several solutions allowing to perform product configuration choices according to a given context. Voelter et al. [15] detail an approach where negative and positive variability are used to remove or add concepts to a custom DSL which seems to correspond to our business model. However, their approach could not solve our concerns because we needed to adapt the feature model. We address the opposite concern, we adapt the feature model to an imposed business model.

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 [12]. 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. [11] 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 [14].

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

[1] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, and J. P. Rigault. Modeling context and dynamic adaptations with feature models. In *4th International Workshop Models@ run. time at Models*, volume 9, 2009.

[2] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. Generative programming for embedded software: An industrial experience report. In D. Batory, C. Consel, and W. Taha, editors, *Generative Programming and Component Engineering*, volume 2487, pages 156–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[3] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their staged configuration. *University of Waterloo*, 2004.

[4] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Lecture notes in computer science*, volume 3154, page 266–283, 2004.

[5] Daniel Kullmann and Henrik W. Bindner. Using rules in high-level communication for the control of power systems. In *Proceedings of the 2nd International Conference on Microgeneration and Related Technologies*, 2011.

[6] P. Fernandes, C. Werner, and E. Teixeira. An approach for feature modeling of context-aware software product line. *Journal of Universal Computer Science*, 17(5):807–829, 2011.

[7] A. Grilo and R. Jardim-Goncalves. Challenging electronic procurement in the AEC sector: A BIM-based integrated perspective. *Automation in Construction*, 2010.

[8] Z. Jaroucheh, X. Liu, and S. Smith. Mapping features to context information: Supporting context variability for context-aware pervasive applications. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 611 –614, 2010.

[9] K. C. Kang, J. Lee, and P. Donohoe. Feature-oriented product line engineering. *IEEE software*, page 58–65, 2002.

[10] T. Possompès, C. Dony, M. Huchard, H. Rey, C. Tibermacine, and X. Vasques. Towards software product lines application in the context of a smart building project. *Proceedings of the 2nd International Workshop on Model-driven Product Line Engineering (MDPLE 2010)*, pages 73—84, 2010.

[11] C. Quinton, S. Mosser, C. Parra, and L. Duchien. Using multiple feature models to design applications for mobile phones. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, page 23:1–23:8, New York, NY, USA, 2011. ACM.

[12] Raphael Romeikat, Bernhard Bauer, and Henning Sanneck. Modeling of domain-specific ECA policies. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 52—58, Miami Beach, USA, July 2011.

[13] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, Feb. 2007.

[14] Thibaut Possompès, Christophe Dony, Marianne Huchard, and Chouki Tibermacine. Design of a UML profile for feature diagrams and its tooling implementation. In *Software Engineering and Knowledge Engineering (SEKE 2011)*, pages 693—698, Miami, FL, USA, 2011.

[15] M. Voelter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 233–242. IEEE, 2007.