Software Product Line Evolution with Cardinality-Based Feature Models

Nadia Gamez and Lidia Fuentes
Dpto de Lenguajes y Ciencias de la Comunicación, Universidad de Málaga
{nadia,lff}@lcc.uma.es

Abstract. Feature models are widely used for modelling variability present in a Software Product Line family. We propose using cardinality-based feature models and clonable features to model and manage the evolution of the structural variability present in pervasive systems, composed by a large variety of heterogeneous devices. The use of clonable features increases the expressiveness of feature models, but also greatly increases the complexity of the resulting configurations. So, supporting the evolution of product configurations becomes an intractable task to do it manually. In this paper, we propose a model driven development process to propagate changes made in an evolved feature model, into existing configurations. Furthermore, our process allows us to calculate the effort needed to perform the evolution changes in the customized products. To do this, we have defined two operators, one to calculate the differences between two configurations and another to create a new configuration from a previous one. Finally, we validate our approach, showing that by using our tool support we can generate new configurations for a family of products with thousands of cloned features.

Keywords: Software Product Lines, Feature Models, Evolution.

1 Introduction

Recently, pervasive systems and Ambient Intelligence environments are gaining popularity to support people’s daily tasks. These systems are composed by a large variety of networked heterogeneous devices with embedded software. For instance, Ambient Assisted Living systems or Intelligent Transportation Systems (ITS) can be formed by a large number of sensor nodes (grouped in Wireless Sensors Networks, WSNs), smart phones, vehicles onboard computers or other devices with RFID s or cameras. Application domains like pervasive systems, where heterogeneity is present at any level, can greatly benefit from Software Product Line (SPL) engineering [1], which is specifically focused on variability modelling. SPLs aim to provide techniques for creating infrastructures that allow the rapid and systematic production of similar software systems, promoting the reuse of common core assets.

Feature Models (FM) [2] have been widely adopted by the SPL community to specify which elements, or features, of the family of products are common, which are variable and the reasons why they are variable, i.e. if they are alternative elements or optional elements. Then, a feature model permits specifying where the variability is,
independently of the core asset, and enables reasoning about all the different possible configurations of a family of products.

Specifically in heterogeneous pervasive environments, the most common variability is the structural variability, defined as variations in type, cardinality or naming of elements [3]. We propose using cardinality-based features models and clonable features [4] to model the structural variability present in the new generation of pervasive systems. The use of clonable features increases the expressiveness of FMs since they allow the creation of different configurations for the same kind of device using only one feature model. Using clonable features we can model so that a system has a variable number of different kinds of devices (e.g. s sensors, c cameras, a alarms, or sm smartphones). The cloning of these device features leads to the cloning of the related structure (e.g. for 3 sensors, the configuration will contain s1, s2 and s3 clones of the sensor feature, joint with its sub-tree), increasing the complexity of the resulting configurations, and moreover the number of possible configurations increases a lot.

Then, as the FM evolves, the impact of propagating changes made in the FM to the possible configurations is much higher in a cardinality-based FM.

Evolving a FM may imply adding or removing a feature (e.g. adding a new encryption algorithm as a mandatory feature), which in a cardinality-based feature model may cause many changes in all the clones. Specifically in pervasive systems, configurations could have hundred of clones composing a single product configuration. So, considering the evolution of a concrete SPL, it would be useful to automatically obtain the evolved configurations according to the changes introduced to the FM. From the point of view of the SPL engineer, it would be useful to know the necessary effort to evolve a previous existing product configuration to a new valid configuration after a FM modification was performed. This effort could be calculated by comparing the previous and the list of new possible configurations; which is not trivial to do at first glance due to the high number of cloned features.

In this paper, we present how we manage automatically the evolution of an pervasive system software product lines using cardinality-based FM and clonable features. To do this, we have defined two operators between FM configurations that are not trivial for cardinality-based FM. The create_configuration operator allows the creation of a new configuration from a previous configuration and the features that must be added or removed in the new configuration. The differences operator calculates the differences between two configurations of a feature model. We use the create_configuration operator to create evolved configurations from the previous configuration and the evolved feature model. Furthermore, we use the differences operator to calculate the effort of evolving the product configurations of a SPL, reusing and preserving the elements of the previous configuration. Finally, we validate our approach showing that by using our tool support we can easily evolve FMs with clonable features, automatically generating new configurations, for configurations with a high number of clones.

The remainder of the paper is organized as follows. In Section 2, we present our motivation example and the challenges for evolving pervasive systems SPLs and how we achieve them. In Section 3, we show our approach and Section 4 details the differences and create_configuration operators. The validation and the tool support of our approach are presented in Section 5. In Section 6, we compare our approach with related work. Finally, in Section 7 we outline some conclusions.