Product Line Unified Modeler (PLUM)

Aitor Aldazabal
European Software Institute,
Zamudio, Bizkaia, Spain
Aitor.Aldazabal@esi.es

Sergey Erofeev
European Software Institute,
Zamudio, Bizkaia, Spain
Sergey.Erofeev@esi.es

1. Introduction

The Product Line Unified Modeler (PLUM) is a tool suite under development in the European Software Institute which is meant to aid creation, management and exploitation of Software Product Lines [1] [2] by making use of the Eclipse framework's wide array of features and projects.

A software product line (SPL) can be defined as a set of software intensive systems sharing a common, managed set of features that satisfies specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [3].

PLUM suite makes use of the concepts obtained as a result from past international research projects like FAMILIES [4], CAFE [5] or ESAPS [6] as well as brand new concepts from ongoing projects like MoSiS and FLEXI to represent Product Lines into models, and the variability of the products inside them. Implementations of these concepts have already been materialized in a wide array of independent tools such as the two versions of VManage from ESI [7], Gears from BigLever [8] and pure::variants from Pure Systems [9].

The main difference between previous implementations and PLUM resides in its objective, which is to **provide several approaches** (direct product variability, model driven variability and so on) to software product line management, unified in a tool, providing several layers of compatibility between each approach. This makes compatibility and interoperability between modeling tools a key feature of our suite.

PLUM is under development using Eclipse and designed to be compatible and embeddable within the framework [10], making use of known and widely used tools and plug-ins both for its development and as building blocks to implement the features required by a SPL tool.

Special emphasis it's being made in the EMF [11] and GMF [12] Eclipse projects and also in the OAW [13] project. These projects compose the core building blocks and tools for the first approach implemented at PLUM, the Direct Product Variability (DPV).

The DPV approach focuses on identifying variability in text based representations of each product, source code for example, and structuring the variable features in model based representations. These models represent the variability for a family. This approach will provide a series of editors to enable resolving the variability. Once variability is resolved, a series of easily configurable transformations can then be applied to these models in order to obtain different products. The Direct Product Variability approach provides the concepts both for building up a product family and for its exploitation.

EMF, GEF and GMF will provide the basic building block for the models, their editors, and the basic transformations, while OAW's engine will provide the tool with a powerful, mature and highly extensible transformation and workflow engine which will be used to create the actual products.

For the integration and interoperability between modeling tools however, the main Eclipse project we are going to use is the Model Driven Development Integration [14] project which features a series of resources like the Model Bus which will allow us for a standardized approach to interoperability among other benefits.

2. Introduction to Variability.

There are several definitions [15] and several approaches to Variability Management, most of them applied or envisioned to be applied in Software Product Line engineering processes. Several tool vendors like BigLever [8] or Pure Systems [9] and research institutes like ESI [7] have created their own approach to variability management. ESI's approach was derived from results obtained in international research projects like FAMILIES [4], CAFÉ [5] or ESAPS [6]. Lately a new project, MoSiS has been launched to create a **unified language for variability**

modelling targeted at Software Intensive Systems. Additionally another international project called FLEXI is focused in applying of variability in large organizations.

This paper will rely on the latest version of one of ESI's approaches for variability called **Direct Product Variability**, since it is the one being currently implemented in PLUM.

The Direct Product Variability is designed to represent the variable features of a product family in what is called a Decision Model. The Decision Model is a structure which conceptually resembles the questions of a test. It defines the type of decision (multiple choice, single choice, range value, etc.), the description of the decision, the type of data expected as an answer and depending on the type of the decision it also defines the range of values or set of choices from which to pick the answer. It also provides features which allow grouping a series of decisions in a hierarchical structure and collections which can be used to create new instances of decisions at resolution time.

Perhaps the feature which most differentiates a decision model from a test is the concept of dependency. Dependencies allow defining a series of actions to take place, like deriving values of other decisions, when certain conditions regarding the elements of the decision model are met. These dependencies are executed at resolution time.

The figure 1 represents the core concepts of a Decision Model.

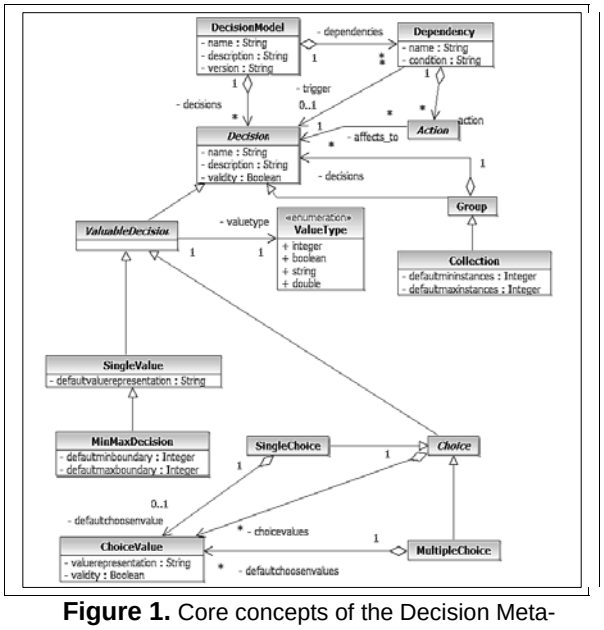

Figure 1. Core concepts of the Decision Meta-Model.

Once the Decision Model is completed, the next step in the Direct Product Variability approach is to generate the Application Model (AM) from it. If previously we have compared the decisions to the questions of a test and the decision model to the layout of the test, the Application Model can be depicted as an instance of the test with the actual answers. An Application Model represents an actual product from the product family by specifying the variable features it has. The process of setting the answers and values in the Application Model is called Variability Resolution.

Finally the resolved Application Model is used as an input to the Flexible Component Architecture (FCA). The Flexible Component Architecture is composed of series of Flexible Components which are scripts like pieces of executable code which create the final product based on the values of the decisions in the resolved Application Model. These scripts are flexible enough to allow the creation of almost any kind of product. Decisions represent the variable features of a product in the family.

The figure 2 is a graphical representation of the workflow in the Direct Product Variability approach.

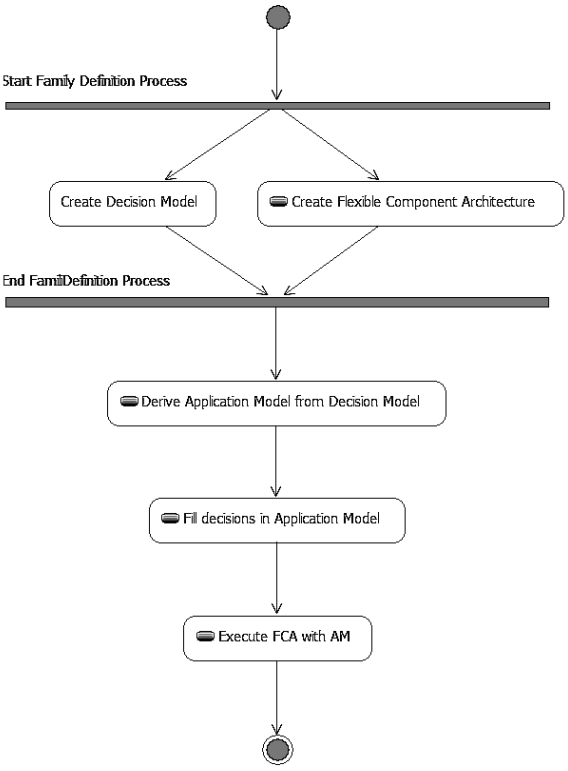

Figure 2. DPV Activity Diagram.

Previous versions of this approach have been supported by ESI's VManage toolkit.

3. Standardized Interoperability

As it has been already stated interoperability between modeling tools is a key feature for our suite. For example the OAW's transformation engine will be used to transform the models containing the decisions into actual products. In order to achieve this we need to interoperate with the OAW tool.

This connection between OAW and PLUM could be implemented by directly using the OAW's API in a custom fashion. However this would impose a direct dependency with the OAW tool and this is not desired since our tool requires functionality which is not proprietary of OAW but also can be implemented by other model transformation tools which use EMF models as their inputs. Acceleo [16] is one of these tools which could be used in substitution of OAW.

In order to implement the much needed interoperability it was decided to use the Model Bus. Model Bus is a product created inside the ModelWare [17] European research project, where ESI also participated, and which has been migrated into the Eclipse community inside the MDDi [14] project.

The aim of Model Bus is to create a standardized way for modeling tools to communicate using SOA principles. Therefore an application can publish a series of services to a registry and another application would be able to easily discover and consume them. The Model Bus helps this process of service creation and deployment by simplifying the tasks needed to define and deploy web services and by standardizing the data types and data mappings the services will use.

However adapters must be created for applications using the Model Bus both to enable them to consume and provide services. In the case at hand, OAW does not provide up to this date any officially supported adapter for the Model Bus toolkit. Hence the first problem we had to solve is to create an adapter for the OAW tool in order to enable it to inter operate with the Model Bus.

Once the OAW tool is made compatible and its services are accessible via the bus, the need to create another middleware tool for PLUM toolkit itself arisen.

Upon integration of the both adapters for the PLUM toolkit to consume the services and for OAW to provide them, the Software Product Line tool implementing the Direct Product Variability Approach will be completed. It will feature both graphical and tree model editors powered by EMF and GMF as well as OCL [17] model constraint enforcement and query capabilities which were also used to implement the Dependency resolution engine. These editors, which are fully integrated into the Eclipse environment, covered both the Decision and Application Modeling phases of the DPV approach. The Flexible components were implemented as model transformations using OAW and integrated with the editors using the process which has been recently explained thereby reducing the costs of creating a transformation engine of our own.

4. Conclusions

During development of the PLUM toolkit it has been made clear that there are tools inside the Eclipse ecosystem which perform different modeling operations. These tools are very efficiently on their own but often middleware adapters have to be developed to combine their strengths in an integrated tool suite.

By combining different series of existing technologies and tools for modeling (EMF, GMF), analysis (OCL, BIRT), integration (Model Bus, SVN) we have been able to produce a SPL supporting tool which manages the whole PL from the beginning to the end with much lower costs and higher quality than if it has been entirely developed from scratch.

Additionally by using Model Bus we have obtained benefits inherent to standardization and SOA development techniques, such as the ability to inter operate with remote applications and the base for a client-server structure.

5. References

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

[2] J. Greenfield and K. Short. Software Factories: Assembling Applications with Patterns, Frameworks, Models and Tools. John Wiley and Sons. 2004

[3] Product Line Definition by Clements and Northrop 2002.

[4] FAMILIES ITEA Research Project http://www.esi.es/Families/

[5] CAFÉ ITEA Research Project http://www.hitech-projects.com/euprojects/cafe/index.htm

[6] ESAPS Research Project http://www.esi.es/esaps/

[7] Gears Tool http://www.biglever.com/solution/product.html

[8] pure::variants Tool http://www.pure-systems.com/Variant_Management.49.0.html

[9] E. Gamma and K. Beck. Contributing to Eclipse : principles, patterns, and plug-ins. Addison-Wesley. 2004

[10] EMF Project http://www.eclipse.org/modeling/emf/

[11] GEF Project http://www.eclipse.org/gef/

[12] GMF Project http://www.eclipse.org/gmf/

[13] OAW Project http://www.openarchitectureware.org/

[14] MDDi Project http://www.eclipse.org/mddi/

[15] Software Product Line Engineering. Foundations, Principles, and Techniques. Klaus Pohl, Günter Böckle, Frank van der Linden. http://www.eclipse.org/mddi/

[16] Acceleo Generator http://www.acceleo.org/pages/home/en

[17] Model Ware research project http://www.modelware-ist.org/

[18] Flexi ITEA2 Research Project http://flexi-itea2.org/