26th Annual INCOSE International Symposium (IS 2016)
Edinburgh, Scotland, UK, July 18-21, 2016

Delta Rhapsody

Oskar Berreteaga
ULMA Embedded Solutions
Tel. 0034 943250300
oberreteaga@ulmaembedded.com

Goiuria Sagardui
Mondragon Unibertsitatea
Tel. 0034 943794700
gsagardui@mondragon.edu

Leire Etxeberria
Mondragon Unibertsitatea
Tel. 0034 943794700
letxeberria@mondragon.edu

Urtzi Markiegi
Mondragon Unibertsitatea
Tel. 0034 943794700
umarkiegi@mondragon.edu

Xabier Perez
Mondragon Unibertsitatea
Tel. 0034 943794700
xabier.perezb@alumni.mondragon.edu

Abstract. Model Based System Engineering (MBSE) has become the pre-eminent paradigm used to improve the development of complex systems. Additionally, Delta Modelling provides an incremental approach to the design and maintenance of models when there is variability in the system. In this context, the market leading IBM Rhapsody® provides a professional and wide range of functionalities. Nevertheless, there is not support for Delta Modelling in IBM Rhapsody®. To meet this need, the Delta Rhapsody solution has been developed. With the tool a new variant model can be automatically generated applying deltas to a core model. The application of the solution in an industrial case study has been undertaken. From this, it was concluded that the Delta Rhapsody solution notably reduces the required time to develop a variant model. To further improve the Delta Rhapsody solution, some future works have also been identified.

Introduction

Nowadays, systems are getting more and more complex. Model-based system engineering is an efficient approach to cope with the increasing system complexity, enhance consistency and allow modelling and simulation of the whole system (Mhenni 2013). "Model-based System Engineering (MBSE) is about elevating models in the engineering process to a central and governing role in the specification, design, integration, validation, and operation of a system" (Estefan 2008). MBSE enhances the ability to capture, analyse, share, and manage the information associated with the complete specification of a product, resulting in the following benefits (Murray 2012): Improved communications among the development stakeholders, increased ability to manage system complexity, improved product quality, enhanced knowledge capture and reuse of the information...

IBM Rational Rhapsody[1] is a model-based system and software engineering environment using the industry-standard Systems Modelling Language (SysML) and Unified Modelling Language (UML) that gives support for code generation, simulation, etc. It is the tool we use in our developments and the tools that several of our customers use.

Model-based and model-driven engineering offers multiple advantages both for the development and V&V of systems. However, it does not deal with another of the challenges in system development: variability. Variability is the ability to change or customize a system (Van Gurp 2001). It is quite common to have families of related systems, which vary from each other in terms of their behaviour, quality attributes, platform, network, physical configuration, middleware, and in a multitude of other ways.

To cope with this challenge, system and software product line engineering (SPLE), in combination with MBSE, can be an adequate alternative to traditional system development to explicitly manage this variability. Systems and software product line engineering is a paradigm to develop system and software product lines (SPL). "SPL is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" (Clements and Northrop 2010). It is a paradigm that systematizes reuse and allows reducing costs and time-to-market and improving the quality of the products (as the artifacts that are reused are reviewed and tested in many products). One of the key aspects of system and software product lines is to manage variability. When talking about SPL variability, two concepts immediately stand out (Trigaux 2003): commonalities (assumptions true for each family member) and variabilities (assumptions about how individual family members differ).

Software product line engineering distinguishes between two types of development processes: domain engineering and application engineering. In domain engineering, software artefacts are developed for reuse; the aim of the domain engineering process is to define and realise the commonality and the variability of the product line. In application engineering, domain artefacts are reused to create specific applications, the aim of the application engineering process is to derive specific applications by exploiting the variability of the product line.

The basis of SPLE is the explicit modelling of what is common and what differs between product variants. Feature modelling is a very common approach for capturing commonalities and variabilities in system families and product lines at problem space. A "feature" is defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system (Kang 1990). A feature model is a compact representation of all the products of the SPL. Feature models have a tree structure, with features forming the nodes of the tree. The arcs and groupings of features represent feature variability. Relationships between a parent feature and its child features (or sub features) are categorized as: Mandatory (child feature is required), Optional (child feature is optional), Or (at least one of the sub-features must be selected) and Alternative or Xor (one of the sub-features must be selected). In addition to the parental relationships between features, cross-tree constraints are allowed. The most common cross-tree relationships are: requires constraints and excludes constraints. See Figure 5.

[1] http://www-03.ibm.com/software/products/us/en/ratirhapfami/

At solution space, there are three variability mechanisms or ways of realizing the variability identified at problem space (Heidenreich et al. 2010):

1. Negative Variability (annotative or 150% model). The negative variability approach relies on having a 150 % model, which means that all the features are allocated in the model, the model contains all the elements used for all variants of the software product line. During product derivation, the elements of the model that are not required according to the selected features are removed and the model of the variant is obtained.
2. Positive Variability (compositional or 100% model). The positive variability approach consists of having a core model with the common or core elements for any product in the product line; and specifying which new elements must be added when a certain feature or set of features is selected. In this case, during product derivation, the core model is taken as starting point and elements are added to this model according to the selected features; and this way the model of the variant is obtained.
3. Modification of Model Elements. In this approach, there is a reference or base model and the variant model is obtained modifying the existing model elements in this model according to the selected features.

Modification of Model Elements allows changes inside the model elements, but not variability at architectural level. For example, modification of model elements could be implemented using parameterised components. This approach could be enough if needed variability could be bound to model elements. In the case of the other two mechanisms, both negative and positive variability, have advantages and disadvantages (Heidenreich et al. 2010). The 150% model or negative variability approach can increase model complexity especially when there are many variants, as it has to contain model elements for any product in the product line. To understand or analyse this model could be quite complex. On the other hand, positive variability or compositional approach helps to cope with model complexity as model elements that are not common ones are in different models. However, in this approach there can potentially be a large number of model fragments describing the SPL as a whole, and this make more difficult to get an overview of the SPL. This makes also more difficult to identify feature interactions. In contrast, in the negative variability approach all the elements are in the same model so to get an overview or spot feature interactions is easier. It will depend on the situation and the SPL which approach is the best choice; even in the same SPL, the variability in some parts could be better managed using one approach whereas in other parts another approach could be more beneficial. Therefore, it would be beneficial for a variability-modelling approach to support all the forms of variability modelling (Heidenreich et al. 2010).

One of the variability modelling that supports all the forms of variability modelling is the Delta modelling approach. Delta modelling (Haber et al. 2013) is a modular and language-independent approach for modelling system variability via transformations. In a delta-oriented SPL, a set of products can be represented as a core model and a set of model deltas (Clarke et al. 2010b; Haber et al. 2013; Haber et al. 2011; Schaefer et al. 2010) explicitly specifying changes to the core model to obtain other system variants. The core model represents a product for some valid feature configuration. The model deltas specify modifications required to apply to the core model in order to implement other features of the

product line. The modifications include adding/removing/replacing model elements. Moreover, the application conditions are defined to specify under which feature configuration the modifications should be applied. The concept of application condition fulfils the need for establishing mappings between features and their realizations in SPL development. During delta application, in order to generate a particular product model for a given feature configuration, the model deltas that have to be applied for this feature configuration are selected and applied one-by-one to the core model. The result is a product model realizing the particular feature configuration. See Figure 1 for a conceptual sample.

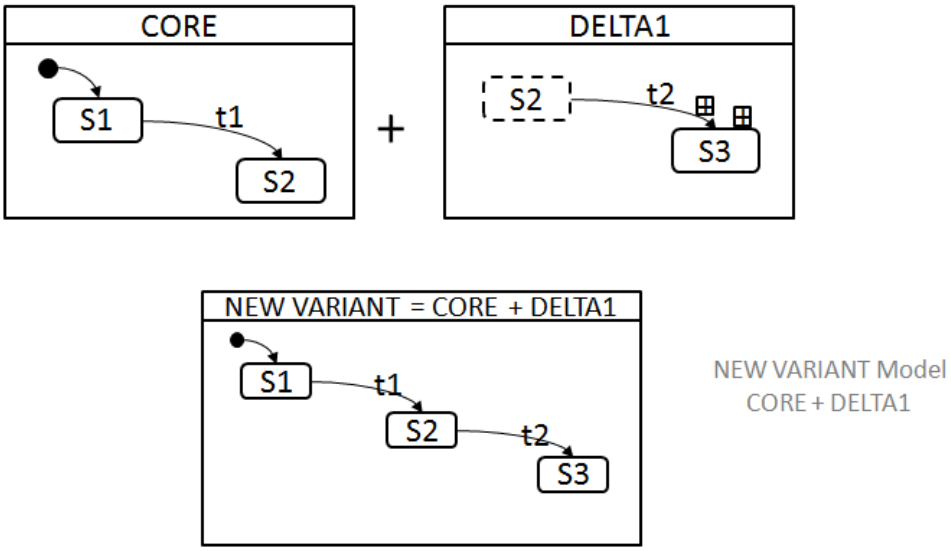

Figure 1. Delta Modelling conceptual sample

Delta modeling has several advantages when used as variability mechanism (Seidl et al. 2014) (Shaefer 2012):

- It can handle configuration (variability in space) as well as evolution (variability in time) within a single notation. Variability in space is related to having several variants or products and variability in time is related to the evolution of the SPL.
- It can deal with an open variant space where not necessarily all configuration options are known in advance. It supports proactive (complete variability management is performed before starting with application engineering), reactive (the product line is updated when new variants appear) and extractive (existing products are taken as base for the core assets) SPLE.
- It is language independent.
- It supports modular and flexible description of variability and change. It is intuitively understandable and well-structured.

This paper presents Delta Rhapsody solution, a tool for Delta modelling in Rhapsody that we have developed. This way, a variability modelling approach is provided that supports positive variability, negative variability and modification of model elements in Rhapsody.

The reminder of the paper is structured as follows. Related work section, presents most remarkable approaches and tools related to Delta Rhapsody. Next sections describes the proposed approach and tools. Case study section introduces the experience of application of the proposal in a real context. Finally, conclusions and future work sections are presented.

Related work

Existing support of Rhapsody for variability

IBM Rational Rhapsody gives some support for designing, reuse and envision product line variants (Scouler and Bakal 2013). Variants are represented using stereotypes and tags that are applied to a model element to indicate how the element changes for the variant. With this approach, the specific product variant can be explained more clearly. The stereotypes and tags can control the code generation of the software of the variant. Moreover, for the automotive domain, the AUTOSAR 4.0 profile of Rhapsody provides support for AUTOSAR variation points in Rhapsody.

There are also tools that integrate Product line engineering tools with Rhapsody allowing Model Driven Product Line Engineering: Pure::variants for Rhapsody[2] and Rhapsody/Gears Bridge[3]. Both tools provide a variability management approach that allow the generation of variants by feature selection.

However, none of these tools provides a delta modelling approach. As far as we aware there is not any tool that support delta modelling in Rhapsody.

Delta modelling tools

There are several tools for delta modelling for specific tools or languages such as Delta Simulink, a tool for delta modelling for Simulink (Haber et al. 2013) and Delta-MontiArc (MotiArc is an existing architecture description language) that offers an integrated modelling language for architectural variability (Haber et al. 2011).

There are also approaches for creating the delta language and tools for a language: (Haber et al. 2013) presents a process that allows deriving a delta language from the grammar of a given base language. And Delta Ecore provides delta modelling on basis of the structured data models conforming to meta models specified in EMF Ecore (Seidl et al. 2014). However, this tool has not been used because Rhapsody's meta model is not specified in EMF Ecore format.

In the context of Delta modelling, the eDeltaMBT tool (Lochau et al. 2014) for incremental model-based component and integration testing uses Rhapsody. It applies Rhapsody in two ways: 1) The component state machine (SM) test models generated in eDeltaMBT are imported to Rhapsody and test cases for testing at component level are automatically generated using the ATG of Rhapsody 2) Integration level test cases are specified using message sequence charts (MSC) in Rhapsody. And both, MSC and SM test cases are imported into eDeltaMBT tool. The Rhapsody library for Java is used for the automated import/export of models and test cases. This tool is oriented to model based testing and not design. Moreover, the delta creation is done in eDeltaMBT, not in Rhapsody.

---

[2] http://www.pure-systems.com/pure_variants_for_Rhapsody.162.0.html

[3] http://www.biglever.com/ecosystem/bridges/ibm.html

Delta programming tools

Delta-oriented programming is a novel programming language approach for software product lines introduced by (Schaefer et al. 2010).

As stated by (Clarke et al. 2010a), this programming approach is a direct alternative to feature-oriented programming (Batory et al. 2004). The feature-oriented programming is a paradigm for variant synthesis in SPLs where a feature is considered an increment in program development or functionality. A variant is derived composing the selected features (software modules associated to features).

The main difference between feature-oriented and delta-oriented programming (Clarke et al. 2010a) is that in delta-oriented programming there is not a one to one mapping between features and deltas. The delta modules that are applied in a variant derivation will depend on the application conditions that are true for a particular feature selection. "By not associating the delta modules directly with features, a degree of flexibility is obtained, resulting in better reuse of code and the ability to resolve conflicts caused by deltas modifying the code base in incompatible ways" (Clarke et al. 2010b).

Schaefer et al. proposed and developed the DeltaJava programming language, a delta-oriented programming approach for Java (Schaefer et al. 2010). The Abstract Behavioural Specification language (ABS) (Clarke et al. 2010a) also proposes a delta approach for managing variability. ABS and its tool suite allow precise specification and analysis of the abstract behaviour and variability of highly configurable software systems (Clarke et al. 2010a). ABS uses the Delta Modelling Language (DML), based on the concept of delta-oriented programming (Schaefer et al. 2010), for expressing the code-level variability of ABS models (using core and deltas). The delta-oriented programming language proposed by ABS is strongly influenced by DeltaJava, one of the differences is that they further separate deltas from features by moving application conditions out of deltas and into a product line configuration language (Clarke et al. 2010a).

Our approach (Rhapsody delta) is used at design level and not at programming level. However, the mapping language of ABS has been the base for the mapping among features and deltas and for specifying the required order of application of deltas in Delta Rhapsody.

Delta Rhapsody

The Delta Rhapsody solution develops and integrates tools to support the Delta Modelling approach within the IBM Rhapsody® tool.

The final solution consists of several programs, with IBM Rhapsody® as the central tool. Within the Delta Rhapsody solution, core and delta models are specified and new variant models are automatically generated. The IBM Rhapsody® tool provides Java API to extend the functionality of the tool. Consequently, the complementary tools and developments for the Delta Rhapsody solution were based on Java, taking advantage of Eclipse as the Integrated Development Environment (IDE).

The development of the Delta Rhapsody Java Application includes all functionalities required to examine core and delta models, feature model creation and configuration and creation of mapping files. It also generates the new variants with all diagrams.

To achieve a visual variability management at problem space, the FeatureIDE Eclipse plug-in was selected for the solution. This plug-in, developed by (Thüm et al. 2014), provides, among

other functionalities, the possibility to design the feature model graphically and configure a particular feature model for a new variant.

After initial research into existing feature-delta mapping solutions deemed current tools too complex for the aim of the project, an alternative was developed. This tool is purpose-specific and is based on ABS syntax for mapping.

Domain Engineering and Application Engineering processes with Delta Rhapsody solution

The Delta Rhapsody solution was designed taking into account the two main roles in model based design for SPL; (i) Domain engineering and (ii) application engineering. The process to be accomplished by the domain engineer comprises three stages. The first stage requires the creation of the feature model in the FeatureIDE java project (the output is the .config file). The second stage consists of a visual design of core and delta models in IBM Rhapsody®. Finally, the feature-delta mapping file has to be defined (the output is the .mapping file). Similarly, the process for the application engineer was defined. The first stage is the feature selection in the FeatureIDE java project, updating the default.config file. The second stage consists of launching the Delta Rhapsody Java Application which automatically generates the new variant model. Finally, the new model can be accessed in IBM Rhapsody®. This is outlined in Figure 2.

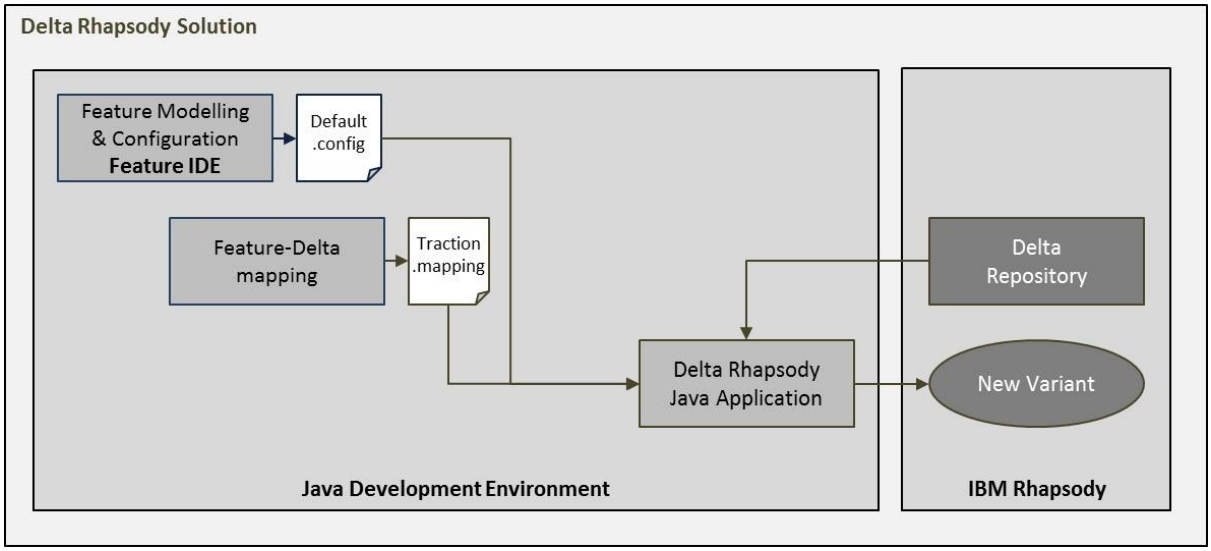

Figure 2. Delta Rhapsody Solution process

Core and delta modelling in Rhapsody

IBM Rhapsody® enables the implementation of various types of diagrams. For Delta Rhapsody three of them have been made available for delta modelling: statechart, class and sequence diagrams. In order to generate new diagram variants via the Java API, dynamic gathering of elements from specific delta type diagrams is essential. In IBM Rhapsody®, each of the mentioned diagrams is composed of distinct elements, which can have a different hierarchical structure depending on the particular requirements of the diagram type. Taking this into account, Delta Rhapsody's diagrams structural hierarchy has a precise and well defined element arrangement. However, the individual structures vary from one diagram type to another.

Firstly, for statechart diagrams, a class must be created in a package and under it, the delta statechart diagrams are placed. For class and sequence diagrams, a new package must be created for the core and for each delta, and where the diagrams are placed.An element name convention was also defined to facilitate understanding. The core model starts with the word "core". Delta models begin with "D" plus the model name itself. As mentioned before, each delta module is associated with a module of program modification (add, remove or modify code), raising the need for some sort of tagging. For this, Rhapsody stereotyping feature allows a high degree of labelling. Both core and delta modules must be stereotyped as the developed Delta Rhapsody Java Application requires stereotypes for its proper functioning. Every core model has been stereotyped as "stereo_Core". Similarly, delta models can have three different stereotypes, depending on their type: "stereo_Delta_add", "stereo_Delta_mod", "stereo_Delta_rem" (see Figure 3). The latter mentioned stereotypes are also applicable to diagram elements. Other complementary stereotypes exist. Their role is to make the Java API distinguish certain diagram elements that are not individualized, such as generalizations and realizations.

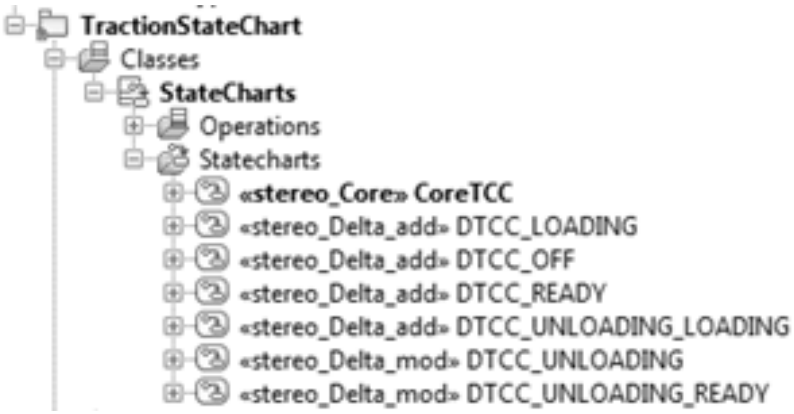

Figure 3. IBM Rhapsody © statechart structure sample

Each delta model diagram has some specific elements relating to its type (states and transitions in statechart diagram, class and associations in class diagram, etc.). Each element that is modified is coloured following the colours reference used by visual merge feature of IBM Rhapsody®. This colouring is directly linked to the stereotypes. Green is used in the case of "stereo_add", blue for "stereo_mod", and "stereo_rem" is highlighted in red. The purpose of doing this is to have an enhanced view of the diagram, emphasising what is modified. See Figure 4.

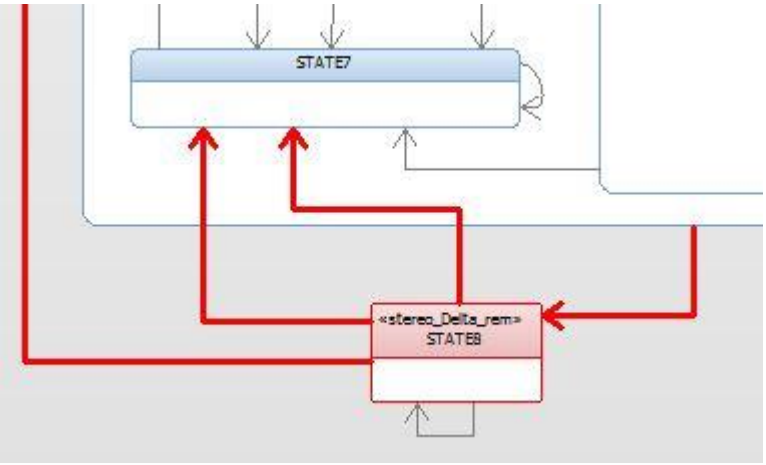

Figure 4. Diagram delta elements highlighting sample

Feature Model and Mapping Files

A feature model is used for capturing the variability and commonality at problem space (see Figure 5). Domain engineer is in charge of designing the Feature Model using the FeatureIDE tool.

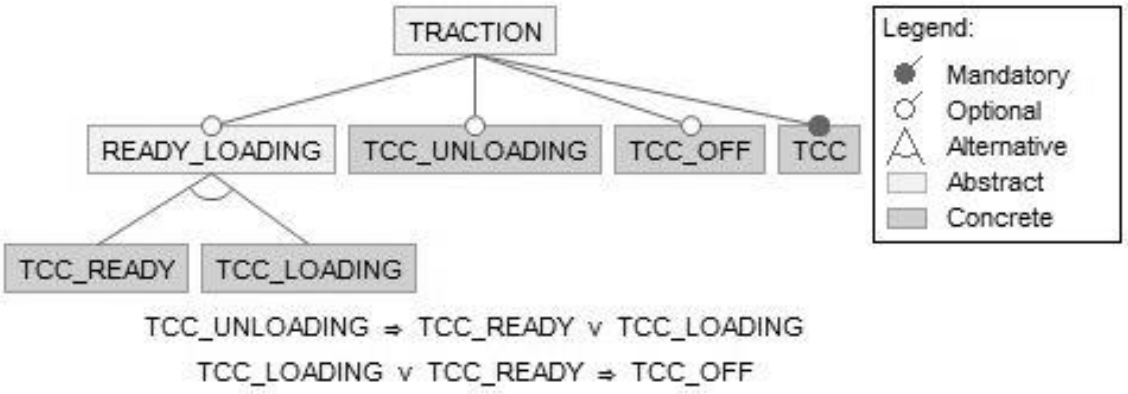

Figure 5. Feature model

Once the Feature Model is designed, the application engineer has the possibility to select a particular feature configuration for a new variant using the FeatureIDE interface. After the configuration has been set and saved, a file named default.config is updated. This file, built as a flat file, contains the list of features selected for the new variant. This will be read by the Delta Rhapsody Java Application to identify the features required by the new variant.

A critical step is the specification of the mapping between features and deltas. A purpose specific tool based on ABS language for mapping was developed to describe this relationship into the mapping files (see Related Work). Each diagram type requires a particular mapping file, as identical delta names can exist in different diagram types.

Each mapping file defines the product line name and the features involved. Moreover, for each feature the required deltas are specified, and the order of the deltas is also defined. See Figure 6.

```
productline Traction;
features TCC_OFF, TCC_READY, TCC_LOADING, TCC_UNLOADING;

delta DTCC_LOADING after DTCC_OFF when TCC_LOADING;
delta DTCC_UNLOADING after DTCC_READY || DTCC_LOADING when TCC_UNLOADING;
delta DTCC_UNLOADING_READY after DTCC_UNLOADING when TCC_UNLOADING && TCC_READY;
delta DTCC_READY after DTCC_OFF when TCC_READY;
delta DTCC_UNLOADING_LOADING after DTCC_UNLOADING when TCC_UNLOADING && TCC_LOADING;
delta DTCC_OFF when TCC_OFF;
```

Figure 6. Mapping file

Delta Rhapsody Java Application

The Delta Rhapsody Java Application was developed to encompass the non-IBM Rhapsody® functionalities of Delta Rhapsody solution. The main functionalities of this applications are (i) Rhapsody models reading and writing through the Java API, (ii) feature configuration integration through FeatureIDE and (iii) feature and delta configuration and mapping files management.

The Delta Rhapsody Java Application software architecture includes two projects, one for feature model configuration, and a second one to generate the new variant. This second one, contains four packages to organise the capabilities. The first package, named diagrams, encloses main classes which launch the diagrams generation of the new variant. The second package, named featureDataManagement, accomplishes the default.config file processing, feature-delta mapping and delta ordering. The third package, named graphicalSettings, provides the graphical arrangement of elements to easier understand the new diagrams. Finally, the fourth package, named resources, contains complementary classes. See Figure 7.

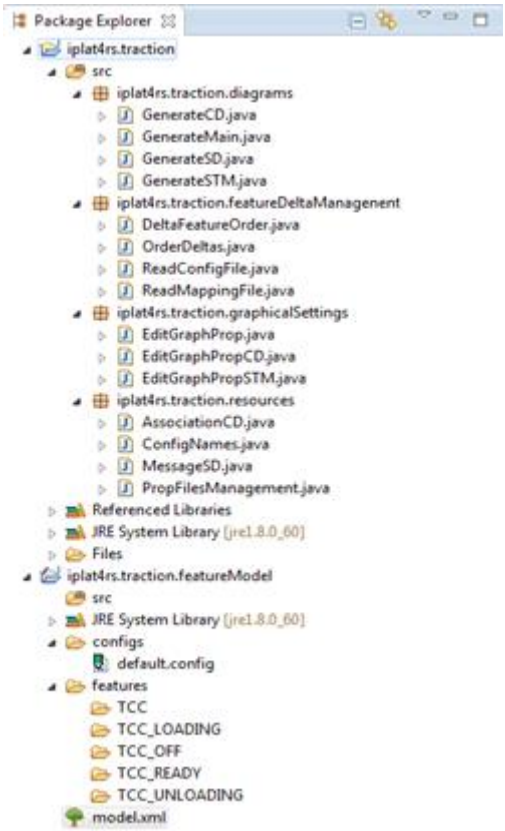

Figure 7. Delta Rhapsody Java Application project structures

The structure of the application was filled out with a certain file directory organization. This directory, named Files, is divided into four subdirectories. The GeneralSettings subdirectory includes a java properties file for general purposes including file names and location. The MappingFiles subdirectory contains a mapping file for each diagram type. In the GraphProperties subdirectory fixed graph properties are specified. Each diagram type requires a particular subdirectory that contains the information taken from the existing core and delta model elements and will distribute the elements of the new variant with an alike layout.

Case Study

For using the solution in a case study, the Delta Rhapsody solution has been deployed into a virtual machine that included a specific software stack. The base of the stack is Windows® 7 operating system and above it runs Java Development Kit 1.8 library. This makes possible both Eclipse Helios and IBM Rhapsody® 8.1 layers. Complementing Eclipse, the FeatureIDE plug-in has been deployed. With the aim of giving access to IBM Rhapsody® functionalities programmatically, Java API has been spread out. On top of the solution the developed Delta Rhapsody Java Application has been deployed. See Figure 8.

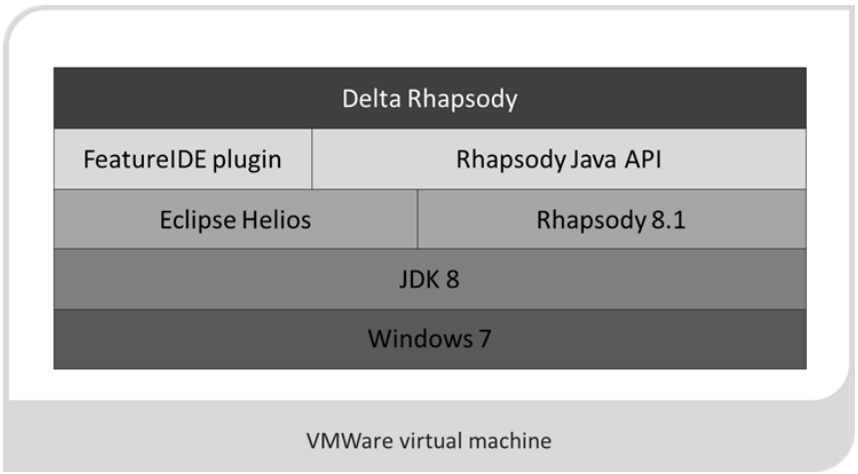

Figure 8. Delta Rhapsody solution stack

The Delta Rhapsody solution has been applied in an operational system in the transport domain, named the Traction Control Core (TCC) system. Due to confidentiality issues, the detailed description of the case study cannot be presented and obfuscated version of diagrams are used. See Figure 9.

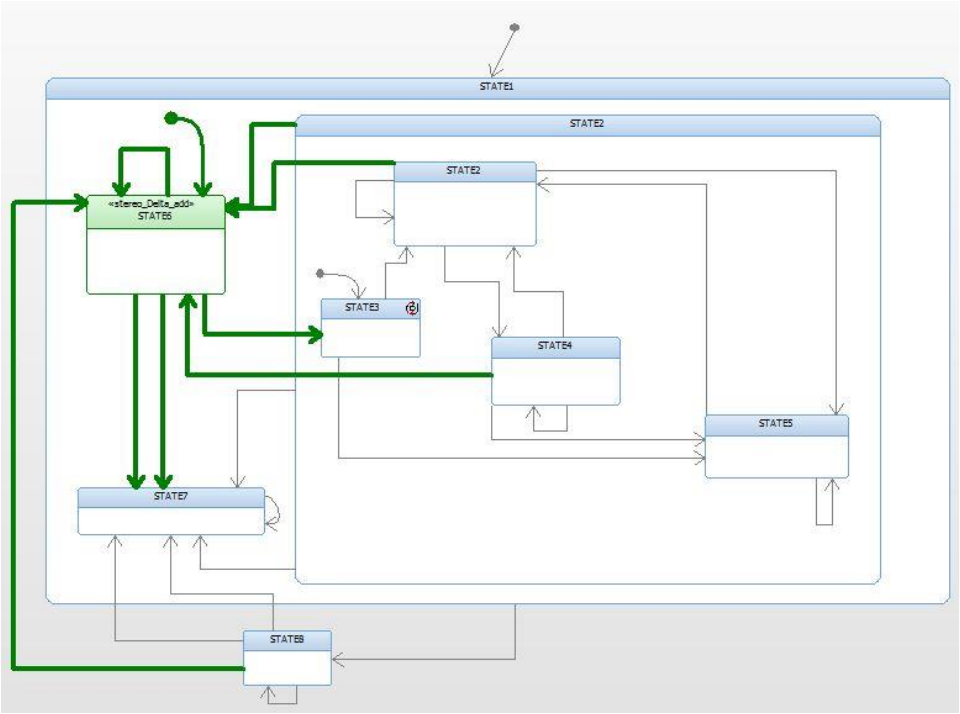

Figure 9. Case study obfuscated delta model

In particular, the core and delta models of the case study were designed in an IBM Rhapsody® project. One package for each diagram type was created following the structure described in the "Core and delta modelling in Rhapsody" section.

A positive variability approach was applied in the case study models design. For Statechart diagrams, besides the core model, six delta models were designed. Four of the delta models were adding deltas, with the other two as modifying deltas. For the class diagram, the core and 4 adding delta models were designed. Finally, the sequence diagrams were supported by one core and one adding delta model.

Conclusions

The developed tool (Rhapsody Delta) provides delta modelling support for class diagrams, sequence diagrams and state machine diagrams in IBM Rhapsody tool. With this support is possible to design product lines with a delta approach in Rhapsody. A process has been defined for the creation of the necessary assets at domain engineering. And the process at application engineering has been also defined, application engineering workload is considerably simplified as variant models are generated automatically using the tool.

Rhapsody Delta solution is based on Rhapsody and in a java application that used the java API of Rhapsody to provide Feature modelling capabilities using the FeatureIDE Eclipse plug-in and feature configuration, mapping and variant generation capabilities.

The results after the application of the tool in a real system of the transportation domain are satisfactory: the current models (core and deltas) are easier to understand that the complete model that was previously used, the required time to develop variant models is reduced, the inclusion of future new products is easier… However, there are also some costs to take into account, especially the time required to model the core and deltas at domain engineering.

To validate the tool, more validation is needed: the application of the tool to other systems in several domains and with other variability approaches (negative, modifying model elements…) as in the case study an incremental or positive variability approach was used (core and adding deltas), empirical validation is also required to measure the benefits of using the tool taking into account the learning curve of the domain engineer.

Future Work

The task of the domain engineer could be quite time-consuming and error prone, so we foresee to provide support to the domain engineer during the creation and checking of deltas in Rhapsody. We will be inspired by existing approaches such as the Difference-based Delta-Modelling[4] that consists of generating the deltas by comparing two models: a given model and the one created by modifying that given model; and using model differentiating technologies. The delta creation in Rhapsody is done with this philosophy and in a future version we will include automatic support for automatic stereotyping (detect if something has been added, deleted or modified and assign corresponding stereotypes and corresponding colour for the enhanced view).

It is not easy for the domain engineer to have an overview of the SPL if it is working with a large number of model fragments or deltas. Automatic support will be developed using the Rhapsody checkers for performing coherence and consistency checkings during delta designing. Moreover, support will be developed for massive generation of variants and automatic checkings of all the valid configurations.

Support for automatic model based test generation will be also developed. This could be useful at domain engineering for testing a set of products or all the valid configurations but also at application engineering for testing a specific product.

---

[4] http://pi.informatik.uni-siegen.de/projects/sipl/index.php

In the current version, the mapping file is textually specified. But for future versions of the tool, a GUI based tool will be developed for mapping file specification. This will help to make the use of the tool friendlier and avoid errors during this specification.

IBM Rational Software Architect Design manager[5] is a collaborative software design and development platform that provides a global repository for storing, sharing and managing models in a central location. In the current version of Rhapsody Delta, the core and model repository is located in Rhapsody but for future version, this repository will be located in a global repository provided by Design Manager. This will facilitate the collaborative work of a team but also to link the design to requirements and to test cases.

Acknowledgments

The research leading to these results has received funding from the Basque Government (UE2014-12 AURE and etorgai iPLAT4RS projects). The project has been developed by the embedded system group supported by the Department of Education, Universities and Research of the Basque Government.

[5] http://www-03.ibm.com/software/products/en/designmanager

References

Batory, D., Sarvela, J., Rauschmayer, A. 2004. “Scaling Step-Wise Refinement.” *IEEE* Trans. Software Eng. 30(6).

Clarke, X. D., Muschevici, R., Proença, J., Schaefer, I., Schlatte, R. 2010a. “Variability Modelling in the ABS Language, Chapterin Formal Methods for Components and Objects.” *Volume 6957 of the series Lecture Notes in Computer Science*: 204-224

Clarke, D., Helvensteijn, M., Schaefer, I. 2010b. “Abstract Delta Modelling.” *ACM Sigplan Notices*: 13-22.

Clements, P. and Northrop, L. 2010. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley.

Estefan, J. A. 2008. “Survey of Model-Based Systems Engineering (MBSE) Methodologies.” *Rev. B, INCOSE Technical Publication, Document No.: INCOSE-TD-2007-003-01* (paper presented at the International Council on Systems Engineering, San Diego, CA, June 10, 2008).

Haber, A., Hölldobler, K., Kolassa, C., Look, M., Müller, K., Rumpe, B., Schaefer,I. 2013. ”Engineering Delta Modelling Languages.”(paper presented at the Proceedings of the 17th International Software Product Line Conference (SPLC), Tokyo, September, 2013): 22–31.

Haber, A., Hölldobler, K., Kolassa, C., Look, M., Rumpe, B., Müller, K., Schaefer, I. 2013. *“*Engineering Delta Modelling Languages*.” Proceedings of the 17th International Software Product Line Conference*: 22-31.

Haber, A., Kolassa, C., Manhart, P., Nazari, P. M. S., Rumpe, B., Schaefer,I. 2013.“First-class variability modelling in matlab/simulink.” (paper presented at the International Workshop on Variability Modelling of Software-intensive Systems (VaMoS).

Haber, A., Kutz, T., Rendel, H., Rumpe, B., Schaefer, I. 2011. “Delta-Oriented Architectural Variability Using Monticore.” *Proceedings of the 5th European Conference on Software Architecture: Companion Volume*: 6.

Haber, A., Rendel, H., Rumpe, B., Schaefer, I., van der Linden, F. 2011. “Hierarchical Variability Modelling for Software Architectures” (paper presented at the Proceedings of International Software Product Lines Conference, August 2011).

Halmans, G., Pohl, K. 2004. “Communicating the variability of a software-product family to customers. Inform. “*Forsch. Entwickl. 18*: 113–131.

Heidenreich, F., Sánchez, P., Santos, J., Zschaler, S., Alférez, M., Araújo, J., Fuentes, L., Kulesza, U., Moreira, A., Rashid, A. 2010. “Relating feature models to other models of a software product line: a comparative study of feature mapper and VML.” *Transactions on aspect-oriented software development VII*: 69-114.

Kang, K.C.,Cohen, S.G., Hess, J.A.,Novak, W.E., Peterson, A.S., "Feature-oriented domain analysis (FODA) feasibility study", Technical Report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University, November 1990.

Katz, S., and Mezini, M. 2011. “A Common Case Study for Aspect-Oriented Modelling*”*. Berlin, Heidelberg: Springer-Verlag.

Lochau, M., Lity, S., Lachmann, R., Schaefer, I., Goltz, U. 2014. “Delta-oriented model-based integration testing of large-scale systems.”*J. Syst. Softw*. 91 63-84

Mhenni, F., Nguyen, N., Choley, J.Y. 2013. “Towards the Integration of Safety Analysis in a Model-Based System Engineering Approach with SysML, Design and Modelling of Mechanical Systems.” *Lecture Notes in Mechanical Engineering* (2013): 61-68 (paper presented at Fifth International Conference Design and Modelling of Mechanical Systems, Djerba, Tunisia, March 2013).

Murray, J. 2012. “Model Based Systems Engineering (MBSE)”, *Media Study*.

Schaefer, I. 2010. “Variability modelling for model-driven development of software product lines.” *VaMoS*: 85-92

Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N. 2010. ”Delta-oriented programming of software product lines.” (paper presented at Proc. of 15th Software Product Line Conference, Sep 2010)).

Schaefer, I., Bettini, L., Damiani, F. 2011. “Compositional type-checking for delta oriented programming.” (paper presented at the 10th International Conference on Aspect-Oriented Software Development, AOSD 2011): 43–56.

Schaefer, I., Bettini, L., Damiani, F., Tanzarella, N. 2010.”Delta-Oriented Programming of Software Product Lines.” *Proceedings of the 14th international conference on Software product lines: going beyond*: 77-91.

Schaefer, I., Damiani, F. 2010. “Pure Delta-oriented Programming”. *FOSD 2010.*

Schaefer, I., 2012, “Efficient Incremental Testing of Variant-Rich Software Systems”, *The 23rd CREST Open Workshop Change Impact Analysis and Testing of Software Product Lines.*

Scouler, J.L., Bakal, M.R. 2013. “Product Design for Variants: Considerations, incentives, and best practices.” http://www.ibm.com/developerworks/rational/library/product-design-variants/.

Seidl, C., Schaefer, I., Aßmann, U. 2014. “DeltaEcore - A Model-Based Delta Language Generation Framework.” *Modellierung*: 81-96

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T. 2014. “FeatureIDE: An Extensible Framework for Feature-Oriented Software Development.” Science of Computer Programming, 79(0): 70-85.

Trigaux, J. C. 2010. “Modelling variability requirements in software product lines: a comparative survey,” *Tech. rep., FUNDP Namur*.

Van Gurp, J., Bosch, J., Svahnberg, M. 2001. “On the notion of variability in software product lines.” *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*: 45

Biography

Oskar Berreteaga, R&D Software Project Manager at ULMA Embedded Solutions since 2009. Experience: 18 years in project management, analysis and development of software for distributed and embedded systems in several domains: industrial automation, renewable energy, vertical transportation.

Goiuria Sagardui obtained the PhD degree in Computer Science from the University of the Basque Country (Spain) in 2000. Previously she finished the BSc in Software Engineering at Deusto University (Spain) in 1995. She worked as a software engineer at ESI (European Software Institute) during 1999-2000. She is nowadays working as lecturer/researcher in the Electronics and Computer Science Department of Mondragon Unibertsitatea. Her main research areas are related to embedded systems and software engineering: V&V, software product lines and model driven engineering. She is the coordinator of the embedded system group of the university.

Leire Etxeberria obtained the PhD degree in Computer Science from Mondragon Unibertsitatea (Spain) in 2008. Previously she finished the BSc in Computer Science Engineering at Mondragon University (Spain) in 2004. She is nowadays working as lecturer/researcher in the Electronics and Computer Science Department of Mondragon Unibertsitatea. Her research topics include software product lines, model driven development, variability and V&V, variability/reuse and safety, etc. in the embedded systems domain.

Urtzi Markiegi is currently a lecturer and a researcher at Mondragon University (MU) (Basque Country, Spain). Computer Science Engineer (2001) at MU, combines lecturing and research with an extensive experience in industrial work. He has actively participated in several projects. His research topics are Web engineering, Software architecture, interoperability, Business Intelligence.

Xabier Perez: is undergraduate student of Bachelor's Degree in Telecommunications Systems Engineering and he is working part time in Mondragon University’s Embedded System Laboratory in an internship program.