Tracking Evolution in Model-based Product Lines

Wolfgang Heider Rick Rabiser Deepak Dhungana Paul Grünbacher
Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler University
Linz, Austria
{heider | rabiser | dhungana | gruenbacher}@ase.jku.at

***Abstract*— Software product lines are complex and need to be maintained and evolved over many years. New customer requirements, new products derived, technology changes, and internal enhancements lead to continuous changes of the artifacts and models constituting a product line. Managing such changes therefore becomes a key issue during a product line's evolution. We propose an approach that supports multi-level monitoring of product line artifacts and models and continuous tracking of changes. We present tool support for evolution tracking in Eclipse workspaces and illustrate our approach with examples from DOPLER, an existing Eclipse-based product line environment.**

Keywords-product line engineering; evolution; change tracking

I. INTRODUCTION

Product lines are typically used for many years and are inevitably subject to continuous evolution. Managing the evolution is success-critical for any product line approach as engineers need to deal with changes and extensions to the product line's assets and the derived products [1]. Feature models [2], decision models [3], extended UML [4], or aspect oriented approaches [5] are typically applied to define product lines. Managing the evolution of models therefore becomes a major concern.

In particular, our research interest is on (i) monitoring and tracking changes to models and product line artifacts, and (ii) establishing traceability between diverse product line artifacts such as product-specific requirements, change requests, or bug reports. Numerous research prototypes and commercial tools are available to support the creation and utilization of product line models, e.g., [6, 7]. However, they provide only limited support for dealing with product line evolution.

A generic approach for tracking the evolution of heterogeneous artifacts and models is still not available. For instance, existing approaches and tools lack support for managing the evolution of product line models at multiple levels of granularity and for managing interdependencies between different product line artifacts. This becomes particularly critical in a multi-team environment if several application engineering projects are conducted in parallel. This can mean that multiple products are derived concurrently from different releases of a product line.

In this paper we propose an approach for evolution tracking which is based on a generic meta-model. The approach is supported by our tool EvoKing. We demonstrate the capabilities of EvoKing using an example of its integration with the DOPLER product line approach and tools [8].

II. A META-MODEL FOR TRACKING PRODUCT LINE EVOLUTION

Many software tools support change tracking at the file or code level. For instance, version control systems and file system journaling mechanisms allow keeping track of changes to artifacts at the file level. Development environments make use of these tools to support change-tracking at the code level. However, tracking changes at this level is tedious. Supporting evolution requires change-tracking at a higher level of granularity and abstraction. It is also important to understand the dependencies between changes. Furthermore, change-tracking needs to cover various types of artifacts such as models, model elements, or structured documents.

From a bird's eye view, tracking evolution is about understanding the changes that are made to different artifacts of interest and establishing traceability between these artifacts based on dependencies between changes. The events and conditions that lead to a certain change are usually as interesting as the change itself. We have devised a generic meta-model for tracking evolution, which comprises artifacts, events, and relations (cf. Fig. 1).

An **artifact** is an element which needs to be monitored to track and manage its evolution. Examples of product line artifacts are meta-models, models, model elements, solution space elements (e.g., reusable code assets), or change requests (e.g., requirements captured during application engineering [9]). In a product line environment, these artifacts are typically managed in files or parts of files. The nature of the artifacts is domain-specific and cannot be generalized. Our evolution meta-model (the top layer in Fig. 1) thus does not specify concrete artifacts such as feature models, configuration files, or component descriptions. Instead we use a layered approach: the generic meta-model defines the basic elements that are then refined to specific domains and technologies using

custom artifacts. Fig. 1 (middle and bottom layer) shows examples of artifacts at multiple levels of abstraction, i.e., in Eclipse-based tools and in product line engineering. Events and relations are created and resolved by implementing the defined custom artifacts (cf. Section 3).

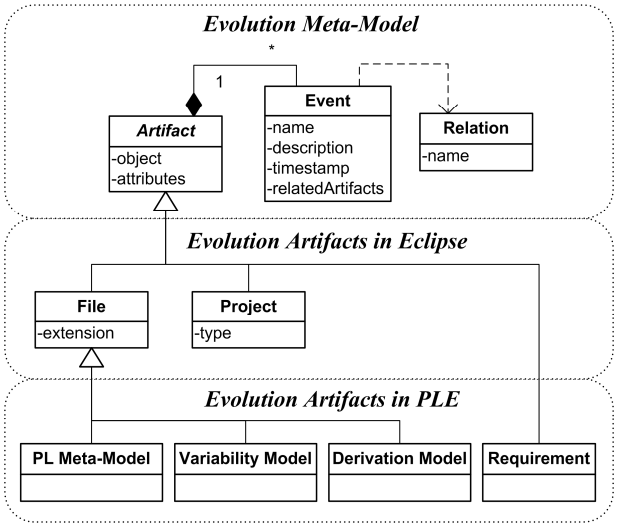

Figure 1. Evolution meta-model for tracking evolution and examples of custom artifacts for product line engineering artifacts in Eclipse.

An **event** causes one or more changes to artifacts. The generic evolution meta-model allows defining arbitrary events for the specified artifacts. Events relevant in product line engineering can typically be derived from existing product line process models and workflows. For example, the addition of a new variation point to a variability model constitutes an event that creates a new version of this model. Events can however also be defined at a much higher level of abstraction: e.g., if a user decides to derive a product using an existing variability model, a new application engineering project will be created, that is e.g., stored in a new model that needs to be tracked.

A **relation** between artifacts is established by an event tracked for specific artifacts. It describes how these artifacts are related with each other. Such links can be structural or temporal in nature. Structural relations between artifacts describe how the artifacts are organized, e.g., a model might be part of another model or a component might be described by a certain document. Temporal relationships are created at certain times during the artifact life-cycle to track their evolution history, e.g., a derivation model is created before a product is derived based on a variability model.

When refining our evolution meta-model to a particular product line development environment, users define different types of trace links as relations. Examples of relations (not shown in Fig. 1) between product line elements are:

- *Project to model*: A specific model (stored for example in a file) becomes part of a project and is marked for change tracking after its creation.
- *Model to model*: A model is related to another model. For instance, a variability model is based on a certain product line meta-model. Since multiple variability models and meta-models can be stored in a workspace it is necessary to establish traceability to ease product line evolution.
- *Model to model element*: A model consists of an arbitrary number of modeling elements.
- *Model element to model*: A model element can be related to different other models. For example, if a requirement is captured in a derivation model [10] or a requirements document during application engineering, it is useful to also establish a trace link from the requirement to the variability model that must be evolved to address the new requirement.
- *Model element to model element*: Model elements are typically related to other model elements. For instance, a newly captured requirement can directly refer to existing model elements like features, decisions, or assets in a product line model.

III. EvoKing: Tool-Support for Tracking Evolution in Eclipse Workspaces

Our approach for tracking and managing evolution of product lines is supported by our Eclipse-based tool EvoKing. We intentionally did *not* use Eclipse libraries to implement the evolution meta-model to keep the core of our approach independent from Eclipse. We describe the refinement of our generic evolution meta-model and the extensions we developed to support tracking of artifact changes in Eclipse.

A. Refining the Meta-model for Eclipse

The **artifacts** tracked by EvoKing are Eclipse workspace entities like `IFile`, `IProject` or `IWorkbench`. They are defined in a refined evolution meta-model as shown in Fig. 1. Users configure EvoKing for an Eclipse-based modeling environment by specifying the artifacts of interest at a higher level of abstraction (the lower level implementation details like `IProject` or `IFile` are transparent to the user). For example, users specify the types of Eclipse projects they want to be tracked (e.g., “Java Project” or “Product Line Project”) or the file types (e.g., “Java source files” or “XY Models”).

Low-level **events** fired by the Eclipse framework (e.g., file change notifications) are automatically captured by EvoKing. EvoKing complements the existing notification mechanisms of Eclipse by adding an explicit meaning to events. For example, users can define in the evolution meta-model that whenever a new file of type “feature configuration” is added to the workspace, this shall be interpreted as the start of product derivation and a relation to a feature model should be created (see Section 4). This way a **relation** from a derivation project (i.e., stored in a feature configuration file) to a variability model (i.e., stored in a feature model file) is established.

B. Tool Architecture

EvoKing works as a consumer and recipient of event notifications coming from Eclipse or other custom event providers (cf. Fig. 3). Based on the incoming events and the

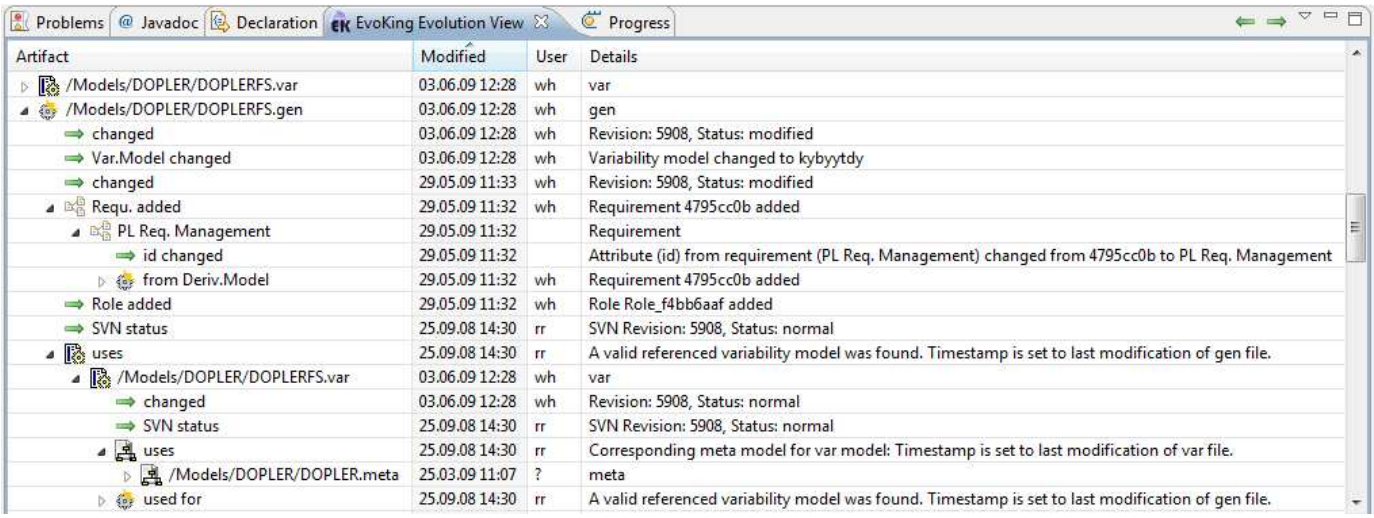

Figure 2. EvoKing Evolution View showing the change history of a DOPLER derivation model (.gen file) and a related requirement, variability model (.var file) and meta-model (.meta file).

defined artifacts, new events with more detailed information regarding context and semantics can be generated. Such *evolution events* are then stored for each artifact and can be browsed using the EvoKing evolution view (cf. Fig. 2). Other tools implementing a specific interface can also be registered as an observer to retrieve evolution events if they wish to be informed about changes and their meaning.

EvoKing supports the user in further refining the evolution meta-model. This includes support for the modeler to add code for resolving relations, to interpret events from Eclipse for specific models, and to enrich change events with context-specific, semantic information. Product line engineers can thereby customize EvoKing to support evolution in arbitrary Eclipse-based product line environments.

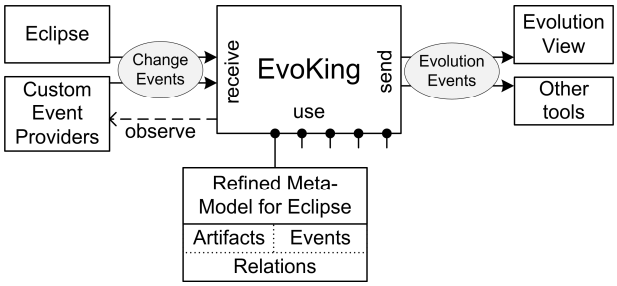

Figure 3. EvoKing's event architecture.

EvoKing recognizes change events based on information from two sources:

Eclipse resource change events such as `file added` or `file changed` and their sources are analyzed. EvoKing for example parses files representing models so that internal changes to models can be recognized using existing model APIs. Such changes are then mapped to artifacts and events defined in a refined evolution meta-model (see Section 4).

Custom event providers for models can send specific events to EvoKing. For example, if listeners have been implemented for a certain model type, they can be extended to explicitly fire change notifications. EvoKing is then registered as a listener for these models and can track changes being made to a model internally (e.g., model elements being added, deleted, or changed). Notifications are automatically transformed to `evolution events` according to the artifact and event definitions found in the evolution meta-model refined for a particular environment (cf. Section 4).

The EvoKing evolution view depicted in Fig. 2 shows all tracked artifacts of a project currently opened in Eclipse. The hierarchically organized representation of dependencies to other artifacts and all corresponding events allows users to quickly get an overview of the changes that have been occurring. Users can display details of a specific artifact at any time by expanding the tree, browsing through event details and related artifacts, and open editors for the elements the artifacts represent.

IV. Example Application of EvoKing: Evolution Management in DOPLER

Our testbed for EvoKing is the DOPLER product line engineering approach and tool suite [8]. We have been developing DOPLER in ongoing research collaboration with industry. The model-based, decision-oriented approach supports variability modeling and product derivation and provides tool support for creating, using, and managing diverse types of product line artifacts and models.

The product line artifacts (cf. Fig. 4) in DOPLER are product line meta-models, variability models, derivation models, and diverse model elements (e.g., assets, decisions, and product-specific requirements). The relevant dependencies between these artifacts are as follows: A variability model (`.var` file in Eclipse) uses a particular meta-model (`.meta` file); a derivation model (`.gen` file) is based on a specific variability

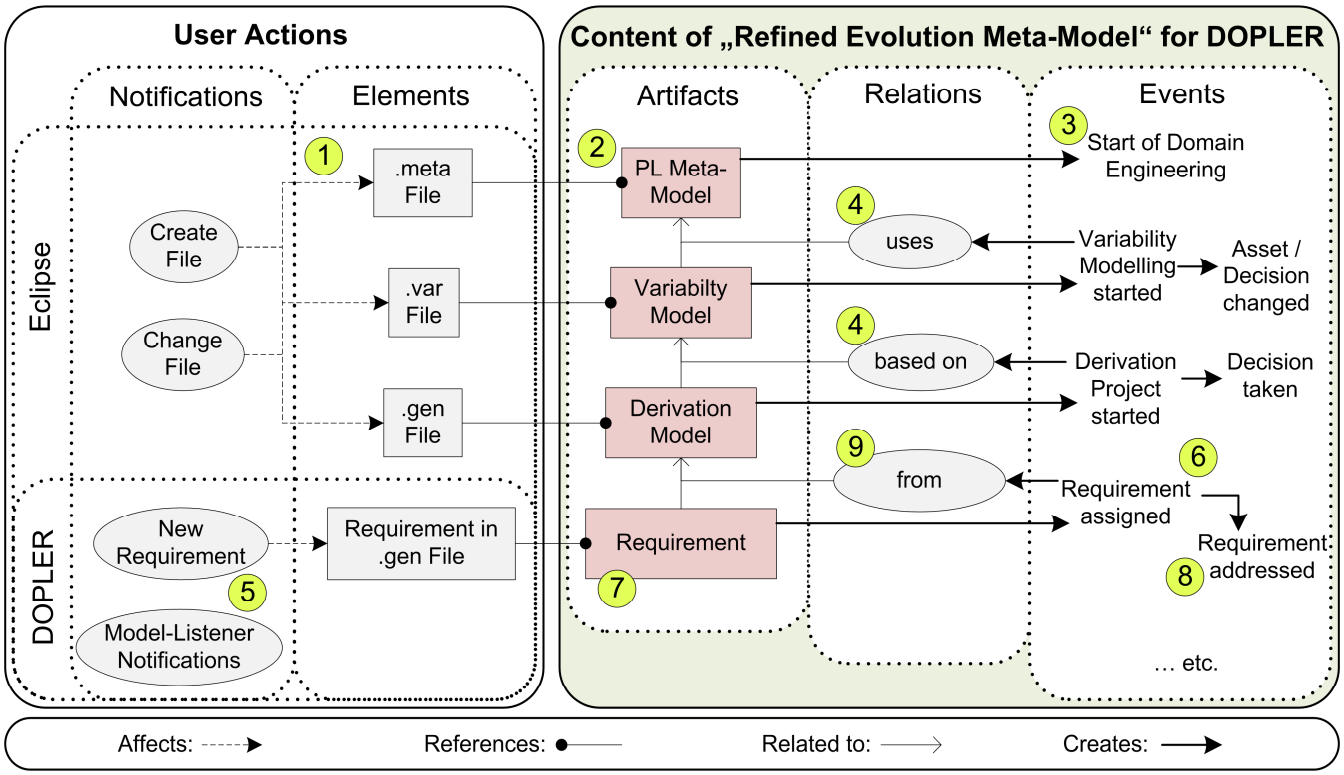

Figure 4. EvoKing customized for DOPLER. The left side shows elements and notifications we see within the workspace and editors. The right side shows artifacts, relations and events that represent the left side enriched with information taken from the refined evolution meta-model for DOPLER.

model; a requirement comes from a particular derivation model.

Evolution in DOPLER is for instance triggered by product-specific requirements captured during application requirements engineering. Requirements are captured in the derivation model representing a particular product derivation project. Implementing a requirement typically causes a change of the variability model (and thereby its elements like, i.e., assets and decisions).

Fig. 4 shows a simplified overview of how we customized EvoKing for DOPLER. Operations on files defined as model containers (`.meta`, `.var`, and `.gen` files) are captured and processed in the corresponding artifact implementations. For instance, for the creation of a `.meta` file (1) the artifact for the contained product line meta-model (2) is created. This leads to an evolution event indicating the start of domain engineering (3). This procedure works similar for other files and models. Starting variability modeling or starting a new derivation project additionally creates trace links between (4) the product line meta-model or variability model respectively. Independent of file changes, DOPLER-specific notifications are processed by the EvoKing artifacts. For instance, the DOPLER tool suite notifies EvoKing about model changes (5) like new model elements (i.e., assets, decisions, requirements) being added. EvoKing stores events containing this information (6) or, according to the refined evolution meta-model, new artifacts, (7) e.g., representing requirements, are held with their own evolution history (8) and relations to their origin (9).

EvoKing allows users to track the evolution of DOPLER product line meta-models, variability models, derivation models, and of the elements these models comprise. The customization of EvoKing to a different (Eclipse-based) product line environment would be pretty straightforward as most Eclipse-based product line environments store models in files in Eclipse projects and different model elements such as features or requirements are contained in the models.

V. Conclusions and Future Work

We presented a tool-supported approach for multi-level monitoring and tracking of changes to facilitate evolution in model-based product line engineering. Based on a generic meta-model for tracking evolution our tool EvoKing supports evolution management in Eclipse-based product line environments. We illustrated the applicability of our approach by customizing EvoKing for the DOPLER product line tool suite.

EvoKing automatically maintains a development history showing what and when was done by whom during development. There are, however, more advanced usage scenarios for the tool which we plan to explore in the future. For instance, we will use of the refined evolution meta-model and evolution information tracked by EvoKing to assist users with their *workflow* of modeling and creating product line

artifacts. We will also use the relations captured by EvoKing as trace links for the purpose of *consistency checking* in and between product line models and artifacts. This will help to point out potential update leaks or inconsistencies after changes to specified artifacts. We plan to improve support for *further development of artifacts and relations*. This way, for example, changes to configuration files, custom service configurations, and component interface definition files can be tracked to ease maintenance tasks. Finally, the information collected by EvoKing allows deriving product and process metrics to facilitate benchmarking, to monitor development processes, and to track variability shifts in product lines.

ACKNOWLEDGMENT

This work has been conducted in cooperation with Siemens VAI Metals Technologies and has been supported by the Christian Doppler Forschungsgesellschaft, Austria.

REFERENCES

[1] D. Dhungana, T. Neumayer, P. Grünbacher, and R. Rabiser, "Supporting Evolution in Model-based Product Line Engineering, "Proc. of the *12th International Software Product Line Conference (SPLC 2008)*, Limerick, Ireland, IEEE Computer Society, 2008, pp. 319-328.

[2] K. Czarnecki and C. H. P. Kim, "Cardinality-Based Feature Modeling and Constraints: A Progress Report, "Proc. of the *International Workshop on Software Factories at OOPSLA'05*, San Diego, USA, ACM Press, 2005, pp. 1-9.

[3] K. Schmid and I. John, "A Customizable Approach to Full-Life Cycle Variability Management," *Journal of the Science of Computer Programming, Special Issue on Variability Management*, vol. 53(3), pp. 259-284, 2004.

[4] H. Gomaa, *Designing Software Product Lines with UML*: Addison-Wesley, 2005.

[5] M. Voelter and I. Groher, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development, "Proc. of the *11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan, IEEE CS, 2007, pp. 233-242.

[6] A. Pasetti and O. Rohlik, "Technical Note on a Concept for the xFeature Tool," P&P Software GmbH / ETH Zurich, PP-TN-XFT-0001 2005.

[7] C. Krueger, "BigLever software gears and the 3-tiered SPL methodology, "Proc. of the *Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA'07)*, Montreal, Quebec, Canada, ACM, 2007, pp. 844-845.

[8] D. Dhungana, R. Rabiser, P. Grünbacher, and T. Neumayer, "Integrated tool support for software product line engineering, "Proc. of the *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, Atlanta, Georgia, USA, ACM, 2007, pp. 533-534.

[9] R. Rabiser and D. Dhungana, "Integrated Support for Product Configuration and Requirements Engineering in Product Derivation, "Proc. of the *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'07)*, Lübeck, Germany, IEEE Computer Society, 2007, pp. 219-228.

[10] R. Rabiser, P. Grünbacher, and D. Dhungana, "Supporting Product Derivation by Adapting and Augmenting Variability Models, "Proc. of the *11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan, IEEE Computer Society, 2007, pp. 141-150.