Documenting Application-Specific Adaptations in Software Product Line Engineering

Günter Halmans[1,*], Klaus Pohl[2], and Ernst Sikora[2]

[1] RDS Consulting GmbH
Mörsenbroicher Weg 200, 40470 Düsseldorf, Germany
guenter.halmans@rds.de
[2] Software Systems Engineering, University of Duisburg-Essen
Schützenbahn 70, 45117 Essen, Germany
{klaus.pohl,ernst.sikora}@sse.uni-due.de

Abstract. 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. In application engineering domain artefacts are reused to create specific applications.

Application engineers often face the problem that individual customer needs cannot be satisfied completely by reusing domain artefacts and thus application-specific adaptations are required. Either the domain artefacts or the application artefacts need to be modified to incorporate the application-specific adaptations. We consider the case that individual customer needs are realised by adapting the application artefacts and propose a technique for maintaining traceability between the adapted application artefacts and the domain artefacts. The traceable documentation of application-specific adaptations is facilitated by an application variability model (AVM) which records the differences between the domain artefacts and the application artefacts of a particular application. The approach is formalised using graph transformations.

Keywords: product line engineering; variability modelling; application-specific adaptations; traceability.

1 Introduction

Software-product line development differentiates between two processes [5] [12]: domain engineering and application engineering. In domain engineering, domain artefacts (domain requirements, domain architecture, domain components, domain test cases, etc.) are developed for reuse. In application engineering, domain artefacts are reused for defining and realising specific applications. Domain artefacts and application artefacts are interrelated by traceability links to support the different domain and application engineering tasks (cf., e.g. [12]). In addition, traceability links are established between requirements, architecture, components, and test artefacts, both,

[*] The work reported in this paper was performed while Günter Halmans was member of the Software Systems Engineering Group at the University of Duisburg-Essen.

Z. Bellahsène and M. Léonard (Eds.): CAiSE 2008, LNCS 5074, pp. 109–123, 2008.

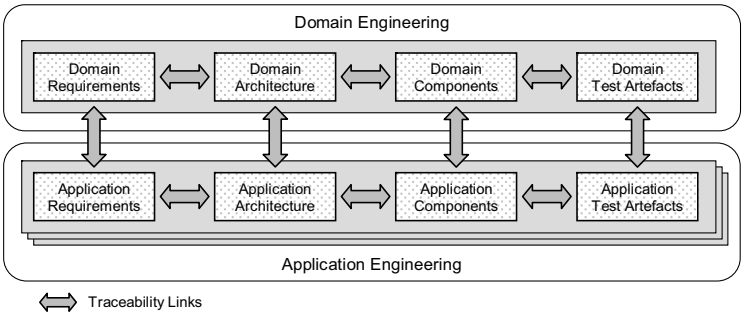

Fig. 1. Overview of domain and application artefacts including traceability links

in domain and application engineering. Fig. 1 provides a schematic overview of the two processes, the key artefacts and the traceability links.

The second, central characteristic of product line engineering is the distinction between common and variable artefacts. *Common domain artefacts* become part of each application derived from the product line. The product line variability denotes the *variable artefacts* and thus defines the possible variations among the products (applications) of a product line. Variable artefacts may (or may not) be selected for a particular application. The process of selecting the variable artefacts required for a specific application during application engineering is called the binding of variability [3]. The selection (i.e. the binding) is documented in order to ensure, for instance, the traceability of the application artefacts to the domain artefacts.

In practice, product line applications can typically not be derived 100% from the domain artefacts. For example, if a customer requirement is not part of the domain requirements, the application engineers have to create additional artefacts and/or adapt existing artefacts in order to realise the customer requirement. Thus, in addition to reusing artefacts from domain engineering, application-specific artefacts have to be defined (see e.g. [1] [4] [8] [12] [15] [16]). By *application-specific artefacts* we denote all development artefacts (including requirements, architecture, components, and test artefacts) needed specifically for a particular application. Application-specific artefacts can be realised either by extending and/or adjusting the domain artefacts and deriving the required application-artefacts from the adapted domain artefacts or by adapting the application artefacts directly.

For instance, in [10] application-specific requirements changes are realised by adapting the domain requirements. In [2], the inclusion of new components in the domain artefacts is addressed in order to realise application-specific changes. Further examples of approaches supporting the adaptation of domain artefacts to realise application-specific adaptations are [6] and [13].

In contrast, [15] and [16] address the realisation of application-specific adaptations by adapting the application artefacts. However, these approaches do not provide a systematic procedure for realising the adaptations. Thus, for instance, no traceability relationships between the adapted application artefacts and the domain artefacts are maintained. The KobrA approach foresees the adaptation of application artefacts, yet does not provide a means to document the adaptations in a traceable manner [1]. The approach in [8] realises application-specific changes by adapting application artefacts

which are represented as UML diagrams. However, the changes are incorporated in each diagram, and thus, there is no central, consistent model where the adaptations are documented. In addition, the approach does not address the issue of providing traceability between the domain artefacts and the adapted application artefacts.

Summarising, none of the stated approaches supports the traceable documentation of application-specific adaptations of application artefacts throughout different artefact types (requirements, architecture etc.).

In this paper, we argue that application-specific adaptations should be incorporated at the application engineering level and that the application-specific changes must be interrelated with the domain artefacts (cf. Section 2). In Section 3, we introduce an application-specific variability model (AVM) as a means to realise and document traceable, application-specific adaptations. In Section 4, we sketch the prototypical realisation of our approach in a tool environment based on typed attributed graph transformation systems. In Section 5, we illustrate our approach using a simplified example. Section 6 summarises the paper and provides an outlook on future work.

2 Adjusting Domain Artefacts or Application Artefacts?

As mentioned in the introduction, application-specific adaptations can be realised either by adapting the domain artefacts (Section 2.1) or by adapting the application artefacts, i.e. at the application engineering level (Section 2.2).

2.1 Adaptation of Domain Artefacts

Application-specific adaptations can be realised by increasing the variability of the product line and by adding and/or adjusting domain artefacts in such a way that ***all*** required application artefacts can be derived from the domain artefacts. This approach is appealing as it simplifies the application engineering process for the specific application. However, an adaptation of domain artefacts leads to an evolution of the product line. As described, for instance, in [5] and [12], the evolution of a product line should be determined by the product line strategy of the organisation rather than by individual customer wishes. Typically, proposed changes of domain artefacts have to be approved by a change control board before being incorporated in the domain artefacts of the product line. The domain artefact change control board enforces the product line strategy and takes into account in its decisions the costs and benefits of each proposed change. For instance, the following "cost drivers" have to be considered, when deciding about the implementation of individual customer wishes through the adaptation of domain artefacts:

- *Increased complexity of domain artefacts*: Incorporating a huge number of individual customer's wishes into the domain artefacts increases the complexity of the domain artefacts as well as the variability of the product line [12]. Realising application-specific artefacts through variable domain artefacts is thus in conflict with a typical product line strategy: to facilitate testing of domain artefacts as early as possible, which requires as little variability as possible [14]. Moreover, an increase of variability leads, to a more complex product derivation process for future applications as, for

instance, the application engineer has to take a higher number of decisions to derive a new application.

- *Adjustment of all applications already derived from the product line*: The applications that have already been derived from the product line or are being derived could be affected by the adaptation of the domain artefacts. One reason for integrating the changes made to the domain artefacts in all existing applications of the product line is to facilitate future maintenance of these applications (cf., e.g. [11]). However, some applications might be affected by the adaptation of the domain artefacts in an unexpected manner. Hence, the effects of the adaptations on each existing application have to be checked. For implementation artefacts the checking can be performed by an automatic regression test of the applications. Yet, requirements and architectural artefacts typically need to be checked manually which is a time and resource-intensive process, in particular, when the product line has a large number of applications.

Several researchers recommend to realise individual customer wishes by adapting application artefacts directly (cf., e.g. [4] [12] [15]) which avoids the high effort related to product line evolution. Even if satisfying the individual customer's wishes matches well the product line strategy and offers a high benefit for the entire product line, the product line organisation may decide to develop a lead product before the proposed changes are incorporated into the domain artefacts. Summarising, there are strong arguments for maintaining a clear separation between product line evolution and the realisation individual customers' wishes. Prior to adapting the domain artefacts one has perform a cost-benefit-analysis that takes into account the product line strategy and the entire set of applications.

2.2 Application-Specific Adaptation of Application Artefacts

The other option, which avoids the high effort of adapting the domain artefacts, is to realise application-specific artefacts at the application engineering level. In this case application-specific adaptations are realised by changing and extending the application artefacts derived from the domain artefacts. Thus, the adaptations only affect the application under development instead of all existing or future applications.

When changing the derived application artefacts to incorporate application-specific requirements, obviously, the resulting application artefacts differ from the domain artefacts. If the product line organisation does not establish a systematic approach for dealing with those differences, severe problems may occur that undermine the advantages of product line engineering (cf. [11]). There is thus a need to record the interrelations between the domain artefacts and the application-specific adaptations of application artefacts. We illustrate this by the following two cases:

- ***Case A:*** Assume that a particular domain requirement is selected for the application and that this requirement is slightly modified to satisfy the customer need. If there is no documentation of this adaptation and no relation to the original domain artefact, the application architect may not be aware of the fact that the application-specific requirement is based on a domain requirement. Thus, the architect might not know that there are domain components which realise the domain requirement (which the application-specific requirement is based on) and which could be modified to fulfil

the application-specific requirement. When designing the application architecture, the architect would assume that the components required for implementing the application-specific requirement have to be developed anew. Likewise, the application tester is not able to identify the domain test artefacts that, with some modifications, could be reused for testing the application-specific requirement. Summarising, if there is no adequate traceability information from the specific application requirement to the reused domain requirement, it is close to impossible to identify, later on in the development process, that the application-specific requirement is based on a particular domain requirement (i.e. that there was a reuse with modifications).

- ***Case B:*** Assume that two variable domain requirements exclude each other, for instance, because each requirement alone causes about 60% system load. One of the two domain requirements is 100% reused for an application and the other is adapted for the application. If the application-specific requirement is not linked with the original domain requirement, it is very difficult for an application architect to recognise the conflict. Consequently there is a risk that an application is developed that does not fulfil its performance requirements since the application developers are not aware of the existing conflict.

We conclude that application-specific adaptations of application artefacts should be traceable and the application artefacts should be interrelated with the domain artefacts. In general, a particular application-specific adaptation most likely influences all or most types of application artefacts (requirements, architecture, components, test cases, etc.). For instance, a home security product line might include two authentication mechanisms for unlocking the front door: to enter a personal identification number in a key pad or to swipe a personal access card through a card reader. Incorporating a new authentication mechanism, for instance, biometric authentication, for a specific home security system would change the requirements, possibly the architecture, the implementation as well as the test cases. In the following, we discuss two approaches to tracing adaptations in application artefacts.

- ***Documenting Adaptations within Application Artefacts:*** Keeping track of the differences between application artefacts and domain artefacts can be achieved by documenting the differences individually for each type of application artefacts. In this case, application-specific changes in application requirements are interlinked with the corresponding domain requirements, changes in the application architecture artefacts are interlinked with the domain architecture artefacts and so forth. Using this approach, the traceability information for a particular change is split across the various artefact types. The splitting entails that the impact of a particular application-specific adaptation on the different application artefacts is difficult to recognise.
- ***Documenting Adaptations in a Dedicated Model*:** In order to avoid the splitting of traceability information across the various artefact types, we suggest to use a dedicated model to document the application-specific adaptations. The key idea behind this approach is to interpret application-specific changes as variation of the application artefacts with respect to the domain artefacts, i.e. to interpret the adaptation as application-specific variability. For instance, the change of the authentication mechanism of a specific home security system can be regarded as a variation

between the specific system and the home security product line, regardless which artefacts have to be adapted in order to realise this change.

We call the dedicated model used to record application-specific variations *application variability model*. The idea of the application variability model is similar to the orthogonal modelling of the variability at the domain level (cf. [12]). Documenting the application-specific variations in an application variability model has, essentially, the same advantages as documenting product line variability in a dedicated model and not in the domain artefacts themselves (cf. [3]), for instance:

- The application variability model can be used to identify, communicate, and reason about application-specific changes as opposed to having to consider the adaptations made in different types of artefacts where each adaptation only provides a partial view on an application-specific change.
- The application variability model serves as an entry point for navigating to all application artefacts (requirements, architecture, test cases etc.) affected by an application-specific change. This makes it easier to keep the adaptations consistent across the different artefact types and development phases.

3 Application-Specific Variability Model

This section sketches our solution for documenting application-specific adaptations using an application variability model (AVM). In Section 3.1, we briefly recapitulate how product line variability is documented using an orthogonal variability model (OVM). In Section 3.2, we introduce the application variability model for documenting application-specific adaptations.

3.1 Orthogonal Variability Modelling

Product line variability is typically represented by variants, variation points and dependencies between variants and variation points. We use our orthogonal variability model (OVM) to document product line variability. A detailed description of the variability modelling language is given in [12]. Fig. 2 depicts a simplified example of a variability model for a home security system. A variation point is represented as a triangle. The variation point named *door lock mechanism* in Fig. 2 is related to the two variants *key card* and *key pad* which are represented as rectangles. The dashed lines between the variants and the variation point represent optional variability dependencies. A variant with an optional variability dependency can, but does not have to be selected during the binding of variability. In contrast, a mandatory variability dependency (represented by a solid line) means that the corresponding variant has to be selected. The model further indicates by the arc annotated with the cardinality *[1..1]* that exactly one of the two variants of the variation point *door lock mechanism* must be selected. Selecting the variant *key pad* means that the resulting home security system requires the user to enter a personal identification number using a key pad for authentication. Selecting the variant *key card* means that the home security system performs the authentication by checking a key card and the identification number saved on it. The variants of the variability model are connected with domain artefacts

by so called *artefact dependencies*, which are represented by dotted lines with arrow heads. The artefact dependencies depicted in Fig. 2 express that, e.g. the requirements *R 12* and *R 13* specify the variant *key pad*. So called constraint dependencies are used to model that two variants, two variation points or a variant and a variation point either exclude each other or that one requires the other.

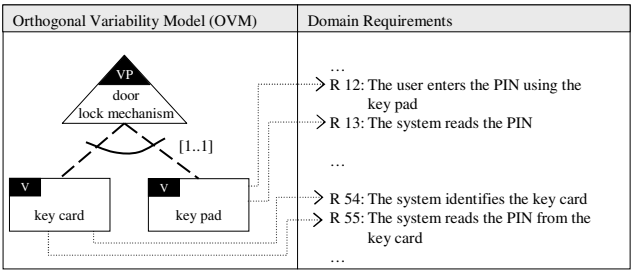

Fig. 2. Example of an orthogonal variability model

3.2 Application Variability Model

As outlined in Section 2, we propose to document application-specific changes (e.g. additional variants or additional variation points) in an *application variability model* (AVM). As indicated in Fig. 3, an AVM is defined for each application to document the application-specific adaptations with respect to the domain variability model (DVM). The AVM captures the application-specific variability of all types of application artefacts such as requirements, architecture, etc. As the application engineering process proceeds, requirements variability is mapped to architecture variability and so forth (cf. [12]). The traceability between the AVM and the different types of application artefacts is established by means of artefact dependency links.

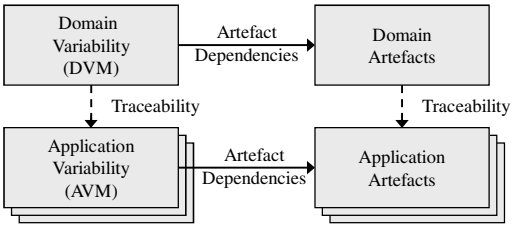

Fig. 3. DVM and AVM

In the following, we outline four main cases of how the AVM supports the traceable documentation of application-specific adaptations. The four cases apply to all types of artefacts. Yet, for simplicity, we use requirements to illustrate the cases.

Case 1 – 100% reuse: The application requirement can be defined completely by reusing domain requirements. In this case, no application-specific adaptation is required. Nevertheless, the binding of the selected variant has to be documented. By

selecting a variant, the domain requirements related to this variant are completely reused. If, e.g., the variant k*ey card* (cf. Fig. 2) is selected, the requirements R54 and R55 become part of the application requirements. The result of this variability binding is illustrated in Fig. 4: The selected variant, the related variation point *door lock mechanism*, and the dependencies to the application requirements *R54* and *R55* become part of the bound AVM.

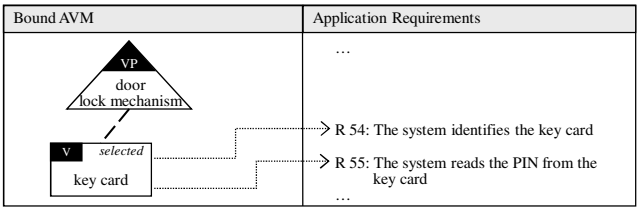

Fig. 4. Bound AVM with complete reuse

Case 2 – No reuse: In this case, an application requirement cannot be fulfilled by reusing domain requirements and must therefore be developed from scratch. Assume that an application requirement demands a door locking mechanism with a finger print scanner, which is not offered by the domain requirements. In this case, a new variant *finger print* is introduced in the AVM and related to the corresponding requirements (cf. Fig. 5). As the variant is required for the application it is also selected for this application. When realising the application-specific variant *finger print*, the application engineers may either relate the application-specific adaptations of system components, test cases etc. to the variant finger print, or define additional variants and variation points that represent the application-specific adaptations in the architecture, test cases etc. The addition of a variation point for representing architectural adaptations is illustrated in Section 5. Since the architectural artefacts, test cases etc. required for the new functionality are related to the elements of the AVM, the impact of the application-specific adaptations becomes traceable across the different artefacts types.

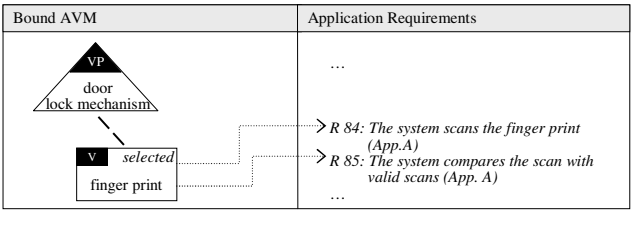

Fig. 5. New variant with related requirements

Case 3 – Partial reuse: In this case, the application requirement can be partially defined by reusing domain requirements. In other words, domain requirements can be reused but some application-specific adaptations are needed. Assume, for example,

that the application-specific requirement *R12a* is defined by partially reusing the domain requirement *R12*. *R12a* contains a restriction of the PIN length whereas *R12* has no such restriction (cf. Fig. 6). To record this adaptation, the artefact dependency from the variant *key pad* to *R12* is marked as deleted and labelled with the string *App.A* to denote that this element is application-specific for the application *A*. Moreover, the new application requirement *R12a* is included, and a new artefact dependency is introduced from the variant *key pad* to *R12a.* This artefact dependency is also labelled with *App.A*, and a reference is added to requirement *R12a* to record that this requirement is the partially reused domain requirement *R12* (cf. *App.A, former R12*) thus providing traceability.

Note that the described solution for the documentation of the specific application requirement *R12a* is one of many possible solutions. Alternatively, a new variant could have been introduced and new artefact dependencies from the new variant to the application requirements *R12a* and *R13* could have been documented. Since different modelling choices exist to document the same kind of adaptation, the documentation of adaptations cannot be automated. Rather, the engineer has to decide, which modelling choice is appropriate for the particular case. However, once the engineer has chosen a particular modelling option, the validity of the resulting AVM with respect to the DVM can be checked automatically.

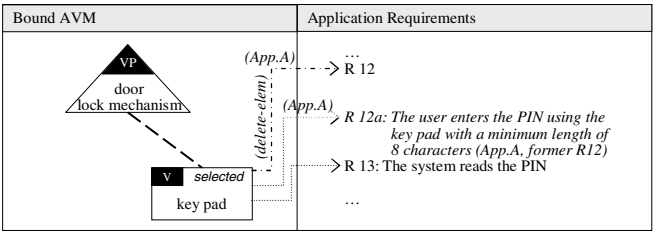

Fig. 6. Partial reuse

Case 4 – Conflict with the product line variability: In this case, the application-specific extension is in conflict with the variability defined in the DVM. This conflict must be resolved in order to realise the application-specific adaptation. For example, assume an application requirement for a specific application requires both door locking mechanism depicted in the DVM of Fig. 6. In this case, a conflict exists between the domain variability model and the specific application requirement. In the DVM, the cardinalities of the alternative group, which encompasses the two variants *key card* and *key pad*, are defined as *[1..1]* and thus one (and only one) of the two variants can be selected for an application. To resolve this conflict, the restriction of the variant selection with respect to the variation point *door lock mechanism* has to be suspended for the application. Thus, in the AVM the upper bound of the cardinalities is changed to 2 (cf. ❶ in Fig. 7) and thereby the selection of both variants for this application is enabled (in terms of the variability model). The label *App.A* on the alternative group represented by the arc of a circle makes the adaptation traceable.

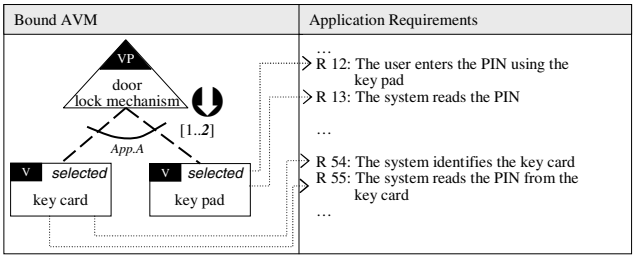

Fig. 7. Conflict with DVM

To summarise, we document application-specific adaptations in the AVM (application variability model) and relate all application artefacts affected by an application-specific adaptation to the corresponding elements in the AVM. Moreover, if an application-specific artefact is based on a domain artefact, the application artefact is additionally related to the domain artefact it is based on thus ensuring traceability. Deleted elements are marked by appropriate labels. Thus, the AVM facilitates the documentation of all application-specific adaptations in one central model and thereby facilitates consistency checks as well as an easier comprehension of the adaptations.

4 The Tool Environment

To support the documentation of the application-specific changes and to ensure the consistency between the AVM and the DVM we have formalised our approach using graph transformations. We decided to use the integrated development environment for graph-transformation AGG (Attributed Graph Grammar System) because, among others, AGG is well-founded by the theory of categories, has been successfully used in various settings and provides a comprehensive set of techniques for validating graphs and graph transformation systems, e.g. parsing, critical-pair-analysis, and consistency checks (cf. [7] for more details).

4.1 Graph Transformation Systems as a Basis for Adapting and Binding Variability Models

We realised the following two graph transformation systems in AGG as a basis for defining variability, recording adaptations of variability models and for binding variability:

- ***VM-GTS*:** The typed attributed graph transformation system *VM-GTS* provides the formalism to model and adapt variability models including the needed traceability. The VM-GTS contains a type graph, which represents the variability meta model. The type graph defines node types (i.e. variation points, variants, application requirements, etc.) and edge types (i.e. variability dependency, constraint dependency, artefact dependency). Moreover, in the type graph, attributes of nodes and edges are defined. These attributes are used for the unique identification of nodes and edges as well as for holding traceability information. In addition to the type

graph, the VM-GTS contains a set of graph transformation rules. These rules represent the permissible operations on a variability model such as the insertion of new variants for a given variation point. The rules are generic in the sense that they contain variables for the different attributes that are instantiated during the application of a specific rule (Section 4.2). Finally, the VM-GTS contains pre- and post-conditions, which ensure that the variability model is well-formed. The pre- and post-conditions prohibit, for instance, that two variants are connected by both, an *exclude* and a *requires* dependency.

- ***BV-GTS*:** The BV-GTS (Binding Variability Graph Transformation System) supports the binding of the variability for the application. The BV-GTS uses an AVM to generate a bound AVM. The input for the BV-GTS is a labelled AVM, i.e. an AVM where the selected variants are indicated by labels. The derivation of the BV-GTS results in a bound AVM that includes only selected variants. Derivation of the BV-GTS means that all graph transformation rules defined in the BV-GTS are applied until no graph transformation can be applied any more. The BV-GTS ensures that only valid bindings are performed, i.e., for instance, all variation points, variability dependencies, and artefact dependencies of the selected variants as well as variants that are required by the selected variants are bound. The resulting AVM is thus a valid variability model.

4.2 Generating an AVM

Based on the VM-GTS we are able to define a graph transformation system specifically for an application, which documents the adaptations, makes the adaptations visible in the AVM, and labels the selected variants. The so called **A**pplication **V**ariability **M**odel **G**raph **T**ransformation **S**ystem (AVM-GTS) includes the type graph of the VM-GTS and a set of instantiated rules from the VM-GTS rule-set including the corresponding pre- and post-conditions. To define an AVM-GTS, firstly, the needed graph transformation rules from the rule-set of the VM-GTS have to be selected. The selection is determined by the modeller, since, typically, more than one solution is available to achieve a required adaptation. Secondly, the selected rules are instantiated. During instantiation, for instance, the generic rule for inserting a variant (from the VM-GTS) is concretised with values such as the short name for the variant. These concrete values for specific attributes are also used to find a particular element in the DVM, e.g. the variation point to which the new variant is related. Third, the AVM-GTS is derived based on the DVM. Derivation of the AVM-GTS means that the selected and instantiated graph transformation rules of the AVM-GTS are applied until no rule is applicable anymore. This activity is performed automatically. The output is an AVM that has been generated from the DVM by applying the transformation rules defined in the AVM-GTS.

4.3 Evaluation

Our experience in defining the VM-GTS and the BV-GTS in AGG indicates that the definition of the different graph transformation rules, the type graph as well as the conditions is fairly simple. We applied our approach to several examples of varying complexity (in terms of the size of the variability model and the number of adaptations)

for generating and binding AVMs. The performance results obtained for generating adapted AVMs from two different DVMs are shown in Table 1. The full set of examples including the performance results has been published in a thesis (cf. [9]). The experimentation with our prototypical implementation based on AGG supports our claim that our approach for the traceable documentation of application-specific adaptations is feasible. In particular, by using the tool prototype, we gained the following insights about our approach:

- Our orthogonal variability model in combination with graph transformations provides a sound basis for the precise documentation of application-specific adaptations in product line engineering.
- The recording and management of inter-model trace links for product line artefacts can be greatly eased by the use of graph transformation rules.
- The generation of application artefacts from domain artefacts including application-specific adaptations and the binding of variability can be accomplished quite easily by using a graph transformation environment.

The observed increase in runtime between the first and the second example depicted in Table 1 can be explained by the complexity of graph matching. The reduction of the time for generating the AVM is a topic of further investigations.

Table 1. Performance results for the prototypical implementation with AGG (excerpt)

#Variation points in DVM	#Variants in DVM	#Artefact dependencies in DVM	#Adaptations in AVM-GTS	Time for generating the AVM [seconds]
10	20	60	5	5
20	40	120	15	45

5 Example

In this section, we illustrate the application of our approach on a simplified example from the e-commerce domain. The example consists of a domain variability model, a domain use case and an excerpt of a domain components model including the relevant artefact dependencies between the domain variability model and the other domain artefacts (see Fig. 8). We omit technical details as these have already been described in Section 4. The domain variability model depicted in Fig. 8 allows to choose between two payment methods allowing the application engineer to derive applications either supporting credit card payment or direct debit payment. The choice affects the example use case (either step 2 [V1] or step 2 [V2] is selected) and the example components model (either the credit card payment plug-in or the direct debit payment plug-in is selected).

In the application engineering process, the customer expresses the following requirements that demand application-specific adaptations:

- The application shall offer both payment methods, credit card and direct debit.
- The user shall be able to choose the payment method before entering the required payment details.

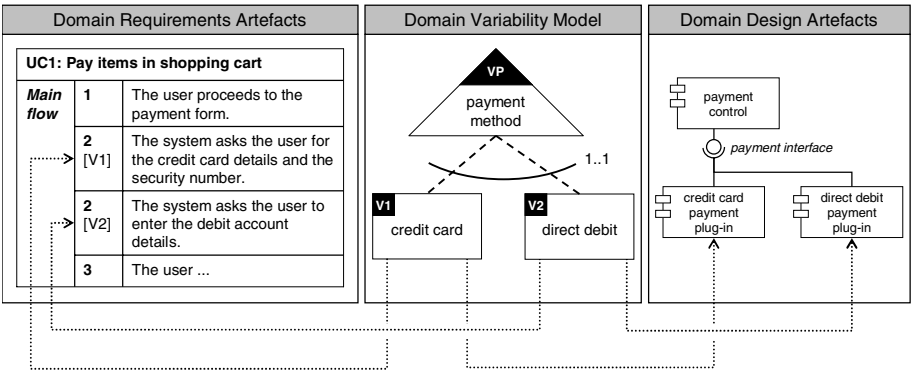

Fig. 8. Example of a domain variability model with associated domain requirements and domain design artefacts (excerpt)

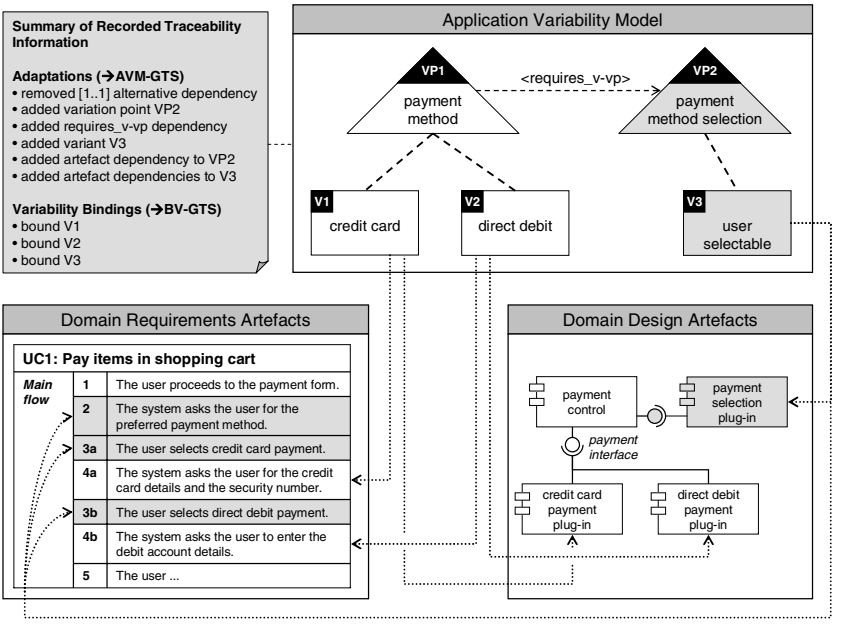

Fig. 9. Example of an application variability model with recorded traceability information (as natural language text) and application artefacts including modifications

Fig. 9 shows the resulting application variability model and the associated application artefacts. In the variability model, variants *V*1 and *V*2 have been bound. To enable the binding of both variants for an application, the alternative dependency between *V*1 and *V*2 had to be deleted. Furthermore, the required adaptations in the application use case and the application design have been documented in the application variability model by inserting the variation point *payment method selection* and a

single variant *user selectable* together with the corresponding artefact dependencies. In addition, variants *V*1, *V*2 and *V*3 have been bound. An informal summary of the traceability information recorded by our approach is shown on the upper left of Fig. 9. In our tool, the stated adaptations and bindings are recorded by selecting and instantiating the needed graph transformation rules from the VM-GTS and the BV-GTS respectively as described in Section 4. With the recorded information, the AVM for the specific application can be generated from the DVM.

6 Summary and Outlook

When developing product line applications, typically not all application requirements can be fulfilled by reusing domain artefacts designed and developed in domain engineering. Thus, application-specific adaptations of the domain artefacts (including extensions) are required. We proposed, in this paper, to record application-specific adaptations in a dedicated model, the application variability model (AVM). Documenting the adaptations in an AVM ensures that the adaptations are defined in a central location (the AVM) and provides the basis for reasoning about application-specific adaptations across different artefact types. To facilitate the reasoning about consistency between the application-specific adaptations and the domain artefacts (including the product line variability) we have formalised our approach using graph transformation systems which have been implemented in a prototypical tool environment. We have applied our prototypical environment to define several applications for a product line with different complexities, including a product line from the home automation domain. When defining application-specific adaptations for several applications, we encountered several inconsistencies which were automatically detected by our tool environment. Those inconsistencies, especially with respect to the domain variability model, were in many case not easy to detect, i.e. without an automated tool support they would have most likely not been detected. In our future work, we will develop a better graphical visualisation of the AVM together with a graphical editor which eases the task of documenting application-specific adaptations. We will then use the extended tool environment in a real case study.

Acknowledgments. This paper was partially funded by the DFG projects PRIME (Po 607/1-1) and IST-SPL (Po 607/2-1). We would like to thank Prof. Dr. Michael Goedicke for the fruitful discussions during the elaboration of our approach.

References

1. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., Zettel, J.: Component-Based Product-Line Engineering with UML. Addison-Wesley, UK (2002)
2. Baum, L., Becker, M., Geyer, L., Molter, G.: Mapping Requirements to Reusable Components using Design Spaces. In: Proc. of the 4th Intl. Conference on Requirements Engineering (ICRE 2000), pp. 159–167. IEEE Computer Society, Los Alamitos (2000)

3. Bühne, S., Lauenroth, K., Pohl, K.: Modelling Requirements Variability across Product Lines. In: Atlee, J.M. (ed.) 13th IEEE Intl. Conference on Requirements Engineering, pp. 41–50. IEEE Computer Society, Los Alamitos (2005)
4. Bosch, J., Ran, A.: Evolution of Software Product Families. In: Van der Linden, F. (ed.) Software Architectures for Product Families, International Workshop IW SAPF 3, Las Palmas de Gran Canaria, Spain, pp. 169–183. Springer, Heidelberg (2000)
5. Clements, P., Northrop, L.: Software Product Lines – Practices and Patterns. Addison-Wesley, Boston (2001)
6. Eriksson, M., Börstler, J., Borg, K.: The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 33–44. Springer, Heidelberg (2005)
7. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
8. Gomaa, H.: Designing Software Product Lines with UML. Addison-Wesley, Boston (2004)
9. Halmans, G.: Ein Ansatz zur Unterstützung der Ableitung einer Applikationsanforderungsspezifikation mit Integration spezifischer Applikationsanforderungen (in German). Doctoral Dissertation, Logos Verlag, Berlin (2007)
10. Mannion, M., Kaindl, H., Wheadon, J.: Reusing Single System Requirements from Application Family Requirements. In: Proc. of the 21th Intl. Conference on Software Engineering (ICSE 1999), pp. 453–462. ACM Press, New York (1999)
11. Mohan, K., Ramesh, B.: Change Management Patterns in Software Product Lines. Communications of the ACM 49(12), 68–72 (2006)
12. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Heidelberg (2005)
13. Padmanabhan, P., Lutz, R.R.: Tool-Supported Verification of Product Line Requirements. In: Automated Software Engineering, vol. 12(4), pp. 447–465. Springer, Heidelberg (2005)
14. Reuys, A., Kamsties, E., Pohl, K., Reis, S.: Model-Based System Testing of Software Product Families. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 519–534. Springer, Heidelberg (2005)
15. Raatikainen, M., Soininen, T., Männistö, T., Mattila, A.: Characterizing Configurable Software Product Families and their Derivation. In: Software Process Improvement and Practice, vol. 10(1), pp. 41–60. Wiley, Chichester (2005)
16. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering, A Family-Based Software Development Process. Addison-Wesley, Boston (1999)