

# ULRR

## A formal approach to reuse successful traceability practices in SPL projects

Item Type	Meetings and Proceedings
Authors	Espinoza, Angelina;Botterweck, Goetz;Garbajosa, Juan
Citation	SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing;pp. 2352-2359
Publisher	Association for Computing Machinery
Download date	2026-04-20 15:58:36
Item License	<a href="https://creativecommons.org/licenses/by-nc-sa/1.0/">https://creativecommons.org/licenses/by-nc-sa/1.0/</a>
Link to Item	<a href="https://hdl.handle.net/10344/761">https://hdl.handle.net/10344/761</a>

# A Formal Approach to Reuse Successful Traceability Practices in SPL Projects

Angelina Espinoza  
Systems and Software  
Technology Group (SYST)  
Technical University of Madrid  
(Universidad Politécnica de  
Madrid - UPM, Spain)  
aespinoza@syst.eui.upm.es

Goetz Botterweck  
Lero, The Irish Software  
Engineering Research Centre  
University of Limerick, Ireland  
goetz.botterweck@lero.ie

Juan Garbajosa  
Systems and Software  
Technology Group (SYST)  
Technical University of Madrid  
(Universidad Politécnica de  
Madrid - UPM), Spain  
jgs@eui.upm.es

## ABSTRACT

Software Product Line (SPL) Engineering has to deal with interrelated, complex models such as feature and architecture models, hence traceability is fundamental to keep them consistent. Commonly, a traceability schema must be started from scratch from project to project. To avoid that, useful traceability practices to solve day to day problems should be modeled explicitly and kept as part of the traceability knowledge gained, and then organizations can reduce time and effort in implementing traceability in new projects. This paper presents an approach for formalizing and reusing traceability practices in SPL Engineering. Using this formalization approach a traceability metamodel is defined, incorporating the particular traceability practices performed in SPL Engineering. Customized traceability methodologies for SPL projects will be systematically and formally generated from this metamodel. These resulting methodologies will have already incorporated the traceability knowledge proven as successful in previous projects, facilitating the reuse of such practices. In this paper, we focus specifically on the product derivation process, to show the advantages of this formalization approach to reuse traceability knowledge.

## Categories and Subject Descriptors

D.2.1 [Requirements/Specifications]: Methodologies;  
D.3.1 [Reusable Software]: Reuse models; I.6.5 [Model Development]: Modelling methodologies

## General Terms

Management, languages, documentation

## Keywords

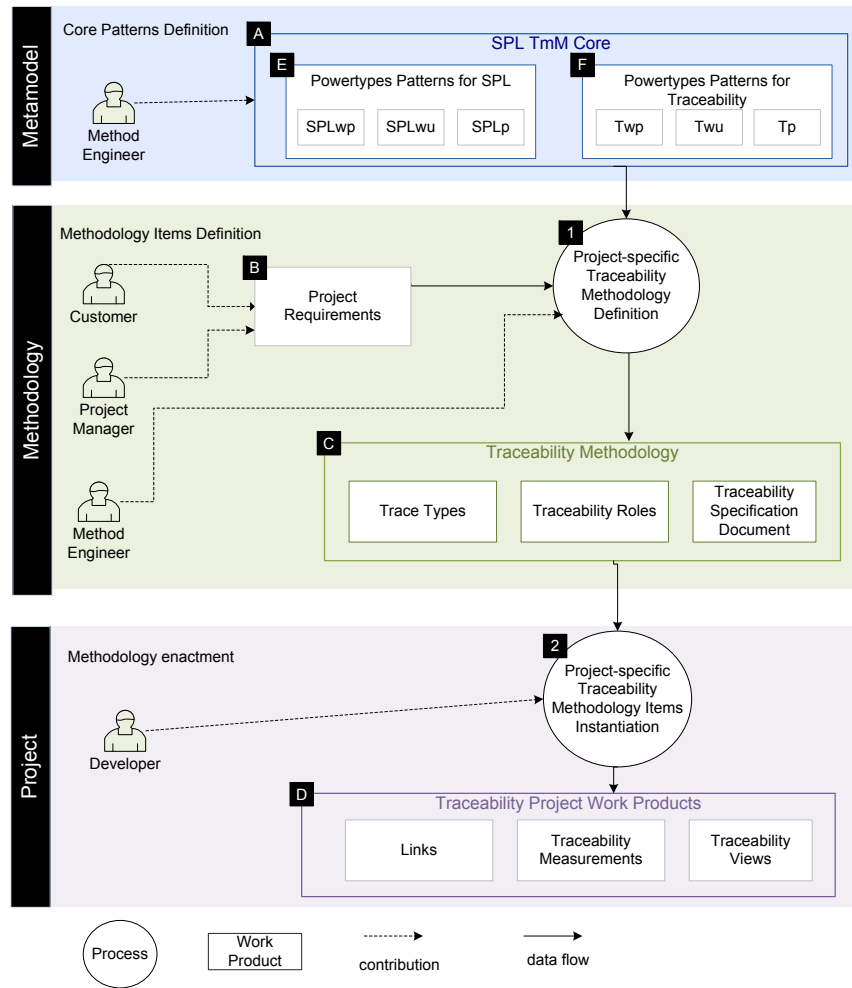
Traceability metamodeling, traceability methodology, software product lines, product derivation, feature configuration, knowledge reuse

## 1. INTRODUCTION

In software process management, traceability support is fundamental and is needed among the different activities, roles, and artefacts characterizing a software development process [14, pg. 169]. Similarly, in SPL engineering [9, 15], traceability support is required both 1) between different software models (e.g., feature and structural models), and 2) between different development phases (e.g., from domain engineering to application engineering) [14, pg. 169]. Consequently, traceability is fundamental to support several tasks of the SPL development methodologies and processes, beyond just being a key factor for requirements engineering.

However, a common problem when traceability is implemented in SPL projects is that it must be started from scratch from project to project, because most of the times there is no documentation and guidelines which explicitly formalizes the best practices and successful experience with traceability in the organization's projects. In other words, traceability knowledge is lost if the traceability practices are not actively captured, documented and applied. Hence, the traceability practices, which have proven to be successful to solve day to day problems, should be kept as part of the traceability knowledge gained. Of course, developers keep and spread this implicit knowledge, however, if we do not make this knowledge explicit, the application of development practices becomes an art without a formal guidance, which leads to wasted time and effort that could be used in other development tasks. Moreover, whenever team members leave the project, they take with them all the knowledge acquired. To avoid this, the knowledge and the fundamental items to implement traceability should be explicitly documented and formalized. This formalization must be a tool to specify the successful traceability practices, the work products to be generated, plus the people and tools involved in traceability tasks. These items, actually compose a traceability methodology, according to [12].

One approach to address this problem is to include high-quality traceability metamodels from which to generate customized traceability methodologies, which every system development process should have. Intrinsically methodology metamodels, provide customization by means of the instantiation, and they provide reuse, since the development specifications and practices can be modeled in the metamodel. Similarly, metamodels facilitate extension to include more methodology items if they are required, just by adding new modeled items.



**Figure 1: Usage process of the SPL traceability metamodel. SPL TmM Core are composed by powertype patterns at Metamodel level: 1) for SPL development (marker E) are *SPLwp* (work products), *SPLwu* (tasks) and *SPLp* (producers), and 2) for traceability (marker F) are: *Twp* (work products), *Twu* (tasks) and *Tp* (producers).**

This paper presents an approach that supports a formal definition of methodology practices and a later reuse. The approach includes several modeling principles as the basis to define a traceability metamodel for SPL development, called SPL Traceability metaModel (SPL TmM). The formalization and reusing of traceability knowledge using these modeling principles will be showed, while SPL TmM is defined. SPL TmM includes work products, producers and tasks corresponding to the traceability practices in SPL Engineering. In order to describe the formalization of traceability knowledge with this metamodel, we use a particular practice to support the feature configuration process: feature model primitives technique implemented in the S2T2 Configurator, a tool for visual interactive feature configuration [4, 5]. This technique facilitates to manage and to keep the features trees, involved in the feature configuration process, consistent.

The traceability metamodel approach provides a clear separation between the expertise areas involved in a method-

ology: *metamodel*, *methodology* and *project*. Then, the SPL traceability metamodel diagrams includes classes to support: 1) the methodology items definition, and 2) the enactment of such items during the project development. An example is included, to illustrate the definition of the project-specific traceability methodology items. These traceability items have already incorporated the traceability knowledge acquired in previous projects.

Some clear benefits are obtained with this approach: 1) if the project's traceability methodology can be generated with these elements, changes to modify the traceability strategy will be easily introduced, just by instantiating new items from the traceability metamodel; 2) knowledge acquisition for new project's members becomes systematic, since the metamodel divides the classes to be use during the methodology definition and during the project development; and 3) those practices proven as successful will be completely reused in new projects, since the metamodel encourages a very systematical methodology instantiation,

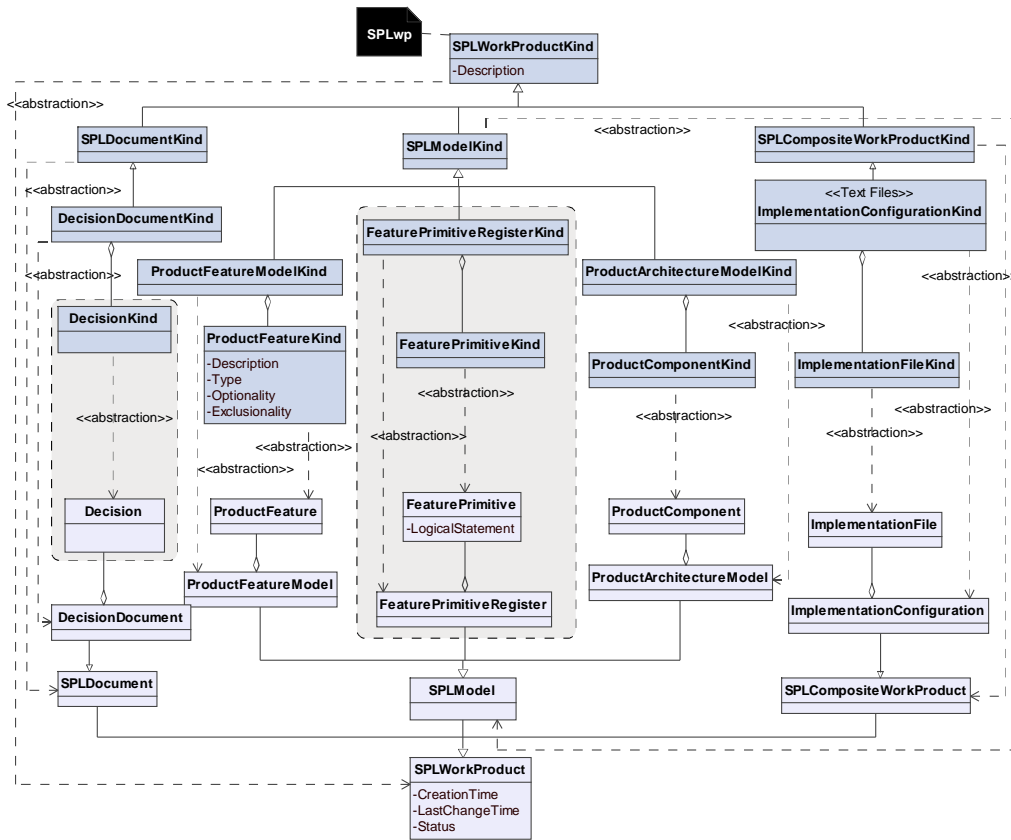


Figure 2: SPL TmM core for product derivation: work products (SPLwp), tasks (SPLwu) and producers (SPLp). See marker E in Figure 1

avoiding knowledge lost.

The rest of the paper is structure as follows: Section 2 introduces the modeling principles to create the traceability metamodel proposal for SPL projects. Section 3 presents the traceability metamodel for SPL Engineering. Section 4 describes, how to capture the traceability knowledge using our approach. Section 5 explains how the modeled elements interact with each other to perform traceability tasks, during product derivation. Section 6 illustrates the metamodel usage to define the traceability methodology's items. Section 7 presents the conclusions and the future work.

## 2. MODELING ELEMENTS

We are interested in assuring that SPL TmM provides a real guidance to both, 1) define the items required to implement traceability in a SPL project, and 2) to describe the interaction between these items during SPL development process. The first step is to select the modeling principles needed to produce such a metamodel that accomplishes the desired objectives. Our approach uses two modeling principles: a) a three-layer modeling hierarchy to represent the abstraction layers involved during a methodology definition process. Metamodel, methodology definition and methodology enactment during the project development, are the abstraction levels used in the three-layer modeling hierarchy used in our approach, see Figure 1. And b) across

these three layers, we use the *powertype patterns* modeling principle, which allows to model concepts through the three-layer modeling hierarchy, respecting strict metamodelling rules [3, 13]. Both modeling principles a) and b) are detailed in the next paragraphs.

The three-layer abstraction levels involved in a methodology definition, in our approach are defined as: metamodel, methodology and project, see Figure 1. The layers are used as follows: 1) The Meta-model layer provides the expressive means for describing the common tasks, work products, producers and practices of SPL methodologies with traceability (as suggested by this paper). 2) In the Methodology layer the method engineers use our meta-model to define their methodology approach (by instantiating concepts from the meta-model layer). 3) In the Project layer, developers in a particular software project use the methodology defined on the level above (again by instantiation of the methodology concepts).

A powertype pattern is an entity of a metamodel composed by two classes representing the same concept [12]. Thanks to the powertype patterns instantiation mechanism, one class of the pattern is instantiated in the methodology layer, producing another class. The other class of the pattern is instantiated in the project layer, producing an object. This technique prevents the class instantiation to only *one* layer. Thus, as SPL TmM is composed by powertype

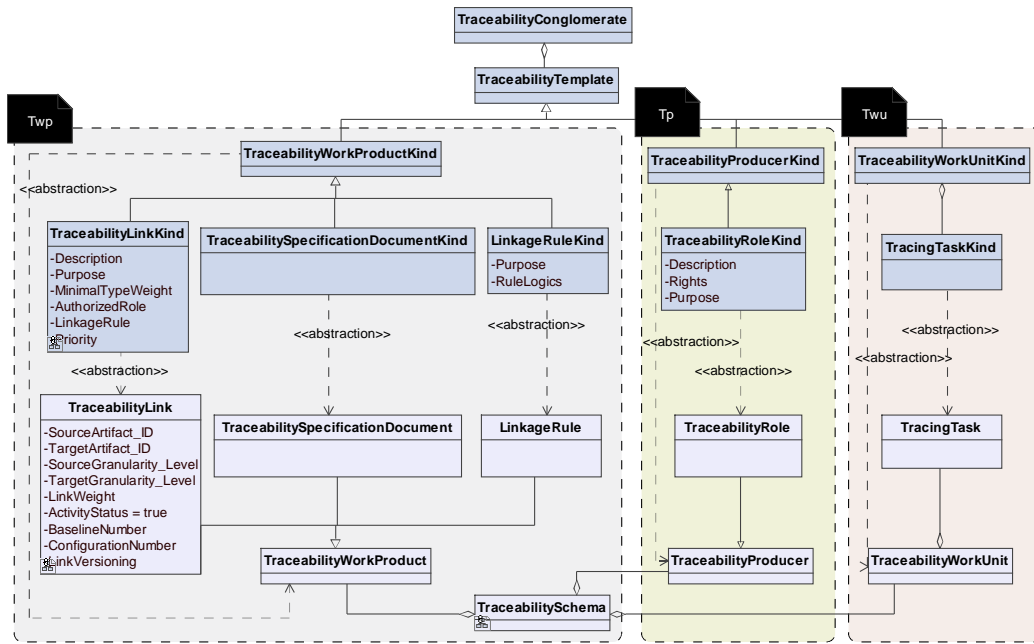


Figure 3: SPL TmM core for traceability: work products (Twp), tasks (Twu) and producers (Tp). See marker F in Figure 1.

patterns, then the first SPL TmM instantiation produces a customized model for traceability in SPL development. Later an instantiation of such model produces project objects, such as traceability links or measurements collected with traceability metrics. Other traceability metamodels approaches such as in [1, 2, 16], propose that the first metamodel instantiation directly produces objects. Thus, from the metamodeling perspective, approaches fail to achieve the main objective of every metamodel: to produce customized models, which later can be used in the target projects, by instantiating their classes into project objects. Consequently, those metamodels do not accomplish the strict metamodeling rules, as stated in [3, 13]. The three-layer hierarchy combined with the powertype patterns principle avoids this and allows to produce project-specific traceability models, overcoming the two main objectives stated in the beginning of this section (in the first paragraph and numbered as 1 and 2). Thanks to the separation abstraction levels of the three-layer hierarchy is possible to provide a guide during the process to define the items required to implement traceability in a SPL project, as it was stated in 1). The use of powertype patterns is possible to clearly model the interaction of the traceability items with SPL items, at the three levels of abstraction, as it was required in 2).

Therefore, the approach structured in three-layers is as follows: In the metamodel layer the traceability metamodel is defined by the method engineer. Here, the project items for the SPL development and for traceability are modeled using powertype patterns (marker A, Figure 1). The powertype patterns corresponding to SPL Engineering (marker E, Figure 1) are: *work products (SPLwp)*, *work units (SPLwu)*, and *producers (SPLp)*. Concerning traceability the powertype patterns (marker F, Figure 1) are: *work products*

(*Twp*), *work units (Twu)*, and *producers (Tp)*.

In the methodology layer, several stakeholders participate in the project requirements definition (marker B, Figure 1), which is performed according to the product requirements stated by the customer. Based on the project requirements, the method engineer defines the traceability items required to support the application development (process 1, Figure 1). Additionally, the traceability metamodel indicates how the traceability items interact with the SPL items, during product derivation. Here, the outcome of the process *project-specific traceability methodology definition*, will be the methodology-specific items to implement traceability in the project (marker C, Figure 1).

In the project layer, the developer uses the traceability methodology items previously defined (process 2, Figure 1). Here, several traceability objects are produced (marker D, Figure 1). For instance, links are created, and this data is used to produce the views to present the information to the stakeholders, according to their roles defined in the project-specific traceability methodology. Similarly, some traceability metrics could be executed to get measurements which are used to check link consistency.

### 3. CORE ITEMS OF THE SPL TRACEABILITY METAMODEL

For all diagrams, the attributes for top classes (to support the methodology layer) are different, from the attributes for bottom classes (to support the project layer). For instance, see the *ProductFeatureKind* and *ProductFeature* classes in Figure 2 or the *TraceabilityLinkKind* and *TraceabilityLink* classes in Figure 3. Each pair of classes that are connected with the *abstraction* relationship constitute a *powertype pattern*, according to the explanation given in Section 2. For in-

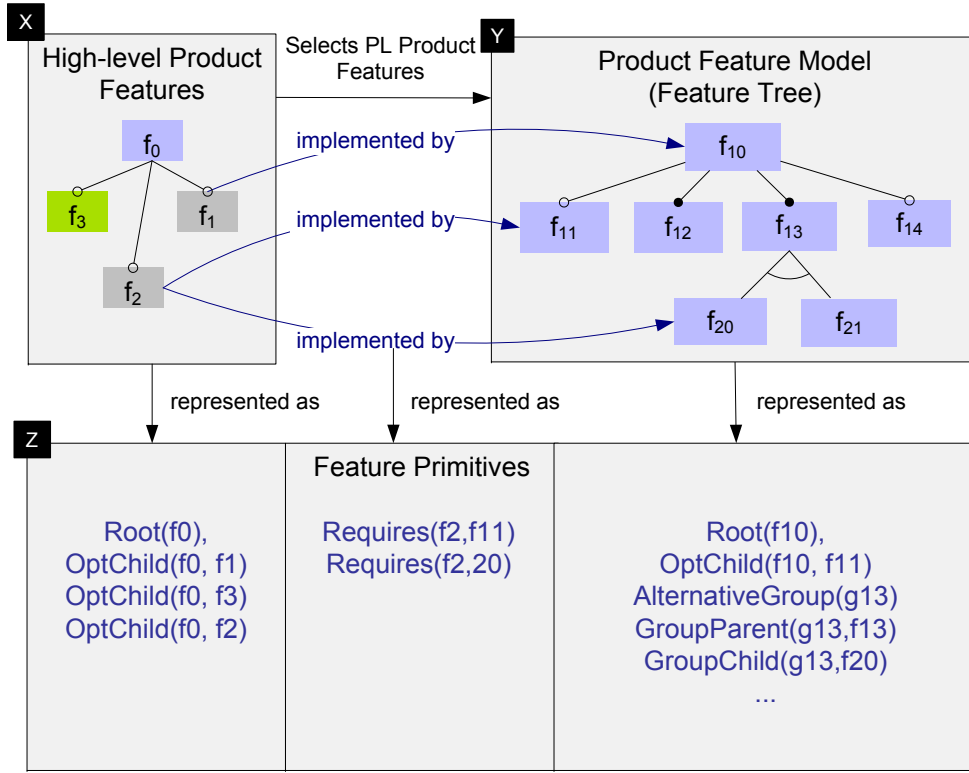


Figure 4: Feature Configuration Process stated in [4].

stance the together *ProductFeatureKind* and *ProductFeature* classes in Figure 2 compose the *ProductFeature* powertype pattern.

The *SPL Traceability metaModel (SPL TmM)* presented in this paper is an extension of a previous traceability meta-model called *TmM*, introduced in [11] and applied in a use case in [10]. *TmM* is a general metamodel for defining traceability methodologies in any context project, independent of life cycle model, or development methodology, or application domain. In this paper, *TmM* has been adapted to include the *SPL Engineering* main concepts, defined as: *work products* (e.g., product requirements specification, feature models and feature configuration models, or implementation components), *work units* (e.g., product feature and product architecture configuration) and *producers* (e.g. product architect or developer). In Figure 1, in the metamodel layer, the main *SPL Engineering* concepts were represented in *SPL TmM* with powertype patterns: *SPLwp* (*work products*), *SPLwu* (*work units*), and *SPLp* (*producers*). Figure 2 shows the specific *SPLwp* powertype patterns (classes linked by the *abstraction* relationship type) corresponding to the work products, specifically for product derivation. For space reasons, the other detailed diagrams for the work units (*SPLwu*) and producers (*SPLp*) modeling are omitted here. Regarding traceability, several items to implement traceability have been considered, such as work products (traceability types, linkage rules or traceability documentation), producers (method engineer who manages the traceability methodology), and traceability tasks (links updating such as creation, deletion or edition). In Figure 1, the trace-

ability main concepts were represented in *SPL TmM* with the powertype patterns: *Twp* (*work products*), *Twu* (*work units*), and *Tp* (*producers*). Figure 3 details these powertype patterns and resources.

#### 4. MODELING A PARTICULAR TRACEABILITY PRACTICE TO MANAGE THE FEATURE TREE

In this section we will describe a particular technique that we use during the product derivation process to manage product features. Figure 4 illustrates the model elements manipulated during the features selection process, when a product is derived from the product line. Typically, in a product derivation process the customer makes decisions to select some high-level features from the product line, which shape the new product (marker X). Later, the product architect will use these high-level product features to select the specific and more technical features from the product line to build the new product (marker Y). Internally these feature trees in the high-level feature model (X) and the technical feature model (Y), are decomposed into *feature primitives* (marker Z), by following our technique. A *feature model primitive* is an elementary constraint in a feature model. Basically, a feature model primitive (1) describes the relation between the referenced features and, hence, (2) defines a constraint for a legal configuration of this feature model. For instance, there are feature model primitives such as *Root(f1)*, *OptChild(f0, f1)*, *Requires(f11)*, *GroupParent(g12, f12)* or *GroupChild(g12, f20)*. Each fea-

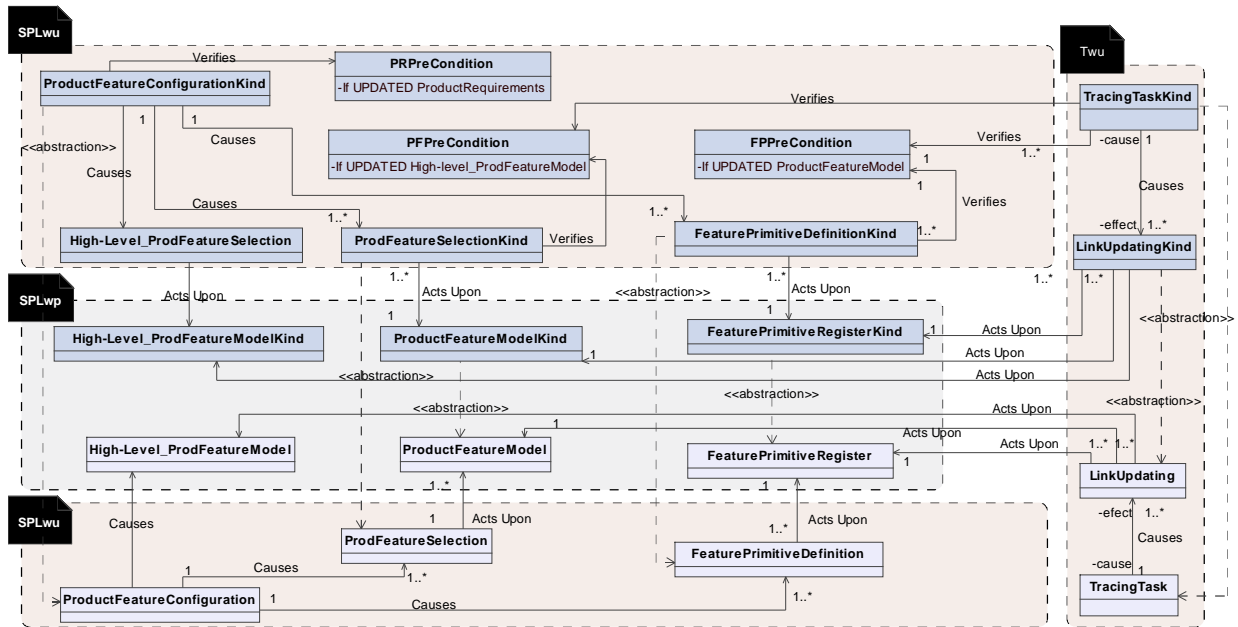


Figure 5: Modeling a traceability task performed in the product feature configuration.

ture model primitive can be directly translated into a corresponding clause in propositional logic. In doing so, the technique allows to define and implement the semantics of the feature tree, e.g., the difference between mandatory and optional features or rules for feature groups. More details of this technique are given in [4], including a description of the corresponding tool “S2T2 Configurator”. In this manner, the data for a feature tree can be easily understood, kept and managed. Note that in this approach traceability is implicit, since each feature primitive intrinsically represents the traceability between the referenced features: the parent and the child, but with a strong semantics. Also, in some sense the feature model primitives types represent traceability types between features.

The feature model primitives technique has been applied successful in our projects, e.g., to configure large and complex models in an consistent manner, while providing explanations for automatically derived consequences [6, 7]. Based on these experiences, our goal for the research presented in this paper was to capture the implicit knowledge on traceability in an explicit form. For that, we need a formal representation to keep this traceability knowledge without ambiguity, assuring its correct reuse in all new projects. With this aim in mind, this traceability technique was represented using the powertype patterns of the *SPL TmM* metamodel. First, we introduced two new concepts into the *SPL TmM* metamodel: 1) the *decision* concept to represent the selected high-level features for a new product (marker X, Figure 4), that is modeled with the *Decision* powertype pattern in Figure 2 (highlighted rectangle on the left); and 2) the *feature primitive* concept (marker Z, Figure 4), that is modeled with the *FeaturePrimitive* powertype pattern in Figure 2 (highlighted rectangle on the right). The product features (marker Y, Figure 4), which are selected according to the high-level product features, are modeled with the

*ProductFeature* powertype pattern in Figure 2.

## 5. MODELING THE INTERACTION BETWEEN THE CORE ITEMS

Once, the features primitives technique is described and the powertype patterns for this task are included in *SPL TmM*, we need to model how the traceability task is performed to keep the related models consistent. Figure 5 models *when* the *SPL* work products are affected by traceability during product feature configuration. Figure 5 includes the powertype patterns for the concepts illustrated in Figure 4. They are shown in the rectangle with the mark *SPLwu*, in both levels, methodology and project. In the diagram, the product feature configuration task is started, if the product requirements has been updated by the customer (verifying the *PRPreCondition*). This will cause that the *High-Level\_ProdFeatureSelection* action (described in Figure 4 marker X) updates the model for the high-level product features selected by the customer. Later, the *ProdFeatureSelection* action (described in Figure 4 marker Y) updates the *product feature model* (also called *implementation-oriented feature model*), according to the new selection of the *high-level product features*. Then, the *FeaturePrimitiveDefinition* action derives feature model primitives (described in Figure 4 marker Z), according to both the high-level feature model and the implementation-oriented feature model. Once the feature model primitives are created, the *LinkUpdating* action kept them in a register, called in the diagram as *FeaturePrimitiveRegister*.

In Figure 5 (right side), in the rectangle with the mark *Twu*, the *LinkUpdating* action, triggered by a traceability task, will update the involved links in a product configuration process. The *LinkUpdating* action is executed whenever the high-level product feature, product feature and feature

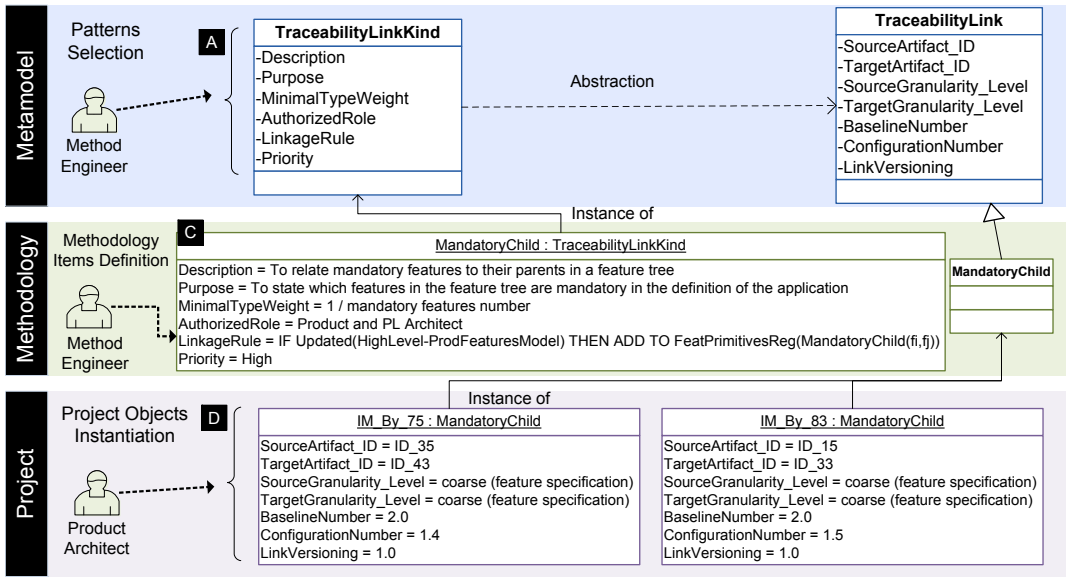


Figure 6: *MandatoryChild* link type definition.

primitive models are updated, by checking the *FPreCondition*, *FPPreCondition* and *FPostCondition*. The *SPL TmM* metamodel prescribes no particular order to execute the tasks, all the tasks are controlled using pre-conditions and post-conditions to execute an action. This permits to reuse this model when is enacting a non-traditional development life-cycle, as it is the case for Agile Methods, in which there is no a explicit Requirements Analysis stage [8].

## 6. SPL TMM USAGE EXAMPLE

In this section we will now discuss a small example to illustrate the viability of the *SPL TmM* metamodel approach as a structured platform and traceability methodology, which can widely support the management of product line models using feature model primitives. First, it is necessary to identify types of model elements that will be used in the *SPL* project. In our case these are feature model primitives. These types of feature model primitives determine the traceability types from which links are created for the project. According to the *SPL TmM* instantiation process described in Figure 1, it is necessary to instantiate the right power-type patterns, to define the concepts for the particular traceability methodology. Since *SPL TmM* has already incorporated the feature model primitives technique, the resulting traceability methodology will provide this knowledge. As discussed earlier, the *TraceabilityLink* power-type pattern is used to define the traceability types to manage the feature tree. Then, the process to define the types is exemplified by using the feature primitive *MandatoryChild(f1, f2)*, which models that feature *f1* has a mandatory child *f2*. Then, the *TraceabilityLink* power-type pattern is used to define the *MandatoryChild* traceability type. Figure 6 shows this type definition and some links created from this type.

In this example, the use of the *SPL TmM* power-type patterns and its interaction models provides guidance to identify which system artifacts will be involved in the creation

of *MandatoryChild* links. See Figure 6 in the methodology layer for this specification. *SPL TmM*'s *TraceabilityLink* pattern determines which artifact types will be related, using the *MandatoryChild* type. For instance, when applying *MandatoryChild* links only features in the current feature model are considered; other elements are ignored. This guidance permits to reduce the number of artifacts to consider when such links are created. Similar principles can be applied for any traceability type defined by using the *TraceabilityLink* pattern. This helps to reduce the effort spent when updating traceability links. This is particularly crucial when the feature model is evolving, due to the introduction of changes to customer selection of the high-level product features and new feature model primitives are generated or the existent ones are modified. Similarly, the *TraceabilityLink* pattern forces to define a linkage rule to automatically create links of a type defined with this pattern. For instance, in the discussed case, the rule *MandatoryChildRule* determines the reasoning which creates the links of the *MandatoryChild* type. Here, the example illustrates the traceability types definition. However, there are other key elements that should be defined in the traceability methodology. For instance consider specific traceability roles (to determine the traceability information to show to each stakeholder) or traceability metrics (to derive aggregated information).

## 7. CONCLUSIONS AND FUTURE WORK

In this paper an approach for formalizing the traceability knowledge gained in a particular feature configuration technique for software product lines has been presented. The aim is to reuse this knowledge in other projects to derive new products, reducing the time to implement traceability, and assuring that helpful traceability practices to manage the feature configuration are completely reused. This approach can also work as a structured guidance for new team

members, when traceability is introduced in other projects. In this paper, the SPL TmM metamodel for traceability has been introduced as means for formalization. The SPL TmM metamodel defines traceability techniques in three areas: metamodel, methodology and project. The SPL TmM metamodel contains core powertype patterns from which the traceability methodology items are derived. The interaction between these core items describe the corresponding traceability practices.

To describe how to formalize a traceability practice we modeled our feature primitive technique, which has been successfully applied to manage relations between elements in related product line models in a practical and consistent manner. A brief example has been introduced to show how to define a traceability methodology, which reuses the feature primitive technique. As an example a traceability type *MandatoryChild*, which we use during the product feature configuration was instantiated. The *MandatoryChild* definition using the *TraceabilityLink* pattern of SPL TmM can be used to define links of this type.

Similarly, it was discussed how the definition of this type following the *TraceabilityLink* pattern of SPL TmM, can reduce the time and effort spent on creating links of this type.

As future work we intend to investigate not just the traceability practices for application engineering, but also the practices for domain engineering, and traceability links between artefacts in these two areas. The introduced of pre and post conditions are intended to be used to automatically monitor changes on the different models involved in SPL Engineering (e.g., feature model, component-oriented models or actual implementation artefacts). The aim is to use this metamodel approach to automate the consistency checking of traceability links in all processes of SPL Engineering.

## 8. ACKNOWLEDGMENTS

This research work has been partially supported by the Mexican National Council of Science and Technology (CONACyT), projects OVAL/PM TIC2006-14840 (MEC, Spain), FLEXI (ITEA2 06022) FIT-340005-2007-37 (MITYC, Spain), and by Science Foundation Ireland grant 03/CE2/I303.1 to Lero – the Irish Software Engineering Research Centre, <http://www.lero.ie/>.

## References

- [1] S. Ahn and K. Chong. A feature-oriented requirements tracing method: A study of cost-benefit analysis. In *Proceedings of the International Conference on Hybrid Information Technology (ICHIT)*, pages 611–616, Washington, DC, USA, 2006. IEEE Computer Society.
- [2] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J.-C. Royer, A. Rummler, and A. Sousa. A model-driven traceability framework for software product lines. *Software and Systems Modeling*, page 25, June 2009.
- [3] C. Atkinson and T. Kühne. Profiles in a strict meta-modeling framework. *Sci. Comput. Program.*, 44(1):5–22, 2002.
- [4] G. Botterweck, M. Janota, and D. Schneeweiss. A design of a configurable feature model configurator. In *VaMoS*, pages 165–168, 2009.
- [5] G. Botterweck, L. O’Brien, and S. Thiel. Model-driven derivation of product architectures. In *ASE ’07 Proceedings*, pages 469–472, New York, NY, USA, 2007. ACM.
- [6] G. Botterweck, A. Polzer, and S. Kowalewski. Interactive configuration of embedded systems product lines. In *International Workshop on Model-driven Approaches in Product Line Engineering (MAPLE 2009)*, collocated with the 12th International Software Product Line Conference (SPLC 2008), San Francisco, CA, USA, August 2009.
- [7] G. Botterweck, D. Schneeweiss, and A. Pleuss. Interactive techniques to support the configuration of complex feature models. In *1st International Workshop on Model-Driven Product Line Engineering (MDPLE 2009)*, held in conjunction with *ECMDA 2009*, Twente, The Netherlands, June 2009.
- [8] L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25, issue: 1:60–67, 2008.
- [9] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. The SEI series in software engineering. Addison-Wesley, Boston, MA, USA, 2002.
- [10] A. Espinoza and J. Garbajosa. A proposal for defining a set of basic items for project-specific traceability methodologies. In *Proceeding of 32nd Annual IEEE Software Engineering Workshop (SEW)*, pages 175–185, Kassandra, Greece, 2008. IEEE Computer Society.
- [11] A. Espinoza and J. Garbajosa. Tackling traceability challenges through modeling principles in methodologies underpinned by metamodels. In *CEE-SET WiP 2008 Proceedings*, pages 41–54, Brno, Czech Republic, 2008. Oficyna Wydawnicza Politechniki Wrocławskiej.
- [12] C. Gonzalez-Perez and B. Henderson-Sellers. A powertype-based metamodeling framework. *Software and System Modeling*, 5(1):72–90, 2006.
- [13] B. Henderson-Sellers and C. Gonzalez-Perez. The rationale of powertype-based metamodeling to underpin software development methodologies. In *Proceedings: Asia-Pacific Conference on Conceptual Modelling (APCCM’05)*, pages 7–16, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.
- [14] P. Lago, H. Muccini, and H. van Vliet. A scoped approach to traceability management. *J. Syst. Softw.*, 82(1):168–182, 2009.
- [15] K. Pohl, G. Boeckle, and F. van der Linden. *Software Product Line Engineering : Foundations, Principles, and Techniques*. Springer, New York, NY, 2005.
- [16] A. Sousa, U. Kulesza, A. Rummler, N. Anquetil, R. Mitschke, A. Moreira, V. Amaral, and J. Araújo. A model-driven traceability framework to software product line development. In J. Oldevik, G. K. Olsen, T. Neple, and R. Paige, editors, *ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings*, pages 97–107, Berlin, Germany, 2008. SINTEF ICT.