Customizable Domain Analysis

Joachim Bayer, Dirk Muthig, Tanya Widen

Fraunhofer Institute for Experimental Software Engineering (IESE),
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
+49 (0) 6301 707 251 {bayer, muthig, widen}@iese.fhg.de

Abstract. Generative and Component-Based Software Engineering are approaches to reuse. Within both approaches the scope of reuse can vary between general purpose and application specific. We argue for scoping the reusable assets based on a product line.
Domain analysis methods provide processes for determining the common and varying requirements for a product line. However, we experienced that existing methods have problems concerning applicability and instantiation support. Therefore, we have developed the Customizable Domain Analysis (CDA) method.
CDA is customizable to the project context where it will be applied. This ensures that the method and workproducts used for modeling are appropriate for the specific needs. CDA also provides full instantiation support for specifying systems in the product line.

Keywords: Domain Analysis, Domain Engineering, Product Line Software Engineering

1 Introduction

Generative and Component-Based Software Engineering are approaches to make software engineering more efficient through reuse.

Generative techniques require developing a high level language for specifying systems that is then used to generate the individual products. The more specific the language, the tighter it fits to the problems being solved (e.g., domain specific languages). Yet, one language can only be used for a particular class of problems. The more general the language, the more applicable it is (e.g., the object-oriented language Java or the functional language Haskel). The trade-off is that the language may contain much that is not needed in a particular situation. Additionally, it does not necessarily contain specifics for the problem (e.g., concepts or terminology). Therefore, general languages are more difficult to use than specialized languages.

Component-Based Software Engineering approaches, on the other hand, advocate building reusable components that can be plugged together to create new systems. The trade-off here is again generality versus specialization. General components can be used in many applications, but may be much more than needed. Specific components, on the other hand, are appropriate for the situation, but then there may be many specific components that you have to search through to find the one that exactly meets your needs, or the components may need to be adapted to the particular situation.

Product line engineering is an approach to reuse that can be used as both the generative approach and the component-based approach to reuse. It is a middle ground between specific assets (one time use) and general assets (applicable anywhere). The idea is to define a product line, which captures the intended scope of reuse. The products included in a product line are determined to have sufficiently common characteristics to make it more efficient to study the commonalities and variabilities once for all products of the product line and to build reusable assets, than to study and build all the products separately. Product line engineering approaches define how to leverage the commonalities and create reusable assets that increase the efficiency of developing the products.

Domain analysis is one part of product line engineering. Domain analysis methods provide processes for eliciting and structuring the requirements of a domain, or product line. The results are captured in a domain model. A domain model must capture both the common characteristics of the products and their variations. The domain model is the basis for creating other reusable assets like a domain specific language or a component-based architecture.

Many domain analysis methods have been proposed. However, we have identified two major problems in trying to use them: applicability and instantiation support.

For a method to be applicable, it must be appropriate and it must provide enough guidance so that it can be carried out. As in other areas of software development, the context for each domain analysis application varies, and methods that are appropriate in one context will not be in others. This fact is especially important for domain analysis because of the compound effects of inappropriate models over multiple products. Therefore, a generally applicable domain analysis method must be customizable to the context of the application.

Unfortunately, most existing domain analysis methods do not provide customization capabilities, and those that do, do not provide adequate support for customizing it and using the results.

Another requirement for domain analysis methods that is often not even addressed is the instantiation of the domain model for application development. Instantiating the domain model provides the requirements for the single system to be developed. Some approaches support making decisions for each variability in the domain model during instantiation. However, additional instantiation support is necessary because variations in the product line are related and selecting one alternative often has consequences on which other alternatives can be chosen during instantiation. These relations should be determined during domain analysis.

1.1 Approach and Context

In this paper, the Customizable Domain Analysis (CDA) method is presented. CDA is the domain analysis component of the PuLSE™ (Product Line Software Engineering)[1] methodology [4]. An overview of PuLSE is given in Section 2.

1. PuLSE is a registered trademark of the Fraunhofer Institute for Experimental Software Engineering

PuLSE-CDA was developed to be adaptable to the project needs, so that the most appropriate models can be used. In addition, the method provides enough guidance to make it can systematically applicable. Finally, the products of PuLSE-CDA support the processes that will use them, namely architecture development and instantiation.

The PuLSE-CDA method has been developed based on a study of existing domain analysis methods, which helped us determine the main variation points that customization will be based on. Additionally our experience trying to apply some of these methods provided insight into which context factors most affect customization of domain analysis.

PuLSE-CDA's support for instantiation is an expansion of the decision model from Synthesis [16]. The decision model concept is expanded with constraints/relationships to model the dependencies among variabilities.

1.2 Outline

The remainder of this paper is structured as follows: Section 2 provides an overview of the PuLSE approach. Section 3 presents the PuLSE-CDA method in detail and explains the domain decision model and customization of the method. Section 4 follows with an analysis of the PuLSE-CDA approach based on our experience so far. Section 6 discusses the related work. Section 6 concludes the paper.

2 PuLSE

PuLSE is a method for enabling the conception and deployment of software product lines within a large variety of enterprise contexts. This is achieved via a product-centric focus throughout its phases, customizability of its components, an incremental introduction capability, a maturity scale for structured evolution, and adaptions to a few main product development situations.

Figure 1 shows an overview of PuLSE.

PuLSE is centered around three main elements: the deployment phases, the technical components, and the support components.

The *deployment phases* are logical stages of the product line life cycle. They describe activities performed to set up, use, and evolve product lines. The deployment phases are:

- PuLSE initialization: PuLSE is customized to the context of its application. The principle dimensions of adaption are the nature of the domain, the project structure, the organizational context, and the reuse aims.
- Product line infrastructure construction: The product line infrastructure is set up. This is done by scoping, modeling, and architecting the product line.
- Product line infrastructure usage: The product line infrastructure is used to create a single product line member. This is done by instantiating the product line model and architecture.

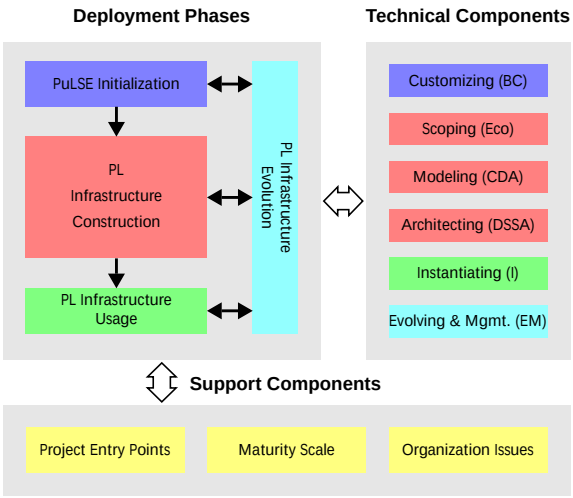


Figure 1. PuLSE Overview

- Product line infrastructure evolution: Concepts within the domain or other requirements on the product line may change over time. The evolution of the product line is handled in this phase.

The *technical components* provide the technical know-how needed to operationalize the product line development. They are used throughout the deployment phases. The technical components are:

- Baselining and Customization (PuLSE-BC): Baseline the enterprise and customize PuLSE. The result is an instance of PuLSE — that is instances of the other technical components — tailored to the specific application context.
- Economic scoping (PuLSE-Eco): Identify, describe, and bound the product line. This is done by determining the products that constitute the product line.
- Customizable Domain Analysis (PuLSE-CDA): Elicit the requirements for a domain and document them in a domain model (a.k.a. product line model).
- Domain Specific Software Architecture development (PuLSE-DSSA): Develop a reference (or domain specific) architecture based on the product line model.
- Instantiation (PuLSE-I): Specify, construct and validate one member of the product line. This encompasses the instantiation of the product line model and the reference architecture, the creation and/or reuse of assets that constitute the instance, and the validation of the resulting product.
- Evolution and Management (PuLSE-EM): Guide and support the application of PuLSE throughout the deployment phases initialization, construction, usage, and evolution.

The *support components* are guidelines that support the other components. They are:

- Project entry points: Project entry points are guidelines to customize PuLSE for a set of standard situations. For example, in reengineering driven PuLSE projects,

legacy assets are a major source of information and guidelines on how to integrate them are given in the respective entry point.

- Maturity scale: This scale provides an integration and evolution path for product line adoption using PuLSE. The levels on the scale are: initial, full, controlled, and optimizing.
- Organization issues: For PuLSE to be most effective, an organization structure has to be set up and maintained that supports the development and management of product lines. Guidelines on how to do that are given here.

3 PuLSE-CDA

The purpose of PuLSE-CDA is to develop a domain — or product line — model that captures the requirements of a product line, and to enable the instantiation of this product line model for single systems that are members of the product line. In this section, different aspects of PuLSE-CDA are described. Section 3.1 starts with a high level process description.

A domain model consists of multiple workproducts that capture different views of a domain. Each view focuses on particular information types and relations among them. In the workproducts, common and distinguishing aspects of the domain are modeled. Workproducts are presented in more detail in section 3.2.

Each workproduct that captures variability has a decision model. Variability and decision models are discussed in section 3.3.

It is not possible to consider all possible future requirements on a product line. Therefore, evolution of the reusable assets and products that are built from them is a key concern. Section 3.4 describes evolutionary aspects for PuLSE-CDA.

The workproducts that are used to capture information in different domains vary. Additionally, processes to capture and document the information in the different workproducts vary. These customization aspects of PuLSE-CDA are discussed in section 3.5.

Due to the amount of information that makes up a domain model and the interrelations among the information, domain analysis requires tool support. In section 3.6, Diversity/CDA, a tool to support PuLSE-CDA is presented.

3.1 Process Description

In this section the PuLSE-CDA process is described. The process is illustrated in Figure Figure 2. It consists of three steps: refine scope definition, elicit raw domain knowledge, and model domain knowledge. These are described below.

The inputs to PuLSE-CDA are the product characteristics information and scope definition from PuLSE-Eco. The scope definition identifies the range of characteristics that systems in the product line cover. The characteristics themselves are listed and described in the product characteristics information. These inputs describe the contents of the product line.

In the first step of PuLSE-CDA, *refine scope definition*, the boundary of the product line is determined. Based on the scope definition that is focused on the contents of the

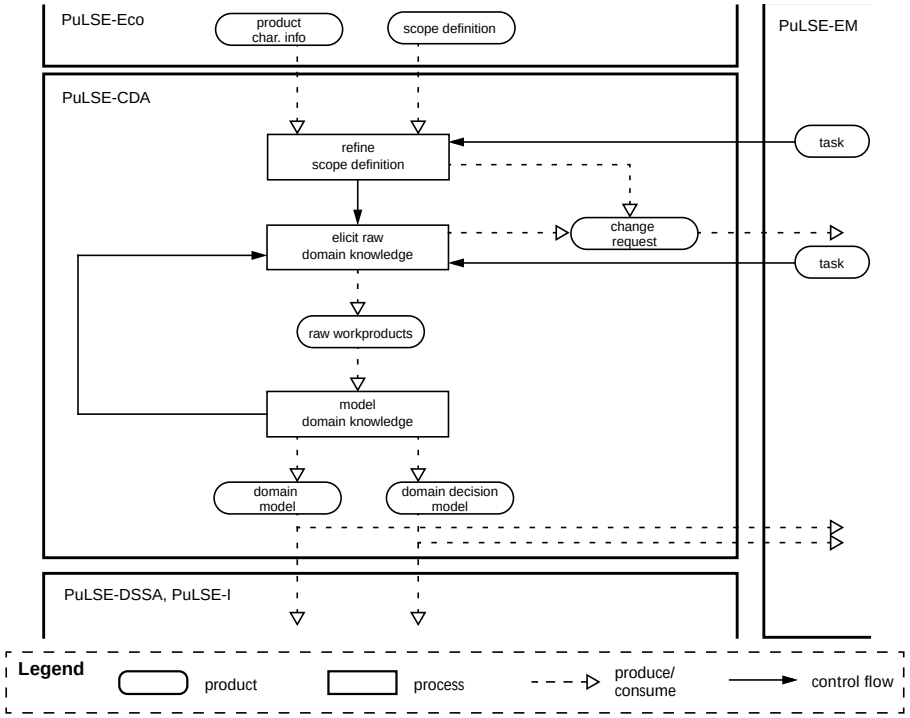


Figure 2. PuLSE-CDA Process Overview

product line, the boundary definition is created that focuses on the interfaces of the product line. Both views are used in the following process steps; the scope definition provides an initial structure for the domain information and the boundary definition limits the area to be analyzed.

In the second process step, *elicit raw domain knowledge*, information is gathered from various sources. Sources may include books and other literature, human sources like domain experts, expected system users, and application engineers, or existing systems.

At this point the information is considered raw, because it is not necessarily well structured. Information may describe single systems separately, or it may be at the wrong level of abstraction and detail. The raw information is captured explicitly in the workproducts.

As information is elicited and captured in the domain model workproducts, links are created to the source of the respective information. This supports evolution and maintenance.

In the final step of PuLSE-CDA, *model domain knowledge*, the elicited knowledge is modeled. This is done using the same workproducts that were used to capture the knowledge in the elicitation step.

The outputs of this step are the domain model — composed of the generic workproducts — which describe the product line as a whole as opposed to single systems,

and the domain decision model, which is used to extract single system descriptions from the generic workproducts.

There are five interrelated activities that are performed to model domain knowledge:

- Abstraction: Higher level models are built to connect low level models that may have been produced during elicitation. Within this activity, the completeness and consistency of the models is also checked.
- Restructuring: Existing abstractions are replaced with better ones (i.e., simpler, clearer, or more appropriate ones) that capture the same information.
- Building generic models: The systems in the product line are integrated through the explicit modeling of variability.
- Consolidation of variability: On top of the different decision models for the different workproducts, a domain decision model is built that combines them.
- Providing traceability: Links are established that explicitly model interrelationships among information in the different workproducts.

Elicitation and modeling are tightly coupled and there is no best way to combine them. Information may be directly elicited and modeled at the same time. On the other hand, during modeling the elicitation step can be revisited when information is not yet complete or considered inconsistent.

The outputs of PuLSE-CDA — the domain model and the domain decision model — are used in PuLSE-DSSA to guide the reference architecture construction and in PuLSE-I to create product specifications for the product line members. Additionally, the product line model can be used to introduce people to the product line.

Whenever, during the scope refinement or elicitation steps, the scope definition is determined to be insufficient, a change request is sent to PuLSE-EM that handles such requests and distributes them to the responsible PuLSE components.

On the other hand, the product line model can be subject to change. The scope of the product line or the product characteristics can change over time. Another reason for changes is inadequately or insufficiently modeled workproducts that have to be improved. In these cases tasks are passed from PuLSE-EM to PuLSE-CDA.

The evolution aspects of PuLSE-CDA are described in more detail in section 3.4.

3.2 Domain Model

The domain model, a set of workproducts, captures the requirements for all systems of a product line. There are two types of requirements: requirements that are common to all systems of the product line (commonalities), and requirements that vary from system to system (variabilities).

Commonalities are not different from requirements for a particular single system except that they hold for all members of a product family. Variabilities are requirements that may differ between members of the product family. There are three types of varying requirements:

- Optional requirements: requirements that do hold for a particular system or do not hold,

- Alternative requirements: a set of requirements of which only one or a subset holds for a particular system, and
- Range requirements: requirements that specify the potential range for a numerical value, which is supported by the domain model, instead of the specific value as required by a single system.

One goal of PuLSE-CDA is to be customizable to different environments (see section 3.5) and to minimize the necessary changes while introducing the product line approach. From a domain model point of view, this means that the workproducts to document requirements for the product line should be as similar as possible to the already established workproducts that have been used for documenting requirements for single systems.

To adopt the established workproducts for the product line approach, it must be decided which workproducts are supposed to capture variabilities and which workproducts capture only information that is valid independent of the particular instance. Additionally, there are workproducts that capture general domain information (e. g., a glossary to define domain-specific terms), but only parts of them are relevant for particular instances (e. g., only the terms that are really used in the instantiated workproducts are integrated into the domain model instance). Therefore, and to document constraints and relationships among information of the domain model, a means to link elements of (different) workproducts must be available.

Because there is no significant difference between commonalities and specific system requirements, the notations for documenting the latter can be adopted to document commonalities. But most of the notations used in practice do not consider variabilities. Hence, notations of generic workproducts (i. e., instantiatable workproducts that capture variant requirements) must be extended in a natural way. This is done by introducing meta elements that indicate points of variation and enable the instantiation of the workproducts.

For each generic workproduct and for each of the three types of variant requirements, meta elements are defined and integrated. They consist of three parts.

- The representation part defines the notation, that is the syntactic integration of variabilities into a workproduct. This includes the definition of representations within the instantiated workproduct dependent on the context of the particular product line instance.
- The transformation logic maps each possible instantiation of a variability to a useful and understandable representation. A simple example is the omission of a textual optional requirement when it is not valid for an instance.
- The decision model interface connects the meta element with a simple decision in the domain decision model. It controls the transformation logic with respect to the current resolution of the connected simple decision.

3.3 Domain Decision Model

The variabilities are connected to decisions that, when completely resolved, specify (via instantiating the domain model) a particular system, a member of the product line. The

decisions are at different levels of abstraction and are hierarchically structured based on constraints among them. This decision hierarchy is the domain decision model. There are two types of constraints: exclusion and (partial) resolution.

Exclusion means that the resolution of a decision becomes irrelevant for the specification of a particular system because another decision has been resolved. For example, decisions concerning variabilities within the specification of an optional requirement are excluded when the option is not chosen.

(Partial) resolution means that the number of possible resolutions for a particular decision is reduced because of the resolution of another decision. When only one possible resolution is left, the constrained decision is also resolved.

There are different types of decisions. For each type of variability described in section 3.2, there exists an analogous decision type. A decision can constrain other decisions independent of its own and the other types. Its possible resolutions are mapped by constraints concerning the possible resolution sets of the constrained decisions.

At the lowest level of the decision hierarchy, there are simple decisions. Each simple decision is directly connected to a meta element, which is part of a domain model workproduct.

A single workproduct is instantiated when all simple decisions that correspond to one of the workproduct's meta elements are resolved. The decision hierarchy reduces the number of decisions that must be made and supports the intellectual control of constraints and dependencies.

The subhierarchy of decisions that is built only upon simple decisions of one workproduct (i. e., no decision concerning a different workproduct is constrained) is called the workproduct decision model. Decisions at the next higher level of abstraction with respect to the top level of a workproduct decision model consolidate variabilities modeled in different workproducts.

The domain model must be completely instantiated to specify a particular system. Therefore, all simple decisions that are not excluded must be resolved. The instance of the domain model is generated by passing resolution by resolution of simple decisions to the connected meta element, which instantiates its corresponding part of the domain model.

3.4 Evolution

PuLSE-CDA expects that the domain model will never be stable. This is founded on a basic idea of PuLSE to scope the domain dependent on products planned to be developed. Additional systems in the future will probably require a scope expansion. Therefore, PuLSE-CDA is designed for handling changes of the scope, as well as of the model. In general, there are three cases in which an evolution of the domain model is necessary:

- Workproducts of the domain model are modeled inadequately or insufficiently, so that it is not possible to construct an acceptable reference architecture. Therefore, PuLSE-DSSA, the technical component for constructing the reference architecture, sends a change request to the product line management. The management, PuLSE-EM, forwards the request as well-defined task to PuLSE-CDA.

- During the lifetime of a product line, the domain is continuously observed to recognize changes and new concepts. This is necessary to also cover anticipated requirements of potential future systems by the product line. The product line management decides whether observed changes and/or new concepts are integrated into the product line. In the case of integration, the scope definition is changed first. Then, the domain model and the domain decision model must be adapted accordingly.
- During the development of systems in the domain (i. e., members of the product line), requirements often arise that are not in the scope of the product line. Hence, the scope of the product line must be either expanded or system-specific assets must be developed.

In all cases, an effective and efficient identification of the domain model parts that are affected by the planned changes is required. For that purpose, traceability within the domain model is needed. First, information units are chosen that are affected by a change. This should be easily possible because the workproducts are customized for capturing concepts of the domain. Following the links starting at the chosen information units, related and hence further affected parts of the domain model can be identified. All affected parts of the domain model are modified according to the change.

One benefit of the product line approach is the idea that all members of the product line are based on a single set of assets. Thereby, the maintenance effort is reduced to this single asset base. Hence, existing products must always follow the evolution of the asset base. This means, existing products must be 'redeveloped' after changing the product line models when necessary. This is done by instantiating the changed domain model while reusing the existing resolution of the domain decision model.

The decisions abstract, as far as possible, from the concrete assets, which are the workproducts of the domain model. This means, the resolution of the domain decision model must be adapted only with respect to simple decisions that are connected to changed workproduct parts, higher level decisions related to changed decisions, as well as decisions that constrain changed decisions. Hence, necessary adaptations can be easily localized and performed. The domain model, the domain decision model, and all specifications of existing products have been adapted to the required evolution.

3.5 Customization

Customization of PuLSE-CDA to the context where it will be applied ensures that the process and products are appropriate. PuLSE-BC defines the customization process used for tailoring the other technical components. Through tailoring the technical components, a customized version of the construction, usage, and evolution phases of PuLSE is created.

The customization process is built upon variation points identified for each technical component. These points are called customization decisions.

Each customization decision has a number of factors that impact its resolution. Factors contain information about the context of application that influence the resolution of a decision. There are two types of factors: customization factors and intermediate decisions.

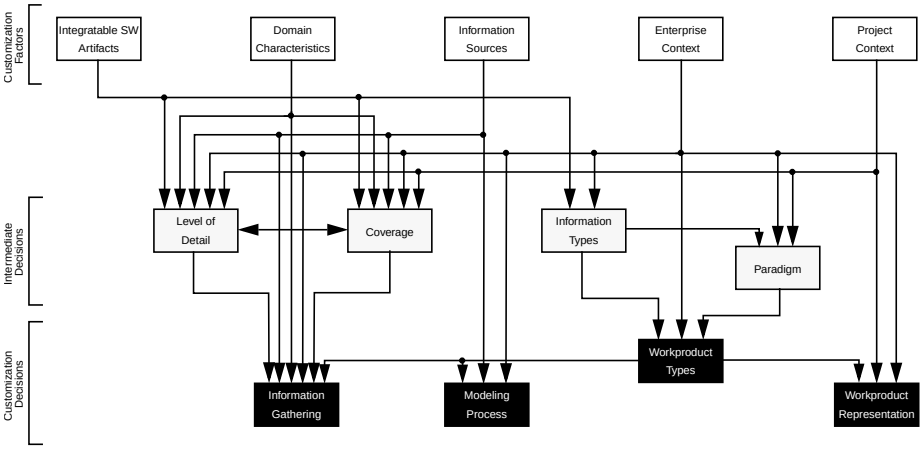


Figure 3. PuLSE-CDA Decision Graph

Customization factors define information about the context that can be directly collected from the enterprise. Intermediate decisions, on the other hand, define information about the context that cannot be collected directly. Rather, they are either (partially) based on subjective judgements or they are influenced by other factors, or both.

Each customization factor, intermediate decision, and customization decision has guidelines associated with it that provides strategies for collecting the information or resolving the decisions.

The goal of a PuLSE-BC application is the derivation of appropriate values for the different decisions and the integration of the adapted components into a coherent process for product line engineering. The assignment of values to the decisions is done using decision graphs and the strategies mentioned above. Figure 3 shows the (simplified) decision graph for PuLSE-CDA. Decision graphs relate the customization decisions that have to be resolved in order to customize a technical component to the customization factors that influence those decisions.

To simplify information gathering, the customization factors are categorized into:

- Domain characteristics: Factors that relate to the domain itself and are independent of implementation aspects.
- For PuLSE-CDA these are: complexity, size, maturity, and stability of the domain, as well as the business area abstractions and the predominant application types.
- Information sources: Factors that relate to the availability of information about the domain.
- For PuLSE-CDA these are: Available books, papers, and standards, as well as accessible users, domain experts, and application engineers.
- Implementation characteristics: Factors that influence implementations in the domain.
- These factors do not have an impact on PuLSE-CDA.

- Integratable software artifacts: Factors that relate to artifacts that can be integrated in a new product line.
- For PuLSE-CDA these are existing systems.
- Project context: Factors relating to the specific project that is planned.
- For PuLSE-CDA these are: resources (in terms of money, time, and personnel), project skills, and the reusable assets that will be produced.
- Enterprise context: Factors relating to the enterprise applying PuLSE.
- For PuLSE-CDA these are: business objectives, enterprise skills, and available tool support.

PuLSE-CDA is customized through the detailed processes for gathering and modeling domain information and through the workproducts that make up a domain model for a project.

The customization of the process is concerned with who will do the activities and how, for example, through group discussions, interviews, literature searches, etc. This is split into two customization decisions: the process for the information gathering, and the process for modeling.

The customization of the products is split into two decisions: the workproduct types and the workproduct representations. The workproduct type captures which information and relations must be modeled. The representation decision defines the notations used to express the information and relations of the workproduct.

3.6 Tool Support: DIVERSITY/CDA

Domain modeling is a complex task for several reasons. During system analysis and modeling for single systems, a large set of complex and interrelated information from various sources has to be managed. In addition to this, domain analysts and modelers deal with multiple systems at once, which requires modeling both commonalities and variations among the systems. Tool support is essential to manage these complexities.

Several types of tool support exist that could be considered promising to automate domain analysis. General purpose analysis and design tools, and requirements engineering tools capture models at the same level of abstraction as a domain model. Another approach is to use specialized analysis tools that are available for different domains. As a third possibility, there are some domain analysis tools available, such as DARE [9], DOMAIN [11], or DOME [8].

Investigating these different types, we determined that there is not one tool that covers all of the required aspects: adaptation to different project situations, automated instantiation, support for intellectual control, and incorporation of existing information. Therefore, we created DIVERSITY/CDA (Domain and Variant Engineering Supporting Technology for the Customizable Domain Analysis) [5].

DIVERSITY/CDA provides support for — beside modeling in general — the four key concepts: customization to the target environment, instantiation of the models to generate specifications of particular applications, traceability among data in the domain model to support intellectual control, and integration of external information.

DIVERSITY/CDA supports *customization* (see Section 3.5) only through the workproducts provided in a DIVERSITY/CDA version for a project. Customization of the process is actually not needed because process guidance is not integrated yet. For each project, the workproducts are determined through applying PuLSE-BC, and then created, adapted, or reused from the workproduct library for integration in the particular tool version. To facilitate customization, DIVERSITY/CDA has been developed as a product line itself. Its general capabilities — for example, database access, searching, and linking — have been separated from the workproducts. The interface between this general part and the workproducts is defined in such a way that different workproducts and combinations of workproducts can be plugged in to create efficiently customized versions of DIVERSITY/CDA for different projects.

To construct applications based on the domain model, DIVERSITY/CDA supports the construction of the domain decision model, as well as the *instantiation* of the domain model. The modeling views, which are part of DIVERSITY/CDA, integrate the workproduct-specific meta elements (see also section 3.2). The meta elements directly correspond to the set of simple decisions, the initial domain decision model (see also section 3.3). In the construction view for the domain decision model, a set of decisions is selected and a new decision is created. The new decision and its possible resolutions must be described. For each resolution, the constraints with respect to the resolutions of all selected decisions must be defined. Using DIVERSITY/CDA, the support for the instantiation of the domain model is separated from the construction support. The clear separation emphasizes the different product line phases, construction (especially PuLSE-CDA) and usage of the product line (i. e., PuLSE-I). The whole instantiation process is guided and interactive. It is guided by generating a useful sequence of decisions whose resolution completely instantiates the whole domain model. It is interactive by updating the representation of the instantiated domain model immediately after each resolution. The user can thus see the impacts of his/her resolution on the final instantiation.

The information captured in the different workproducts is interrelated. To explicitly capture these interrelations and exploit them, DIVERSITY/CDA has a linking mechanism. By providing a means to relate information in a domain model, links support *traceability* within the domain model. A link in DIVERSITY/CDA is a directed, typed reference from one workproduct element to another one. For each workproduct, it is defined which workproduct element (or part) can be linked. The reverse direction of a link can be navigated, too. For some links, the reverse direction expresses the same relation as the initial link, for others it is a reverse relation. Based on the workproducts present, DIVERSITY/CDA provides an initial, extensible set of link types, for example, ‘Uses’/’Used by’ (i. e., the use of a concept is described somewhere else) or ‘Defines’/’Defined in’ (i. e., a term is defined in a glossary entry). Through links, a network of relations is created connecting the information in a domain model. This network can be browsed using link inspectors.

To incorporate information that is not modeled with DIVERSITY/CDA in a domain model, we introduced *external workproducts*. There are basically three purposes of external workproducts: reuse of information from an already established application engineering process, linking information that is related but not part of the domain model

(e. g., PuLSE-Eco spreadsheets), and usage of external modeling tools for workproducts that are not supported by DIVERSITY/CDA (yet). An external workproduct is integrated by creating a representation of its internal structure within DIVERSITY/CDA using tags. These tags are linked to related parts of the domain model using the link mechanism described above and refer to positions in the external document. Hence, connections between domain model elements and positions in external documents can be established.

4 Analysis

The motivation for developing PuLSE-CDA was due to two problems we experienced with existing domain analysis methods: applicability and instantiation support.

The problem of applicability is supported through systematic customization of PuLSE-CDA to the context where it will be applied. Through PuLSE-BC, the framework for customization is supplied. The specifics for PuLSE-CDA are captured in the decision graph and strategies that support resolving the decisions. These specifics were developed based on our experience applying domain analysis and from literature. These will evolve over time with more experience.

The goal of PuLSE-CDA is not to necessarily create new concepts and models for domain analysis, but also to reuse existing models through defining what information models really capture and which models are needed depending on the context.

We have had success working with PuLSE-CDA on different projects. From these projects, we have developed workproducts for business process workflows, business rules, rationales, commonalities and variabilities, issues, glossaries, and to do lists. For a current project we are developing workproducts for variability extended UML diagrams.

All of these workproducts can potentially be reused in other projects if they are determined to be appropriate.

Instantiation support is handled with the domain decision model. Capturing the dependencies among variabilities and creating higher level decisions guides the instantiation and reduces the amount of decisions to be resolved. In addition, it ensures that single systems specifications are valid with respect to the constraints. Single system specifications are generated from a resolved domain decision model.

This generation can be expanded to support generation of an instantiated architecture as well. PuLSE-DSSA, the architecture development component of PuLSE, also provides a decision model for instantiating the architecture. This expanded instantiation can be achieved through mapping the domain decision model to the architecture decision model.

In addition to the customization and instantiation aspects of PuLSE-CDA, evolutionary aspects and tool support have been described above. These aspects are important for the practical application of PuLSE-CDA.

The requirements for the product line will evolve over time, therefore it is imperative to plan for this. Appropriate models and capturing relations explicitly increase understandability and traceability throughout the domain model. In addition, external

links from sources of information and to other reusable assets support the evolution of the product line.

Tool support is essential for keeping intellectual control of the large amount and complexity of information involved. Therefore, Diversity/CDA has been developed. It makes applying PuLSE-CDA practical.

5 Related Work

In this section work related to PuLSE in general, PuLSE-CDA, customization (PuLSE-BC), and the PuLSE-CDA domain decision model is presented.

Domain engineering methods cover most of the same aspects as PuLSE. However, their focus is different, they lack customizability and they are complex to deploy. Domain engineering methods include Model-Based Software Engineering (MBSE) [14], Organizational Domain Modeling (ODM) [17], Synthesis [16], the Domain-Specific Software Architecture (DSSA) program [18], and the Evolutionary Domain Life-Cycle (EDLC) [8].

Domain analysis, requirements engineering, and knowledge engineering are areas that are related to the PuLSE-CDA component. There are many domain analysis methods, including those provided by each of the domain engineering methods mentioned above. Most stem from the work of Arango and Prieto-Diaz [1]. However, the models are either too specific to be applied in the different contexts of domain engineering projects, or they are meant to be tailorable, but lack support for this. CDA aims to improve upon these methods by providing a flexible domain analysis approach with adequate support. Requirements engineering has also provided input into the modeling workproducts, such as use-cases or state transition diagrams [13]. However, these models only support modeling of single systems; no variability modeling is provided.

The goal of PuLSE-CDA is not to develop new modeling technology per se, but to determine which models work best in which contexts and also to expand models developed for capturing single systems with variability capabilities for modeling families of systems.

The PuLSE-BC process for baselining and customization is grounded in work done on the CMM [12], the Reuse Adoption Guidebook [15], and Experience Factory packages [3]. This work is used in the BC component to support the baselining of an enterprise and customizing, or packaging, an appropriate process for the situation.

Decision models for supporting the instantiation of domain models do appear in Synthesis and in the Commonality Analysis [16,2]. The PuLSE-CDA domain decision model is based on this work, but extended with higher level decisions, relations among decisions, and variability from other workproducts (both Synthesis and the Commonality Analysis focus on creating models for a specific workproduct type known as Commonalities and Variabilities).

6 Conclusion

In this paper we presented PuLSE-CDA, a method to determine and capture the common and varying requirements for a product line of systems. PuLSE-CDA is an im-

provement over existing domain analysis methods because it is customizable to the context in which it will be applied and because it provides full instantiation support.

PuLSE-CDA can be used for both generative and component-based software engineering approaches, that is, the models created in PuLSE-CDA can support architecture and component development as well as language design.

Acknowledgements

The authors would like to thank all the other contributors to PuLSE: Oliver Flege, Cristina Gacek, Peter Knauber, Roland Laqua, and Klaus Schmid.

References

[1] Arango, G. and Prieto-Diaz, R. (eds.) Domain Analysis Concepts and Research Directions. In *Domain Analysis and Software Systems Modeling*, pp. 9-31, IEEE Computer Society Press, 1991.

[2] Ardis, M. and Weiss, D. Defining Families: The Commonality Analysis. *Proceedings of the Nineteenth International Conference on Software Engineering*, pp. 649-650, IEEE Computer Society Press, May 1997.

[3] Basili, V., Caldiera, G., and Rombach, D. Experience Factory. *Encyclopedia of Software Engineering Volume 1:*469-476, Marciniak, J. ed. John Wiley & Sons, 1994.

[4] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.M. PuLSE: A methodology to develop software product lines. In *Symposium on Software Reusability*, May 1999.

[5] Bayer, J., Muthig, D., and Widen, T. Support for Domain and Variant Engineering: Diversity/CDA. *Submitted to Automated Software Engineering*, May 1999.

[6] DeBaud, J.M., Flege, O., and Knauber, P. PuLSE-DSSA: A Method for the Development of Software Reference Architectures. *In Proceedings of the 3rd International Software Architecture Workshop*, November 1998.

[7] DeBaud, J.M. and Schmid, K. A Systematic Approach to Derive the scope of Software Product Lines. I*n Proceedings of the 20th International Conference on Software Engineering*, pp. 34-43, IEEE Computer Society Press, 1999

[8] Domain modeling environment. The official web site. http://www.htc.honeywell.com/dome/. April 1999.

[9] Frakes, B., Prieto-Diaz, R., and Fox, C. DARE: Domain Analysis and Reuse Environment. In *7th Annual Workshop on Software Reuse*, 1995.

[10] Gomaa, H., Kerschberg, L., Sugumaran, V., Bosch, C., Tavakoli, I. and O'Hara. L. A knowledge-based software engineering environment for reusable software requirements and architectures. *Automated Software Engineering, 3(3,4)*, pp. 285–307, August 1996.

[11] Higgins, J., Tracz, W., and Newton, E. Domain (DOmain Model All Integrated) User Guide. Technical Report ADAGE-LOR-94-06A, Loral Federal Systems, September 1994.

[12] Paulk, M., Curtis, B., Chrissis, M., and Weber, C. Capability Maturity Model for Software (Version 1.1). *Technical Report CMU/SEI-93-TR-024*, February 1993.

[13] Potts, C., Takahashi, K., and Anton, A. Inquiry-Based Requirements Analysis.

IEEE Software, pp. 21-32, March 1994

[14] Software Engineering Institute, Model-Based Software Engineering. http://www.sei.cmu.edu/technology/mbse/is.html, April 25, 1998.

[15] Software Productivity Consortium Services Corporation. Reuse Adoption Guidebook, Version 02.00.05, November 1993.

[16] Software Productivity Consortium. Reuse-Driven Software Processes Guidebook, Version 02.00.03. *Technical Report SPC-92019-CMC*, Software Productivity Consortium, November 1993.

[17] Software Technology for Adaptable Reliable Systems. Organization Domain Modeling (ODM) Guidebook, Version 2.0. *Unisys STARS Technical Report STARS-VC-A025/001/00*, Reston VA, June 1996.

[18] Tracz, W. and Coglianese, L. Domain-Specific Software Architecture Engineering Process Guidelines, *Technical Report ADAGE-IBM-92-02*, Loral Federal Systems, 1992.