Experiences with Model-based Product Line Engineering for Developing a Family of Integrated Control Systems: an Industrial Case Study

Tao Yue[1], Lionel Briand[1,2], Bran Selic[1,3]
[1]Certus Software V&V Center
Simula Research Laboratory, Oslo, Norway
[2]University of Luxembourg, Luxembourg
[3]Malina Software Corp., Ottawa, Canada
{tao, bselic}@simula.no, lionel.briand@uni.lu

Qitao Gan[4]
[4]FMC Technologies
Kongsberg, Norway
Qitao.Gan@fmcti.com

***Abstract*—Integrated Control Systems (ICSs) are often large-scale and highly configurable software-intensive systems-of-systems, with software and hardware components integrated to control and monitor physical devices and processes (e.g., oil and gas production platforms). Employing Product Line Engineering (PLE) is believed to bring potential benefits with respect to reduced cost, higher productivity, higher quality, and faster time-to-market. However, relatively few industrial field studies are reported regarding the application of PLE to develop large-scale systems, and more specifically ICSs. In this paper, we report about our experiences and insights gained from investigating the application of model-based PLE at FMC Technologies, a company developing subsea production systems (typical ICSs) to manage the exploitation of oil and gas production fields. We first discuss the benefits and challenges of applying systematic domain analysis-the first major step of PLE, report on the key domain analysis results and then assess the benefits and challenges of employing model-based PLE at FMC, as a means to improve the quality of their products and the productivity of their product development process.**

Keywords-Product Line Engineering; Integrated Control System (ICS); Domain Analysis; Model-based Product Line Engineering.

1 INTRODUCTION

Integrated Control Systems (ICSs) are heterogeneous systems-of-systems, where software and hardware components are integrated to control and monitor physical devices and processes, such as process plants or oil and gas production platforms. FMC's Subsea Production Systems (SPSs)[1] are large-scale, highly-hierarchical, and highly-configurable ICSs for managing the exploitation of oil and gas production fields. FMC Technologies, Inc[2] is a leading global provider of technology solutions for the energy industry. One of its key technologies is subsea production systems, used to develop new energy reserves and for managing and improving producing fields. They are composed of hundreds of mechanical, hydraulic, and electrical components and configured software to support various field layouts ranging from single satellite wells to large multiple-well sites (more than 50 wells). The main components of the system are subsea control modules, which contain software, electronics, instrumentation, and hydraulics for safe and efficient operation of subsea tree valves, chokes, and downhole valves.

Our industrial collaboration with FMC started with a discussion about their integration issues, which were described as issues arising when integrating configured software and hardware components, especially hardware components provided by third-party suppliers. This discussion initiated the analysis we conducted and which is reported in this paper. The initial results of this analysis revealed that the majority of the issues is due to improper configuration of deployed software that is used to monitor and control physical devices. It also became clear that the FMC product development lifecycle is an instance of Product Line Engineering (PLE) [16, 17]. PLE aims at supporting the reuse of product assets while differentiating individual products in a product line family. It has been suggested that PLE can provide important benefits such as reduced cost, higher productivity, higher quality, and faster time-to-market [16]. To achieve these benefits, one of the most important features of PLE is to provide an architecture that accurately captures the commonalities and variabilities of all the products in the product line family. Therefore, we narrowed down our analysis scope specifically to collect information required to build such an architecture and understand the current configuration process used at FMC. Our ultimate goal was to propose a suitable PLE solution for FMC to support product configuration. In addition, based on the domain analysis results, our goal was to assess

[1] FMC SPSs: www.fmctechnologies.com/subsea
[2] FMC Technologies Inc. http://www.fmctechnologies.com/

whether Model-Based PLE (MBPLE) is suitable for developing ICSs and identify the benefits and challenges of adopting existing model-driven engineering technologies for that purpose.

Domain analysis in PLE is considered to be a key process to identify commonalities and variabilities of a family of systems [12]. However, considering that PLE requires a thorough understanding of the product development process, including product configuration, understanding this process should be also addressed by domain analysis. In this paper, we first report, in Section 2, how we conducted the domain analysis, including the process we followed and the timeline of each activity. We also report lessons learned based on our experience at FMC.

In Section 3, we discuss the key results of the domain analysis, including identified characteristics of FMC SPSs, which are largely similar to other ICSs in many industry sectors, and their current product development practices. Configuration requirements are derived from the analysis results. These requirements are subsequently used to derive objectives that a PLE solution should address.

In Section 4, we discuss the benefits as well as the practical and research challenges of applying a MBPLE approach to address the objectives identified through domain analysis. We conclude that model-based technologies [9] is appropriate in our context. We conclude the paper in Section 5.

2 DOMAIN ANALYSIS PROCESS

Domain analysis is defined as "*the process by which information used in developing software systems within the domain is identified, captured, and organized with the purpose of making it reusable (to create assets) when building new products*" [4]. More specifically, in our particular context, we consider domain analysis to be a set of activities which are conducted to understand FMC SPSs with the following four objectives in mind: 1) confirm whether FMC SPSs can be considered to be members of a product-line family, 2) identifying key characteristics of FMC SPSs, 3) identifying, gathering, organizing, and specifying the commonalities and variabilities of a family of FMC SPSs, and 4) understanding the current product development processes of FMC, including the identification of the main activities of the process, stakeholders involved in each activities, and requirements for configuring a FMC SPS.

An overview of key domain analysis activities as we performed them in our study is provided in Figure 1, where we divide them into two groups: activities conducted by the Certus V&V center at Simula Research Laboratory[3], and activities performed by FMC in interaction with Simula. In the following section, we will discuss each activity in detail and describe the timeline of these activities.

[3] http://simula.no/department/certus

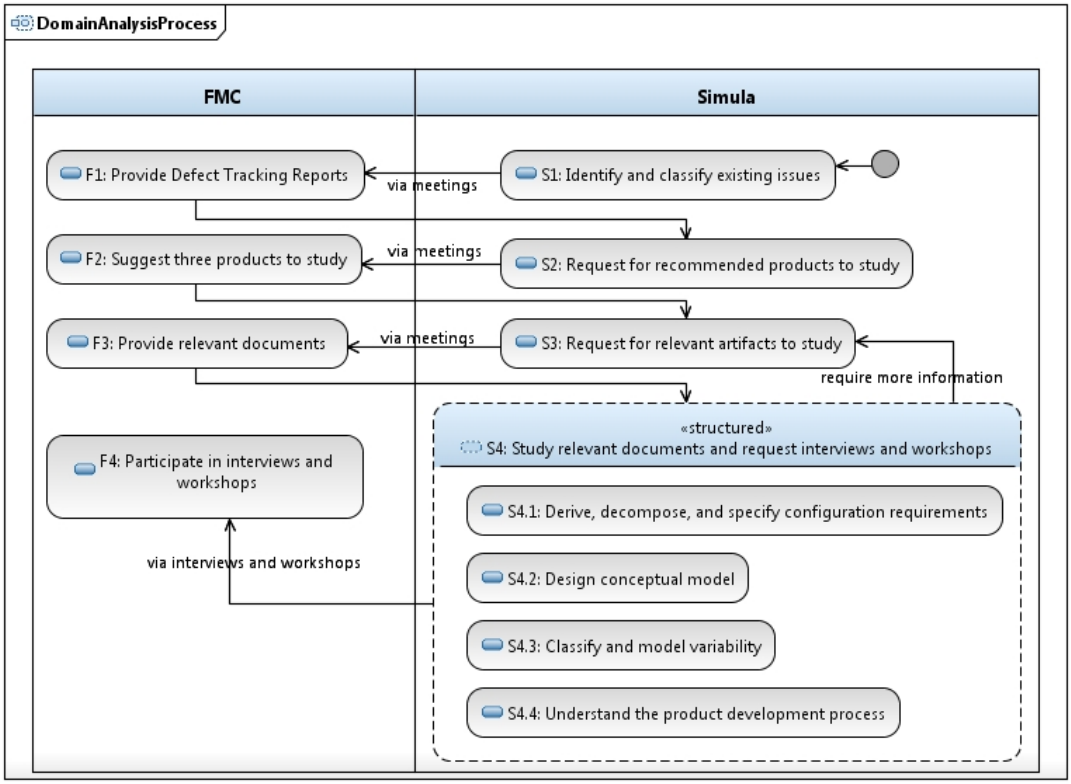

Figure 1 Key domain analysis activities

2.1 ACTIVITIES S1 AND F1

The first domain analysis activity was to identify and classify existing issues by studying the defect tracking systems of FMC and discussing with their software and configuration engineers. This activity was required to indicate the root causes of the problems and pinpoint the places where the FMC project development process can be improved. Results revealed that the key issues that FMC faces during product development are configuration-related issues such as incorrect and inconsistent configuration data, unclear configuration inputs to configuration engineers. The results of this root cause analysis was used to define our research objectives and will be used later to assess the extent to which our proposed solution matches the needs. This very initial step is considered as one of the most important steps in the domain analysis. In particular, analyzing defect tracking systems (if an organization has such systems) provides us with an opportunity to identify critical challenges or open issues in a more objective and systematic way, which might not be easy to obtain just through discussions or interviews with engineers. In this step, we collected an initial set of issues after analyzing FMC's defect tracking systems, which turned out to provide us with valuable insights. We then clarified and confirmed each of these issues with FMC engineers.

Lesson learned: Analyzing defect tracking systems or similar kinds of systems is a very effective way to more objectively and systematically identify and classify existing issues. Our experience showed it provided very different and complementary insights from informal discussions with engineers.

After this, it was also clear that the FMC product development lifecycle is an example of PLE, even though this was not explicitly perceived as such.

2.2 ACTVITIES S2 AND F2

Simula asked FMC engineers to recommend products to study (*S2*), based on the following criteria. First, the selected products should be representative in the sense that they should cover as many commonalities and variabilities of the product line family as possible. Second, the selected products should belong to the same product line family, as FMC has multiple product lines. Third, to ensure accuracy of the analysis, they should be recently completed projects such that relevant knowledge was still fresh in the minds of FMC engineers, in order to hold effective and efficient discussions and interview with them.

After three meetings, three representative products to study were selected by highly-experienced system engineers (*F2*). During the meetings, we spent much effort to first understand the subsea production domain, and then to convey the above selection criteria to the FMC engineers and to assist them in making their selection. Note that the selection process was iterative.

To have a concrete understanding of FMC subsea control systems, a guided workshop tour was arranged at the beginning of the domain analysis process for Simula researchers to see how a product instance was developed and what phases were involved in the process.

Lesson learned: It is crucial to gain a thorough understanding through direct interaction and experience (i.e., discuss concrete artifact examples with engineers and guided workshop tours) rather than just go through indirect descriptions (e.g., reading documents).

2.3 ACTIVITIES S3 AND F3

The information collection was kicked off by requesting relevant artifacts such as requirements, configuration guidelines and manuals, standards, regulations, the in-house configuration tool, and hardware schematics. Collecting these was an iterative process and about half dozen meetings were organized to this end.

Recall that the objective of this domain analysis was to gather information relevant to PLE. We targeted two types of documents: ones relevant to the product line, which should be applicable to all its products when they are developed, and those relevant to specific products. These documents included system and software requirements, architecture design, process documents, and a product family glossary.

Lesson learned: In our experience, it is not a good idea to blindly trust the provided requirements and documents. In particular, variabilities across different products are often not explicitly documented so that it is critical to gather them during meetings, interviews, and workshops.

We also carefully studied the configuration manual guidelines, while also experimenting with the in-house configuration tool developed by FMC. Two of the meetings were with a configuration engineer who is highly experienced and is responsible for providing configuration guidelines and creating initial configuration files for the software development team. The meetings were to understand how configuration is actually performed at FMC and what are the inputs and outputs of each configuration step. The meetings involved tool demonstrations, simulations of the real configuration process, and discussions. Subsequently, we studied some of the configuration files of the identified products in activities S2 and F2. These configuration files provided distinct sets of values to configure the three identified products. This exercise helped us identify variabilities across the three products.

One of the inputs of the configuration process is hydraulic and mechanical hardware design schematics. A configuration engineer was required to review these schematics to glean information necessary to configure the corresponding software to be deployed to the computer hardware. We received two hours of training on how to interpret such schematics and how they are related to the configuration process. After that, we tested our understanding by checking some of the schematics and identifying configuration requirements from them. This process also helped us understand how the software controls instruments and monitors sensors. This information is very important for ICSs, as ICSs typically include hundreds and even thousands of variability points, most of which are related to the configuration of software controlled instruments and monitored sensors.

Lesson learned: Simulating the configuration process with the assistance of experienced configuration engineers helped us to quickly and easily understand the configuration process. Using the existing configuration tool helped us to pinpoint the source of configuration problems and what support was missing. This is very important as they identify the limitations of the current practice and therefore eventually lead to future research questions.

We also studied several standards and regulations to understand the oil and gas production domain. Some of these are intended to assist in making early-stage design decisions about products, and are often sources of variabilities across different products.

We also carefully inspected the source code of the software of the subsea control modules. As for other typical ICSs, this software is shared by all the members of the SPS product line family. In other words, the same source code (e.g., a set of highly-parameterized C++ classes) is shared for all the product line members and the software is configured differently for each product, mainly based on the hardware topology and configuration. Based on the source code, we manually derived an architecture-level structural model to understand the software architecture and to identify the configurable parameters of the software. Later on, the structural model was refined as part of the product line architecture model we derived for FMC. We

ensured the right level of abstraction in the architecture model by making sure that the model only contained information that is 1) necessary to understand how the software is connected to its relevant hardware components, 2) relevant to configuration, and 3) is used in the specification of constraints restricting a configurable component. Other details not related to the above three points were omitted from the model.

2.4 ACTIVITIES S4 AND F4

After having gathered relevant PLE information, we studied the collected documents, which led, whenever more information was required, to requesting more artifacts to study.

Activity *S4* involved eight interviews and five workshops with various numbers of FMC participants (between one and ten). To make optimal use of time and resources, interviews were requested only after when a sufficient number of questions was collected, or, for workshops, at a point when significant domain analysis results were achieved. The objective of the interviews was to resolve questions brought up during the analysis. They involved a small number of participants (less than four). The workshops, however, had more participants (between five and ten), and aimed to get early feedback on our analysis results. They were run as brainstorming sessions. The four workshops were on average three hours long and the interviews were on average one and a half hour long. The interview and workshop participants included system, software, configuration, and quality engineers and their managers.

Lesson learned: According to our experience, it is very difficult to ask an industrial partner to commit time for meetings and workshops. We spent much effort on this during the analysis process. For organizing a workshop or an interview, it is important to get the audience interested first. To that end, we did the following: First, during a workshop, to explain product line concepts and notations, we used examples that were familiar to FMC engineers. Second, we carefully prepared for each interview and asked very specific and concrete questions. In addition, before each interview, we also prioritized the questions to ensure that there was sufficient time for the most important ones.

On our part, we further decomposed activity *S4* into four sub-activities (*S4.1*-*S4.4*). The first activity includes deriving, decomposing, and specifying configuration requirements (*S4.1*), which are the requirements that should be met by a configuration process and used to define evaluation criteria to assess it. We used a conceptual model, as recommended in [12], to identify, define, and organize the concepts relevant to configuration, and to assist in formulating a precise and concise description of the concepts and their relationships (*S4.2*). A conceptual model [1] (also known as a domain model) captures domain concepts and the relationships among them. It attempts to clarify the meaning of these concepts to minimize possible misunderstanding among stakeholders. A conceptual model can be described using different notations, Unified Modeling Language (UML) [15] being a frequently used one. The resulting conceptual model is also an important artifact for the configuration tool design since it is an initial step towards the information model to be manipulated by the tool. In Section 3.3, we discuss a subset of the conceptual model.

Lesson learned: For large-scale ICSs, during the domain analysis, a large amount of information related to various aspects such as software, hardware, and configuration needs to be analyzed. Capturing the analysis results into a conceptual model helps cope with such complexity.

In addition to incrementally developing the conceptual model, we also built an initial product line architecture model of the FMC SPSs, for the purpose of collecting the commonality and variabilities identified during the domain analysis process. UML constructs, including package, template, and class diagrams, was found sufficient to build this product line architecture model. At a later stage, the model was refined and a modeling methodology [5] was proposed.

One of the important tasks of the domain analysis in PLE is to identify variabilities (*S4.3*) across all the different products of a product line family. We identified and classified all the variabilities of the three selected products of the family of a FMC SPSs. This step was important as its outcome was required later to assess whether a particular variability modeling solution can be used to specify all the different types of variabilities in a concise but precise way.

Domain analysis was also needed to understand the current configuration process at FMC (*S4.4*), which was the basis for recommending improvements and a new way to handle configuration.

2.5 TIMELINE

The domain analysis spanned roughly six months. During this time, we finished all the activities shown in Figure 1. The effort (hours) spent on each activity is given in Figure 2, amounting to a total of hours. The diagram also indicates roughly the sequence and overlap of these activities. Most of the time (80%) is spent

on the sub-activities of activity *S4*. Two researchers from Simula were involved and more than seven FMC engineers participated in discussions, meetings, and workshops.

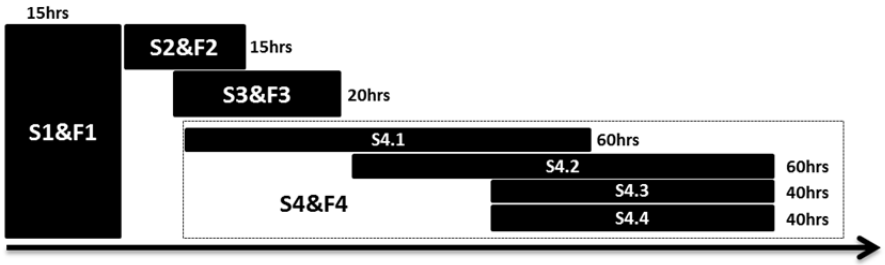

Figure 2 Domain Analysis Activities Timeline

3 DOMAIN ANALYSIS RESULTS

In this section, we discuss the key domain analysis results, which formed the foundation for overall quality and productivity improvement recommendations to the FMC product development practice. We first discuss (in Section 3.1) why we believe that developing FMC subsea production systems can be best described as a PLE process. Second, we summarize relevant characteristics of FMC SPSs in Section 3.2. Third, in Section 3.3, we formulate the key required configuration concepts as a conceptual model. In Section 3.4, we discuss in detail the current FMC product development lifecycle.

3.1 SOFTWARE-INTENSIVE SYSTEM PLE

The current FMC practice involves a series of refinements of their products to adapt them to the specific needs of a particular customer. The adaption process is actually a product configuration process, which includes: 1) configuring the hardware topology (e.g., making decisions such as how many wells to construct and how they are connected), and 2) configuring the software that is deployed to the hardware computing resources. The software controls and monitors the oil and gas production process, given a specific set of values for the configurable parameters of the software. These values are different from product to product and are jointly referred to as *configuration data*. The configurable parameters have to be properly configured before the software is loaded and executed to operate hardware devices. These parameters represent variabilities of individual products, and therefore, we can conclude that FMC SPSs and the current FMC practice of developing their products fit well with PLE and that, based on this, a more systematic, reliable, and scalable way to facilitate the configuration process can be introduced.

3.2 CHARACTERISTICS OF FMC SUBSEA PRODUCTION SYSTEMS

FMC SPSs are large-scale, highly-configurable, and highly-hierarchical, and software-intensive systems-of-systems, in which software controls and monitors the operation of electrical and mechanical instruments. They are large-scale and highly-configurable in the sense that a FMC subsea control system is composed of up to hundreds of control modules and thousands of instruments. This type of system is large-scale also in terms of its complicated configuration process; completely configuring a product may require resolution of hundreds or even thousands of variability points. Furthermore, since a hardware component often contains other hardware components, FMC SPSs have a hierarchical topology.

In a family of FMC SPSs, the hardware topology can vary from one product to another, with each topology being a specific configuration of the generic family design. Hardware is configured based on customer requirements, environmental conditions, and different regulations and standards.

As previously mentioned, members of a family of FMC SPSs share the same software code base (e.g., a set of highly-parameterized C++ classes). This software is configured differently for each product, mainly based on the hardware topology and configuration. For example, the number of electrical and mechanical instruments, as well as their properties (e.g., resolution of a sensor) affects the number and values of run-time objects in the software configured for a specific product instance. Each configuration of the software forms a unique installation (i.e., a set of deployable and executable software modules) provided for a specific product. Notice that this characteristic is unique to software-intensive system configuration and dependencies between the hardware and software should be captured and accounted for during the configuration process.

Software and hardware variability points occur at different levels of detail and are typically resolved by different specialists in different phases of the product development lifecycle. For example, high-level hardware decisions (e.g., number of wells) are made by domain experts after tendering and front-end

engineering design phases. Low-level variability points (e.g., the operating range of a device) are typically configured by configuration engineers or software engineers during the configuration, testing, or operation phases.

We believe that the above characteristics are not specific to FMC SPSs and our experience suggests they can be generalized to many other ICSs.

3.3 PROBLEM FORMULATION

The product line development process and product development process contain various development phases, involving different stakeholders, with various configuration rights and covering various disciplines. In this section, we formalize the key concepts of these two processes and their relationships as a conceptual model, as shown in the UML class diagram in Figure 3.

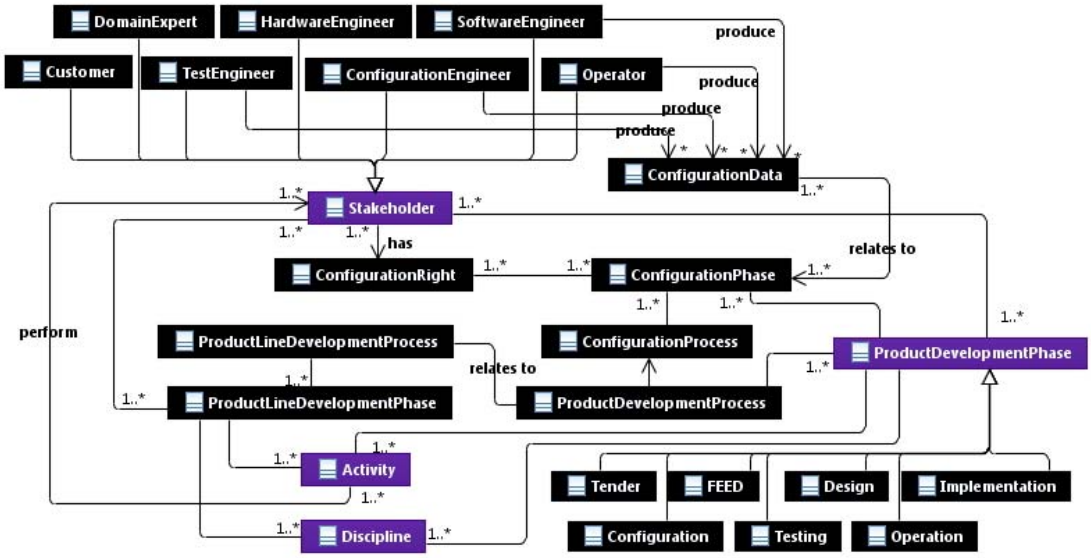

Figure 3 The conceptual model

As shown in Figure 3, both the `ProductLineDevelopmentProcess` and the `ProductDevelopmentProcess` have a set of distinguishable phases. Both processes have multiple development phases (`ProductLineDevelopmentPhases` and `ProductDevelopmentPhases`), involve different `Stakeholders`, relate to various `Disciplines`, and contain many activities (instances of `Activity`). FMC does not have explicit product line development phases and, therefore we do not discuss them further in the conceptual model. However FMC has a very explicit product development lifecycle, which is composed of seven phases: `Tender`, `FEED`, `Design`, `Implementation`, `Configuration`, `Testing` and `Operation`. These seven phases are discussed in detail in Section 3.4. The processes consist of a set of activities (instances of `Activity`) that should be performed by `Stakeholders`. There are typically six types of stakeholders in the FMC project development lifecycle: `Customers`, `DomainExperts`, `HardwareEngineers`, `SoftwareEngineers`, `TestEngineers`, `ConfigurationEngineers`, and offshore `Operators`.

Different stakeholders might have different `ConfigurationRights`. By configuration rights, we mean that not everyone has the knowledge required for all configurations and, therefore, we need to restrict certain configuration decisions to stakeholders who have the appropriate expertise. For example, hardware engineers might not have configuration rights to configure the software. The `ConfigurationProcess` is integrated with the product development process. The configuration process has multiple `ConfigurationPhases` corresponding to different configuration purposes such as testing and operation. Detailed discussion of these concepts is provided in Section 3.4.

It is worth noting that this conceptual model was used in our domain analysis to collect/organize/formulate information, as discussed in Section 2.4. However one can see that the conceptual model is general and therefore can be reused and extended for other kinds of domain analyses in PLE, except that different types of stakeholders or phases might be different from case to case.

3.4 PRODUCT DEVELOPMENT LIFECYCLE

In FMC, there is no explicit product line development process. However the lifecycle of the product development process is quite explicit and it is defined according to four dimensions: *Phases*, *Stakeholders*, *Activities*, and *Disciplines*, which are captured as four concepts in the conceptual model, as shown in Figure 3. In this section, we describe each product development phase, its related stakeholders, activities, and covered disciplines, and identify requirements for improving the configuration process.

3.4.1 TENDER AND FEED PHASES

The tender phase is a procedure for obtaining a contract. Domain experts are mainly involved in this phase to prepare the tender document. The Front-End Engineering Design (FEED) is a process focused on the conceptual development of a product and decisions made in this stage have a very high impact on cost and time estimates. Both the Tender and FEED phases are actually front-end phases, including activities mainly conducted by domain experts and focusing on product definition and analysis. These activities span over multiple disciplines such as requirements engineering, design, and configuration. For example, domain experts need to identify and define product requirements (by interacting with customers), analyze them, propose a product architecture, and configure the product by making top-level configuration decisions, concerning mainly the top-level hardware topology. ***Req1***: A solution should support different levels of configurations: top-level hardware topology configuration, lower-level hardware component configurations, and parameterized software configuration.

During these two phases, domain experts also need to estimate the cost and resources required, and assess the reliability and availability characteristics of the hardware topology configuration. For an ICS, the top-level hardware topology should have acceptable reliability, which can be achieved by, for instance, redundancy and by using highly-reliable hardware components. ***Req2***: A solution should support the assessment of reliability and availability of the top-level hardware topology configuration.

3.4.2 DESIGN AND IMPLEMENTATION PHASES

Hardware engineers, including mechanical, hydraulic, and electrical engineers, along with software engineers, are the primary actors in these two product development phases.

Hardware engineers take the product documents and partially configured product architecture (mostly textual documents), both of which are outputs of the FEED phase, as input to design the hardware. One of the most important artifacts produced in this activity is the schematics of the hardware design.

Software engineers first need to define software requirements based on the product documents provided during the Tender and FEED phases. Second, they need to design, analyze and verify the software design against the requirements, followed by creating and modifying the software when necessary. Notice that, as we discussed in Section 3.2, members of a family of FMC SPSs share the same software code base and the software is configured differently for each product, mainly based on the hardware topology and configuration. Therefore, creating and modifying the software is very rare, except when the product under development needs new types of hardware devices, which might require changes in software drivers for instance.

From the point of view of configuration, there is no direct, explicit, and easily manageable transition from the Tender and FEED phases to the Design and Implementation phases. Decisions made during the tender and FEED phases are scattered across product documents, with no consideration given to facilitate subsequent, low-level configuration decisions. ***Req3***: A unified platform is required for explicitly and systematically sharing and managing configuration information, including commonalities, variabilities, and configuration data, across all the product development phases, which is particularly important for systems like ICSs, since configuration activities are performed by different stakeholders, in different phases, and for different purposes.

3.4.3 CONFIGURATION, TESTING, AND OPERATION PHASES

As previously discussed, software has to be configured to form a unique installation provided for a specific product. Configuration and test engineers are the main actors in this activity. Configuration engineers need to configure software based on the system and software requirements of the product and the selected hardware design. The information required to perform such configuration is again scattered in requirement documents, which are not always relevant to configuration, and decisions made earlier during the tender and FEED phases about the hardware topology are not explicitly captured. For example, hardware engineers need to select a specific type of pressure sensors and this information is captured in the hardware

schematics of a product. However this information is very important to correctly configure some parameters (e.g., pressure range) of a software class representing this type of sensors. In order to collect this kind of information, a configuration engineer has to go through the hardware schematics, identify such information, and then perform the configuration. Such a process is inefficient and error-prone, which explains the large proportion of configuration errors in defect reports. Such configuration-relevant information should be captured explicitly in a way such that it can be easily used for subsequent configuration steps (***Req3***).

Outputs of the configuration mainly include configuration data and a configuration manual (a guide in selecting correct classes to configure and entering values for configurable parameters for each class), which is provided to software engineers, test engineers, and offshore operators for configuring their system. In the FMC product development lifecycle, there are different testing phases including software testing, hardware testing, Environmental Stress Screening (ESS), and Factory Acceptance Testing (FAT). Different testing phases require different configuration data. For example, ESS only tests each individual subsea control module and therefore the configuration is limited to the subsea control module under test only, not the whole subsea production system. The test environment for hardware testing provides stimuli and responses during test execution, including a sensor simulator with a correct hardware interface. As for software, a so-called generic sensor makes it possible to test the hardware on a broader basis than a real sensor. This is required especially in ESS to test a hardware component under extreme conditions. Therefore the software should be configured according to the generic sensor instead of the real sensor. During the FAT testing of an XmasTree (a mechanical component that physically contains instruments connected to control modules), real sensors are connected to the control modules and are tested together with the control modules. Therefore, in this case, the software should be configured for the real sensors this time. Configuration data should be systematically captured such that they can be reused as much as possible. In addition, certain subsystem or component instances of an ICS may have identical or similar configurations, for example, in case of redundancy for fault tolerance. ***Req4***: A solution should be able to systematically and automatically support configuration information reuse.

Another issue regarding the configuration process is that assumptions and the rationale for design choices should be captured explicitly rather than merely residing in the minds of engineers. For example, during the hardware testing phase of a subsea control module, devices under control by the control module are simulated since the devices are often unavailable during the hardware testing phase. In such case, configuring software according to the simulated devices should be considered to be an assumption regarding the actual device properties. Such information is necessary when performing subsequent configuration steps (e.g., the FAT testing) or fixing defects. For example, configuration problems reported by system operators would be easier to analyze and fix if such information is properly captured in the first place. ***Req5***: A mechanism is required to capture assumptions and decision rationales along with configuration data.

In complex ICSs (e.g., FMC SPSs), we have observed that configuration errors are very difficult and costly to locate and fix, due to the large number of variabilities, their interdependencies, the many stakeholders involved, and inadequate automation of the configuration process. In particular, lack of automation creates many opportunities for human errors, such as the need to enter identical configuration information multiple times, which can result in inconsistencies. ***Req6***: A solution should help automatically guarantee the correctness and consistency of configuration decisions.

Recall from Section 3.2 that FMC SPSs are large-scale ICSs, which are composed of hundreds and thousands of hardware components and devices and configurable software containing thousands of configurable parameters. ***Req7***: A solution should be able to handle hundreds or even thousands of interrelated configuration decisions.

3.4.4 PRODUCT LINE DEVELOPMENT LIFECYCLE

As we mentioned previously, though FMC has a product line development lifecycle, it is not explicitly structured as such. Product-line activities are scattered across the various activities of the product development lifecycle. For example, during the design and implementation phases, software might be updated for one product due to a new instrument type. This update might be propagated to other products of the same product line family under consideration, stating that this new type of instrument will be used in the future for all other products. In that case, the updated software should then be a product line asset, instead of a product-instance artifact. Though some of the product-line artifacts (e.g., product line software) are distinguished from product-dependent artifacts (e.g., product software), there is currently no support for the co-evolution of these two types of artifacts. In other words, there is no clear separation between domain and application engineering teams at FMC. This makes it hard to distinguish product line artifacts from product-

specific artifacts and to support their concurrent evolution. However, since we focus on the configuration process in this paper, we do not discuss this further.

4 FEASIBILITY ANALYSIS OF EMPLOYING MBPLE

Based on the domain analysis results discussed in Section 3, we opted for Model-Based Engineering (MBE) technologies to cope with the configuration issues that FMC is facing and developed a product-line architecture model. The model was created using UML and its extensions by applying a product line architecture modeling methodology we proposed for ICSs [5].

In this section, we assess how our proposed MBPLE approach matches the needs for configuration support in FMC. We first decompose our overall objective into sub-objectives. Second we discuss relevant the relevant properties of a MBPLE approach based on existing MBE technologies. Last, based on our experience at FMC, we discuss how a MBPLE approach addresses each sub-objective by relating the MBPLE properties to the sub-objectives. The mapping among the configuration requirements (Req1-Req7), the sub-objectives, and the MBPLE properties are summarized in Table 1.

Table 1 Mapping between the configuration requirements, configuration objectives, and key MBPLE properties

Configuration Requirements	Configuration Objectives				MBPLE Properties				
	Obj1	Obj2	Obj3	Obj4	Precision	Automation	Separation of Concerns	Abstraction	Change Analysis
Req1	√				√			√	
Req2				√		√	√	√	√
Req3			√		√		√	√	
Req4		√			√	√	√	√	
Req5			√		√		√	√	
Req6				√		√	√	√	√
Req7			√		√		√	√	

4.1 OBJECTIVES

We decompose our overall objective into the following sub-objectives:

- **Obj1**: Provide a reliable, scalable, systematic, automated, hierarchical and end-to-end configuration process to configure a product from determining its top-level hardware topology to assigning values to the configurable parameters of the software.
- **Obj2**: Provide a way to systematically manage the configuration process, focusing on enabling effective reuse of configuration data—the main artifacts of the configuration process.
- **Obj3**: Provide a scalable and unified knowledge-sharing platform to capture the architecture of a product line, including commonalities and variabilities of its products, and specify decision rationales and assumptions,
- **Obj4**: Enable automated analysis, including the reliability and availability analysis during the tender and FEED phases, consistency checking of configuration data, and change impact analysis.

These four objectives were derived from the seven configuration requirements (Req1-Req7) that we obtained through domain analysis (Section 3). The mapping between the objectives and requirements is provided in Table 1. In the following subsections, we discuss each of these objectives and their targeted configuration requirement(s).

4.1.1 OBJ1: SYSTEMATIC AND AUTOMATED CONFIGURATION PROCESS

As explained in Section 3.3, the configuration process involves multiple stakeholders, i.e., domain experts, software engineers, test engineers and configuration engineers. Different stakeholders have different configuration rights and configure the system at different phases for different purposes. Therefore, the configuration process is highly complex and a systematic solution is needed to handle such a complex function.

To specifically address ***Req1*** identified during the domain analysis process (Section 3.4.1), we recommend the hierarchical structure presented in Figure 4 as a basis for the configuration process, which can be characterized as a layered and end-to-end process: the configuration process is organized into layers and spans all configuration decisions. Assuming one top layer, *N* middle layers, and one bottom layer for hardware configuration, the number of intermediate layers (N) depends on the complexity of a system and the depth of the hardware topology hierarchy is. The configuration process starts naturally with configuring

the top level hardware topology (Top layer), gradually decomposing top-level hardware components into sub-components and configuring them according to their hierarchical structure until the complete hardware topology is configured (Middle and Bottom layers). When all the hardware topology configurations are completed, the parameterized software classes are then configured (Software Parameter Layer).

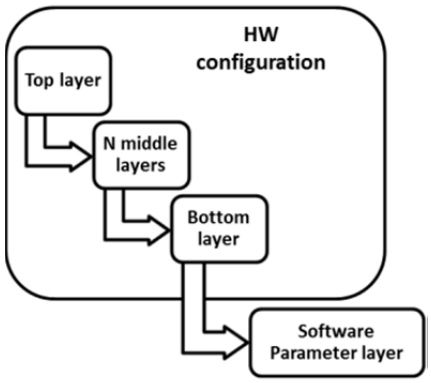

Figure 4 Layered and end-to-end configuration process

During the top layer, hardware configuration phase, high-level decisions regarding the field topology should be made (e.g., number of subsea control modules to deploy according to the number of wells to explore in a field). These decisions usually have a high business impact. Factors related to policies, history, regulations, legacy applications, and previous product experiences can all influence this level of decisions. These decisions are typically made by domain experts and the output of this step is a partially configured product. The middle and bottom layer of hardware configuration phases make subsystem-level decisions and unit or instrument-level decisions (e.g., type of sensors and valves). Domain experts and hardware engineers are typically involved in these two configuration phases. During the software configuration phase, the main tasks include assigning correct values to thousands of parameters of parameterized software classes, and ensuring the consistency of these values with the hardware configuration and among themselves. Configuration engineers and software engineers are the main stakeholders of these two tasks.

One can see that a lower layer configuration phase (e.g., software parameter layer) depends on the configuration performed in its previous phases (e.g., top, middle, and bottom hardware configuration layers). Such dependencies should be explicitly captured to ensure higher quality of the configuration and to improve productivity. However, the current configuration practices of our industry partner separate the hardware configuration from the software configuration and the dependency between the software and the hardware still remains as tacit knowledge in the heads of configuration experts. Such a practice results in many configuration errors that are often detected very late in the project development lifecycle. Therefore, a more systematic configuration process is required to tackle the complexity caused by the involvement of multiple stakeholders, various phases, different layers of configuration, various activities, and multiple disciplines, with preferably automated tool support.

4.1.2 OBJ2: CONFIGURATION PROCESS MANAGEMENT

The main artifacts of the configuration process are configuration data resulting from the different configuration phases, generated by different stakeholders with different configuration rights, for different purposes, as shown in Figure 3. In addition, the same configuration data may need to be reused across different configuration phases. Therefore, two key issues in configuration process management are how to ease the reuse of configuration data and make sure correct configuration data is used at the right place for the right purpose, are two key issues in configuration process management.

There is no explicit specification of relevant configuration knowledge and therefore the reuse of configuration data across different configuration phases of the same product, or across different products of the same product line family, becomes very difficult and entirely reliant on the expertise of engineers. In addition, configuration data for different testing purposes (e.g., unit test, hardware test, ESS test, FAT test) bear similarities since these tests are all for the same product. Therefore, it is clearly necessary to systematically manage such configuration data such that we know when and how to reuse data to minimize duplication and simplify evolution (***Req4***).

4.1.3 OBJ3: UNIFIED KNOWLEDGE-SHARING PLATFORM

A product line architecture captures the commonalities and variabilities of all products of the product line family (***Req3***). It should also form the basis for automated and systematic product configuration and derivation. It should help engineers make design decisions and track their rationale. It should allow tracing back to design rationales, requirements, regulations, and standards, which are all input of the architecture design and configuration phases of the product development lifecycle. It should also provide a common understanding and unified knowledge-sharing platform for different stakeholders, and support evolution by easing the identification of configuration problems and their solutions.

Considering ***Req5***, a proposed approach should also provide a mechanism to represent assumptions and decision rationales and connect them to other configuration artifacts such as specific decisions made during a specific configuration step. A unified platform is expected to specify and help manage all these artifacts such that, for example, querying across different configuration artifacts can be facilitated.

To address the scalability issue (***Req7***), the general principle of separation of concerns should be applied to handle complexity. For example, organizing the product line architecture of a system in a hierarchy of views, sub-views, and models, where each package has a clear focus (e.g., representing a well-defined view, or grouping models and model elements related to a particular subsystem or subcomponent), is one way to handle through MBPLE.

4.1.4 OBJ4: AUTOMATED ANALYSIS

As indicated by ***Req2***, during the tender and FEED phases of the FMC product development lifecycle, domain experts need to assess the reliability and availability of the top-level hardware topology configuration of a product. For example, they need to estimate whether the proposed top-level hardware topology has acceptable reliability, which can be achieved by different complementary means, e.g., redundancy, locating critical sensor systems on subsea trees, and/or using high-reliability hardware components. Redundancy is particularly important if component replacement is difficult, or if significant production availability or operating capability is lost through single-component failures. Therefore, it is important to provide an automated approach that helps domain experts assess the reliability and availability of a top-level hardware topology configuration at the early stage of the product development lifecycle. This kind of analysis should be based on the product line architecture model and the configuration data (representing the top-level hardware topology configuration under assessment) with additional information such as the mean time to repair of a unit or component.

As indicated by ***Req6***, inconsistencies among configuration data are mainly caused by human mistakes due to inadequate automation during the configuration process. Tool support is expected to automatically guarantee the correctness and consistency of configuration data by performing run-time consistency checking each time when a configuration input is provided.

Impact analysis [7] is in general useful to support the planning of changes, making changes, and tracing the effect of changes, as a measure of the cost of a change, and to drive regression testing. Impact analysis is particularly important in PLE since it can assist domain engineers during the tender and FEED phases in making high-level design choices. During the configuration process it provides insights regarding the cost and resources entailed by decisions. Second, it helps determine which features/variabilities lead to large change impacts on the other assets in the product line.

4.2 MBE PROPERTIES RELEVANT TO PLE

We discuss, in the section, the key MBE properties that are a priori relevant to PLE. We call them MBPLE properties, as summarized in Table 1.

Precision: Modeling languages (e.g., UML) typically have rich and full-bodied notations and provide a systematic way to extend themselves (e.g., UML profiling mechanism). This can be used to precisely model both structural and behavioral aspects of a system in a way that is suited to both context and objectives.

Automation: Mature MBE technologies such as metamodeling [6], constraint checking, and model transformation [13], provide systematic mechanisms to facilitate the automation of many tasks based on models, including automated model analysis and verification, and code and test data generation.

Separation of concerns: Standard modeling languages such as the UML and its extensions, provide mechanisms for modularity and encapsulation, which help implement the principle of separation of concerns. This can be used to model a system in a well-organized architecture where each part fulfills a meaningful role. Consequently, better scalability can be achieved.

Abstraction: For a given objective and context, to avoid distracting and irrelevant details about the system under study, abstraction is a key mechanism. With separation of concerns, this is a key mechanism to achieve scalability. Abstraction is a key feature of standard modeling languages, which enables the definition of modeling methodologies that are suited to a given context and objective, by specifying what must be modeled and at what level of detail.

Change analysis: MBE technologies (e.g., consistency checking and change impact analysis) provide mechanisms to reliably and predictably handle changes (e.g., introducing new features, fixing a defect). For example, the Object Constraint Language (OCL) [14], initially used as a precise modeling language complementing UML specifications, can be also used to evaluate queries or check constraints on model instances. This usage of OCL is important for rule-based and model-based consistency checking and impact analysis. There exist OCL evaluators [2] to enable this type of analysis and they are applied in our context.

4.3 MAPPING FROM MBPLE PROPERTIES TO OBJECTIVES

4.3.1 PRECISION

We need a configuration process to systematically configure a product (Obj1). To make the configuration process systematic, we need to precisely capture the commonalities and variabilities of a family of products as a common asset (usually as an architecture model) (Obj3). We have devised a product-line modeling methodology (SimPL) for families of ICSs [5] that we applied at FMC for the above purpose. SimPL provides notation and guidelines for creating product line models at a sufficient level of precision. Most notably, our experience at FMC has shown it can be used to thoroughly specify all types of variabilities identified in ICSs.

To facilitate configuration data reuse (Obj2), one possible solution is to 1) group variabilities related to a subsystem or component into distinct packages and 2) organize and maintain resolution/configuration of the grouped variabilities into reusable groups of variability resolutions (i.e., configuration data). To do so, we need a systematic approach to provide, represent, and manage configuration data and connect them to the product line architecture model. In our SimPL methodology [5], we rely on UML Packages and Templates, characterized with a specific stereotype called <<ConfigurationUnit>>, to group and organize variabilities for each configurable component. We have also proposed another stereotype (<<Inherit>>) to facilitate reuse of configuration information by applying it to UML Dependencies connecting two groups of variabilities corresponding to two configurable components, respectively. This implies that configuring one configurable component requires resolving variabilities in dependent configurable components. These two stereotypes were used extensively in the FMC case study without particular difficulty. Note that the SimPL methodology is part of an ongoing project to realize MBPLE in the organization of our industry partner and is not the focus of this paper.

4.3.2 AUTOMATION

In families of ICSs, at FMC and elsewhere, we face large numbers of interdependent variability points, multiple stakeholders' involvement, and complex analysis needs (e.g., reliability and availability assessment of top-level hardware topology configuration, configuration data validation, and consistency checking of configuration decisions) (Obj4), configuration reuse needs (Obj2), and this entails automated support. Modeling environments enable high degrees of automation through capabilities for model analysis and transformations (Section 4.2). The SimPL profile proposed as part of the SimPL methodology [5], including a set of stereotypes and constraints, has been implemented in the IBM Rational Software Architect (RSA) [11] modeling environment. This profile was fully applied to the FMC SPSs. RSA is embedded with model transformation technologies (e.g., Eclipse Modeling Framework (EMF) [10]) and provides model analysis support (e.g., consistency checking and impact analysis), thus supporting the configuration of products and their verification. We are currently in the process of developing a model-based, automated, user-interactive configuration tool to automate the proposed configuration process. The configuration tool builds on product line models created with SimPL, and uses constraint solvers to interactively guide engineers to construct and verify configuration data. In the future, industrial strength tool support for employing MBPLE in our context will be available to automate the proposed configuration process to the maximum extent.

4.3.3 SEPARATION OF CONCERNS

A scalable configuration process (Obj1) is expected to resolve thousands of variabilities; a scalable solution is needed to specify, represent and manage large amount of configuration data (Obj2); a unified knowledge sharing platform requires a solution capable of handling such complexity (Obj3); an automated

analysis approach should be scalable to enable, for example, the querying of large-scale models and processing of large amounts of computation (Obj4). In our SimPL methodology [5], we have implemented the principle of separation of concerns by using UML packages to organize the architecture of the FMC SPSs in a hierarchy of views, sub-views, and models, where each package has a clear focus (e.g., representing a well-defined view, or grouping models and model elements related to a particular subsystem or subcomponent). The SimPL methodology also organizes the generic product line architecture description into two models: base and variability models, which are loosely connected through inherent UML features. The above mechanisms were widely applied in modeling the FMC SPSs and helped different stakeholders identify parts of the product line architecture that were relevant to them.

4.3.4 ABSTRACTION

Suitable mechanisms for abstraction are needed for all objectives. In the FMC case study, the level of abstraction specified by our modeling methodology, based on extensions of UML, enabled us to model a very large product family with reasonably-sized models. For the industrial case study we conducted [5], the product line architecture model in total contains five views and subviews and is visualized in 17 class diagrams. The model elements contain a total of 71 classes. In total, 109 variabilities are specified for 22 configurable components. A total of 16 OCL constraints are captured. The evaluation of the case study shows that the product line architecture model was created at the right level of abstraction, such that it could facilitate product-line configuration and product derivation.

4.3.5 CHANGE ANALYSIS

Based on existing MBE technologies (Section 4.2), model-based analysis can provide an automated approach to perform model-based reliability and availability analysis of top-level hardware topology configurations and consistency checking of configuration data, and change impact analysis (Obj4). In the FMC context, we are still investigating model-based change analysis and have at this stage little experience to report. However according to our previous experience on change impact analysis in different contexts [8, 18], we can foresee that existing MBE technologies [2] are likely to address the change analysis requirements at FMC.

5 CONCLUSION

In this paper, we reported on experiences and insights gained from investigating the application of model-based product line engineering to large-scale, highly-configurable Integrated Control Systems (ICSs). Our case studies took place in the context of subsea production systems developed by FMC technologies. We first presented the steps of the systematic domain analysis we conducted at FMC, the lessons learned from the process, and the effort spent on each domain analysis activity. It is worth noting that the activities involved in domain analysis and the lessons learned are quite general and, therefore, can be applied to the study of similar kinds of families of ICSs, including estimating the effort required.

Second, we discussed the key results of the domain analysis and specified seven configuration requirements, based on which we derived four objectives that should be met by a Product Line Engineering (PLE) solution. These domain analysis results, including the derived conceptual model, the configuration requirements, and the identified objectives expected from a systematic configuration process, are not specific to our FMC context. They are generally applicable to any large-scale ICS where hardware and software are integrated to control and monitor physical devices and processes. In other words, our domain analysis results are relevant and valuable beyond the context of our industry partner.

Last, we report our experience in using such a PLE solution based on model-driven engineering for a family of ICSs. Our assessment results suggest that model-based PLE, such as the one we applied, is indeed suitable to achieve all four objectives. As part of the ongoing project, based on the domain analysis results, we have proposed a UML based, product-line modeling methodology for families of ICSs [5], which we named SimPL. We are currently developing a SimPL-based, user-interactive, automated configuration tool to facilitate the automated configuration of families of ICSs. In the future, we plan to integrate SimPL and the configuration tool to the model-based product-line engineering practice of FMC for the purpose of improving the overall quality and productivity of their system development.

6 ACKNOWLEDGEMENT

We are grateful to engineers at FMC Technologies for their support and help on performing the domain analysis.

7 REFERENCES

[1] Conceptual Model (computer science): http://en.wikipedia.org/wiki/Conceptual_model_(computer_science)
[2] Eclipse OCL: http://www.eclipse.org/modeling/mdt/?project=ocl
[3] Papyrus UML Modeler, http://www.eclipse.org/modeling/mdt/papyrus/
[4] America P., Thiel S., Ferber S. and Mergel M., "Introduction to Domain Analysis, http://www.esi.es/esaps/public-pdf/CWD121-20-06-01.pdf," 2001.
[5] Behjati R., Yue T. and Briand L., "SimPL: A Product Line Modeling Methodology for Families of Integrated Control Systems," Simula Research Laboratory, 2011.
[6] Bezivin J., "On the unification power of models," *Software and Systems Modeling*, vol. 4 (2), pp. 171-188, 2005.
[7] Bohner S. A., "Software change impact analysis," *IEEE Computer Society Press*, 1996.
[8] Briand L. C., Labiche Y., O'Sullivan L. and Sowka M., "Automated Impact Analysis of UML Models," *Journal of Systems and Software*, vol. 79 (3), pp. 339-352, 2006.
[9] Dhungana D., Neumayer T., Grunbacher P. and Rabiser R., "Supporting evolution in model-based product line engineering," pp. 319-328, 2008.
[10] Eclipse Foundation, Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/ (Last accessed April 2010)
[11] IBM, Rational Software Architect
[12] Khurum M. and Gorschek T., "A systematic review of domain analysis solutions for product lines," *Journal of Systems and Software*, vol. 82 (12), pp. 1982-2003, 2009.
[13] Kleppe A., Warmer J. and Bast W., *MDA Explained - The Model Driven Architecture: Practice and Promise*, Addison-Wesley, 2003.
[14] OMG, "OCL 2.0 Specification," Final Adopted Specification ptc/03-10-14.
[15] OMG, "UML 2.2 Superstructure Specification (formal/2009-02-04)."
[16] Pohl K., Bockle G. and Van Der Linden F., *Software product line engineering: foundations, principles, and techniques*, Springer-Verlag New York Inc, 2005.
[17] Van Der Linden F., Schmid K. and Rommes E., *Software product lines in action: the best industrial practice in product line engineering*, Springer-Verlag New York Inc, 2007.
[18] Yue T., *Towards Vertical Impact Analysis of UML Models*, Thesis, Carleton University, Systems and Computer Engineering, 2006