A Component-based Product Development Process for a Workflow Management System Product Line

Itana Maria de Souza Gimenes[1], Ruy Nishimura[1], Edson Alves de Oliveira Junior[1], Fabrício Ricardo Lazilha[2], Uirá Kulesza[3], Carlos J. P. Lucena[3]

[1]Departamento de Informática
Universidade Estadual de Maringá
Av. Colombo, 5790
87020-900 Maringá-PR Brazil
{itana, edson}@din.uem.br
ruy@dc.unifil.br

[2]Departmento de Informática
Centro Universitário de Maringá (CESUMAR)
Av. Guedner, 1610
87050-390 Maringá-PR Brazil
fabricio@cesumar.br

[3]Departmento de Informática
Laboratório de Engenharia de Software
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Brazil
{uira, lucena}@inf.puc-rio.br

Abstract. *The product line approach has been pointed out as a means to improve software organization's productivity. It offers both technical and managerial support to design applications for a specific domain from a well-defined infrastructure. A specific application can be developed from this infrastructure by customizing its variable aspects. This paper presents a process to develop members of a component-based product line for Workflow Management Systems (WfMS). The activities of the process are described within the context of a case study that develops a WfMS to manage service orders in a software house. Aspects and research issues relevant to the product development process are discussed.*

Resumo. *A abordagem de linha de produto tem sido indicada como uma das formas de aumentar a produtividade nas organizações. Ela oferece apoio técnico e gerencial para projetar aplicações de um domínio específico a partir de uma infra-estrutura bem definida. Uma aplicação específica pode ser obtida por meio da resolução de aspectos variáveis definidos nesta infra-estrutura. Este artigo apresenta um processo para desenvolvimento de aplicações específicas a partir de uma linha de produto baseado em componentes para sistemas de gerenciamento de workflow (WfMS). As atividades do processo são descritas no contexto de um estudo de caso que visa desenvolver um WfMS para gerenciar ordens de serviço em uma software house. Aspectos e questões de pesquisas relevantes sobre o processo de desenvolvimento de produtos são discutidos.*

1. Introduction

The product line (PL) approach [Clements and Northrop 2001] has been providing suitable solutions to improve productivity in software development. However, there are still several open issues [Bosch 2004]. According to the Product Line Practice of the Software Engineering Institute (PLP/SEI) [Clements and Northrop 2001] there are three main processes in the product line context: artifact development (domain engineering), product development (application engineering) and management. Current PL research results have been more related to domain engineering whereas there are fewer works related to the process of product development. However, the combination of results from component-based techniques and generative programming provides important and complementary concepts and mechanisms that can be used to enhance the PL approach. The main issue dealt with in this paper is concerned with the product development process following a component-based approach.

A product line for Workflow Management Systems (WfMS) [Gimenes et al. 2003] was built to comply with an industrial need and the effort of the Workflow Management Coalition (WfMC) [WfMC 1995] to provide customized WfMS made up of independent and interoperable components. The WfMC established a generic architecture and a reference model for WfMS to enable the customisation of workflow products according to market needs. Organisations need workflow products that have similar features but with some different aspects. They need simple and adaptable products which do not have the complexity of the broad and general purpose ones. Examples are workflow products with either traditional or web user interface, products for small versus large enterprises and products with different task scheduling algorithms.

The artifact development process of the product line for WfMS was described in Gimenes et al. (2003). The work presented in this paper describes the process for product development. This process is used to produce members of the WfMS family from the product line architecture and its components according to client's requirements. The process was designed based on: (i) terminology of the PLP/SEI; (ii) principles of design and analysis of Catalysis [D'Souza and Wills 1999]; (iii) some extensions of Kobra regarding variability management and the decision model [Atkinson et al. 2001]; and (iv) UML (Unified Modelling Language) artifacts tracing of the Rational Unified Process (RUP) [Kruchten 2000]. Thus, the results of the work presented in this paper show both the evolution of a product line based on novel technology and the application of proposed process in a practical case sudy.

This paper is organized as follows. Section 2 presents the existing product line (PL) for WfMS. Section 3 presents an overview of the product development process whereas Section 4 presents its activities illustrated with the case study developed. Section 5 presents lessons learned. Section 6 presents related work and discussion about relevant issues regarding the product development process.

2. The PL for WfMS

The design of the product line for WfMS [Gimenes et al. 2003] was mainly based on the Catalysis method [D'Souza and Wills 1999]. Catalysis was used, as it is a general purpose component-based development method, based on UML [Rumbaugh et al. 1999] that encompasses important concepts such as the central role of software architecture, frameworks and patterns. The fact that we represent all artifacts of the PL in UML makes it easier for designers of traditional approaches to understand our specification and also enable us to take profit of current support tools such as IBM Rational Rose [IBM 2005].

The artifact development process of the PL is composed of the following phases: (i) requirement analysis; (ii) system specification; (iii) architectural design, and (iv) component internal design.

In the requirement analysis, the domain model representing the objects and actions of the domain were developed. The domain analysis was based on the generic architecture and reference models for WfMS of the WfMC [WfMC 1995]. The similarities and variabilities amongst product line members were identified and represented at the level of use case model following the notation proposed in Jacobson et al. (1997).

In the system specification phase, the analysis of the system's actions led to the identification of the types and related actions. Sequence diagrams were designed for each use case representing the interactions between objects. The correspondent variation points were represented in the system specification phase based on the notation proposed in Morisio, Travassos and Stark (2000) which extends UML with a variability stereotype.

From the system specification several refinements were made to reach the level of architecture of components. The product line architecture is presented in Figure 1.

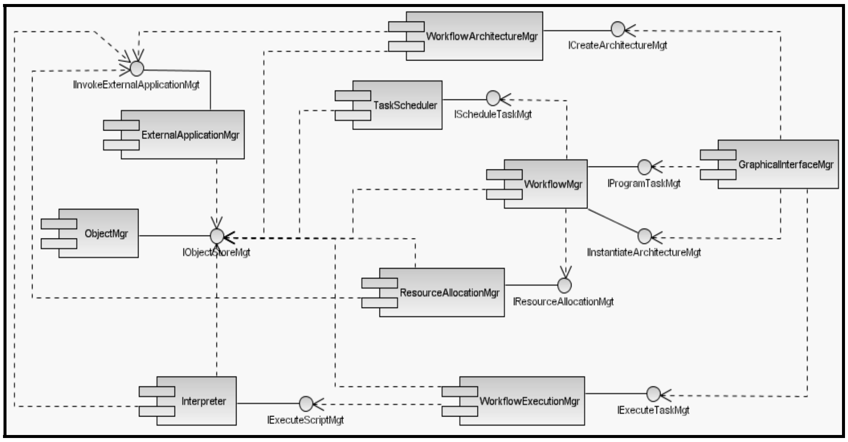

Figure 1: Architecture of the PL for WfMS.

The components of the architecture and their respective variation points are described as follows:

- ***GraphicalInterfaceMgr:*** responsible for user interface management. One variation point is the user interface being via web browser or conventional.
- ***WorkflowArchitectureMgr:*** supports the definition and maintenance of workflow architectures. This component makes the workflow definition more flexible as the workflow types are not static. Variation points include the resource type, the tool type, and the process language supported. Resource can be specialised into actor, tool and material types. Tool type can be either internal or external. Different process programming languages can be supported depending on the interpreter.

- ***WorkflowMgr:*** responsible for the instantiation and management of projects that are associated with a workflow. A project includes an instantiation of workflow architecture. For each workflow element in the architecture there is an object in the workflow instance. No variation points were defined for this component.
- ***WorkflowExecutionMgr:*** responsible for the control and management of workflows. The main variation point in this component is the possibility of executing different scheduling algorithms.
- ***TaskScheduler:*** responsible for the scheduling of tasks. It allows the interaction between the users and the tasks. Variation points include resources to be used: types of resources and tools to be used (external or internal).
- ***ResourceAllocationMgr*:** responsible for resource allocation (e.g. actors, tools and material). In addition to the resource type and tool type, variation points include resource allocation policies.
- ***ExternalApplicationMgr:*** responsible for the management of external applications during the workflow definition and task execution. Variation points include different mechanisms to adapt external applications to the workflow.
- ***ObjectMgr:*** responsible for the object management support. It maintains workflow data such as: control data, information data and even a whole workflow. All other components of the architecture use its services. This makes the architecture independent from the object management system. Variation points include adapters for different databases management systems.
- ***Interpreter:*** responsible for the execution of a workflow script written in a process programming language [WfMC 1995].

The specification of invariants, preconditions and post-conditions for the components was also carried out based on OCL [OMG 2005a]. The product line architecture was evaluated based on Rapide and its support tools [Computer Science Lab 1997].

Currently we are populating the architecture by either developing novel components or reengineering components developed previously not following the PL approach. Each component internal design also follows the Catalysis method. As far as we progress in their design, the architecture is revaluated. The implementation of the product line components is being carried out based on open source tools which include: Java, the Swing (Java 2) toolkit [Sun 2005] and the JHotDraw framework [JHotDraw 2005]; CORBA JacORB [JacORB 2005], an Object Request Broker (ORB) implemented in Java that has many functionalities not identified in similar products; and, MySQL [MySQL 2005] together with the ObjectBridge framework [Apache 2005].

3. The Product Development Process

In this section, we describe the product development process defined to enable the instantiation of the product line infrastructure described in Section 2. The definition of the process was mainly based on the concepts of the PLP/SEI [Clements and Northrop 2001] and [Atkinson et al. 2001]. The PLP/SEI was chosen because it offers a conceptual structure and terminology for product lines that we consider as a standard. Kobra was adopted as a basis model both because it follows UML and is component based. These are also characteristics of our existing product line. Thus, the adaptation of an existing product development process with similar basis was considered more applicable. Kobra is referred to as an object oriented customization of PULSE [Bayer et al. 1999] that contains a software engineering process easier to be adapted to organizations. It follows two main activities: framework engineering and application engineering. The representation of the artifacts in both activities is based on UML. Variabilities are captured by decision models which guide the application engineering. As described in Section 2, our product line already contains the artifacts that represent its main infrastructure also based on UML. Thus, we inherit the structure of the decision model from Kobra to conceive our product development process. A prototype to support the product development was also built based on IBM/Rational Rose [IBM 2005].

An overview of the process is presented in Figure 2. It is composed of the following phases: (i) requirement analysis; (ii) product instantiation, and (iii) product packing. In this section, we present an overview of the process whereas in section 4 we illustrate the development of some activities, within the phases, for a case study.

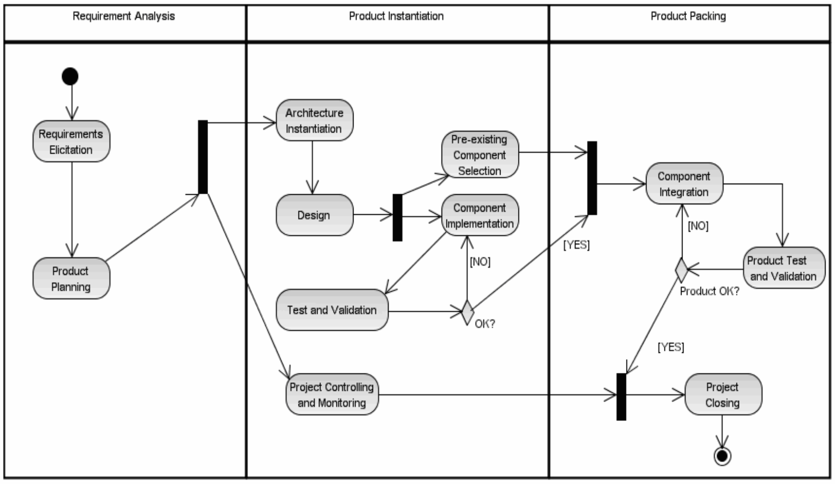

Figure 2: Product Development Process.

The first phase or our process is the requirement analysis. As pointed out by Clements and Northrop (2001), this phase affects both the specific process to produce the product and the product line itself, as new requirements might be introduced. This phase is composed of the following activities: requirements elicitation and product planning. The requirements elicitation takes the requirements of the product line and the decision model as input and produces the resolution of the decision model and the initial requirement document for the specific product as output. This activity identifies the differences between the specific product and the product line infrastructure. It resolves most of the variabilities, thus, it impacts the instantiation of the product line architecture. The participation of a product line specialist is important as he/she can estimate the effort and cost of the product development. The product planning activity includes issues regarding scheduling, resource allocation, cost, monitoring and control of the product development process.

The second phase is the product instantiation. It aims at resolving variabilities at the architectural level. It takes as input the artifacts generated in the first phase, so the variabilities are resolved according to the content of the document resolution of the decision model. It generates as output an instance of the architecture for the new product, the updated design and eventual new components. It is composed of the following activities: (i) architecture instantiation; (ii) design; (iii) pre-existing component selection; (iv) component implementation; (v) component test and validation and (vi) project controlling and monitoring. In this phase, it is possible to find out that requirements are not satisfied by the components of the product line. These requirements can be supported by the acquisition of COTS (Commercial off-the-Shelf) components, adaptation of legacy software or the development of a new component. In any case, the design or adaptation of the component has to follow the product line architecture. Moreover, the generalization of the component so that it could be incorporated in further versions of the product line infrastructure must be considered. The activity of monitoring and control provide managerial support for the product instantiation.

The product packing phase involves: (i) component integration; (ii) product test and validation, and (iii) project closing.

Variability management in the proposed process is guided by the UML artifacts of the product line infrastructure which are represented following the RUP framework of the IBM Rational Rose [IBM 2005]. Thus, the effect of decisions made at the use case model is traced up to the product line architecture. It follows the use case realizations and affects their respective classes, attributes, methods and sequence (or collaboration) diagrams. This is an additional support proposed in our process that was not dealt with in Kobra. It follows our guideline that by keeping the product line approach closer to existing methods and tools, its introduction in industry becomes easier. In order to support this tracing, new stereotypes were introduced to UML to represent the variation points. Figure 3 represents the stereotypes developed to represent the variation points in the use case model. They are described as follows:

- a Use Case Variability or an Actor Variability is represented by a dot in the use case ellipse;
- the stereotype Use Case Variability Deleted indicates that an alternative use case related to a variation point will not be implemented for the product;
- the stereotype Variability Use Case Updated indicates that a use case related to the variation point was modified;
- the stereotype Variability Use Case Inserted indicates that a new use case was included at a variation point.

In order to identify and trace the classes related to a use case, a script was implemented in the IBM Rational Rose. By using this mechanism, at the end of the generation process, we have the UML design documentation of the specific product updated. It shows the changes made in the product line to achieve that product. This aspect is important for further maintenance.

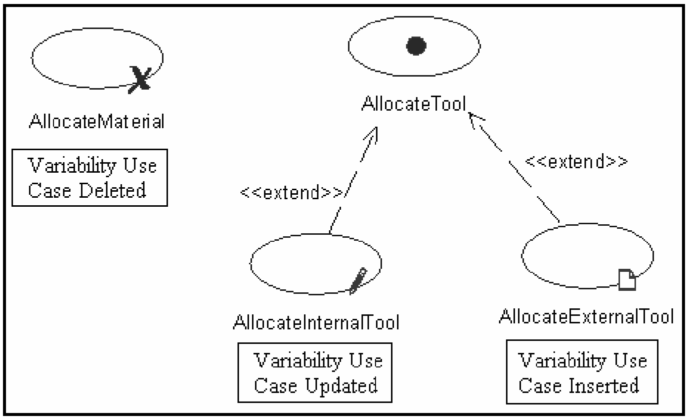

Figure 3: Stereotypes developed for the Use Case Model.

4. Case Study

In this section, we present the case study used to assess the product development process described in Section 3. Our case study consisted of the development of a Service Order Management System (SOMS) for a software house. The system controls enquiries for services, elaborated by the customer care personnel, such as maintenance changes, addition of extra functionalities and bug fix requests. The features of the system were obtained from a real software house, fictitiously named here XYZ due to the company requests. Thus, the case study consists of generating the SOMS based on the proposed product development process for the existing WfMS product line. It explores the instantiation of the artifacts along the process. The issues regarding implementation are not dealt with in this case study. The author of the proposed process played the role of the product line engineer. A complete description of the artifacts produced for the case study can be found in Nishimura (2004).

As all the artifacts and activities of the case study could not be fully described in this paper, we chose to present a process excerpt related to the use case **ExecuteWorkflow** and the activities requirement elicitation, architecture instantiation and design.

4.1. Requirement Elicitation

The requirement elicitation was supported by interviews with the XYZ manager, manuals and by observing some of the company service order procedures in vivo. The interviews were guided by questionnaires elaborated from the artifacts of the product line. The results obtained were registered in the artifact called **Initial Requirements**. This artifact contains: objective, context, general overview, an analysis of the problem, a description of the functionalities required for the product and technical constraints. Throughout the interviews it was observed that the most important issues to the company's managers were: task planning, assignment and execution control; task resource allocation and scheduling; and control over the service orders. They were mainly worried about task delays and inappropriate resource allocation. Thus, the required functionalities listed for SOMS were: definition of workflow architectures;

workflow instantiation; task definition and programming; resource allocation; definition of criteria for task progress, workflow execution; and task monitoring and control (mainly tasks delay alerts).

The **Decision Model** is represented as a table in which the rows contain the decision to be taken to resolve the variabilities. The columns represent the attributes of the decision as follows:

- **ID**: each decision has a unique identifier.
- **Question**: a question formulated according to the variabilities of the product line artifacts.
- **Variation Points**: the decisions are related to the variable element of an artifact.
- **Alternatives**: each decision may provide a set of alternative solutions to resolve the variability.
- **Effects**: set of effects that results from the possible solution. They describe how the artifacts of the product line have to change to comply with the product requirements.

Table 1 shows the decisions with respect to the use case **ExecuteWorkflow**. The scripts referred to in the table were implemented to allow the tracing in the IBM Rational Rose, as mentioned before.

Table 1: **Decision Model for the use case ExecuteWorkflow.**

ID	Question	Variation Point	Alternatives	Effect
WE1	Which is the scheduling algorithm?	WorkflowExecutionMgr	serial or priority control	If serial: 1) change “ExecuteWorkflowWithPriorityControl” use case stereotype to “Variability Use Case Deleted”; 2) remove stereotypes from “WorkflowExecuteSerial” use case; 3) execute script “VariabilityHandler.ebs”; 4) remove “PriorityControlMgr” class; 5) remove “PriorityControlScheduling” class; 6) remove “ExecuteWorkflowWithPriorityControl” use case. If priority control: 1) change “ExecuteWorkflowSerial” use case stereotype to “Variability Use Case Deleted”; 2) remove “WorkflowExecutionWithPriorityControl” use case stereotypes; 3) execute script “VariabilityHandler.ebs”; 4) remove “SerialScheduling” class; 5) remove “ExecuteWorkflowSerial” use case.
WE2	Communi-cate with users?	CommunicateWithUsers	Yes or No	Yes: 1) remove “Variability Use Case” stereotype from “CommunicateWithUsers” use case; 2) resolve the decisions “WE3”, “WE4” and “WE5”. One of these must be “Yes”; 3) execute script “VariabilityHandler.ebs”; No: 1) change “CommunicateWithUsers” use case stereotype to “Variability Use Case Deleted”; 2) resolve the decisions “WE3”, “WE4” and “WE5” with “No” response; 3) execute script “VariabilityHandler.ebs”; 4) remove “CommunicationMgr” class;

				5) remove “CommunicateWithUsers” use case.
WE3	Communi-cate via tele-conference?	CommunicateViaTeleConference	Yes or No	Yes: 1) remove stereotype “Variability Use Case” from the use case “CommunicateViaTeleConference”; 2) execute script “VariabilityHandler.ebs”; No: 1) change “CommunicateViaTeleConference” use case stereotype to “Variability Use Case Deleted”; 2) execute script “VariabilityHandler.ebs”; 3) remove “TeleConference” class; 4) remove “CommunicateViaTeleConference” use case.
WE4	Communicate via e-mail?	CommunicateViaEMail	Yes or No	Yes: 1) remove “Variability Use Case” stereotype from “CommunicateViaEMail” use case; 2) execute script “VariabilityHandler.ebs”; No: 1) change “CommunicateViaEMail” use case stereotype to “Variability Use Case Deleted”; 2) execute script “VariabilityHandler.ebs”; 3) remove “EMail” class; 4) remove “CommunicateViaEMail” use case.
WE5	Communi-cate via Chat?	CommunicateViaChat	Yes or No	Yes: 1) remove “Variability Use Case” stereotype from “CommunicateViaChat” use case; 2) execute script “VariabilityHandler.ebs”. No: 1) change “CommunicateViaChat” use case stereotype to “Variability Use Case Deleted”; 2) execute script “VariabilityHandler.ebs”; 3) remove “IRC” class; 4) remove “CommunicateViaChat” use case.

The issues related to requirements of the **Decision Model** were resolved. Table 2 shows the resolution of the decision model with respect to the use case **ExecuteWorkflow**. It indicates that the serial scheduling algorithm was chosen and that the managers opted for no communication between SOMS users.

Table 2: Resolution of the decision model for the use case ExecuteWorkflow.

ID	Question	Variation Point	Resolution	Effect
WE1	Which is the scheduling algorithm?	WorkflowExecutionMgr	serial	1) change “ExecuteWorkflowWithPriorityControl” use case stereotype to “Variability Use Case Deleted”; 2) remove “WorkflowExecuteSerial” stereotypes; 3) execute script “VariabilityHandler.ebs”; 4) remove “PriorityControlMgr” class; 5) remove “PriorityControlScheduling” class; 6) remove “ExecuteWorkflowWithPriorityControl” use case.
WE2	Communi-cate with users?	CommunicateWithUsers	No	1) change “CommunicateWithUsers” use case stereotype to “Variability Use Case Deleted”; 2) resolve decisions “WE3”, “WE4” e “WE5” with “No” response; 3) execute script “VariabilityHandler.ebs”; 4) remove “CommunicationMgr” class; 5) remove “CommunicateWithUsers” use case.
WE3	Communi-cate via teleconference?	CommunicateViaTele-Conference	No	1) change “CommunicateViaTeleConference” use case stereotype to “Variability Use Case Deleted”; 2) execute script “VariabilityHandler.ebs”; 3) remove “TeleConference” class;

				4) remove “CommunicateViaTeleConference” use case.
WE4	Communi-cate via e-mail?	CommunicateViaEMail	No	1) change “CommunicateViaEMail” use case stereotype to “Variability Use Case Deleted”; 2) execute script “VariabilityHandler.ebs”; 3) remove “EMail” class; 4) remove “CommunicateViaEMail” use case;
WE5	Communi-cate via Chat?	CommunicateViaChat	No	1) change “CommunicateViaChat” use case stereotype to “Variability Use Case Deleted”; 2) execute script “VariabilityHandler.ebs”; 3) remove “IRC” class; 4) remove “CommunicateViaChat” use case;

Figure 4 shows the use case model after the variability being resolved. In this model only the functionalities regarding serial scheduling are kept and no communications are represented anymore.

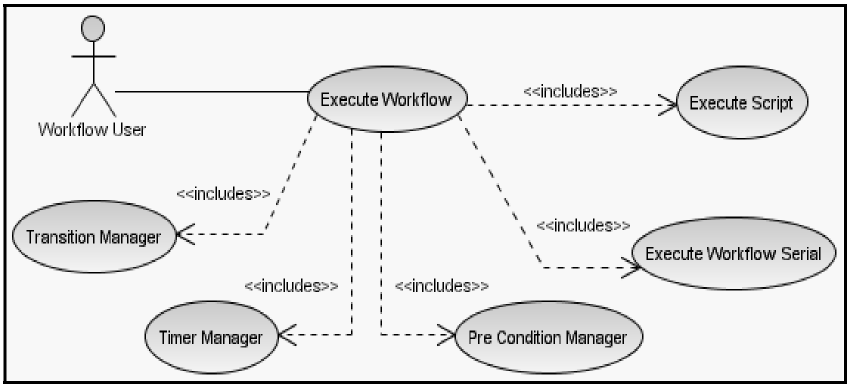

Figure 4: Use case model for ExecuteWorkflow.

4.2. Architecture Instantiation

As a follow up of the decisions taken with respect to the use cases variabilities, the architecture of the new product is obtained. Note that this is possible due to the tracing mechanism implemented in the IBM Rational Rose which follow the main framework RUP template available in this tool. Figure 5 shows the component architecture SOMS after variabilities resolved (see Figure 1). The component **ExternalApplicationMgr** of the product line architecture was removed as a result of decisions made for the use case **DefineWorkflowArchitecture** and **DefineWorkflow**. This component was not considered important from the point of view of the stakeholder.

Our case study has dealt with the following types of variabilities [Anastasopoulos and Gacek 2000]: positive, negative and optional. Taking the workflow manager component (**WorkflowMgr**), the following are examples of variabilities resolved: (i) positive – the addition of the functionality for material; (ii) negative – the removal of the functionality for tool allocation; (iii) alternative – the choice of the serial scheduling algorithm.

4.3. Design

The design activity updates the class diagram according to the resolved decision model. The updates in the class diagrams may result in alterations in the sequence diagram.

Figure 6 shows the class diagram for SOMS that already has all the variabilities resolved. Following the excerpt of the case study related to the use case **ExecuteWorkflow**, we can see that as a result of the choice of the serial scheduling algorithm the classes `PriorityControlMgr`, `PriorityControlScheduling`, `CommunicationMgr`, `TeleConference`, `EMail` and `IRC` were removed. As a consequence, the sequence diagrams were also updated to reflect the decisions made regarding actions sequence and collaborations.

It is important to mention that although we didn't come across to difficult cases of design tracing in this case study, the problem of specifying interacting features makes this issue much more difficult. This is a current open problem in product lines. It is an area where UML specifications become difficult due to the necessity of constraints specification in addition to stereotypes.

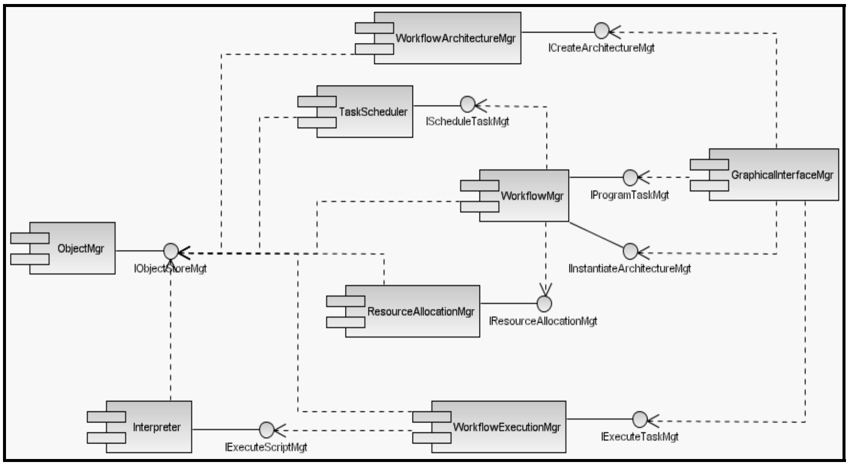

Figure 5: SOMS Architecture.

5. Lessons Learned

In this section, we present the lessons learned from the development of the product development process and the overall product line for WfMS. These lessons take into account the benefits of the proposed process and the difficulties faced.

UML Models: our approach to the artifact and product development process is based on UML. The artifacts of the PL are represented as UML models to which the variability stereotypes were added to represent the differences between products. This approach allowed us to resolve the questions of the decision model and, at the same time, to trace the decisions to the artifacts of the product line. Thus, at the end of the product development, an updated UML model of the specific product is obtained. So, UML acted both as a design language and as a DSL (Domain Specific Language) to support the application engineering. As far as maintenance is concerned this is an adequate approach because a proper documentation of the specific product can be used to support further changes throughout the product operational life cycle. Kobra [Atkinson et al. 2001] and Gomaa (2005) also confirm this advantage. However, there are still subtle questions to be discussed in this view. There is currently a hot discussion of the advantages of DSL over UML or vice-versa. It is worth reading Alan Will's blog on this matter, although it is informal [Wills 2005].

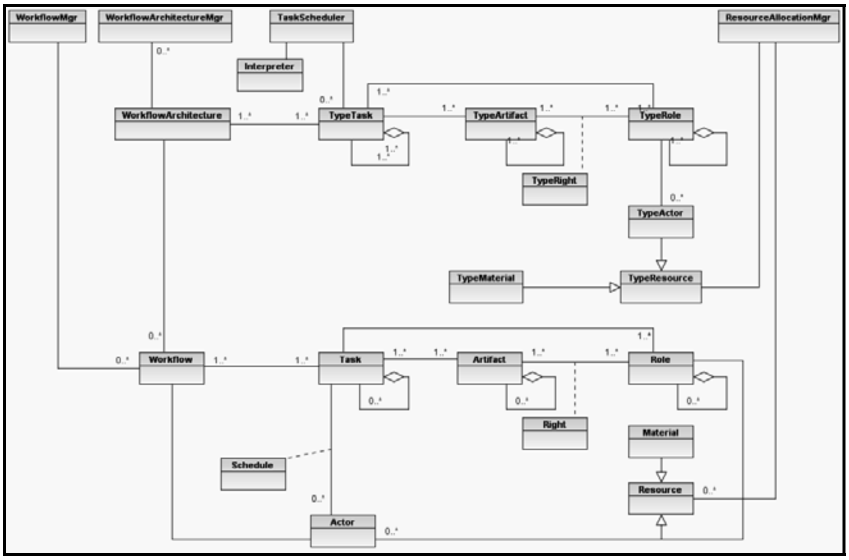

Figure 6: Instantiation of the Class diagram for SOMS.

<u>Variability Management</u>: this work did not use the feature model. We use the use case model and decision models which contain the features and their variabilities, as described in section 4. This approach is also followed in Kobra. However, in recent works [Oliveira Junior 2005], where a variability management process was introduced, the feature model turned to be essential as an initial path to trace variability as indicated in most of the domain analysis works. We can always represent variability from the use case models but the feature model offers an important higher abstraction level. The current problem is that software engineers are not familiar with feature models yet.

<u>Tool support</u>: In our work, we use IBM Rational Rose as a support tool [IBM 2005]. We are aware that several tools to support the PL approach have been developed, such as Ménage [Van der Hoek 2000], Holmes [Succi et al. 2001] and pure:variants [Beuche et al. 2003]. However, they still lack support for the whole PL life cycle as well as are not in accordance with popular tools that developers have been using such as IBM Rational Rose [IBM 2005], Together [Borland 2005] and Argo UML [Trigis 2005]. Our experience has shown that the UML artifacts of the product line can be traced with scripts of the IBM Rational Rose. We are aware that there are still difficult issues to be dealt with such as how to specify and resolve constraints. However, we believe that, for the success of the PL approach, it is important to minimize its impact on current software development technology normally used by software engineers in industry.

6. Related Work

There are still many discussions related to how novel is the product line approach [Poulin 1997]. The PL approach encompasses existing techniques of reuse from several areas such as domain engineering, software architecture, frameworks, patterns and generative programming. The lessons learned from the development of this work show that these techniques are complementary, thus, the PL constitutes a systematic approach that aims at integrating these techniques to offer automated mechanisms to modelling application domains and generating specific applications. Well-known methods for domain analysis [Kang 1990] and [Arango 1994] are used as a basis for the identification of concepts and functionalities required for a family of products in order to represent them as a generic model that is the main infrastructure to support reuse. These methods use the concept of features to represent the common functionalities and variabilities of a domain. The product line approach is directly related to frameworks [Pree 1995]. The idea of variability is analogous to hot spots. Moreover, frameworks constitute building blocks of a PL infrastructure. Patterns [Buschmann 1996] can be used both from the point of view of architectural styles as well to make the development of components easier. The software architecture is the main asset of a PL, so methods and techniques to design, represent and evaluate architectures are relevant to the PL approach. Generative programming [Czarnecki and Eisenecker 2000] and [Batory 2004] is based on similar concepts of those of PL. There have been workshops [Butler 2001] to discuss how one approach can benefit from the other. It can be observed that the domain modelling of both approaches are rapidly evolving as both take benefits of work already available from domain engineering. Current PL approaches have attempted to be closer to existing software development methods by adopting UML based concepts whereas generative programming looks for DSL and generators. In our view the PL approach can benefit from generative programming to improve the application engineering process.

Some product lines methods are based on the specification of artifacts in UML [Kruchten 2000] and [Gomaa 2005]. Thus, their product development processes are supported by UML models which are modified to represent variability. We have considered the use of the artifact models of Kobra [Atkinson et al. 2001] due to several reasons which include: (i) it is based on UML; (ii) its architecture is component-based; (iii) its decision model is relevant and helpful, in particular to support the interviews between stakeholders and the product line engineer. In addition to adding a product process to the existing product line, we have adapted Kobra's artifacts to our needs and also conceived the main ideas towards using a commercial tool to support application engineering. PLUS [Gomaa 2005] was recently released. It takes a similar approach to ours. However, it is mainly based on RUP whereas we inherit many concepts from Catalysis, in particular regarding frameworks and architecture. The vertical and horizontal partitioning of the architecture of the Catalysis method [D'Souza and Wills 1999] offers a better path to find the architecture components.

Many recent works have focused on the automating PL development processes [Czarnecki and Eisenecker 2000] and [Greenfield and Short 2004]. These works address not only the definition of a production plan to help the instantiation of the product line architecture, but also investigate alternative means to automate the instantiation process. The main goals of generative programming [Czarnecki and Eisenecker 2000], for example, are: (i) development of a proper means to specify members of a software product line; and (ii) modeling of the configuration knowledge (mapping between problem and solution space) in detail in order to automate it by means of a code generator. In this way, approaches based on generative programming and software factories [Czarnecki and Eisenecker 2000] and [Greenfield and Short 2004] have emphasized the use of DSLs and code generators as an effective way to customize and instantiate members of a software PL. We have not considered these ideas in our product line as yet.

Model-driven development (MDD) [Greenfield and Short 2004] motivates the use of appropriate models to capture different features of software systems. Models that follow MDD approaches are used to generate executable code rather than only with the representation of artifacts for documentation. Thus, in the context of product lines, MDD approaches can bring benefits to allow the automation of product development processes. However, many current works in MDD, such as Model Driven Architecture (MDA) [OMG 2005b] have concentrated efforts on addressing only technical variability. Thus, there is still a need for these approaches to address application domain variability using DSLs. Important questions arise in this context, such as: (i) how to use UML models as DSLs to specify application domain variability; and (ii) how to provide new DSLs frameworks that can be integrated with the existing UML meta-model.

Currently, we are investigating the introduction of the feature model in the product development process to work together with the existing artifacts [Oliveira Junior 2005]. Our goal is to use the feature model with two purposes: (i) to manage the PL variabilities; and (ii) to automatically instantiate and customize the product line components. In this latter case the feature model can be viewed as a configuration DSL [Czarnecki and Eisenecker 2000]. Some integrated development environments (IDEs) have initiated the incorporation of feature models. Together [Borland 2005] already indicates that they can encompass support for product line in the future. For instance, RequisitePro [IBM 2005] already offers support for feature modelling, although still primitive. Thus, if the application engineering process is supported by UML models, it may not be difficult to design a support tool for the Rational suite or similar tools. There are market issues that may influence the future of this area. The overall gain is that, in this area, both industry and academy are close to provide real improvements to software development.

7. Summary

As the interest in the product line approach, both from academia and industry, grows more works that show its advantages and drawbacks are necessary. This paper presents the conception and application of a product development process for an existing product line within an important domain, which is WfMS [Gimenes et al. 2003]. The process is mainly based on Kobra [Atkinson et al. 2001], a relevant component-based product line approach. It builds on this approach by providing details on how to develop its activities and by investigating means to support artifact tracing with a commercial tool. Moreover, the paper discusses important issues regarding the relationship between product line and other areas of research.

References

Anastasopoulos, M.; Gacek, C., Implementing Product Line Variabilities, IESE-Report No. 089.00/E, Version 1.0, Fraunhofer Institut Experimentelles Software Engineering, Kaiserslautern, Germany, Nov. 2000.

Apache DB Project: ObJectRelationalBridge - http://db.apache.org/ojb - 2005.

Arango, G. Domain Analysis Methods. in: Software Reusability, Schafer, W.; Pietro-Diaz, R.; Matsumoto, M. (Eds.), Ellis Horwood, New York, NY, 1994, pp. 17-49.

Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthing, D.; Paech, B.; Wurst, J.; Zeitel, J. *Component-Based Product-Line Engineering with UML.* [S.l.]: Addison-Wesley, 2001.

Batory, D. The Road to Utopia: A Future for Generative Programming, In: Domain Specific Generation, Lengauer et al. (eds.), LNCS 3016, p1-18, 2004.

Bayer, J.; Flege, O.; Knauber, P.; Laqua, R.; Muthig, D.; Schmid, K.; Widen, T.; DeBaud, J. PuLSE: A Methodology to Develop Software Product Lines, in 1999 Proc. Symposium on Software Reusability - SSR99 Conference, May 1999.

Beuche, D.; Papajewski, H.; Schroder-Preikschat, W. Variability management with feature models. In: Software Variability Management Workshop, 2003, Portland. Proceedings of the ICSE-2003 Workshop on Software Variability Management. Portland, 2003. p. 72-83.

Borland USA, Together, http://www.borland.com/together - 2005.

Bosch, J. On the Development of Software Product line Components. In: Proceedings of theThird International Conference of Product Line, Nord, R. (ed.), LNCS 3154, p146-164, 2004.

Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerland, P.; Stal, M. *Pattern-oriented software architecture: a system of patterns*. 1. ed. John Wiley & Sons, 1996. 476 p.

Butler, G. Generative Programming and Product Lines. Software Engineering Notes, vol. 26, no. 6, ACM SIGSOFT, 2001, pp 74-76.

Clements, P.; Northrop, L. *Software Product Line: Practices and Patterns*. Addison Wesley Longman, 2001.

Computer Science Lab. DRAFT Guide to Rapide 1.0 – Language Reference Manuals, Rapide Design Team – Program Analysis and Verification Group. Stanford University, 1997.

Czarnecki, K., Eisenecker, U., *Generative Programming. Methods, Tools, and Applications*. Addison-Wesley, 2000. 832 p.

D'Souza, D. F.; Wills, A. C. *Objects, Components and Frameworks with UML – The Catalysis Approach*. Addison Wesley Publishing Company, 1999.

Gimenes, I. M. S.; Oliveira Junior, E. A.; Lazilha, F. R.; Barroca, L. M. *A Product Line Architecture for Workflow Management Systems with Component-based Development*, in 2003 Proc. The IEEE Conference on Information Reuse and Integration, pp. 112-119.

Gomaa, H. *Designing software product lines with UML: from use cases to pattern-based software architectures*. Addison-Wesley, 2005, 736 p.

Greenfield, J.; Short, K. Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools", John Wiley & Sons, 2004.

IBM Rational Software – http://www-306.ibm.com/software/rational/ - 2005.

Jacobson, I.; Griss, M.; Jonsson, P. *Software Reuse – Architecture Process and Organization for Business Success*, New York: Addison-Wesley, 1997.

JacORB - http://www.jacorb.org - 2005.

JHotDraw - http://www.jhotdraw.org - 2005.

Kang, K. Feature-oriented domain analysis (FODA) - feasibility study. Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh, 1990.

Kruchten, P. *The Rational Unified Process – An Introduction, Second Edition*. Ed. Addison-Wesley Pub Co, 2000.

Morisio, M.; Travassos, G. H.; Stark, M. *Extending UML to Support Domain Analysis*, in 2000 Proc. IEEE International Conference on Automated Software Engineering, pp. 321-324.

MySQL - http://dev.mysql.com - 2005.

Nishimura, R. T. Geração de Produto em uma Abordagem de Linha de Produto para Sistemas Gerenciadores de Workflow. Maringá, 2004. 125p. Master Dissertation, Departamento de Informática, Universidade Estadual de Maringá, Maringá.

Object Management Group. OMG Document: UML 2.0 OCL 2nd Revised submission – http://www.omg.org/cgi-bin/doc?ptc/2003-10-14 - 2005.

Object Management Group. MDA – Model Driven Architecture, http://www.omg.org/mda/ visited at April 2nd 2005.

Oliveira Junior, E. A. Um Processo de Gerenciamento de Variabilidade para Linha de Produto de Software. Maringá-PR, 2005. 157p. Master Dissertation, Departamento de Informática, Universidade Estadual de Maringá, Maringá-PR.

Poulin, J. *Software Architectures, Product Lines, and DSSAs: Choosen the Appropriate Level of Abstraction*, in Proc. 1997 WISR8.

Pree, W. *Design patterns for object-oriented software development*. Addison-Wesley, 1995. 268 p.

Rumbaugh, J.; Jacobson, I.; Booch, G. *The Unified Modeling Language Reference Manual,* Addison-Wesley Pub. Company, 1999.

Succi, G.; Yeip, J.; Pedrycz, W. Holmes: an intelligent system to support software product line Development. in: International Conference on Software Engineering, 23., 2001, Toronto. Proceedings of the International Conference on Software Engineering, 2001, p.829-832.

Sun Microsystems. Java Technology - http://www.java.sun.com - 2005.

Trigis.org Open Source Software Engineering. http://argouml.tigris.org -2005.

Van der Hoek, A. Capturing product line architectures. in: International Software Architecture Workshop. 4., 2000, Limerick. Proceedings of the 4th International Software Architecture Workshop, 2000. p. 95-99.

Wills, A. C., http://blogs.msdn.com/alan_cameron_wills - 2005.

Workflow Management Coalition. *Workflow Reference Model*. Document number TC00-1003, January, 1995.