A Component-based Product Line for Workflow Management Systems

Itana Maria de Souza Gimenes
Universidade Estadual de Maringá
Departamento de Informática
Maringá, Brasil, 87020-900
itana@din.uem.br

Fabrício Ricardo Lazilha
Centro Universitário de Maringá
Departamento de Informática
Maringá, Brasil, 87050-390
fabricio@cesumar.br

Edson Alves de Oliveira Junior
Universidade Estadual de Maringá
Departamento de Informática
Maringá, Brasil, 87020-900
edson@din.uem.br

Leonor Barroca
The Open University
Department of Computing
Milton Keynes, MK7 6AA, England
l.barroca@open.ac.uk

Abstract

This paper presents a component-based product line for workflow management systems. The process followed to design the product line was based on the Catalysis method. Extensions were made to represent variability across the process. The domain of workflow management systems has been shown to be appropriate to the application of the product line approach as there are a standard architecture and models established by a regulatory board, the Workflow Management Coalition. In addition, there is a demand for similar workflow management systems but with some different features. The product line architecture was evaluated with Rapide simulation tools. The evaluation was based on selected scenarios, thus, avoiding implementation issues. The strategy that has been used to populate the architecture and experiment with the product line is shown. In particular, the design of the workflow execution manager component is described.

Keywords: Product line, Reuse, Software Architecture, Workflow Management Systems.

1. Introduction

A software product line [1] is a collection of systems that share a manageable set of features amongst its main artefacts. These artefacts include a base architecture and a set of common components that populate the architecture. The design of a product line must consider similarities and variabilities amongst its products.

Product line is still a recent approach, which demands new architectures and components design methods. These methods should provide mechanisms to capture and represent domain features and variabilities. Existing methods include:

- **Synthesis** [2] - a wide approach to construct software systems representing system family instances with similar descriptions.
- **Family-Oriented Abstraction, Specification and Translation** (FAST) [3] - a common feature analysis of the domain which is important to: identify the context; describe the domain; provide a set of key terms; identify common features and variabilities; quantify variability providing variation parameters; and identify and register useful information during analysis.
- **Product Line Software Engineering** (PuLSE) [4] - a method to construct and use product lines. PuLSE's general structure includes the following stages: development, technical components, and support components.
- **Feature-Oriented Domain Analysis** (FODA) [5]: a method to support reusability on architectural and functional levels.

The product line approach is appropriate for domains where there is a demand for specific products that can be modelled from a set of common features and well-defined variability points. In this paper, we consider the domain of workflow technology [6]. This technology meets the current needs of organisations as the reengineering of legacy processes and the modelling and automation of business processes, supported by workflow systems, are means to improve the productivity and the quality of processes and products. In addition, workflow systems allow rapid development and modification of systems to comply with the transient and unexpected variations of the business environment.

Workflow systems are applications supported by WfMS (Workflow Management Systems). These systems support definition, management and execution of workflows. WfMS interpret process definitions, interact with the users (the human agents), and, when necessary they invoke tools and applications to execute parts of the workflow.

The WfMS domain is appropriate to the application of the product line approach due to both organisations' needs and the efforts of the Workflow Management Coalition (WfMC) [7]. This regulatory board established a generic architecture and a reference model for WfMS that can be used to customise products to market needs. Organisations need workflow products that have similar features but with some different aspects. They want simple and adapted products avoiding the complexity of the broad and general purpose ones. Examples are workflow products with either traditional or web user interface, and products with different task scheduling algorithms.

This paper presents a component-based product line for WfMS and its design and development process. Section 2 describes the product line design process. This includes extensions made to represent variability throughout the process. Section 3 presents the strategy to populate the product line architecture. In particular, the workflow execution manager component is presented. Section 4 describes related works. Finally, section 5 presents the conclusions.

2. The Product Line Architecture Design Process

Most of the existing product line design methods are based on domain engineering. In general, they are strong in domain modelling, but less efficient to represent architectures and components. We defend that Component-Based Development (CBD) methods can be used in the design of product lines to bridge the gap between domain analysis and the architecture and component design and implementation. General-purpose CBD methods are easy to understand and use. In addition, there are commercial tools that may be used to support them. In this work, we use the Catalysis method [8] to guide the WfMS product line architecture design process.

The product line design process proposed in this paper considers:

- domain analysis based on the generic architecture and reference models for WfMS of the WfMC [7];
- design of the product line architecture and its components based on Catalysis [8]; and
- evaluation of the architecture with Rapide [9] [10] language and tools.

Catalysis was used, as it is a general purpose CBD approach based on UML [11] and encompasses important concepts such as the central role of software architecture, frameworks and patterns.

Product line approaches that also take CBD into account are Kobra [12] and GenVoca [13], but they are, as yet, less disseminated than Catalysis.

The proposed product line design process is composed of the following phases: requirements analysis, system specification, architectural design and component internal design. The following sections describe the application of this process to design a component-based product line for WfMS.

2.1 Requirements Analysis

The design of a product line for WfMS starts from the domain model representing the objects and actions of the domain. In this phase, it is already possible to identify similar aspects and variation points amongst product line members. According to the product line terminology we are proposing an

architecture and components for a WfMS family of products. Each specific product that can be generated from this product line is a member of that family.

The reference model and generic architecture for WfMS of the WfMC were used to extract the main set of features of the WfMS family. These models define, at a higher-level, the main components and interfaces that a WfMS should have in order to allow interoperability of sub-products from different suppliers. In addition, the WfMC defines important characteristics to allow specific WfMS to be built according to an organisation's needs, such as workflow for software production or workflow for financial administration. Figure 1 presents the generic architecture for WfMS of the WfMC [7]. The WfMC models also indicated the potential components of the product line architecture and their interfaces. The Process Manager Pattern [14] was used, in our approach, to exploit the WfMS domain. This is an architectural pattern for definition of Process-Centred Software Engineering Environment (PSEE) process managers developed from studies of existing environments and from experiences obtained in the development of one of these environments [15].

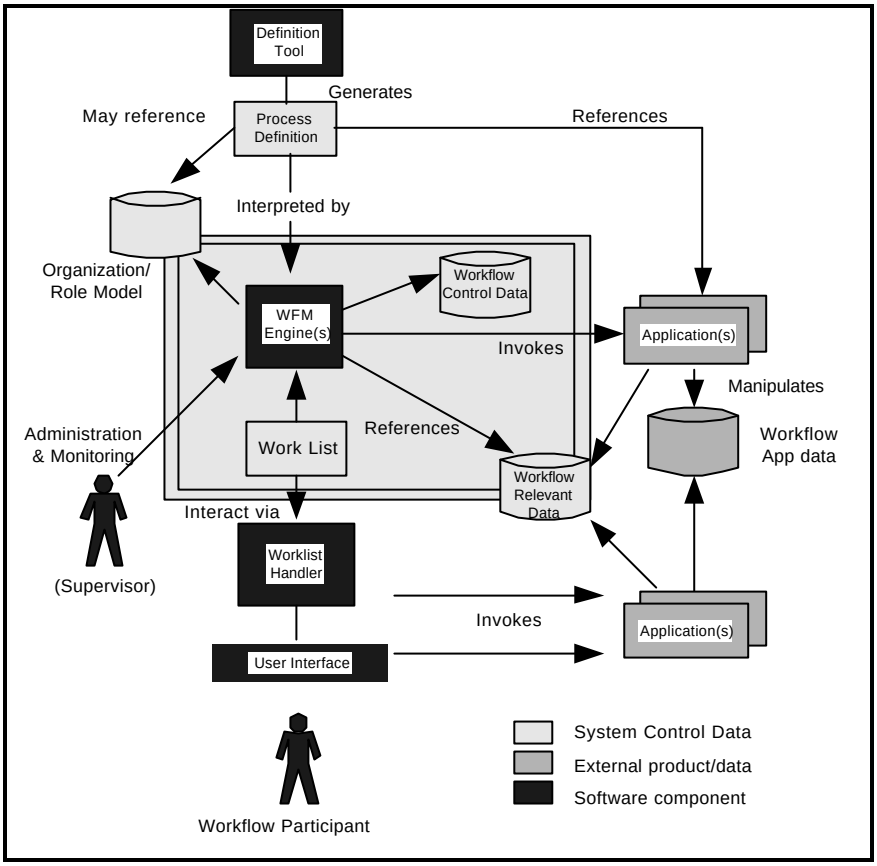

Figure 1: Generic architecture for WfMS [7].

A domain model in Catalysis is composed of objects and actions at a high level of abstraction and is independent of the software solution for the problem. This model is represented by UML use cases. There are three main actors in the WfMS domain: the workflow architecture manager, who defines reusable workflow architectures; the workflow manager (supervisor), who controls the instantiation, resource allocation and task assignment for the workflow; and the workflow user, who executes workflow tasks. The main use cases associated with these actors are represented in Figures 2, 3 and 4, respectively. In addition, a sequence diagram was produced for each use case.

One of the main aspects of a product line development is to capture and represent variabilities associated with the architecture and its components. In requirements analysis, the notation followed is the use case variability of Jacobson et al. [16] that suggests the stereotype «extend» to represent variation points in use cases. The extended use case that represents variations is annotated with a mark (blob). Examples of this representation can be seen in Figures 2, 3 and 4.

The use case Define Workflow Architecture represents the action of defining workflow architectures by the workflow architecture manager. The workflow architectures defined are used by the workflow manager to generate workflow instances. The Define Workflow Architecture use case, shown in Figure 2, is marked as a variation point to represent the optional feature Allow Dynamic Changes. This optional feature allows dynamic updating of workflow architecture. Another example appears in the use case Define Tool Type that specifies two extensions: Define Internal Tool Type or Define External Tool Type.

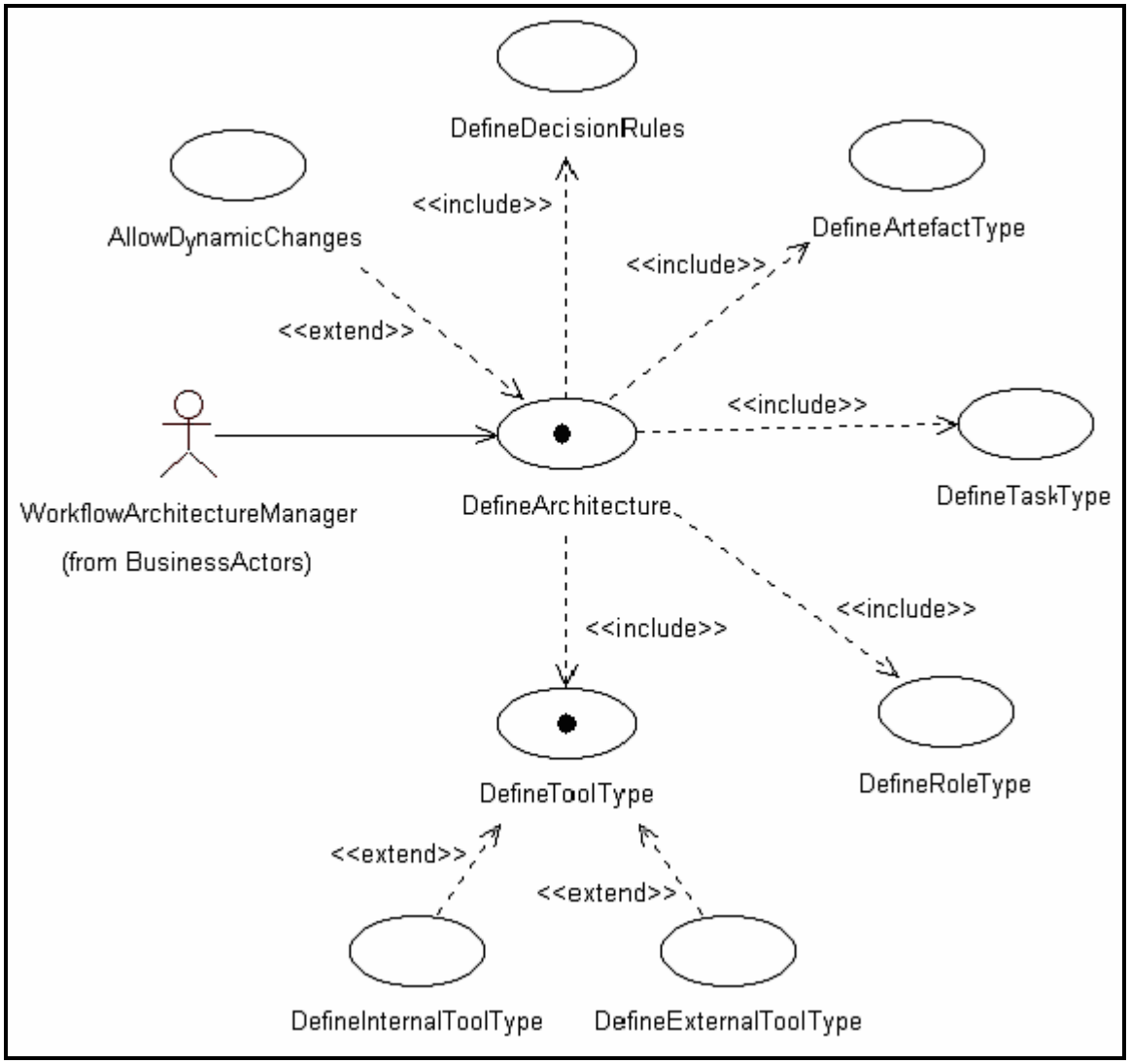

Figure 2: Use case diagram for the Workflow Architecture Manager.

The use cases presented in Figure 3 represent the instantiation of workflow architectures, definition of associated elements and, workflow monitoring and testing. The use case Test Defined Workflow is marked as a variation point to represent the possibilities of applying prototyping, statistics generation or simulation mechanisms. The use cases Allocate Resource and Allocate Tool are also marked to represent the allocation of either internal or external tools.

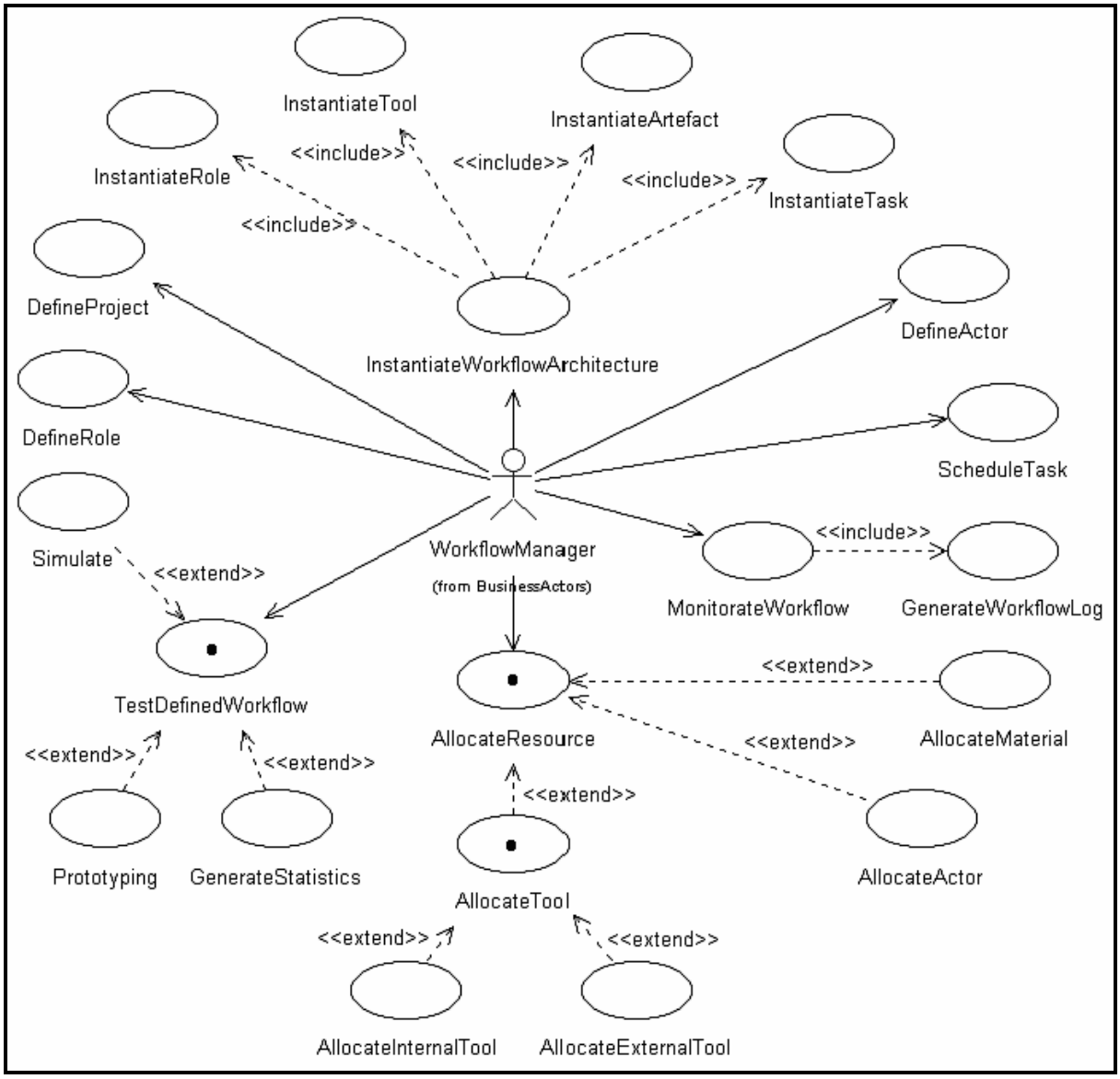

Figure 3: Use case diagram for the Workflow Manager.

Figure 4 represents the actions related to the execution of workflow by the workflow user. He/she can visualise, select, execute, cancel, stop, restart, finalise tasks and communicate with other users. The latter is marked as a variation point to represent communication by teleconference, email or chat.

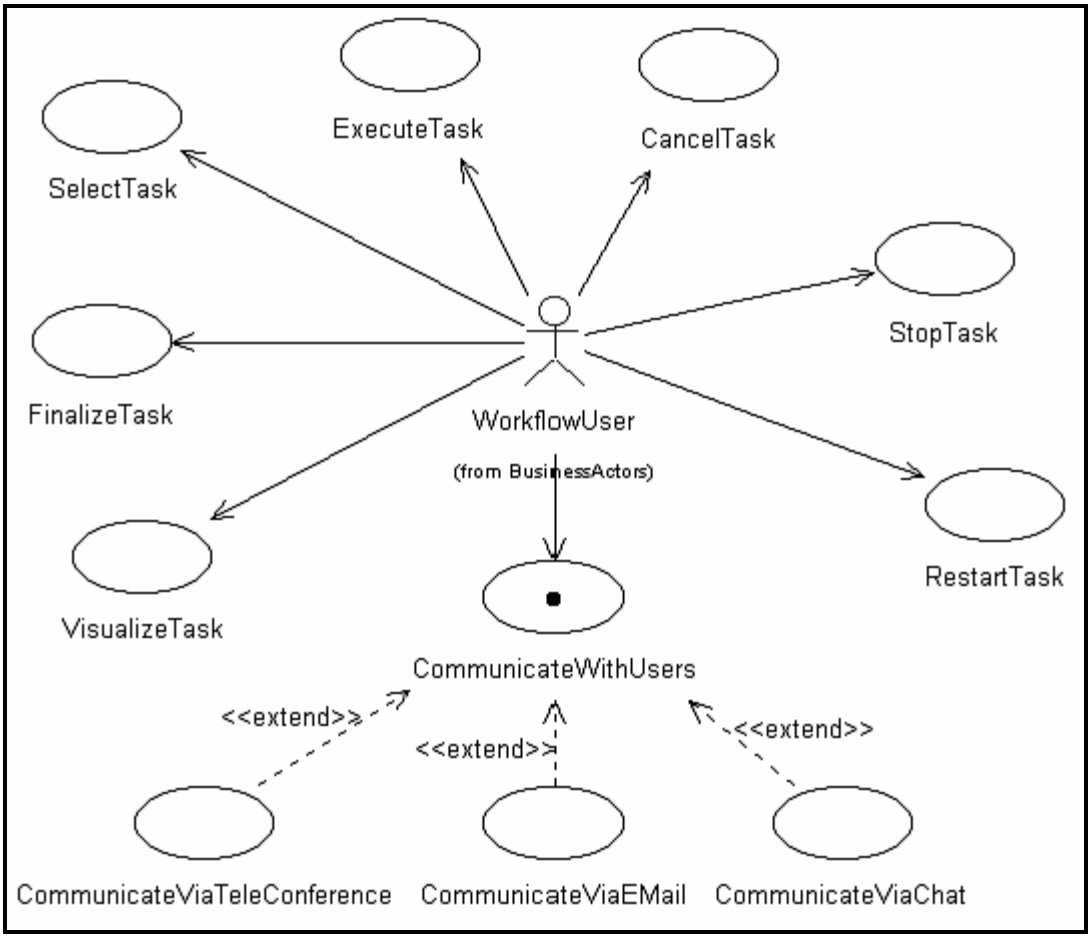

Figure 4: Use case diagram for the Workflow User.

2.2 System Specification

This stage specifies the software solution identified from the domain model. The analysis of the system's actions leads to the identification of the types and related actions. Types, in Catalysis, are specifications of the external view of an object's behaviour; types are represented by a class diagram. A sequence diagram was designed for each use case representing the interaction between objects. The main artefact of this phase is the static type model.

The representation of variation points is carried through to the system specification phase. Figure 5 shows the static type model for WfMS. Morisio [17] extended UML with a variability stereotype, indicated as a «V». This stereotype is related with the concepts of specialization and aggregation. In Figure 5, «V» is used to represent types that can vary according to the product features. For instance, the stereotype «V» is used to represent a variable resource type, TypeResource, which can be specialised to material, actor or tool types of resources. The tool type is also extended to represent internal or external types of tool.

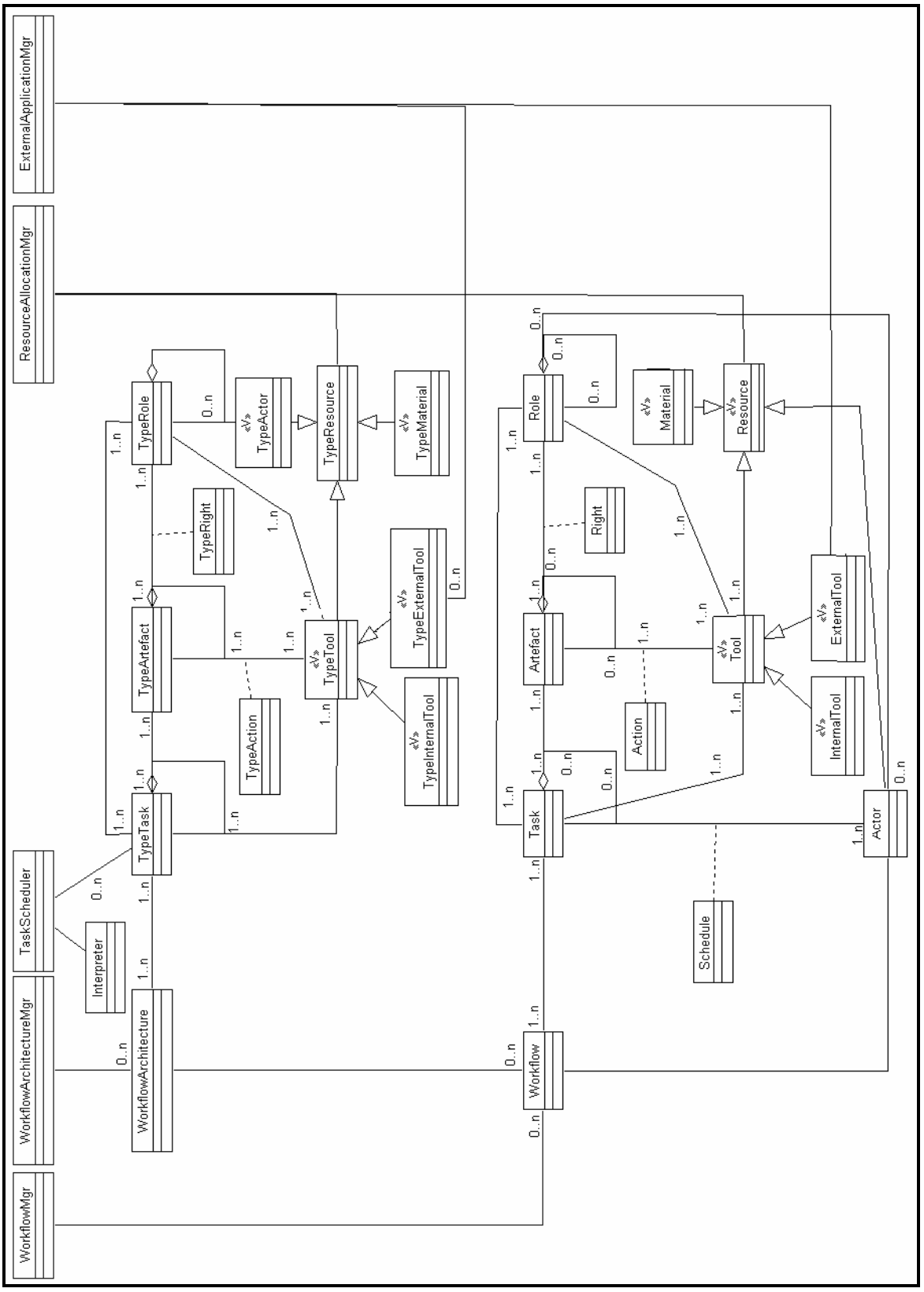

Figure 5: Static type model for WfMS.

2.3 Architectural Design

From the static type model several refinements need to be made to reach the level of components. Catalysis uses packages as a high-level decomposition unit. A package is an independent unit whose relationships to the rest of the system can be well established. These packages can be derived from the static type model and their relationships can be represented as import dependencies. The package partitioning follows the vertical and horizontal slices approach of Catalysis. The vertical slices represent the business level partitioning of the generic architecture according to the actions undertaken by the main actors that interact with the WfMS.

Figure 6 shows the high-level vertical slice diagram, which follows a multi-layer architecture style. The horizontal slices represent the partitioning of the architecture separating higher-level business model packages from the infrastructure packages (e.g. middleware or system software). This partitioning identifies service packages that are shared with higher-level packages. It also aims at reducing the package importation across the layers. The elements of the WfMC generic architecture are represented by the packages: WorkflowArchitectureMgr, ObjectMgr, TaskScheduler, WorkflowExecutionMgr, ResourceAllocationMgr, Interpreter, ExternalApplicationMgr and WorkflowMgr.

The next step in Catalysis is to refine the high-level vertical slice diagram down to the vertical layer diagram that represents the final partitioning of the system. The vertical layer diagram maps the packages of the high-level vertical slice diagram to the types associated with each package. Figure 7 presents the TaskScheduler package and its types. The whole diagram is not shown for lack of space.

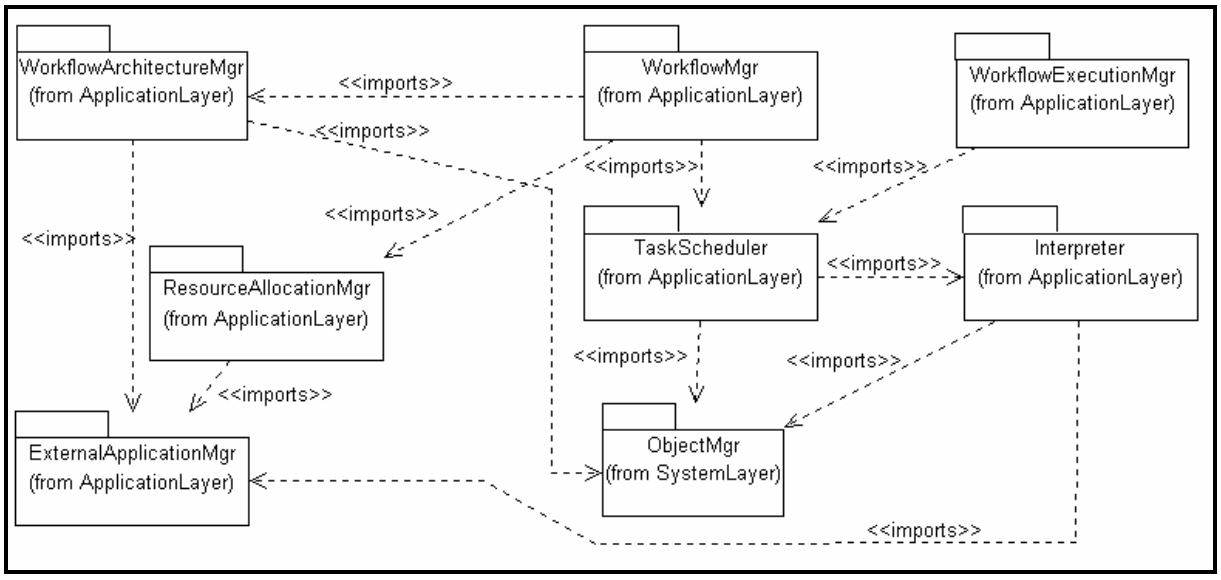

Figure 6: High-level vertical slice diagram.

The TaskScheduler package is responsible for the control and management of tasks and actions to be undertaken in the workflow, such as modifying an artefact using an external tool. Thus, some types that appear in the static type diagram (Figure 5) are associated with this package as shown in Figure 7. Variation points are associated with types Resource and Tool.

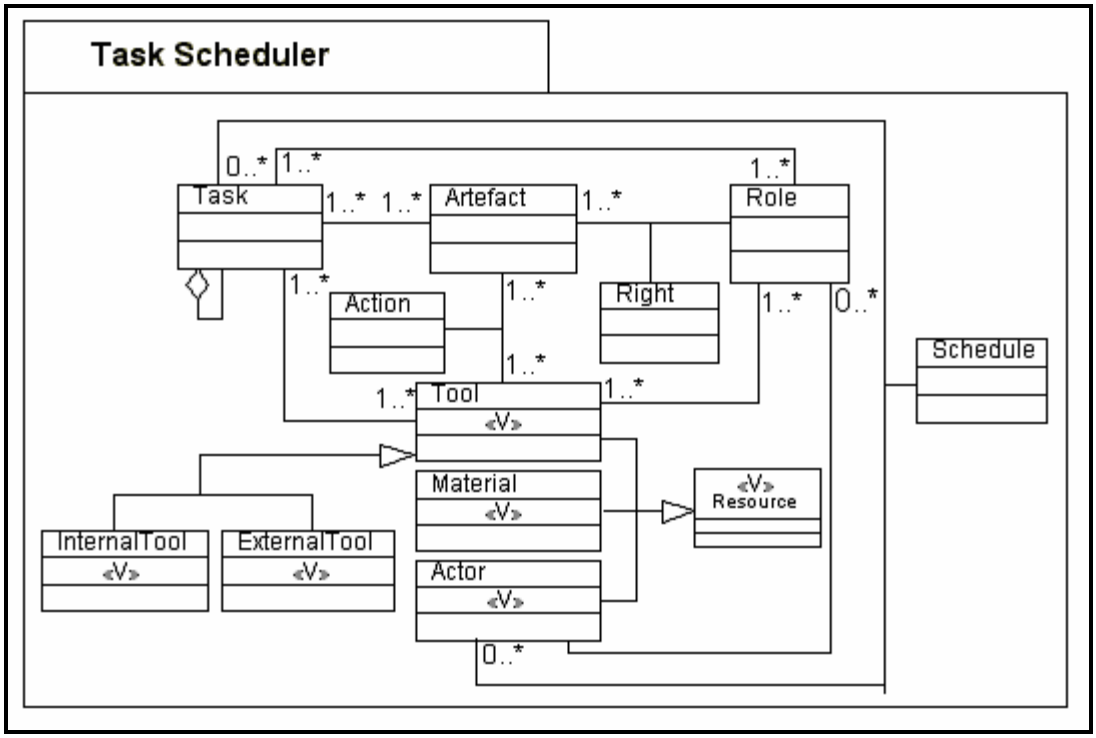

Figure 7: Specification of the TaskScheduler package.

2.3.1 Component Architecture

The vertical layers diagram is the result of the identification and specification of components. The components are represented by the generic packages encompassing their types and relationships. The component architecture designed from the diagram in Figure 6 is shown in Figure 8.

Most of the packages were mapped to components. This is a result of our experience with the domain. Several iterations of this system specification were carried out. Therefore, the specification architecture is closer to the component architecture.

A description of the components and the variabilities associated with this architecture is 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 a 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 as yet.
- ***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).

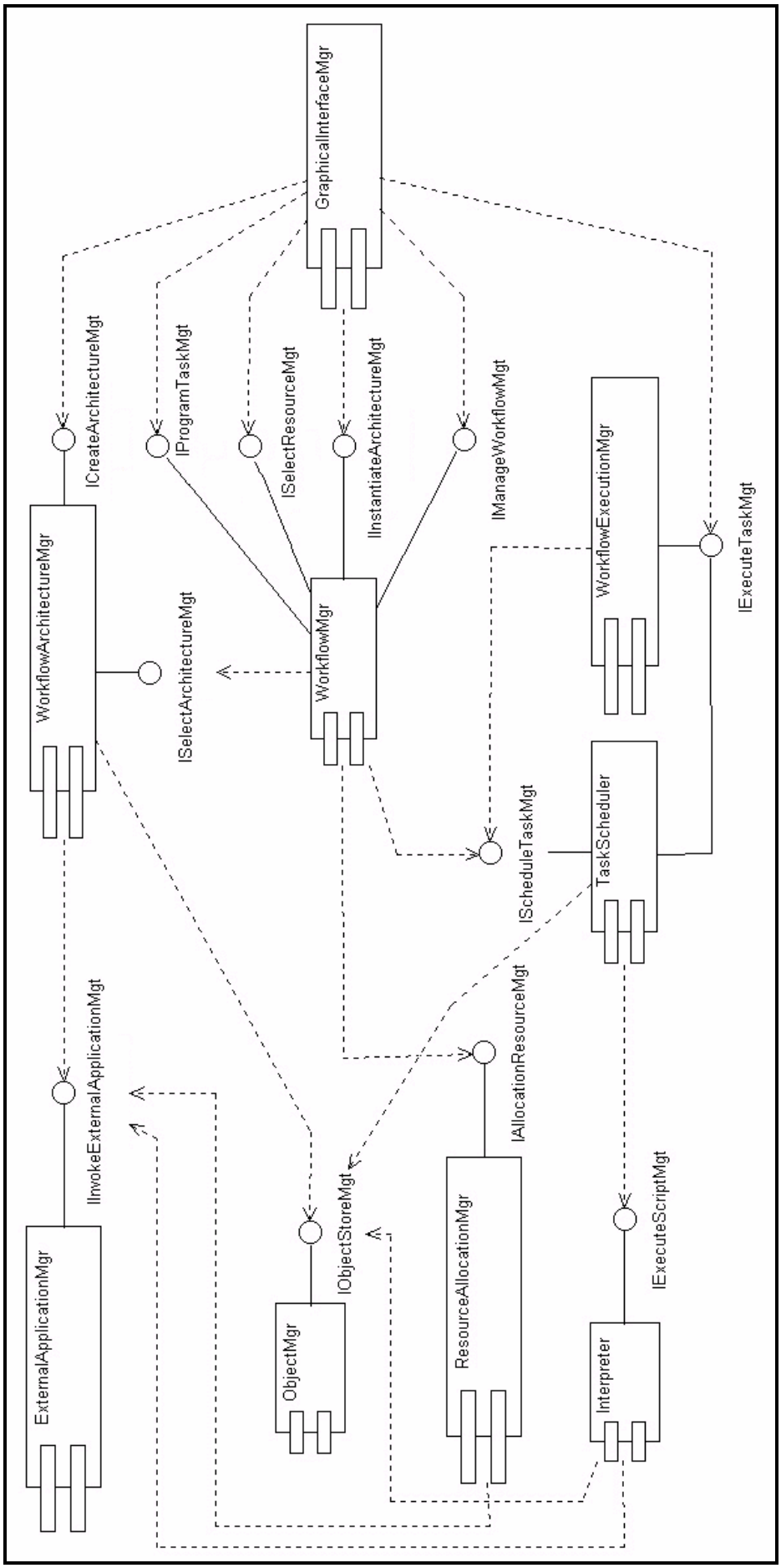

Figure 8: Component architecture for WfMS [19].

- **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 mechanism 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.
- ***Interpreter:*** responsible for the execution of a workflow script written in a process programming language [7].

The following section shows the specification of invariants, preconditions and post-conditions for the TaskScheduler component and the representation of the technical architecture in CORBA IDL.

2.3.2 Specification of Invariants, Precondtions and Post-Conditions

The use of invariants, preconditions and post-conditions are important in this phase. Either a textual description language or a more formal language can be used. As Catalysis suggests, OCL (Object Constraint Language) [20] is used as it allows the specification of well-defined constraints, associated UML, from the beginning of the design process. Figure 9 presents the OCL specification for the methods of the interface IExecuteTaskMgt of the TaskScheduler component.

```
Context IExecuteTaskMgt :: selectTask(task : Task) : Task
pre : -- The task must exist.
  Task.allInstances->includes(task)
post : -- The parameter task is returned as the result of the operation
  result = task;

context IExecuteTaskMgt :: executeTask(task : Task) : Boolean
pre: -- The task must be in state 4 (Ready)
  task.status = 4
post: -- The task is in state 6 (Executing)
  task.status = 6 and
  result = true

context IExecuteTaskMgt :: cancelTask(task : Task) : Boolean
pre: -- The task must be in state 6 (Executing)
  task.status = 6
post: -- The task is in state 9 (Terminated)
  task.status = 9 and
  result = true

context IExecuteTaskMgt :: interruptTask(task : Task) : Boolean
pre: -- The task must be in state 6 (Executing)
  task.status = 6
post: -- The task is in state 5 (Suspended).
  task.status = 5 and
  result = true

context IExecuteTaskMgt :: restartTask(task : Task) : Boolean
pre: -- The task must be in state 5 (Suspended)
  task.status = 5
post: -- The task is in state 6 (Executing)
  task.status = 6 and
  result = true;

context IExecuteTaskMgt :: finalizeTask(task : Task) : Boolean
```

```
pre: --  The task must be in state 8 (finalized).
 task.status = 8
post: -- The task is in state 9 (terminated).
  task.status = 9 and
  result = true;

context IExecuteTaskMgt :: visualizeTask(actor : Actor, role : Role) :
Set(Task)
pre: -- The role name must be "WORKFLOW MANAGER" or "WORKFLOW USER".
  role.roleName = "WORKFLOW MANAGER" or role.roleName = "WORKFLOW USER"
post:
  if role.roleName = "WORKFLOW MANAGER" then
    result = Task.allInstances
  else
    if role.roleName = "WORKFLOW USER" then
       result = Task.allInstances->collect(t: Task|t.Role.includes("WORKFLOW
USER"))
    endif
  endif

context IExecuteTaskMgt :: postponeTask(task : Task, newDate : String, actor
: Actor) : Boolean
pre: -- The task must be in state 6 (Executing) and the new end date is
greater than the old end date
  task.status = 6 and newDdate > task.endDate
post: --
  task.endDate = newDate and
  result = true;
```

Figure 9: OCL specification of the TaskScheduler component.

2.3.3 Technical Architecture

According to Catalysis, the modelling of the logical architecture, as shown in Figure 8, is followed by the design decisions regarding the implementation mechanisms to be used, for instance the middleware. The result is a technical architecture as shown in Figure 10. In this case a CORBA Object Request Broker (ORB) [21] was considered as the middleware.

The components of the architecture have well defined interfaces in the CORBA Interface Definition Language (IDL). These IDL interfaces can be generated from tools such as Rational Rose [22]. The representation of the component interface in IDL allows for the individual implementation of the components in any language, operating system or network. The requested component replies or not according to communication constraints defined in the component architecture.

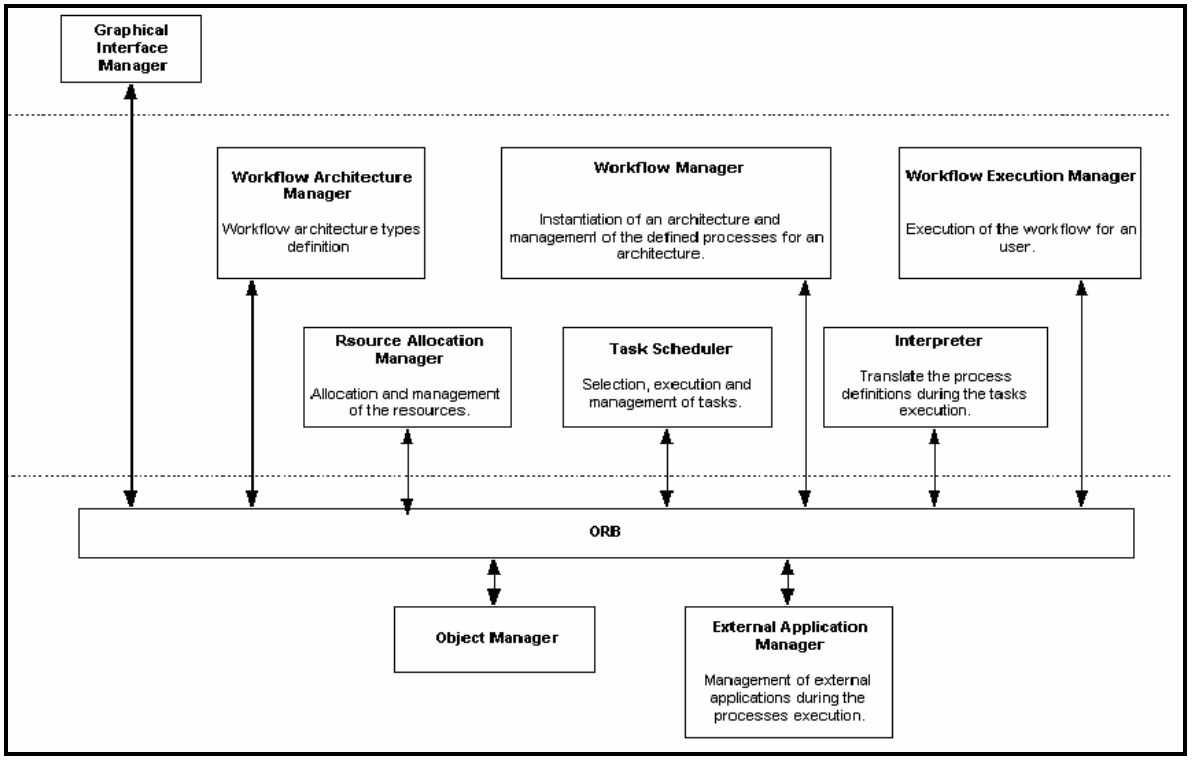

Figure 10: The WfMS product line architecture in CORBA.

Figure 11 presents the CORBA IDL for the TaskScheduler component generated by Rational Rose.

```
#ifndef __TASKSCHEDULER_DEFINED
#define __TASKSCHEDULER_DEFINED

#include "Interpreter.idl"
#include "WorkflowArchitectureMgr.idl"
#include "ResourceAllocationMgr.idl"
#include "WorkflowExecutionMgr.idl"
#include "ExternalApplicationMgr.idl"

module TaskSheduler {
  interface IScheduleTaskMgt {
    boolean requestConnection (String userName, String passWord);
     boolean scheduleTask (Task task, Actor actor, Role role, Workflow
workflow);
  };

  interface IExecuteTaskMgt {
    Task selectTask (Task task);
    boolean executeTask (Task task);
    boolean cancelTask (Task task);
    boolean interruptTask (Task task);
    boolean restartTask (Task task);
    boolean finalizeTask (Task task);
    Task[] visualizeTask (Actor actor, Role role);
    boolean postponeTask (Task task, String newDate, Actor actor);
  };
};

#endif
```

Figure 11: CORBA IDL for the TaskScheduler component.

The communication between components is carried out by the ORB. The requested component answer or not according to the restrictions applied to the requested communication based on the architectural style.

2.4 Architecture Evaluation

Bosch [23] identified four ways of evaluating product line architectures: scenarios, simulation, mathematical models and evaluation based on past experiences. In this section we described the process of defining the proposed architecture. This process is based on Catalysis; Catalysis, however, does not support simulation. Thus, in order to evaluate the proposed architecture without delving into implementation details, we opted for an Architecture Definition Language (ADL). This language is focused on the representation of high-level structures abstracting away from implementation details. The Rapide ADL [10] was chosen to specify the proposed architecture, as it is a general purpose ADL for the modelling of component interfaces and their externally visible behaviour. In addition, there is a support environment that allows the definition of the architecture and its simulation. Figure 12 presents the Rapide code for the TaskScheduler component used to simulate the architecture.

```
TYPE TASK_SCHEDULER is INTERFACE
action
  in Schedule_Task();
  out Insert_Task();
  in Insert_Task_OK();
  out Schedule_Task_OK();
  in Execute_Task1();
  out Select_Task();
  in Select_Task_OK();
  out Execute_Script();
  in Execute_Script_OK();
  out Execute_Task1_OK();

BEHAVIOR
  action animation_Iam (name:string);

BEGIN
  start => animation_Iam("TASK_SCHEDULER");;
        Schedule_Task() => Insert_Task();;
        Insert_Task_OK() => Schedule_Task_OK();;
        Execute_Task1() => Select_Task();;
        Select_Task_OK() => Execute_Script();;
        Execute_Script_OK() => Execute_Task1_OK();;
END;
```

Figure 12: Rapide code for the TaskScheduler component [24].

The simulation was carried out based on the selection of relevant scenarios for WfMS. Sequence diagrams were drawn to represent the interaction of each specific scenario using components in the place of objects. The architecture was executed according to the scenarios to simulate the behaviour of the system. Each scenario represented the view of each WfMS user: workflow architecture manager, workflow manager (supervisor), and workflow user. Once an erroneous message was observed, the sequence diagram was adjusted to correct the components communication and the overall simulation was repeated.

Figure 13 presents a snapshot of the simulation in which the TaskScheduler sends the message Select_Task to the Object Manager component.

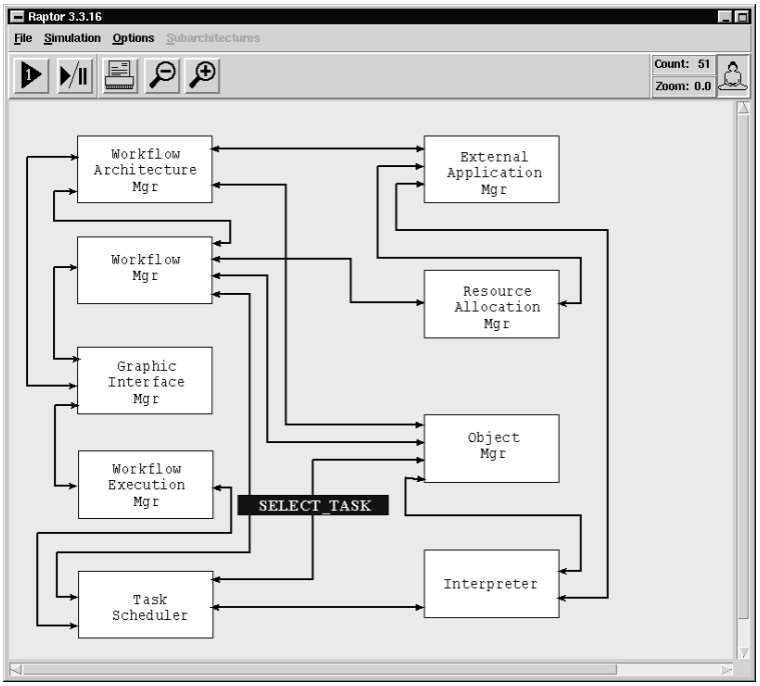

Figure 13: Snapshot of architecture simulation.

The analysis of the dynamic behaviour of the components of the proposed architecture allowed the evaluation of both the communication between components and the general features of the system avoiding implementation issues. This analysis permits the debugging of communication based on scenarios from the use cases. However, it was not sufficient to demonstrate that the architecture is correct. The complete evaluation needs more precise techniques for architecture evaluation [25]. Criteria need to be established to allow derivation of statistical data from the simulation.

3. Component Internal Design

The strategy followed to populate the product line has been to develop each component also based on the Catalysis method. The design of the components has been easier because there was a legacy implementation of a software engineering environment, the ExPSEE [15], which has many similar classes to those of the new components.

In the following subsections we present the mechanisms used to implement the product line and the design of the WorkflowExecutionMgr component.

3.1 Mechanisms used to Implement the Product Line

In order to experiment with the product line an implementation of it has been carried out. The mechanisms used to implement the product line were chosen based on open source tools so that the product line can be broadly used. The mechanisms chosen are as follows [26]:

- **Programming Language**: Java, as it is part of a technology that has many resources to be used in both scientific and commercial applications [27].
- **GUI**: the Swing (Java 2) toolkit [27] and the JHotDraw framework [28]. Swing has a broad set of graphical elements and it is continuously updated. JHotDraw has a very simple API that allows easy implementation of Java applications.
- **Communication services:** CORBA JacORB [29], is an ORB (Object Request Broker) implemented in Java that has many functionalities not identified in similar products.
- **Database Management System:** MySQL [30] together with the ObJectBridge framework [31]. MySQL provides services for the manipulation of persistent data. ObJectBridege was written in Java and allows the mapping of Java Objects to MySQL based on XML (Extensible Markup Language).

3.2 Design of the WorkflowExecutionMgr Component

The WorkflowExecutionMgr was designed by Halmeman [32]. The development of this component affected the overall architecture as presented in Figure 14. As a result, the main focus of the TaskScheduler component is the scheduling of the tasks to be executed by the WorkflowExecutionMgr.

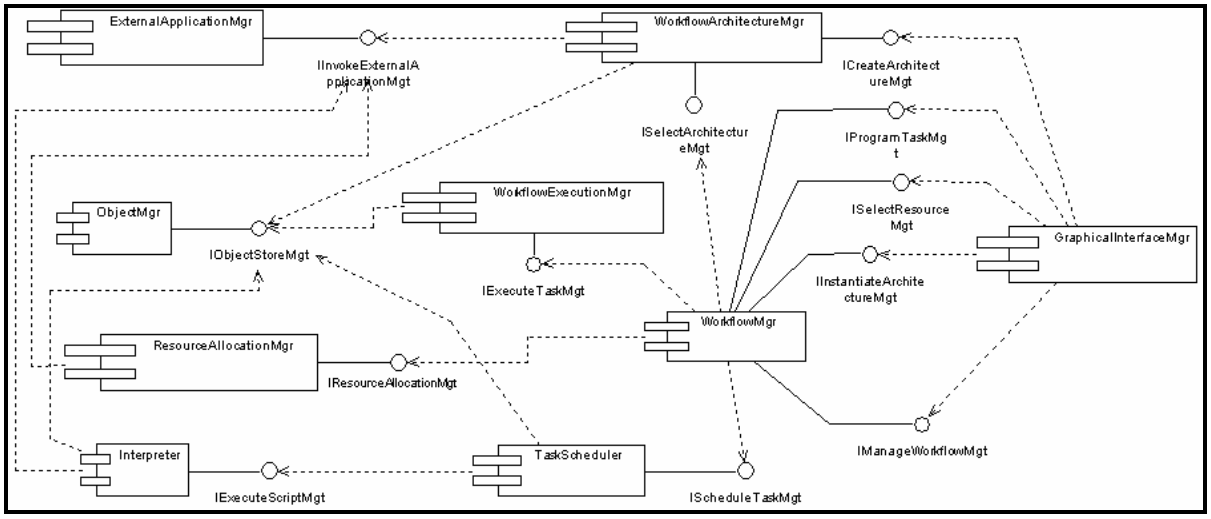

Figure 14: Component architecture for WfMS revised [32].

The WorkflowExecutionMgr focuses on the management of the execution of the tasks from a previously instantiated workflow. The workflow user requests the task execution to the WorkflowExecutionMgr through the WorkflowMgr. As a result the WorkflowExecutionMgr verifies if the precondition is satisfied. If they are satisfied it changes the task state and instantiates the time preconditions and transition managers. When the task is finalised its state is changed to satisfy the postcondition.

Two types of tasks were considered: automatic and manual. The state transitions are the same for both task types. The difference is in the invocation of tasks by the users. The manual tasks are activated by human intervention whereas the automatic tasks are activated by the WorkflowExecutionMgr.

The requirement specification for the WorkflowExecutionMgr component produced both the business and use case models. The system specification phase produced the details of the software solution. Figure 15 presents the component model for the WorkflowExecutionMgr component. The TimeMgr, PreConditionMgr and TaskTransitionMgr classes control the execution time, preconditions and state transitions respectively. The main variation point identified in the WorkflowExecutionMgr component is the possibility of executing different scheduling algorithms. In this case two variations are possible: SerialScheduling or PriorityControlScheduling. The WorkflowExecutionMgr component interface was specified in OCL.

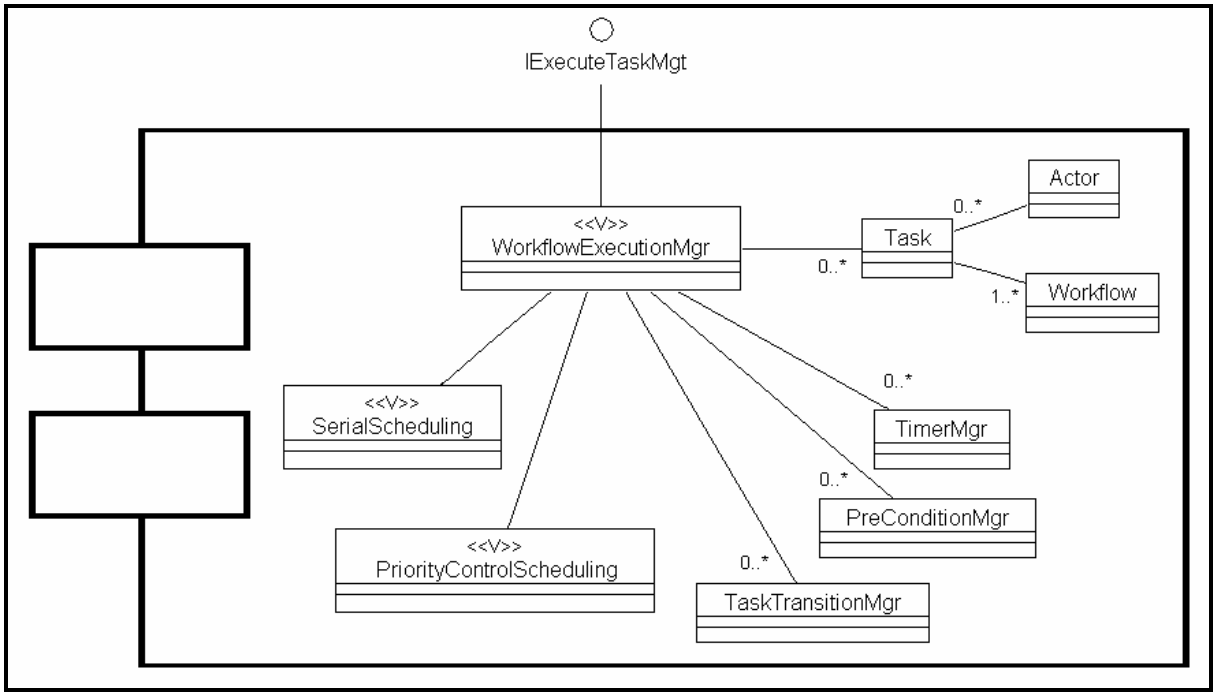

Figure 15: WorkflowExecutionMgr component [32].

In order to automatically decide which algorithm to implement for the component in the experiment carried out, an XML script containing parameters necessary for the variability instantiation was specified. An example of the script for the WorkflowExecutionMgr is shown in Figure 16. This figure describes the XML structure to represent the variability correspondent to the scheduling algorithm used for the component. The tags `<variabilityDescriptor>` and `</variabilityDescriptor>` represent the beginning and end of a variability. The tags `<variability.class>`, `<variability.name>` and `<variability.value>` represent the component name (WorkflowExecutionMgr) to which the variability is associated, the variability identification (SchedulingAlgorithm) and variability decision respectively. In this case a SERIAL schedule was chosen.

```
<!-Variability for WorkflowExecutionMgr -->

   <variabilityDescriptor>
     <variability.class>WorkflowExecutionMgr</variability.class>
     <variability.name>SchedulingAlgorithm</variability.name>
     <variability.value>SERIAL</variability.value>
   </variabilityDescriptor>
```

Figure 16: An example of an XML script for the WorkflowExecutionMgr [32].

In addition to the WorkflowExecutionMgr, the TaskScheduler component had already been developed [33] although not following the product line approach; it had only to be refactored.

4. Related Works

There is not an established and systematic relationship between the techniques for reuse, domain engineering, product line, software architecture and frameworks [34]. They are basically seen as complementary techniques. Well-known methods for domain analysis [5] are used for the identification of concepts and functionalities required for a family of products in order to represent them as a generic model. This model 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. Our approach evolves from previous work both on frameworks and components. Gimenes et al. [18] proposed techniques for defining

frameworks within the WfMS domain that offer guidelines for the architecture design presented in this work. These techniques involve the concept of *model framework* from Catalysis as a base to generate components.

The application of a product line approach to the workflow management system domain, as presented in this paper, is novel. As far as we know there are no previous works that take a similar approach in this domain of application. The product line approach itself presents extensions to previous works. We introduce the use of the Catalysis method [8] to encompass the explicit use of component-based concepts as well as UML based representation to the product line approach. These ideas are also applied in Kobra [12], which has been developed almost in parallel with our work. Kobra is a component-based evolution of Pulse [4].

GenVoca [13] is a mature method based on the concepts of virtual machine, component layers (representing an implementation of the virtual machine) and an architecture-realm (as a set of components). We diverge from GenVoca in that one of the guidelines of our approach is to stay closer to current well-known UML-based software development methods, instead of generating an overall new method, support tools and formalisms. We envision product line concepts incorporated in current successful tool suites in the future [22].

5. Summary

This paper presented the process followed to define a component-based product line for WfMS. Extensions made to represent variability across the process were presented. These extensions were based on Jacobson [16] and Morisio [17].

We argue that CBD can be used in the design of product lines to bridge the gap between domain analysis and the architecture and component design and implementation. General-purpose CBD methods are easy to understand and use. In addition, there are commercial tools that may be used to support them. In this work we have used Catalysis [8] to design the product line component architecture. The domain analysis was carried out based on the generic architecture and reference models for Workflow Management System (WfMS) of the Workflow Management Coalition (WfMC) [7]. The evaluation of the architecture was carried out by its specification in the Rapide ADL and simulation within the Rapide environment. This allowed the evaluation of the architecture based on selected scenarios avoiding implementation details. However, further investigation is needed to extract statistical data from the simulation.

Members of the WfMS family can be generated from the proposed product line by providing their specific requirements. These requirements are used to select and instantiate the variabilities of the component architecture. The complete construction of the product line is a large project. Currently we are populating the architecture by either developing novel components or adapting previously developed components [15]. We are also working on formalising the product generation process.

References

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

[2] Software Productivity Consortium. *Reuse-Driven Software Processes Guidebook.* SPC-92019-CMC version 02.00.03, Nov. 1993.

[3] D. M. Weiss, T. R. L. Chi. *Software Product-Line Engineering: A Family-Based Software Development Approach*. Addison-Wesley, 1999.

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

[5] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, (CMU/SEI-90-TR-21, ADA 235785), Pittsburgh, PA: SEI CMU, 1990.

[6] D. Georgakopoulos, M. F. Hornick, A. P. Sheth. *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. ACM Distributed and Parallel Database, No. 3, p. 119-153, 1995.

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

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

[9] D. C. Luckham, J. J. Kenney, L M. Augustin, J. Vera, D. Bryan, W. Mann. *Specification and Analysis of System Architecture Using Rapide*, IEEE Trans. on Software Engineering, Special Issue on Software Architecture, vol 21, No. 4, pp. 336-355, 1995.

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

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

[12] C. Atkinson, J. Bayer, O. Laitenberger, J. Zettel. *Component-Based Product Line Development: The Kobra Approach*, in 2000 Proc. 1st International Software Product Line Conference. 2000, pp. 289-309.

[13] Batory, D. *Product Line Architectures*. Erfurt, Germany. Smaltalk and Java in Industrie and Ausbildung. 1998.

[14] I. M. S. Gimenes, G. Weiss, E. H. M. Huzita. *Um Padrão para Definição de um Gerenciador de Processos de Software*, in 1999 Proc. II Workshop Ibero Americano de Engenharia de Requisitos Y Ambientes de Software, San José, Costa Rica, Ideas'1999 Memorias, 1999, pp. 30-46.

[15] I. M. S. Gimenes, E. H. M. Huzita, Steinmacher, I. F., Takano, E. T. *ExPSEE – An Experimental Process Centred Software Engineering Environment*, Technical Report, UEM/CTC/DIN, Feb. 2002.

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

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

[18] I. M. S. Gimenes, L. Barroca. *Enterprise Frameworks for Workflow Management Systems*. Software Practice & Experience. No.32, 2002, pp.755-769.

[19] I. M. S. Gimenes, E. A. Oliveira Junior, F. R. Lazilha, L. M. Barroca. *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.

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

[21] Object Management Group. Catalog of OMG CORBA®/IIOP® Specifications. Available: http://www.omg.org/technology/documents/corba_spec_catalog.htm - 2005

[22] Rational Software – http://www-306.ibm.com/software/rational/ - 2005.

[23] J. Bosch. *Design & Use Of Software Architectures. Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[24] F. R. Lazilha, I. M. S. Gimenes, E. A. Oliveira Junior. *Uma Arquitetura de Linha de Produto para Sistemas de Gerenciamento de Workflow de Acordo com a Abordagem de Desenvolvimento Baseado em Componentes*, in 2003 Proc. 3as Jornadas Ibero-Americanas de Engenharia de Software e Engenharia de Conhecimento, Nov. 2003).

[25] P. Clements, R. Kazman, M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley Pub. Company, 2002.

[26] E. A. Oliveira Junior, I. M. S. Gimenes. *Especificação de um Sistema Gerenciador de Workflow de Acordo com a Abordagem de Desenvolvimento Baseado em Componentes*, Revista Eletrônica de Iniciação Científica da SBC (REIC), Porto Alegre-RS, Set. 2003.

[27] Java Technology - http://www.java.sun.com - 2005.

[28] JHotDraw - http://www.jhotdraw.org - 2005.
[29] JacORB - http://www.jacorb.org - 2005.
[30] MySQL - http://dev.mysql.com - 2005.
[31] The Apache DB Project: ObJectRelationalBridge - http://db.apache.org/ojb - 2005.
[32] R. J. Halmeman, I. M. S. Gimenes. *Projeto do Componente Gerenciador de Execução de Workflow Segundo a Abordagem de Linha de Produto de Software.* Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba-PR, julho de 2003.
[33] I. M. S. Gimenes, S. A. Tanaka, J. P. M. Oliveira. *An Object Oriented Framework for Task Scheduling,* in 2000 Proc. TOOLS Europe, vol 1, pp. 383-394.
[34] J. Poulin. *Software Architectures, Product Lines, and DSSAs: Choosen the Appropriate Level of Abstraction*, in Proc. 1997 WISR8.