An Object Oriented Framework for Task Scheduling

Itana M. S. Gimenes
Universidade Estadual de Maringá
Department of Informatics
Joint supervisor of the PPGC-UFRGS
itana@din.uem.br

Sergio A. Tanaka
CESULON
Department of Informatics
PPGC-UFRGS
tanaka@filadelfia.br

José Palazzo M. de Oliveira
Program of Post-Graduation in Computer Science– PPGC
Institute of Informatics
UFRGS
palazzo@inf.ufrgs.br

Abstract
This paper presents an object oriented framework for task scheduling. This framework can be used in domains such as Process-centred Software Engineering Environments, Workflow Management Systems or Project Management Systems. The framework was conceived based on both current methods for the development of frameworks and an existing architectural pattern for process managers. A prototype of the framework was developed using the Java Language. The lessons learnt show the experience of extracting a framework from well-known applications that can be reused in practical domains. Thus, this work contributes not only in the production of a framework but also giving insights in the application of novel techniques to the development of frameworks.

1. Introduction

The objectives of software engineering are centred both on improving the quality of the software process and the quality of the software product itself. In addition, techniques for increasing productivity and reducing costs and production efforts must be taken into account. Frameworks and Components reuse is an important issue in this context. It allows the development of software based on components, frameworks and models (templates) that are already well specified and tested.

Software engineering techniques used in the definition and application of software architectures, frameworks, patterns and components are rapidly evolving. These concepts have already reached the commercial organisations triggering a new market of products.

An object oriented framework is generally characterised as a set of abstract and concrete classes, plus their collaboration relationships, which offer an implementation scheme for applications [8]. The Catalysis method [2] proposes a wider concept of frameworks, called model frameworks. Model frameworks are designed at a higher level of abstraction establishing a generic scheme that can be imported, at the design level, with substitutions and extensions in order to generate specific applications. This article follows this approach presenting a framework for workflow applications.

Workflow technology [6] 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 participant users (the human agents), and, when necessary they invoke tools and applications to execute parts of the workflow [16, 17]. A software process is composed of an ordered set of tasks. PSEE (Process-centred Software Engineering Environments) are environments that support the management and automation of software processes. The software process can be seen as a workflow. WfMS and PSEE have many similar features.

This paper presents a framework for task scheduling that can be reused in applications such as PSEE, WfMS, or any application which involves some form of task management (ex. building or course management). Section 2 presents the domain analysis of the proposed framework. Section 3 and 4 describe the definition and the design of the framework for task scheduling respectively. Section 5 discusses the prototyping and validation of the framework. Finally, section 5, presents the conclusions and future work.

2. Domain Analysis of the Proposed Framework

The domain analysis was carried out taking a similar approach to the Example Driven Project[7] where frameworks are identified using application examples and are generalised in a iterative process. The main application examples explored for the domain analysis were WfMS and PSEE. Although these applications are distinct, they have many common elements. Thus, the objective of the domain analysis is to identity these common elements as they constitute potential frameworks.

2.1. The Process Manager Pattern

The Process Manager is an architectural pattern for definition of PSEE process managers [4]. It was developed from both studies of existing environments [3] and from experiences obtained in the development of the ExPSEE (Experimental Process-centred Software Engineering Environment) project (funded by CNPq - a Brazilian research funding agency). The pattern is based on a process model that allows the definition and reuse of process architectures. It is described according to Buschmann [1], using UML (Unified Modelling Language) [12, 13] to represent the diagrams that compose it. Figure 1 shows the main structure of the pattern through its class diagram.

The top part of the diagram represents the classes correspondent to the modules of the process manager. The central and inferior parts of the diagram were defined to support process architecture reuse. The central part involves the definition of the process architecture and the object types related to it whereas the inferior part represents the software process instantiation, according to the defined architecture, and its related objects.

2.2. The WfMC Reference Model

The WfMS were studied based on the WfMC (Workflow Management Coalition) reference models[17]. These models include a generic architecture for WfMS, as shown in figure 2, and a reference model which encompasses 5 categories of interoperability and communication standards. This architecture contains the main components and interfaces that a WfMS should have in order to allow interoperability of subproducts from different suppliers, such as process definition tools.
Figure 1. Class diagram of the Process Manager pattern [4].

Figure 2. Generic architecture for a WfMS [17].
2.3 Comparative Synthesis

The class diagram of the Process Manager, shown in figure 1, is more detailed as it was taken from the specification of the ExPSEE [4, 5]. In contrast, the models of the WfMC are generic architectures which represent a reference model composed of large and abstract blocks.

Table 1 summarises the comparison undertaken between WfMS and PSEE, based on the WfMC generic architecture and the Process Manager pattern. The table rows indicate the equivalence between the common elements of both models. The number of common elements and functionalities of these systems provide evidences that the Process Manager pattern can be refined in order to achieve small frameworks that can be used in WfMS, PSEE and even extended for related applications. The identification of these frameworks represents a significant contribution to facilitate the development of this category of systems.

According to the Taligent approach [15] for framework development, small and more specific frameworks for each main function of an application or domain can be produced based on a more generic framework. In the case of the process management domain dealt with in this paper, we may have frameworks for task management, artifact management, actor management and so on. These smaller frameworks have more chances to be reused in other contexts. This paper defines a framework for task scheduling with this feature. This is an important contribution in the area as it explores the process of identification and generalisation of frameworks aiming at increasing their reuse potential.

Table 1. Comparison between WfMC reference models and the Process Manager pattern.

<table>
<thead>
<tr>
<th>WfMC Generic Architecture Elements</th>
<th>Pattern Structure Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFM Engine</td>
<td>Process Manager</td>
</tr>
<tr>
<td>Process Definition Tools</td>
<td>Process Architecture Manager</td>
</tr>
<tr>
<td>Worklist Handler</td>
<td>Task Manager</td>
</tr>
<tr>
<td>Workflow Control Data and Relevant Workflow Data</td>
<td>Artifact Manager</td>
</tr>
<tr>
<td>Applications</td>
<td>Tool Manager</td>
</tr>
<tr>
<td>Workflow Enactment Services</td>
<td>Tool Manager</td>
</tr>
<tr>
<td>Administration and Monitoring Tools</td>
<td>Actor Manager</td>
</tr>
<tr>
<td>Organisation Role Model</td>
<td>Role Manager</td>
</tr>
</tbody>
</table>

The detailed comparison undertaken between the WfMS and the PSEE components indicated that the Worklist Handler of the WfMC generic architecture and the Task Manager of the Process Manager pattern have great similarity regarding the functionalities for the management and the execution of tasks. Thus, we have focus our attention in the development of a framework for task scheduling as described in the following sections.

3. Definition of the Task Scheduling Framework

The domain analysis showed that the structure of the Process Manager pattern contains many of the types, actions and collaborations that constitute the framework for task scheduling. Thus, the pattern structure was chosen as a reference for the design of the framework.

After the domain analysis, a package partitioning based on the Catalysis method [2] was carried out. Other methods for framework development also influenced the framework development such as Taligent[15]. This approach is interesting, intuitive and emphasises the idea of designing small frameworks, but a consistent notation does not support it. Thus, Catalysis proved to be better defined and more complete. The Catalysis method uses UML, and is package partitioning and model framework approaches, are key techniques in the
identification and design of frameworks. In addition, the method takes into account the reuse of patterns, the extensibility of diagrams and the interactions between diagrams.

The partitioning of the Process Manager structure was based on the class diagram presented in figure 1 and the use cases developed within the ExPSEE project. The package separates the work in areas that can be treated individually with explicit dependencies. The package partitioning helps to control the propagation of modifications thus allowing traceability and reducing maintenance costs. The contents of the packages represent the business rules that give a general definition of the target package. In our work we have mainly used the following partitioning approaches of Catalysis: vertical slices, horizontal slices and different domains.

The first step of the package partitioning is to analyse the existing dependencies in the classes (types, in Catalysis) diagram. The next step consists of partitioning the packages keeping a structure that facilitates and reduces the number of package importation.

Figure 3 presents the vertical slices, at a high level, of the Process Manager obtained by partitioning the system according to the actions undertaken by the main agents which interact with the environment. The agents were derived from the general use cases identified in the ExPSEE project. The vertical slices are: Defining Process Architecture, Instantiating Process Architecture, Defining and Allocating Resources and Scheduling Tasks.

The vertical slices diagram also contains a basic package called Managing the Elements of the Process, which contains the managers of the basic elements of the PSEE. This package is imported by the others throughout the horizontal layers. There are two intermediate horizontal layers: the Managing Process Architecture and the Managing Projects.

Throughout the partitioning process we could clearly isolate the potential for reuse of the framework for task scheduling from the package called Scheduling Tasks, thus, identifying its types, actions and collaborations. The model framework proposed represents the scheduling of tasks used in applications such as PSEE, WIMS and project management systems.

![Figure 3. Vertical slices from a common base of the Process Manager.](image-url)
4. The Design of the Framework for Task Scheduling

This section presents the use cases, the design of the framework, the model of its generic package and the application of the framework in different domains.

4.1 Use Cases

The use cases are presented in two figures, followed by their respective description tables, to facilitate their explanation. The first set of use cases is presented in figure 4 and its description in table 2. This use case has a main actor, the Project Manager, who interacts directly with the task scheduling framework.

![Figure 4. Use case diagram for task scheduling.](image)

The stereotyped relationships <<include>> and <<extend>>, defined in UML, and also considered in [2, 14] are applied to the use cases. The <<extend>> is used to represent the dependency relationship between Schedule Tasks and Visualise Scheduled Tasks. The <<include>> is applied to represent the fragmentation of the use cases. For instance, Visualise Scheduled Tasks is split into Postpone Task, Update Status, Cancel Task and Set Progression Rate.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Who starts the action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualise Scheduled Tasks</td>
<td>Project Manager</td>
<td>Visualisation of all scheduled tasks.</td>
</tr>
<tr>
<td>Cancel Task</td>
<td>Project Manager</td>
<td>Cancellation of previously defined tasks.</td>
</tr>
<tr>
<td>Update Status</td>
<td>Project Manager</td>
<td>Updating of the task status.</td>
</tr>
<tr>
<td>Postpone Tasks</td>
<td>Project Manager</td>
<td>Postponing the initial or final date of a task.</td>
</tr>
<tr>
<td>Set Progression Rate</td>
<td>Project Manager</td>
<td>Set the progression rate for a certain task, ex.: (0%) allocated, (25%, 50%, 75%) executed.</td>
</tr>
<tr>
<td>Schedule Tasks</td>
<td>Project Manager</td>
<td>Scheduling of a task to an actor who plays a role in a project.</td>
</tr>
<tr>
<td>Define Actor</td>
<td>Project Manager</td>
<td>Definition of a new actor.</td>
</tr>
<tr>
<td>Define Role</td>
<td>Project Manager</td>
<td>Definition of a new role.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Who starts the action</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Define Project</td>
<td>Project Manager</td>
<td>Definition of a new project.</td>
</tr>
<tr>
<td>Define Task</td>
<td>Project Manager</td>
<td>Definition of a new task.</td>
</tr>
<tr>
<td>Define Status</td>
<td>Project Manager</td>
<td>Set the status of a certain task, ex.: allocated, ready, in execution, stopped or finished.</td>
</tr>
<tr>
<td>Select Actor</td>
<td>Project Manager</td>
<td>Selection of a previously registered actor.</td>
</tr>
<tr>
<td>Select Role</td>
<td>Project Manager</td>
<td>Selection of a previously registered role.</td>
</tr>
<tr>
<td>Select Project</td>
<td>Project Manager</td>
<td>Selection of a previously registered project.</td>
</tr>
<tr>
<td>Select Task</td>
<td>Project Manager</td>
<td>Selection of a previously registered task.</td>
</tr>
</tbody>
</table>

Figure 5, followed by table 3, describes the second use case diagram of the framework for task scheduling. In addition to the actor project manager, this diagram has the actor software engineer who can only access some of the functionalities presented in the use case Visualise Personal Schedule.

![Use case diagram for personal task visualisation.](image)

**Table 3. Description of the use case diagram for Individual Task Visualisation.**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Who starts the action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualise Personal Schedule</td>
<td>Project Manager or Software Engineer</td>
<td>Visualisation of all tasks scheduled for an actor who plays a role in a certain project.</td>
</tr>
<tr>
<td>Visualise Project Status</td>
<td>Project Manager or Software Engineer</td>
<td>Visualisation of the status of a certain project including number of tasks of the projects, number of completed tasks and expected project conclusion date.</td>
</tr>
<tr>
<td>Request Task Change</td>
<td>Project Manager or Software Engineer</td>
<td>Request a task change from a certain actor.</td>
</tr>
<tr>
<td>Select Tasks to Execute</td>
<td>Project Manager or Software Engineer</td>
<td>Selection of the tasks to be executed.</td>
</tr>
</tbody>
</table>

4.2. The Model Framework

According to the Catalysis method, not only pieces of code can be reused but also specifications and designs. The set of types, relationships and constraints that are specified within a package can be seen as a framework. Frameworks are represented through a
generic package called model framework or template package. The framework defined in
this paper is a model framework.

The framework for task scheduling is shown in figure 6. The package diagram of the
model framework was developed from the use cases described in the previous section and
the types, actions and collaborations derived from the Process Manager pattern. The figure
shows the interaction between the use cases and the package that is a characteristic of the
Catalysis model. A type defines the object by specifying its externally visible behaviour.
The concept of type in Catalysis is different from the class one that describes the
implementation of an object. A type is more abstract and does not prescribe an
implementation.

The types with names written in brackets are defined as placeholders. These types can be
substituted in the specific application. The concept is similar to the extensibility of classes
of the object oriented paradigm. There are also no placeholders types, such as the
FrPassword which is used by the <FrIntSchedule>, represented by a dotted line. The type
<FrIntSchedule> (the Schedule Interface) is related to <FrSchedule> by an aggregation.
The <FrSchedule> is an association between <FrActor> and <FrTask>. It has attributes and
methods through which it can reference the types <FrActor>, <FrRole>, <FrProject> and
<FrTask>. The <FrTask> is related to itself by an aggregation, indicating that one task may
be composed of several tasks.

The use cases are represented by ellipses that define the actor responsible for the action
execution. The use case diagrams as described in figures 4 and 5 represent the detailed
actions. The main actions of the use cases are represented in figure 6: Visualise Personal
Schedule, Schedule Task and Visualise Scheduled Tasks.

Figure 6: Generic framework package diagram for the task scheduling.

Figure 7 shows the application of the framework for task scheduling in the ExPSEE
context, according to the Catalysis notation. In this figure we can see the application of the
types defined as placeholders. The application of the framework is carried out by importing
the placeholders with substitutions. This is represented by labelled arrows from the package
to each substitution. It can be observed that the placeholders are substituted by the
correspondent classes of the ExPSEE Process Manager presented in figure 1.

![Figure 7: Application of the Task Scheduling Framework in the ExPSEE context.](image)

The framework for task scheduling can be reused in several domains. Figure 8a shows
the application of the framework in building management whereas figure 8b shows the
application of the framework in the control of subjects offered in a course. In the first
application, the types <FrProject>, <FrRole>, <FrTask>, <FrActor>, <FrSchedule> and
<FrIntSchedule> are replaced by the classes: Building, Role, Task, Person, Schedule and
BuildingSchedule. In the course control application the same types are substituted by
Course, Teacher, Subjects, Person, Schedule and SubjectSchedule.

![Figure 8a: Application of the framework in the building management domain.](image)

![Figure 8b: Application of the framework in the scheduling of subjects in course management applications](image)

The application of a model framework is further unfolded to represent the complete
model of the specific application. The unfolding includes the substitutions in the target
context as well as additional implementation classes.

A case study was carried out where the framework for task scheduling was unfolded,
down to the level of implementation, considering the ExPSEE context. This unfolding,
shown in figure 9, contains in addition to the types presented in figure 6, implementation
classes. For instance, the class FrSchedule_Task contains the actions for personal scheduling of a certain actor, role, task and project as well as it allows the insertion of new registers. The class FrVisualise represents the visualisation of the tasks from a certain actor who works in a given project playing an established role. The class Request was added to represent the requests made by the software engineers to the project manager, informing the problem description.

The class ExPSEESchedule is the result of the substitution and specialisation of the type <FrIntSchedule>. This class represents the visual interface through which the project manager and the software engineers visualise all the scheduled tasks.

Figure 9: Unfolding of the application of the task scheduling framework in the ExPSEE context.

5. Prototyping and Validating the Framework

A prototype was developed to validate the framework for task scheduling. The prototype was implemented in the Java Language and used the Microsoft Access 97 [9] to manage the database. The task scheduling prototype contains the visual interfaces through which the actors may select the tasks to be executed as well to obtain information about their tasks. Figure 10 shows the interface Visualise Scheduled Tasks. In addition to the tasks, the interface shows attributes such as: start and end date, root task, progression rate and task status. The interface also provides buttons to start actions such as apply, cancel, schedule, remove, postpone, visualise status and updates, and exit. These actions correspond to the use cases presented in figure 4. When the option Visualise is chosen, a password is
requested from the actor. Afterwards, the interface shows the actor schedule that contains the tasks that he/she can execute. This corresponds to the software engineer use cases presented in figure 5.

![Figure 10: Interface of the framework for task scheduling.](image)

The connection of a framework with the specific applications is made through plug points. In the prototype, the plug ins were implemented by class extensions.

The lessons learnt with the development of the task scheduling framework considering the ExPSEE context forced a re-evaluation of the implementation of the ExPSEE so that its architecture become more flexible, thus allowing the reuse of other frameworks.

6. Conclusions

This work shows the similarities between the reference models of WfMC and PSEE. The comparative analysis of these systems indicates that it is possible to define generic frameworks that can be reused in these domains. Thus, this paper proposes a framework for task scheduling, which can be used in these domains or related applications such as task scheduling in building or course management, as well as it shows the process of defining the framework.

The Process Manager pattern was used as a basis to derive the proposed framework. The main development method used was Catalysis. The method was used due to its emphasis in factoring and composing frameworks and components. The package partitioning of Catalysis and the model framework proved to be very useful. The package partitioning is important to reason about how to refactor the system respecting the users category views and structuring the system into horizontal layers. The model framework approach enables the design of framework abstracting away from any implementation details. In addition, the model framework can be used as a design template to generate components for more specific applications. This facilitates the production of components for any domain.

The use of an architectural pattern was important as it gave the main structure from which the types, actions and collaborations of the framework were derived. Architectural patterns provide generic structures that can be used to derive small and more specific
frameworks. These smaller frameworks are usually less complex and easier to understand. Therefore, they have higher potential for reuse.

The development of frameworks requires time, experience and techniques that ranges from the domain analysis to the implementation and use. In the development of the framework for task scheduling the experience in the previous development of the ExPSEE and the Process Manager pattern proved to be a key issue.

The possibility of reuse this framework in several domains was emphasised in sections 2 and 4. Further work will be concerned with applications of task scheduling in WfMS to manage courses. This work will take place in the context of the Tapejara Project [10] (a cooperative project funded by CNPq/PROTEM-CC) which aims at developing an environment for distance learning, training and education supported by the workflow technology. In this environment the task scheduling is a key issue to obtain successful results of the courses. This work will allow us to make an even deeper analysis of the similarities and adaptations necessary to the application of the framework proposed in this paper.

References