SADIO Electronic Journal of Informatics and Operations Research

http://www.dc.uba.ar/sadio/ejs

vol. 3, no. 1, pp. 23-32 (2000)

Distributed Task Management by Means of Workflow Atoms

Federico García[1] Roberto Moriyón[1]

[1]Escuela Técnica Superior de Informática
Universidad Autónoma de Madrid
28049 Madrid (Spain)
{Federico.Garcia, Roberto Moriyon}@ii.uam.es

Abstract

In this paper we describe Wf-ATOMS, a framework for the specification and management of workflows, whose engine is integrated in a multi user and distributed task management system. The process models include features standard in other workflow management systems, concerning the form in which the activities forming the processes interact, and other features usually managed by user-task management systems that model the different activities available in interactive applications. The conjunction of these two models provides several benefits. On one hand, there is a simplification in the development of workflow-based applications. On the other hand, it allows the systematic development of training applications for work teams that collaborate in the accomplishment of distributed processes. In this paper we describe both the framework from the point of view of the specification of distributed processes and the underlying architecture of the process management system. Wf-ATOMS has been developed as an extension of ATOMS, a previous framework for the management of user tasks in interactive applications.

Keywords: Workflow, User-Task Models, Distributed Tasks, Client/Server

1. Introduction

Workflows are sets of tasks that are executed either in a computer or through specific methods by process entities, which can be people or computer applications. Tasks in a workflow must satisfy constraints that are specified in the corresponding process models. Workflows represent procedural knowledge related to business processes, administrative processes, etc., that take place in any type of organizations. Usually, the most common activities that take place in organizations are suitable for their inclusion in a workflow. Depending on the type of activities developed by the organization, the percentage of activities that can be included in a workflow can be very high.

During the last years, many computer-based systems for the automatic control of workflows have appeared. Several approaches for the modeling of workflows have been proposed, depending on the predominant technique in their development (Alonso 1996). We shall point out those based on data base transactions (Kamath 1995), events (Weske 1996) and Petri nets (Van der Aalst 1994) as the most relevant ones. Among the benefits of using a workflow management system, the ability to automate very different aspects of the activities that are usually accomplished in the organization is especially important. Some of these aspects are assigning work to users, the information flow among them, the control of timing and priorities of pending activities, and parts of the execution of those activities.

This paper introduces Wf-ATOMS, a framework for the specification and management of workflows whose model management engine is integrated in a distributed multi-user task management system for interactive applications. Wf-ATOMS manages tasks to be assigned to users. These tasks form a hierarchy of composed and basic tasks. The basic ones can be task assignments to other users or simple interactive tasks like clicking on the mouse on a button in the screen or moving a graphic component in a window with the mouse. As a consequence of the granularity of basic tasks, Wf-ATOMS allows a bigger control of user activities than usual workflow systems do. At the same time, the inclusion among them of the possibility to start remote tasks simplifies the development of applications that must be highly integrated with the workflow management system, like monitoring applications for different kinds of processes.

Wf-ATOMS offers a framework for the creation of workflows. This framework isolates the designer from the management of different instances of workflows, and also from the assignment of pending tasks. Wf-ATOMS allows the specification by means of a process editor of workflow processes formed by several activities that are executed taking into account different temporal relations among them. The main management engine takes care of task assignment. Moreover, Wf-ATOMS also allows the design of specific applications that are part of some process or kind of processes, and are able to create and assign specific tasks to some users or groups of them. The use of a highly dynamic underlying object system, ORE (Myers 1997), based on the prototype-instance paradigm, is essential for the ability to build new prototypes of tasks with different structures on the fly. The ability to use the Wf-ATOMS task framework is another factor that simplifies the creation of applications that interact with the workflow system in a specific way.

The rest of this paper is organized as follows: first, we describe two different pieces of work that are related to the Wf-ATOMS framework. After this, the framework for the specification of Distributed Process Models is explained. The next section describes the overall architecture that is in charge of the management of the distributed process in our system. Finally, some conclusions are given, together with comments on the perspectives of future work.

2. Related Work

In this section we shall introduce two previous experiences of a completely different nature. First, we shall give a brief description of the essential aspects of the *Workflow Reference Model*, defined by the *Workflow Management Coalition*, WfMC. In the second part of the section some aspects of the In∇esFlow workflow management system prototype will be explained. That prototype has been the basis for the In∇esFlow product, that is commercialized at international level by the first Spanish software corporation, Informática El Corte Inglés SA, IECISA. The In∇esFlow prototype was developed as part of a join project between IECISA and the Group of Interactive Tools and Applications, GHIA, from the Universidad Autónoma de Madrid.

The Workflow Management coalition was created in 1993 by several developers of workflow management systems. Its main goal is to encourage the use of workflow technology through the creation of standards that allow interoperability and connectivity among this kind of products. The standardization work has been centered on the Workflow Reference Model, WfRM (Hollinsworth 1994). The main goal of the WfRM is the

specification of a general context for workflow systems, by identifying their characteristics, functions, and interfaces.

Figure 1 shows a scheme of the main components of the WfRM; four of them are relevant for this work: *Workflow Enactment Service*, *Workflow Client Applications*, *Invoked applications*, and *Management and Monitoring Tools*. In the next paragraphs we shall describe these components briefly.

As Figure 1 shows, the main component of the WfRM is the *Workflow Enactment Service*, which is enabled for the creation, management and execution of instances of workflow processes. The Workflow Execution Service can include one or more *Workflow Engines*, which constitute the basic operation environment. In case that several engines coexist, process executions are distributed among them according with a specific partition logic (for example, depending on the type of workflow or the type of process). *Workflow Client Applications* are the interface between the final user and the *Workflow Execution Service*, through what usually is known as the *Work List Manipulator*. This manipulator allows users to select tasks, to ask for details about the work to be accomplished, and to invoke applications needed in order to accomplish that work, among other things. *Invoked applications* are a broad range of preexisting services from heterogeneous environments that are integrated into the workflows. Automatic invocation activities and applications that are able to interact at the workflow level, that are developed specifically to interoperate with the *Workflow Execution Service*, are also included in this component. Finally, the *Management and Monitoring Tools* allow the management of the whole *Workflow Enactment Service* from a single tool.

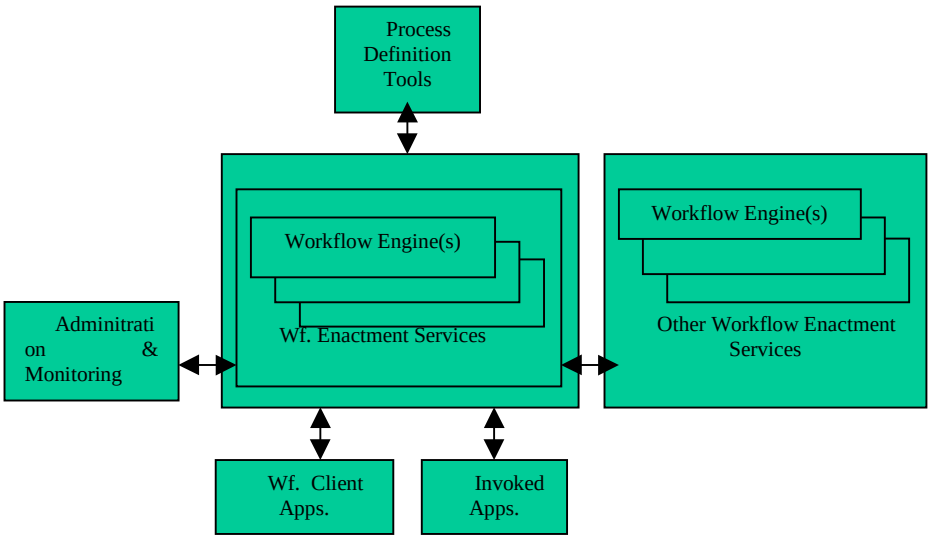


Figure 1: The WfMC's Workflow Reference Model.

After this brief description of the WfRM, we shall describe the essential aspects of the prototype of *Workflow Management System* developed by the GHIA group as part of the design of In∇esFlow. This work has been conducted in the context of the continued collaboration between the IECISA R+D department and the GHIA group since 1995. This collaboration continues nowadays with the participation in join R+D projects that include areas such as the development of last generation techniques for the management of Java distributed processes (Apolo project) and the design and development of systems for the management and resolution of problems based on constraints and their integration in high level tools that support negotiation processes (Logap project).

The In∇esFlow prototype (Alfonseca 1997) uses a mixed approach based on the technology of application interface modeling based on user-task models, together with a classical client-server communication mechanism. This communication mechanism includes the use of transactions on a centralized relational database that supports the persistency of all the objects handled by the system. Using user task modeling techniques has some advantages with respect to other workflow modeling techniques. On one hand, it allows a representation of the processes that form the workflow in terms that are close to the ones used by their

protagonists; this simplifies their design and modification, and allows a bigger collaboration of the users in both activities. On the other hand, the representation of the information flow is simpler to comprehend and to handle, since the information appears in the form of data associated to objects (usually the representation of activities being accomplished). This allows an important simplification with respect to other approaches, like the ones based on Petri nets, where the data themselves carry the responsibility for the evolution of the system. Although the In∇esFlow prototype models the activities that make up a process model in a way that is similar to the one used by user tasks management systems, the original framework did not include enough inner functionality as to link it with a management system of this type. Although the approach used in this prototype allowed the accomplishment of its requirements, it has limitations that can not be overcome with respect to the possibility of keeping track of the activities performed by the users in order to offer them help, automation or other additional services. In the next sections we describe the steps we have given in this direction.

3. Distributed Process Models

Wf-ATOMS offers an object-oriented framework for the specification of workflow models. These models involve the different organization components. This framework is based on a prototype/instance inheritance paradigm. By instantiating this framework, the existing relationships between the different sub processes that constitute a workflow model are specified. These framework instances are named *Distributed Process Models*. As will be seen in the following section, the run-time management environment of the system interprets the *Distributed Process Models* to offer automatic support for process coordination and process assignment between the different components of the workflow. As a consequence, this framework isolates the designer both from the management of the different workflow active instances and from the assignment of the pending processes to the users responsible for carrying them out.

The *Distributed Process Models* in the Wf-ATOMS environment are generalizations of the *Task Model* concept. *Task Models* are increasingly used in the Human-Computer Interaction (HCI) research field (Szekely 1993). *Task Models* are descriptions, usually declarative, of the set of tasks a user can accomplish by means of a particular interactive application. In our case, the *Task Models* we are focussing on are hierarchical *Task Models*. These models decompose abstract user tasks in terms of simpler ones. This process is repeated until a process can be no longer decomposed; that is, until the process is an action directly related with an user interaction in the application interface. A detailed dissertation on the advantages and disadvantages for using hierarchic user task models in the Human-Computer Interaction field can be found in (Kosbie 1994). In the generalization we are describing in this paper, we allow not only to specify local *Task Models*, as in the current models, but also we allow specify relationships between different software and hardware environments within a client/server distributed system. Our aim is to allow the specification of more complex behaviors, including the ones present in workflows, in a similar way as they are modeled in graphic user interfaces.

The framework incorporated into ATOMS (Rodríguez 1997, García 1998b), *Advanced Task-Oriented Management System*, a mono-user and non-distributed user-task management system, has been adapted for the specification of multi-user and distributed tasks in the Wf-ATOMS system. Next, we will explain in detail the structure of ATOMS *Task Models*. After that, we will explain the improvements we have made to those models so as to reuse the ATOMS' framework design, adapting it to a multi-user and distributed environment.

ATOMS *Task Models* are hierarchical representations that indicate which sub processes have to be carried out to complete a complex process. These models are specified by means of a declarative specification language. To specify these hierarchical relationships, the *Task Models* are compelled of a set of processes, with different abstraction levels, and a set of rules that hierarchically relate those tasks with each other. The rules not only define the hierarchical decomposition, but also temporal and contextual constraints among the processes and sub processes, as will be explained later in this section.

There are two types of tasks a *Task Model* has to define: *Atomic Tasks* and *Composed Tasks*. The first type models those interactions the user can directly perform on the application user interface; that is, those actions that can not be decomposed into simpler ones. In order to specify this type of tasks, the ATOMS framework provides designers with a set of *Atomic Task* prototypes. These prototypes allow to easily define *Task Models* by an instantiation and a parameterization procedure of the instances just created. Within the set of atomic prototypes the framework provides, we can cite *AtomicTask*, *WidgetButtonTask*, *WidgetListTask*, *WidgetRadioPanelTa*sk, *WidgetCheckPanelTa*sk, and so on.

Apart from *atomic tasks*, the second type of tasks models those abstract tasks the users have in mind when they complete some set of actions. To specify these *composed tasks*, ATOMS provides the *ComposedTask* prototype. Both types of tasks, atomic and composed ones, can have contextual information associated, in the form of parameters.

ATOMS *Task Models* also incorporate the inheritance concept. This inheritance is based on the prototype/instance paradigm. In this environment, every task can be seen also as a prototype for the definition of new instances. For example, the *WidgetButtonTask* can be used as a prototype to instantiate those tasks associated with pressing a button and, at the same time, it is an instance of the *AtomicTask* prototype. By means of this instantiation procedure, the creation and management of task repositories is facilitated. Thus, it is encouraged the model reuse and designers are helped during the task modeling phase.

ATOMS' *Task Models*, in addition to define the user tasks, define also the rules hierarchically relating those tasks to each other. Each rule indicates the way a particular composed task can be decomposed as a certain set of subtasks, atomic or composed ones. The ATOMS framework provides three types of rule prototypes: *SequenceRule*, *AndRule* and *XorRule*. The designer chooses the prototype that suits the best to her/his needs and instantiates it, according to the temporal relationship between the execution of the subtasks. Thus, in the rule instances of the *SequenceRule* prototype, a given subtask will be blocked until every previous subtask has finished. In the case of rule instances of the *AndRule* prototype, the execution of each subtask is independent from the execution of the rest of subtasks, and all the subtasks in the rule have to be performed so as to finish the process they belong to. Finally, for instances of the *XorRule* prototype, the ending of any subtask implies the ending of the task it belongs to.

Moreover, each rule includes other types of information, such as:

- The parameter flow descriptions between the different subtasks of a task. This description determines how the parameter values are obtained and how these values are propagated among the task hierarchy. A parameter flow description is encapsulated in an instance of a *ProcessParameter* object. The framework provides a library of standard *ProcessParameter* instances that can be extended by *Task Model* designers.
- Which preconditions and post conditions, if any, should hold before and after the execution of each subtask. Similarly, the framework provides designers with a library of instances of *PreCondition* and *PostCondition* prototypes, which can be customized to the particular designer needs before being associated to rule instances.

Similarly to the case of tasks, the specification of rules can also take advantage of the inheritance concept to obtain a higher level of reuse of partial models. However, the use of rule inheritance goes a little further beyond the reuse of the initial prototypes facilitated by the framework. The use of rule inheritance allows the designers to specify a certain relationship between tasks and, afterwards and depending on the particular context for which the rule will be used, turn the conditions imposed on the rule either more rigid or more flexible. Once this specialization has been made, there are two choices for using the new rule instance and its prototype. First, the new rule can completely replace the old one. And, second, both rules may coexist, the old one being applied only when the new one can not be applied.

We have described the architecture of the ATOMS framework, including the different prototypes and constraints provided. However, since *Task Models* are declarative specifications of tasks, this framework has a significant difference with respect to other ones. Most frameworks are procedural, that is, the behavior of the instances of the different classes/prototypes is guided by the methods associated with them. In contrast, the ATOMS framework is declarative, that is, the functionality of the instances is determined by the values of their attributes at execution time. All the methods of the prototypes facilitated by the system are highly parameterized methods that, at execution time and depending on the values adopted by the attributes of each instance, make the instances to behave in one way or another.

The Wf-ATOMS framework extends the one in ATOMS in the sense that, while ATOMS allows to manage *Task Models* referring only to local tasks and involving a unique application, Wf-ATOMS' models allow the use of *Distributed Process Models*, that represent multi user distributed processes within a client/server platform.

Up to now, we have described the framework incorporated in ATOMS for the definition of *Task Models*. The use of a prototype/instance based inheritance model has highly facilitated the work to extend the ATOMS framework to the Wf-ATOMS framework. In this way, the prototypes provided by the Wf-ATOMS framework are the same that the ones described for the ATOMS framework. The main difference is that each task has a boolean attribute associated, *Local*, that indicates whether that task is local to the application or it has to be performed by using another different application, possibly in a remote system. In case a task is not

tagged as *Local*, there are another two attributes, *Application* and *UserOrGroup*, that identify which user or group of users is responsible for carrying out the task and which *Client Application* is to be used. A big part of the effort for the extension of the ATOMS system to the Wf-ATOMS system has been devoted to the extension of the execution environment to a distributed client/server model, as will be explained in the following section.

4. Architecture

This section details the architecture used for the execution environment to manage multi user and distributed tasks. The *Distributed Process Models* allows to specify workflows that are coordinated by the run-time environment we introduce in this section. As it has been already commented, the architecture of the system is based on a client/server environment. First, we will explain the modules included in the server of the architecture and the functions each one fulfills. After that, the structure of the client systems will be described. In particular, the modules that compose each *Client Application* will be shown. Finally, we will describe the communication protocol between the clients and the server for the correct operation of the environment.

4.1. The Server

As shown in Figure 2, the server incorporates the functionality for the coordination and process assignment services.

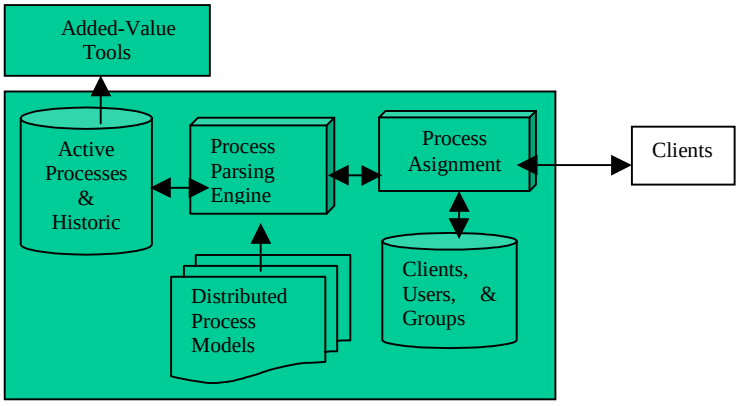


Figure 2: Server Architecture.

The server is in charge of coordinating the accomplishment of the processes by the different clients of the system, as well as the assignment of the pending processes to the clients connected to the server at each moment. The core of the system is the *Processes Parsing Engine*, that uses the information in the rest of components to coordinate the accomplishment of the processes.

For the assignment of processes to the different clients connected during the accomplishment of the workflows, the *Processes Assignment Module* makes use of the *Clients, User and Groups* database. This database contains static information referring to users and groups, and dynamic information such as, for example, which client systems are connected at each moment.

The coordination of processes carried out by the clients involves three main modules: the *Distributed Process Models*, the *Active Processes and Historic* database and the *Process Parsing Engine*. The rest of this section is devoted to explain the functionality of these modules.

The *Distributed Process Models* are formal representations of the processes that can be accomplished. These specifications model the parts of the organization workflows susceptible of being accomplished via software. They have been already described in the previous section.

The *Active Processes and Historic* represent the different activities that are being carrying out (*Active Processes*), as well as those processes that have been carried out in the past (processes *Historic*). For the *Active Processes*, the database includes specific information related to each process, such as its process identification, its execution state, the values for its associated parameters, which client is accomplishing it or must accomplish it, which client should be notified at the end of the process, and so on. As it happens with

user tasks in the user-interface field, the main objective of the *Active Processes and Historic* database is to allow *Added-Value Tools* to analyze and reason about some features of the accomplishment of the processes. In this case, for example, to optimize the performance of processes or to detect repetitive patterns so as to study and automate them.

Each time a client accomplishes the process it has been assigned to, it notifies the fact to the *Process Parsing Engine* in the server. This notification includes all the relevant information dealing with the accomplished process and its context. When this module receives the information of the process accomplished by the client, using the *Active Processes* database and the *Distributed Process Model* in the server, it coordinates the processes to be accomplished by the clients; that is, it determines the set of pending processes. Afterwards, it requests the *Processes Assignment* module to assign the new pending processes to appropriate clients. Furthermore, when it is necessary, it informs the clients that are blocked waiting a specific process to be finished to continue their activities. Finally, the *Process Parsing Engine* is also in charge of updating the contents of the *Active Processes and Historic* database to reflect changes in the processes.

4.2. The Clients

The modules in each client system of the Wf-ATOMS environment are represented in Figure 3. As can be seen there, each client incorporates a *Process Manager* and a set of *Client Applications*. In this section we will explain in detail the paper that fulfils each component and, in the case of the *Client Applications*, the different modules that constitute each one.

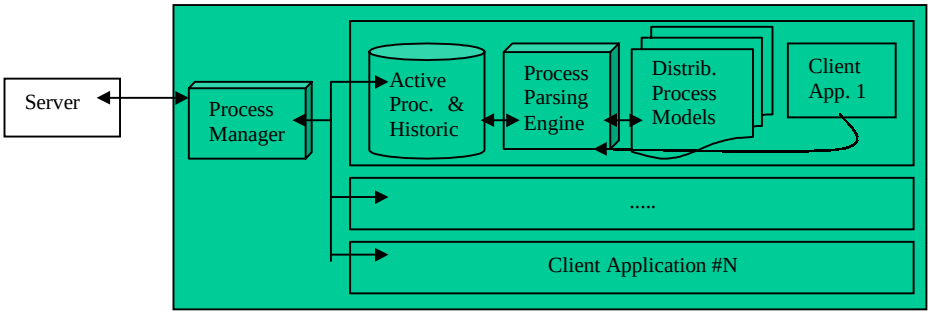


Figure 3: A Client Architecture.

Each client includes a *Process Manager*. This module is in charge of relating, within each client system, the processes the client has been assigned to perform with the applications that should be used to accomplish them. The Process Manager is in direct communication with the *Process Assignment* module in the server, which is entrusted with assigning the pending processes to be accomplished. Once the *Process Manager* in the client knows the task the client has been assigned to perform, it opens the application that is adequate for executing the process.

In each client, there is a set of *Client Applications*. Each *Client Application* is an application for which all or part of its functionality has been modeled in the form of *Distributed Process Models*. These models are accessible via the server or the client to allow external systems to reason about them. As it happens with the *Distributed Process Models* in the clients, *Client Applications* are executed by the client systems, but they do not need to be found physically in the client. At the same time, each *Client Application* incorporates a *Distributed Process Model*, an *Active Processes* and *Historic* database and a *Process Parsing Engine*.

As it has been mentioned, the *Distributed Process Models* are formal representations of hierarchic processes. As we detailed in Section 3, the *Distributed Process Models* represent the tasks that can be carried out in the associated application and relate those tasks with the execution of other tasks in other applications, probably located in different client systems. When, as part of a process being carried out in a given *Client Application*, another external process has to be accomplished, then the *Client Application*, through *the Process Manager*, informs of the situation to the server. Afterwards, the server is in charge of assigning the performance of the task to an available client. From then on, the server coordinates the execution of the

external process and informs eventually the *Client Application* who requested the performance of the process of its end.

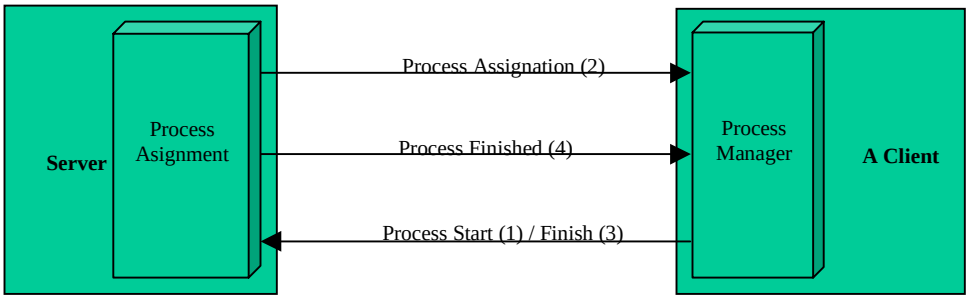
With respect to the *Active Processes* and *Historic* database, it stores, in a similar way than its homonymous in the server, those processes that are being accomplished or that have already been performed in the past. This representation only details those processes that are being accomplished directly in the *Client Application* associated with the *Active Processes* and *Historic* database. In contrast, those parts of processes that are being remotely accomplished are treated with the greatest possible degree of abstraction, since they are not directly related to that *Client Application*. Similarly, *Added-Value Tools* can interact with this database so as to provide high-level services for free to the user-interface. These services may include task automation (Zeiliger 1997) and/or user-task oriented help (García 1998a).

Finally, each *Client Application* incorporates a *Process Parsing Engine*. We have seen that the role of the *Process Parsing Engine* in the server was to analyze the information coming from the clients to determine the states of the processes from the information in the *Active Processes* database and the *Distributed Process Models* in the server. In the case of the *Process Parsing Engine* in the clients, it analyzes the information coming from the interactions of the users with the *Client Applications*, and uses the *Distributed Process Model* and the *Active Processes* database to determine the accomplishment state of the processes assigned to the client.

4.3. Client/Server Communication Protocol

In this subsection we describe the existing communication protocol within Wf-ATOMS. This protocol allows the communication between the server and the different clients, making possible the process coordination and the process assignment roles of the server. As can be seen in Figure 4, the protocol is basically composed by the four communications represented by arrows (1) to (4).

Figure 4: Communication Protocol



In order to begin the accomplishment of a process, a request for starting such a process has to be sent to the server, as indicated by (1). This request will normally be produced by the client used by the supervisor of the workgroup, but it may be sent by any other client, as will be explained later. Once such a request is received, the server will determine which processes have to be carried out, and it will assign those pending processes to appropriate clients so as to accomplish them (2).

When a *Client Application* is carrying out a process, it may imply the accomplishment of sub processes on behalf of other clients or other applications in the same machine. In case a process needs a particular sub process to be executed by other application or other client, the active application (the one performing the process) will send a communication of type (1) to the server. This (1)-type communication is similar to the one used by workgroup supervisors, but it includes some additional information. Thus, depending on the type of sequencing indicated by the rules in the *Distributed Process Model* of the active application, the application may have to wait, or it may not, until the process requested will finish. Then, this kind of (1)-type communication differs from the one explained in the previous paragraph in the fact that the server will inform of the ending of a process to the *Process Manager* in the client who requested its accomplishment. This advice, represented in Figure 4 by a (4)-type communication, will be later routed from the *Process Manager* to the actual *Client Application* who requested the remote process accomplishment. Afterwards, that remote process will be considered as finished and, in case the *Client Application* were waiting for its ending, it would continue its execution.

Once a *Client Application* has finished the accomplishment of a process that has been assigned to, it will inform the server of the fact by using a (3)-type communication. Then, the server is in charge of determining

if any *Client Application* is waiting for the ending of that process or there is not. If the affirmative case, the server will inform that *Client Application* of the fact that the process has already finished.

As a resume of this section, processes are started by the server as a result of a client request. This request may come from a workgroup supervisor or it may come implicit by a process being executed in a *Client Application* that starts the execution of a remote process. Once a process is started, both the coordination and the assignment of the different processes involved are carried out by the *Process Managers* of the clients and the *Process Assignment* and the *Process Parsing Engine* modules of the server.

5. Conclussions and Future Work

In this paper we have described a framework for the representation of distributed tasks. These tasks are a generalization of user tasks, used in the field of user interface modeling, and they have the functionality of workflow models. The *Distributed Process Models* are interpreted in an environment based on a client/server architecture.

The main advantage provided by the modeling of processes by means of these techniques is the possibility to model the different processes as a whole, including the way in which each single activity should be accomplished by means of the appropriate applications. This allows external systems to reason about the models and to offer added-value services. For example, it is possible to semi-automatically generate tutoring systems about the execution of workflows as it is possible to do it for the use of interactive applications.

In the next future we are planning to modify the structure of the client systems to incorporate a unique *Process Parsing Engine*, an unique *Processes Model*, and an unique data base of *Active Processes* and *Historic* for each client machine. This will result in a substantial increment in efficiency, due to the avoidance of unnecessary replications, and to the fact that a part of the communication flow between the clients and the server can be avoided.

CACTUS (García 1999, García 2000), Creating Application *Courses about Tasks Using Scenarios*, a tool for the semi-automatic generation of tutoring courses about the accomplishment of user tasks in interactive applications, is part of our current work along the same line of the one presented in this paper. This tool is also based on the ATOMS task management system; it provides an interactive interface that includes techniques of programming by demonstration in order to specify the contents of the courses. We are planning to adapt CACTUS for the generation of tutors about the accomplishment of interactive processes involved in workflow processes. This environment will allow the creation in a semi-automatic way of courses for working teams to learn and practice how to work together in some activity using a client/server based environment.

6. Acknoledgements

This work has been partially supported by the Plan National de Investigación, projects TIC96-0723-C02-01/02 and TEL97-0306.

References

Alfonseca, A., Contreras, J., Moriyón, R. and San José, P. “Un Sistema de Gestión de Flujos de Trabajo basado en una Jerarquía de Objetos de Gestión Persistentes”. JIDBD’97. Univ. Carlos III de Madrid, 1997.

Alonso, G., Agrawal, A., El Abbadi, A., Kamath, M., Günthör, R. and Mohan, C. “Advanced Transaction Models in Workflow Contexts”. In Proceedings of the 12th International Conference on Data Engineering, Nueva Orleans, Luisiana, USA, Febrero 1996.

García, F., Contreras, J., Rodríguez, P. and Moriyón, R. “Help generation for task based applications with HATS”. In Proceedings EHCI'98, Creta (Greece), September 1998.

García, F., Rodríguez, P., Contreras, J., and Moriyón, R. “Gestión de Tareas de Usuario en ATOMS”. IV Jornadas de Tecnología de Objetos, JJOO'98, Bilbao, Octubre 1998.

García, F., “Towards the Generation of Tutorial Courses for Applications”. 5th ERCIM Conference on User-Interfaces for All. Dagstuhl (Germany), October 1999.

García, F., “CACTUS: Automated Tutorial Course Generation for Software Applications”. Accepted for publication in ACM Conference on Intelligent User Interfaces, IUI2000, New Orleans (USA), January 2000.

Hollinsworth, D.: “The Workflow Reference Model”. Technical Report TC00-1003. Workflow Management Coalition, December 1994. http://www.aiai.ed.ac.uk/WfMC/DOCS/refmodel/rmv1-16.html.

Kamath, M., Alonso, G., Günthör, G. and Mohan, C. “Providing High Availability in Very Large Workflow Management Systems”. Proceedings of the Fifth International Conference on Extending Database Technology (EDBT’96), Avignon, Francia, Marzo 1996. También disponible como IBM Research Report RJ9967, IBM Almaden Research Center, July 1995.

Kosbie, D.S. and Myers, B.A. “Extending Programming by Demonstration with Hierarchical Event Histories”. In Proceedings EWHCI'94, St. Petersburg, Russia, August 1994. East-West Human Computer Interaction.

Myers, B.A., McDaniel, R.G., Miller, R.C, Ferrency, A.S., Faulring, A., Kyle, B.D., Mickish, A., Klimovitski, A. and Doane, P. “The AMULET Environment: New Models for Effective User Interface Software Development”. IEEE Transactions on Software Engineering, Vol. 23, no. 6. June, 1997. pp. 347-365.

Rodríguez, P., García, F., Contreras, J. and Moriyón, R. “Parsing Techniques for User-Task Recognition”. 5th International Workshop on Advances in Functional Modeling of Complex Technical Systems, Paris (France), July 1997.

Szekely, P., Luo, P. and Neches, R. “Beyond Interface Builders: Model-Based Interface Tools”. Proceedings of INTERCHI'93, 1993, pp. 383-390.

Van der Aalst, W.M.P. , Van Hee, K.M. and Houben, G.J. “Modeling Workflow Management Systems With High-Level Petri Nets”. In G. De Michelis, C. Ellis, and G. Memmi, editors, Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pages 31--50, 1994.

Weske, M. “Event-Based Modeling and Analysis of Distributed Workflow Executions”. Fachbericht Angewandte Mathematik und Informatik 16/96-I, Universität Münster, 1996.

Zeiliger, R. and Kosbie, D. “Automating Tasks for Groups of Users: A System-Wide “Epiphyte” Approach, in INTERACT’97 (ed. S. Howard, J. Hammond and G. Lyndgaard), Chapman & Hall Press, IFIP, Sydney, 1997.