TASK HANDLING IN WORKFLOW MANAGEMENT SYSTEMS

Pinar Karagoz, Sena Arpinar, Pinar Koksal, Nesime Tatbul,
Esin Gokkoca, Asuman Dogac
Software Research and Development Center
Dept. of Computer Engineering
Middle East Technical University (METU)
06531 Ankara Turkiye
asuman@srdc.metu.edu.tr

ABSTRACT

Workflow management systems aim to automate the execution of business processes. One of the objectives of the workflow systems is to include the already existing applications such as legacy applications as well as new applications, which are termed as tasks, into the system and provide synchronized execution among them. To achieve this, a mechanism is necessary to support the communication between the tasks and the system. The communication mechanism should handle the transfer of data necessary for the execution of the tasks and for the scheduling of the tasks. Another point to be noted is the necessity of the handling user tasks that have to be performed by the users of the workflow system. Since the trend is toward distributed execution to avoid the bottlenecks due to the nature of central systems, we considered these issues in a distributed execution environment. Therefore, in this paper, task handling in a truely distributed workflow management system that is being developed at METU, namely METUFlow, is described. Yet the techniques described are general enough to be applicable to any workflow management system.

1 INTRODUCTION

Workflow Management Systems (WFMS) achieve considerable improvements in critical, contemporary measures of performance such as cost, quality, service, and speed by coordinating and streamlining complex business processes within large organizations.

A workflow system can be defined as a collection of processing steps (also termed as tasks or activities) organized to accomplish some business process. A task can be performed by one or more software systems, or, by a person or a team, or a combination of these. In addition to the collection of tasks, a workflow defines the order of task invocations or condition(s) under which tasks must be invoked, i.e. control flow, and data flow between these tasks.

In general, a workflow task is considered to be a black box that is functional in nature, i.e., the functionality of the task is orthogonal to that of the workflow process. The tasks could be transactional or non-transactional in nature. Transactional tasks are those that access data controlled by resource managers with transactional properties (i.e. ACID). Non-transactional tasks access data controlled by resource managers without transactional properties such as file systems.

Workflow systems are expected to work in distributed heterogeneous environments which are very common in enterprises of even moderate complexity. Furthermore, the applications running on these environments may be legacy applications, which are meant to work standalone, and some other new applications may be needed for the workflow system. In addition to these applications, tasks which are to be performed by the users, namely user tasks, may take part in the workflow management system. Each of these activities should be wrapped appropriately to work in harmony with the workflow system.

In this paper, we describe task handling in a workflow management system, namely METUFlow, that is being developed at METU, based on a CORBA compliant ORB. Yet the techniques described are general enough to be applicable to any workflow system.

CORBA provides a standard communication mechanism which enables distributed objects to communicate with each other. METUFlow, by allowing CORBA-IDL to be used in task specification, makes it possible to invoke tasks in distributed heterogeneous environments and meets the need for a task wrapper.

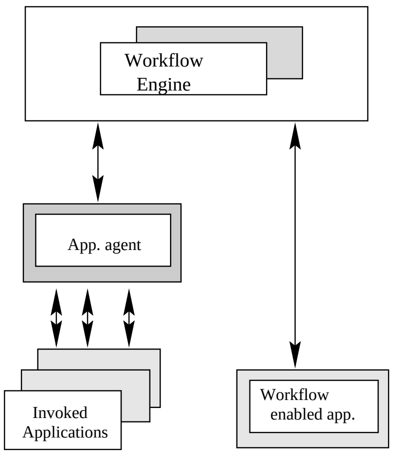


Fig. 1 Invoked Application Interface of WfMC

The paper is organized as follows: In Section 2, the related work on task handling in workflow systems is provided. METUFlow architecture is explained briefly in Section 3. In Section 4, task handling mechanism is explained. Section 5 presents user task handling together with worklist handling. Finally, the paper is concluded in Section 6.

2 RELATED WORK

Although task management is an integral and important part of workflow systems, it is not very much emphasized in the literature.

In workflow management system design, the primary guide is the standards that are published by Workflow Management Coalition (WfMC). WfMC has been established to identify the functional areas and develop appropriate specifications for implementation in workflow products (Holligsworth, 1994). Among these, the specifications for invoked applications are also included.

According to WfMC, any particular WFMS implementation will not have sufficient logic to understand how to invoke all potential applications which might exist in an heterogeneous product environment. As depicted in the Figure 1, WfMC proposes two ways to cope with this heterogeneity:

- Using Application Agents - Application agents contain variety of method invocations behind a standard interface into the workflow enactment service.

- Developing Workflow Enabled Application Tools - These tools use a standard set of APIs to communicate with the workflow enactment service to accept application data, signal and respond to activity events, etc.

The detailed semantics and syntax of an API set for application invocation have not been presented by the Coalition yet.

In METUFlow task handling mechanism, task handlers correspond to the application agents of WfMC. Task handler's methods provide parameter passing, starting and aborting the task which match the command set of WfMC API.

As presented in (Krishnakumar, N. and Sheth, A., 1995),(Sheth, A. and Kochut, K., 1997), METEOR (Managing-End-To-End OpeRations) project is one of the works in which task integration is discussed.

In METEOR workflow model, TSL (Task Specification Language) supports the detailed specification of each task and its interaction with the interfaces / processing entities in a distributed environment. Each task specification is executed by a task manager.

There are some common points in the task handling techniques of METEOR and METUFlow. However, in METUFlow, the communication mechanism between the task and the task handler in terms of status message and parameter passing with a well-defined task handler interface is emphasized. As another difference, in METUFlow, handling of user tasks in collaboration with worklist handlers is also considered.

3 METUFlow ARCHITECTURE

A simplified architecture of METUFlow system is given in Figure 2. In METUFlow, first a workflow is specified using a graphical workflow specification tool which generates the textual workflow definition in METUFlow Definition Language (MFDL). The functionality of the scheduler is distributed to a number of guard handlers which contain the guard expressions for the events of the activity instances. Guards are temporal expressions defined on events (Gokkoca, E. et al., 1997),(Tatbul, N. et al., 1997). There exists a task handler for each task which acts as an interface between the activity instance and its guard handler. In a workflow management system, there are activities in which human interactions are necessary. In METUFlow, user task handler manages such interactions. User task handler is a kind of task handler that is responsible for progressing work requiring user attention. User task handler uses the authorization service to determine the authorized roles and users. History manager provides the mechanisms for storing and querying the history of both ongoing and past processes. It communicates with the scheduler through a reliable message queue to keep track of the execution of processes. In METUFlow, the work-

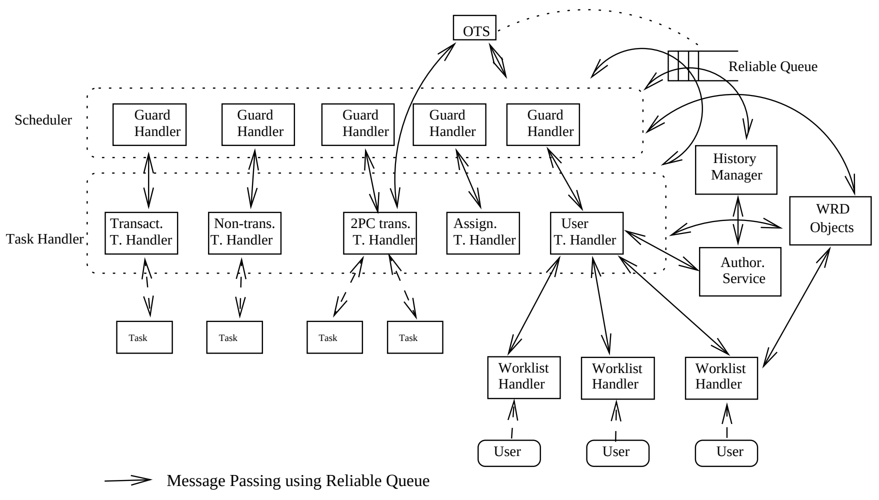


Fig. 2 The simplified architecture of METUFlow

flow relevant data of the workflow management system that are defined in the system specification are kept as CORBA objects. WRD objects keep the values and versions of the variables of the workflow system that are to be used by the underlying tasks.

The communication infrastructure of METUFlow is CORBA, but CORBA does not provide for reliable message passing, that is, when ORB crashes, all of the transient messages are lost. For this reason, a reliable message passing mechanism which uses Object Transaction Service (OTS) based transaction manager to commit distributed transactions is implemented. Note that reliable message passing is necessary among all the components of METUFlow such as between guard handlers and task handlers as indicated in Figure 2. Detailed description of the system is explained in (Dogac, A et al., 1997).

4 TASK HANDLING IN METUFlow

In METUFlow, a distributed environment is provided using the guards, guard handlers and task handlers. The scheduling of the tasks are distributed on guard handlers through guards. Task handlers encapsulate the tasks as objects and each task handler binds its task to the associated guard handler. Thus, tasks, having their own guards on their guard handlers, have the responsibility of correctness of their own execution.

The guard evaluation is done by the guard handler. The skeleton of a guard handler for each task is generated at compile time of the workflow definition and the guard handler is created at run time. The tasks communicate with each other and with the other components of the workflow through guard handlers. Guard handler keeps the guards in the form of conjunction or disjunction of boolean expressions, and the messages coming from the other tasks are kept in a queue. An event of a task (start, abort, commit) can execute only if its associated guard evaluates to true. According to the incoming messages, if the guard evaluates to "true", the guard handler sends the appropriate message to its task handler.

A task handler is, in fact, the wrapper for the task implemented as a CORBA object. As for guard handler objects, skeleton of a task handler object is generated at compile time and object itself is created at run time. It acts as a bridge between the task and its guard handler. While the task handler integrates the task to the workflow system, the guard handler obtains the relevant information from the system. The guard handler sends the information necessary for the execution of the task, like the arguments, to the task handler and the task handler sends the information about the status of the task and new values of arguments which are set by the task to the guard handler. The status messages are as follows: When a task starts, its status becomes *Executing.* If it can terminate successfully, then its status is changed to *Committed* or *Done* depending on whether it is a transactional or a non-transactional task. In case the task fails, its status becomes *Aborted* or *Failed.*

In METUFlow, six types of tasks are defined as follows:

- Transactional Tasks - these are the tasks which are controlled by a transactional resource manager. These tasks can either commit or abort. The final status of these tasks can be controlled by the resource manager that has ACID properties.

- Non-Transactional Tasks - these tasks access data controlled by resource managers without transactional properties such as file systems. Therefore, their final status are not controllable.
- Non-Transactional Tasks with Checkpoint - they are the same as non-transactional tasks with the exception of checkpoints. Such an extension is the result of observation that in some business applications, it is necessary to take checkpoints and guarantee the correctness of execution until that point so that recovery becomes easier in case of a failure.
- Two-Phase-Commit Tasks - these are transactional tasks which communicate with transaction manager that takes the final decision on whether to commit the distributed transaction.
- User Tasks - this type of tasks need human intervention and are handled via the worklist handler.
- Assignments - these tasks make necessary assignments in the workflow process instance. They are not external applications, but provide accesses to workflow relevant data.

Each task is integrated to the system via its task handler. The implementation of the task handlers may differ according to the type of the task they wrap, as explained in the following section.

4.1 GENERAL STRUCTURE OF THE TASK HANDLER

Each task of a workflow system is accompanied by a task handler. Task handler is a CORBA object and has a generic interface which contains the following methods to communicate with its associated guard handler:

- **Init** This method is used for passing initial data such as name of the task and signature of parameters to the task handler.
- **Start** This method is called by the guard handler when the start guard of the task evaluates to true. This causes the task handler to invoke the actual task.

Task handlers for each different type of tasks inherit from this interface and provide overloading of these methods and/or further methods when needed as explained in the following:

- **Transactional task handler** This type of task handler is for the transactional tasks. Even if a transactional task terminates successfully, its task handler should wait for the commit or abort message from the guard handler. For this purpose, in addition to the common methods described above, this type of task handler provides two more methods namely, *Commit* and *Abort* to be called by the guard handler to commit or abort the task respectively.
- **Non-Transactional task handler** This type of task handler handles tasks which are either non-transactional or non-transactional with checkpoint. The difference between non-transactional and non-transactional with checkpoint is that in the latter in case of a failure the application is rolled back to the latest checkpoint and not to the beginning. Since this does not affect the communication between the task and the task handler, only one type of task handler is defined for both of them. Note that, non-transactional tasks terminate without waiting for any confirmation from the guard handler. They only inform the task handler about their status (*Done* or *Failed*).
- **Two-Phase-Commit task handler** This type of task handler is required for two phase commit transactional tasks. The difference between this type of task and transactional tasks is that, the former provides an additional status message, namely *Prepared.* Thus, this type of task handler provides a method called *Prepare* to be called by the transaction manager.
- **User task handler** User tasks are handled by user task handler and worklist handler. The user task handler stores the name of the task and the other necessary information to the repository and distributes these tasks to the users according to role resolution information retrieved from the authorization service. The worklist handler informs the user about the tasks that she/he is responsible for and sends the status of the task to the user task handler.
- **Assignment task handler** This task handler does not cause any task to begin, but only a workflow relevant data assignment is done within the scope of a transaction.

4.2 COMMUNICATION BETWEEN TASK AND TASK HANDLER

Generally, it is desired for a workflow to encapsulate the existing applications. However, these applications are generally developed to execute as standalone applications on specific platforms, such as DOS, Windows, or UNIX. Their error messages and parameter structures

```
register_patient()
{
  TaskExecuting();
   Connect_to_Database();

   /* This part of the code gets patient information from the user.
      A new patient_id is generated for this  patient      */

  Insert_Into_Database(patient_info,status);
  if(status == True){
      ReadyToCommit();
      if(Status() == Commit) {
        Commit();
        TaskCommitted();
        Return(patient_id);
      } else {

      Abort();
      TaskAborted();
      }
  } else {
     Abort();
     TaskAborted();
  }
}
```

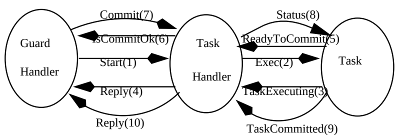


Fig. 3 Communication among Guard Handler, Task Handler and Task

are not common. Therefore, a common form of communication structure between the tasks and the system is needed. This can be achieved by interfering the code of the application where possible. For the applications where such an interference is not possible, an outer shell should be created on the top of the existing application. This outershell is a part of the task handler. It provides for the status information and parameter passing.

4.2.1 STATUS INFORMATION PASSING IN TASK HANDLER

Status information transfer is handled by adding necessary calls either to the code of the task, or to the shell which covers the task and acts as a translator between the task and the associated task handler. For the correct scheduling of tasks, the workflow system needs to know the status of the tasks. For this reason, as described in the previous section, a set of status messages is defined (*Initial, Executing, Committed, Aborted, Failed, Done*). Through this set, the communication message structure is unified.

In Figure 3, we provide the modified code of a transactional task, register_patient to illustrate the case where interference to the code is possible. Register_patient is a task which is a part of a workflow process, Check-Up-Patient. In this process, all the activities and their flow are defined, from the point the patient applies for a check-up until the check-up of the patient finishes. In the register_patient task, the information about the patient who applied for check-up is registered to the patient database. The calls which are written in boldface are added to the original code of the task. The functionalities of these calls are as follows:

- **TaskExecuting()** informs the task handler that it has started executing.
- **ReadyToCommit()** informs the task handler that operation is terminated successfully.
- **TaskAborted()** informs the task handler that the final status is Abort.
- **TaskCommited()** informs the task handler that the final status is Commit.
- **Status()** gets the status message from the task handler.

In Figure 3, the arrows show the message passing between the involved entities. The labels of the arrows are numbered according to the order in which the calls are made and the figure describes the flow of messages for the scenario in which the task terminates successfully and its commit guard evaluates to true. When the start guard of the task evaluates to true, the guard handler of the task calls **Start** method of the task handler. This causes the task handler to start execution of the task. When the task starts executing, task handler is informed of this fact through the **TaskExecuting** call. The status of the task is sent to the guard handler by the task handler in **Reply** method of the guard handler with the parameter *Executing*. Then the normal flow of the task begins. If patient information is written to the database successfully, the task handler is sent **ReadyToCommit** call. The task handler informs the guard handler that the task is ready to commit by calling its **IsCommitOk** method. If the commit guard of the task evaluates to true, the guard handler informs task handler about this situation by calling its **Commit** method. Otherwise, **Abort** method is called. Task checks whether the message sent is *Abort* or *Commit* by the **Status** call. If task handler sends *Commit*, then the actual commit of the task occurs, and the task claims the final state as *Commit*. In case of *Abort* message, task aborts and sends the final abort status. When final status is claimed by the task, the task handler informs the guard handler about the final status by calling **Reply** method of the guard handler again.

4.2.2 PARAMETER PASSING IN TASK HANDLER

In METUFlow, two kinds of data pass among

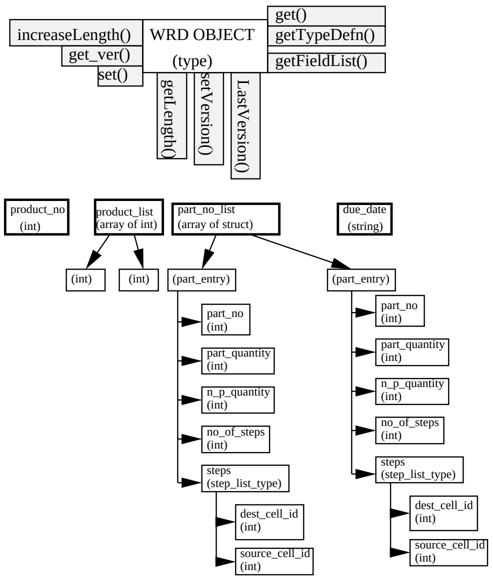


Fig. 4 The structure of WRD object

the components of the system. The first one is the data which the applications need for execution. The flow of this type of data is called **Data Flow**. The second one is the data which the workflow needs for the correct execution of the whole system. The flow of the second type of data is **Control Flow**. The status information which passes among task, task handler and guard handler is a part of the control flow. Similarly, the data flow between task, task handler and guard handler should be defined.

When the task starts execution, it has to receive the value of the parameters. At this point, it communicates with the workflow relevant data objects. Variable declarations correspond to Workflow Relevant Data (WRD) that provide for the data flow between the tasks and the workflow system. In METUFlow, a WRD is implemented as a CORBA object with an interface for getting, setting values and versions of the variables. The interface of a WRD object is depicted in Figure 4.

A WRD object contains the value and the versions of the variable. If the variable is of simple type, the value of the variable is retrieved directly from the corresponding WRD object. For the variables of complex types, WRD object points to other WRD objects which correspond to the subcomponents of the complex type. By this way, an object tree is formed. In Figure 4, sample WRD objects are visualized. Among them, product_no, and due_date are variables of simple types. Product_list is an array of integer and part_no_list is an array of structure. The relevant information of these complex-typed variables are kept in in corresponding

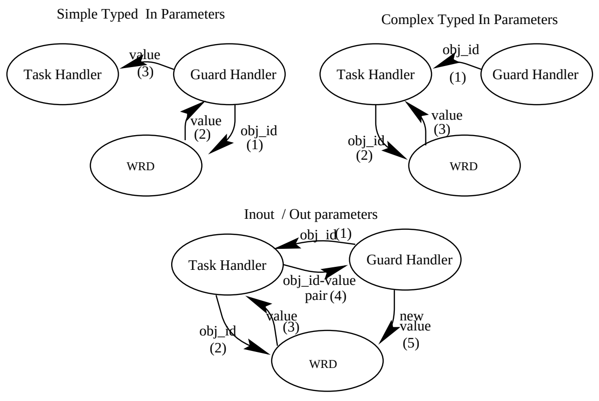


Fig. 5 Parameter Passing

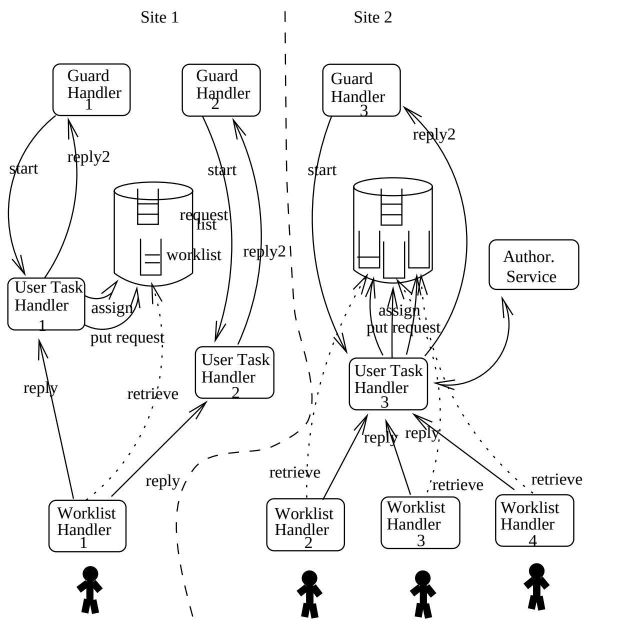
WRD object trees.

Parameters of a task is defined as either *In, Inout* or *Out* according to the direction of flow. The structure of the parameters affects the way they are retrived from WRD objects. As shown in Figure 5, the task handler obtains the values of simple typed parameters from the guard handler, whereas the values of complex typed parameters are retrieved directly from the WRD objects. If new values are to be returned from the task, the task handler sends these values to the guard handler as *ObjectId-Value* pairs and they are set by the guard handler on WRD objects.

The reason behind accessing to corresponding WRD objects differently for simple and complex typed variables is to decrease the communication traffic between the objects. For one complex typed variable, more than one values are retrieved. By leaving the guard handler out, these values are transmitted directly to the task handler. Thus, the cost of carrying these values from the workflow relevant data handler to the guard handler, and from the guard handler to the task handler is prevented. On the other hand, the new values are always sent to the guard handler and the updating of WRDs with new values is performed by the guard handler. This provides a more secure and resilient way against the failures.

5 HANDLING USER APPLICATIONS IN METUFlow

User task handler is the component of the workflow system which transmits the activities to the appropriate users (workflow participants) of the system. Like other tasks of the workflow management system, user tasks communicate with the system through task handler which embodies the task. Therefore, the system



retrieve	:	retrieves the worklist contents for the presentation to the user
put request	:	puts a work item into request list
reply	:	sends reply back (from worklist handler to user task handler)
reply2	:	sends reply back (from user task handler to guard handler)
assign	:	inserts work items into the worklists of the user

Fig. 6 Worklist Manager in METUFlow

does not realize the implementation differences between a user task and other types of tasks.

The user task is not an external task. Therefore, task handler of the user task does not need to call an executable program. It stores the information about the task to the persistent storage and makes the appropriate distribution of user tasks according to the role resolution information of authorization service. The worklist handler uses this storage to get the appropriate information of activities to inform the users when necessary.

In METUFlow, the worklists are distributed, that is, a worklist at a site contains the work items to be accessed by the users at that site.

When a user activity is to be invoked by the guard handler, a user task handler created for this purpose stores the request (work item) into a request list within the scope of a transaction. Request list is a CORBA object and its implementation in a particular site depends on the persistent storage available in that site, that is, this CORBA object is implemented on a DBMS if it is available, otherwise it is implemented as a file. As the first step, user task handler decides on the assignment of work items to the worklists of the users in cooperation with the authorization service. The second component, worklist handler is responsible for retrieving work items to be presented to the user for processing.

A point to be noted over here is the following: CORBA provides location transparency, in other words the users need not be aware of the location of the objects to be created. However, CORBA does provide mechanisms to affect the object creation site although the specifics depend on the ORB at hand. First, by default, an object is created at the local site if it is possible. Therefore, whenever there is a request to create a work list handler, it is created at the same site with the user task handler. In order to be able to create worklists at the same host with the involved user (or role), a list is kept which stores the association between the user-ids and host-ids. In METUFlow, lookup method of Orbix's locator class is used for this purpose (Iona, 1996). Figure 6 depicts the distributed worklist management in METUFlow.

5.1 WORKLIST MANAGEMENT

When user interactions are necessary within the process execution, the related work items are placed onto the worklists of related users by the user task handler. Users are defined in the system as actors. Each actor is assigned one or more roles. Role resolution information is retrieved from the authorization service and the distribution of the work items is done according to this information. Worklist handler is a communication mechanism between the user and the task handler. Worklist handler retrieves a work item from the worklist of the user and after completion of the work item, and informs the task handler about the status of the work item.

Each worklist is accessed through a specific worklist handler. For this reason a worklist handler factory is developed. There must be as many worklist handlers as the number of users in the system. This provides a neat load distribution on worklist handlers.

Worklist handler created by the factory serves to the users of the system. It interacts with the user through the interface implemented using Java. Worklist handler reads the worklist of the user and presents the activities assigned to the user.

Methods of the worklist handler are:

- **GetList:** This method retrieves the worklist of this user.
- **GetRequest:** This method parameter and returns the information about the request whose identifier is given.
- **Reply:** Reply method is used to inform the task handler about the change in the status of the request.

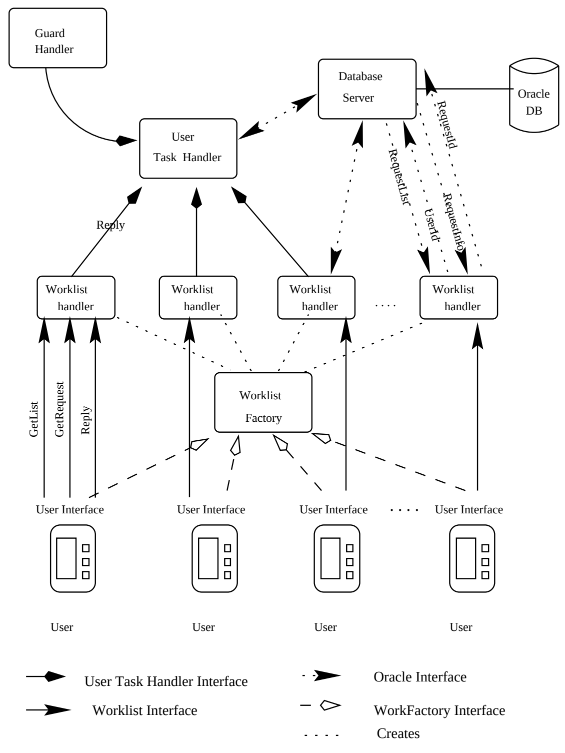


Fig. 7 Relations Among Components

The interaction of components and the usage of methods are depicted in Figure 7.

The user interface provides a screen for the user to show which work items are assigned to him/her by the user task handler. Each user has a separate set of work items which may contain same or different work items depending on the role of the user and the work item distribution, therefore, each user will access to a different user interface, which gets the work items to be presented, from a certain worklist.

One of the objectives of METUFlow is to provide access to the system through WWW (World Wide Web). Therefore, the user interface should be implemented on a web browser, like Netscape. For this reason, Java is chosen as the language for this implementation. Java is a language designed for programming on the Internet. The most important features of Java are its being object-oriented, distributed, portable, multithreaded and dynamic (Flanagan, 1996). These features of Java provide also ease to connect to a CORBA compliant ORB, which is used in worklist implementation.

For administrative purposes, a user screen which accesses to all worklists can be created. The workflow administrator can add, delete work items to and from the worklists of users.

6 CONCLUSION

In this paper, the techniques to integrate the underlying tasks to the workflow system are discussed. These techniques are implemented in the workflow management system that is being developed at METU, namely METUFlow. Tasks that are usually developed to work in heterogeneous, autonomous and distributed environments participate in a workflow system. In addition to this, some tasks are performed by the users of the workflow system. To handle such a diverse domain of tasks, we define a generic task object to introduce tasks to the workflow system. To simplify the mapping of different types of tasks to a generic interface, these objects, i.e task handlers, are divided into five subgroups (transactional, two-phase-commit transactional, non-transactional, assignment and user task handlers). Each of the groups needs additional methods. Among them, user task handler differs in that it works in collaboration with worklist handlers to inform the users about the tasks they have to perform.

The overall approach provides a single interface of tasks from workflow system point of view. Hence, any kind of task that works in any environment can easily be integrated in the workflow system.

REFERENCES

Dogac, A , Gokkoca, E., Arpinar, S., Koksal, P., Cingil I., Arpinar B., Tatbul N., Karagoz, P., Halici U., and Altinel, M. (1997). Design and implementation of a distributed workflow management system: Metuflow. In *NATO-ASI Advances in Workflow Management Systems and Interoperability*, Istanbul.

Flanagan, D. (1996). *Java in a Nutshell.* O'Reilly & Associates, Inc.

Gokkoca, E., Altinel, M., Cingil, I., Tatbul, N., Koksal, P., and Dogac, A. (1997). Design and implementation of a distributed workflow enactment service. In *Conference on Cooperative Information Systems (CoopIS)*, Charleston.

Holligsworth, D. (1994). The workflow management coalition, the workflow referencence model. Technical report, Workflow Management Coalition.

Iona (1996). *Orbix Reference Guide.* Iona Technologies Ltd.

Krishnakumar, N. and Sheth, A. (1995). Managing heterogenous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3.

Sheth, A. and Kochut, K. (1997). Workflow applications to research agenda: Scalable and dynamic work coordination and collaboration systems. In *NATO-ASI Advances in Workflow Management Systems and Interoperability*, Istanbul.

Stevens, W. R. (1990). *Unix Network Programming.* Prentice-Hall, Inc.

Tatbul, N., Arpinar, S., Karagoz, P., Cingil, I, Gokkoca, E., Altinel, M., Koksal, P., and Dogac, A (1997). A workflow specification language and its scheduler. In *International Symposium on Computer Information Systems (ISCIS)*, Antalya.