# Implementing Support for Work Activity Coordination within a Distributed Workflow System

J. J. Halliday, S. K. Shrivastava and S. M. Wheater
Department of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, England.

*Appeared in the Proceedings of the 3rd IEEE/OMG International Enterprise Distributed Object Computing Conference (EDOC'99), University of Mannheim, Germany, pp. 116-123, September 27-30, 1999.*

***Abstract* - There is growing interest in providing computer support for an organisation's business processes such as customer order processing, product support, stock taking and so forth. Workflow systems are normally used for this purpose to co-ordinate and monitor execution of multiple tasks arranged to form business processes. In this context computer support for work activity co-ordination will be taken to mean tools and services made available to the employees of an organisation to enable them to carry out their tasks that form the part of these workflows. The paper illustrates how a transactional workflow system can be augmented with support for reliable management of worklists in an arbitrarily distributed environment. Ideally, the workflow enactment service should be neutral to the types of work activity coordination services to be made available to users. This way the needs of a wide variety of organisations can be supported by building organisation specific services. The approach taken here is to separate, as much as possible, organisational aspects of workflow management from the workflow enactment service**

## I. INTRODUCTION

We address the problem of providing computer support for work activity coordination within an organisation. We observe that organisations are increasingly making use of electronic means for conducting business and therefore need to automate their businesses processes. In the domain of electronic commerce for example, the business processes would include interactions between customer-to-business organisation (e.g., product support, customer complaint handling) as well as between business organisation-to-business organisation. Most organisations make use of workflow management systems for automating their business processes. Workflow systems are intended to direct, coordinate and monitor execution of tasks arranged to form workflow applications representing business processes. Tasks (activities) are application specific units of work, and may themselves be workflows. Many of these tasks require human intervention. In this context 'computer support for work activity co-ordination' will be taken to mean tools and services made available to the workers of an organisation to enable them to carry out their tasks that form the part of these workflows. We indicate below the kind of interactions that are required.

We assume that workers within an organisation choose, or have assigned to them, *roles* which have associated responsibilities. A worker can have more than one role to fulfil, and a given role can be fulfilled by more than one worker. The responsibilities associated with a role can be discharged by performing *tasks*, where each task forms a part of some business process. Associated with a role therefore is a *tasklist* (*worklist*) containing the list of outstanding tasks that currently need attention from that role. As business processes are enacted by the workflow system, relevant lists of tasks get populated and notifications appear on the workstations of workers (very much like new email message notifications) who have taken on the corresponding roles. A worker can browse the list and select a particular task for execution. Once the task is finished, he/she uses the work activity coordination software for notifying the completion of the task, thereby enabling forward progress of the corresponding workflow process. A worker with appropriate role (for example a supervisor), can initiate a workflow, monitor its progress and if necessary, send reminders, change the priority of tasks on tasklists, change role to worker bindings and even modify the structure of the process.

Our work activity coordination system has been designed to support the above types of interactions. Three aspects of our system are worth noting: (i) the system is ideal for distributed organisations, as the entire system architecture - the workflow system as well as the work activity coordination system - is decentralised and open and, to provide interoperability, implemented using CORBA technology; (ii) both the workflow and the work activity coordination systems are fault tolerant as they make use of transactions where appropriate; and (iii) the coupling between the workflow system and the work activity coordination system has been kept quite narrow to minimise change dependency between workflow definitions (these describe logical aspects of a process) and organisational aspects of workflow execution concerned with resource allocation and work activity coordination. The paper describes these aspects of our system illustrating them with the help of a detailed example, a customer complaint handling process within an organisation.

## II. UNDERSTANDING THE REQUIREMENTS AND THE APPROACH TAKEN

At the heart of a workflow management system is the workflow enactment service that provides a workflow

execution environment [1]. We assume that the organisation under consideration is distributed across multiple sites, so business processes require a distributed workflow execution environment. The workflow execution environment must also be fault tolerant, as the communication environment (e.g., the Internet) as well as the processing environment (workstations, servers) will be assumed to suffer from failures which can affect both the performance and consistency of applications. The work activity coordination environment must also be decentralised and fault tolerant.

Unfortunately, most currently available workflow systems possess monolithic structure, so do not provide distributed execution environments [2,3]. Further, they offer little support for building fault-tolerant applications, nor can they inter-operate, as they make use of proprietary platforms and protocols.

We have therefore built a transactional workflow system whose architecture is decentralised and open: it has been designed and implemented as a set of CORBA services to run on top of a given ORB [4]. Furthermore, the system has been structured to provide fault tolerance at *application level* and *system level*. Support for application level fault tolerance has been provided through flexible task composition facilities that enable an application builder to incorporate alternative tasks, compensating tasks, replacement tasks etc., within an application to deal with a variety of exceptional situations. Support for system level fault tolerance has been provided by recording inter-task dependencies in (CORBA) transactional shared objects and by using transactions to implement the delivery of task outputs such that destination tasks receive their inputs despite a finite number of intervening machine crashes and temporary network related failures; this also provides a durable audit trail of task interactions. Thus our system naturally provides a fault-tolerant 'job scheduling' environment that maintains a durable history of application interactions. The work activity coordination environment has also been designed using distributed objects and transactions to ensure that the tasklists faithfully represent the status of workflow tasks.

Ideally, the workflow enactment service should be neutral to the types of work activity coordination services to be made available to users. This way, needs of a wide variety of organisations can be supported by building organisation specific services. Existing workflow management systems have failed to do this because they each support a fixed organisation model [3]. The architecture defined by the WfMC (Workflow Management Coalition) [1] has integrated worklists with the enactment service, making distributed worklist management problematical [2]. The approach taken here is to separate, as much as possible, organisational aspects of workflow management from the workflow enactment service. We discuss this below.

In general terms, a *workflow definition* (*workflow schema*) represents the structure of a business process in terms of constituent task definitions; each *task definition* specifies details such as the task type (for example, 'Transfer Funds'), input data required for starting the task, sources of these inputs, output results produced, and temporal dependencies between tasks. In addition, it is also necessary to specify organisation related information, essentially stating *who* is responsible for carrying out that task and *where* it is to be performed (e.g., the task 'answering customer queries' is to be carried out by a worker with the role 'help desk operator' located at the central headquarter). This organisation dependent information should be decoupled from the task definition, to ensure that changes to workflow definitions (respectively organisation structures) have limited effects, if any, on the organisation structures (respectively workflow definitions) [5]. In our system, this decoupling has been achieved by using the familiar' concept of a role. Each task in a workflow definition is allocated to a role, which defines the logical person/entity responsible for performing the task. An organisation structure database (that could be quite independent from the workflow system) is responsible for the grouping of roles and their mapping to actual person/entity and also contains location specific information.

A workflow is executed by instantiating the corresponding workflow schema which has the effect of the creation of sets of objects that control and represent tasks. At this time, the workflow execution service needs to know the location (computers) where these objects need to be created. This is organisation specific information, therefore we assume that the organisation structure database of an organisation contains the appropriate location information where these objects are to be created. As an example of location policy, in a highly distributed environment it is quite likely that the creation of the objects that control and represent tasks will be located 'close" to the person/entity who will perform the task. We will describe in a subsequent section, how organisation and role specific 'task factories' query the organisation structure database to instantiate a workflow and also create worklists objects for roles.

Finally, we make a general remark concerning activity management [6] which is that two interfaces are required (a) process management interface for instantiation and monitoring of workflows; and (b) worklist management interface. The work activity coordination system should provide easy, integrated way of using both. We provide an integrated GUI that can be tailored to the requirements of the business process. This approach is similar to that taken by [10].

## III. A MOTIVATING EXAMPLE

The trouble ticket scenario [7] covers quality assurance teams or customer support teams. A "bug" or "problem" is identified; it must be recorded; the record must be checked for accuracy; from a single instance of a problem, the underlying cause is identified; a resolution is identified, which must be communicated back to the original party with the problem. We describe the process and one interaction scenario as presented originally in [7], except

that a few simplifications have been made to keep the example manageable for the purposes of this paper.

### A The Process

*Step 1: Recording the Problem.* The problem may be found by an internal or external person. There are two ways that a problem can come from the external person: by phone or by email. For the internal person, there must be a screen that can be called up without delay that presents a form for entering the details of the problem. Submitting this form will cause the creation of the workflow process, and at the same time generate a unique ID for the trouble ticket. The ID is used as a way to call up the trouble ticket when that person calls in again to check on progress.

*Step 2: Reproduce the Problem.* This step is designed to check the trouble ticket report, and to see if it describes accurately a reproducible problem. This activity is simply to follow the instruction on the report, and to see if the described behaviour occurs. If the behaviour cannot be reproduced, then this process goes to step 3, otherwise it goes to step 4.

*Step 3: Correcting the Report.* This step is reached only if the problem cannot be reproduced. This step is assigned to the originator if internal. If external, this must be assigned to a person who can contact the originator and get more clarification on the problem. There are two results of this step, either back to step 2, or to give up on the process and go to step 5.

*Step 4: Identifying the Problem and Resolution.* This is where the specialist is called in. The problem details should narrow down the area of the problem. If the expert determines that the area is wrong, it should be able to be reassigned, and the person assigned to the activity should change. The problem stays in this state until a resolution is determined. Either the problem is identified and it will be fixed, or it be fixed later due to schedule constraints, or it is determined to be a misunderstanding and is actually the correct behaviour. In all cases the resolution must be communicated to the originator, either via email, or else through a phone call. The process goes to step 5.

*Step5: Communicate Results.* The results of the process are communicated back to the originator here. This step contains a rule that the result must be communicated within 3 days of being known. If not, an email message is sent to the support manager.

### B Interaction Scenario

*Day 1*. An important customer calls up with a problem. Person A takes the call, and brings up the form, enters the information, submits it, and sees that it is currently assigned to Person B. Later Person B sees it on the worklist, but does not call it up.

*Day 2*. Person A checks on the status of the trouble ticket, and sees that Person B has not taken any action. Person A composes an email message to Person B, with the processes status page attached. Person B receives the email, clicks on the link to call up the process in the browser. Person B reproduces the problem, determines that person C should fix the problem, and then completes the activity. It disappears from B's worklist; a new activity appears in C's worklist.

*Day 3*. The important customer calls up again but this time to the Vice President of the division, who talks to Person A, who checks on the status. Person A raises the priority data field in the process in response. Person C sees it immediately, and gets to work. A short time later a solution is found; he enters the resolution, and marks it fixed, passing the data back to Person B since B was the one who reproduced the problem. B verifies the fix, and the process goes to step 5.

*Day 9*. Person A checks back in on the process. The process is finished, but A is still able to access the history and final state of the process.

Although the above interaction scenario has been described in terms of Persons', in the following we will put a role based interpretation; so Person A'will mean the the person who is acting out role A.'

## IV. WORKFLOW SYSTEM OVERVIEW

This section presents a brief overview of the workflow system and describes how the trouble ticket process can be represented and executed using the system's repository and execution services respectively.

### A System Overview

The workflow system's structure is shown in fig. 1. Here the box represents the structure of the entire distributed workflow system (and not the software layers of a single node). The most important components of the system are the two transactional services, the workflow *repository* service and the workflow *execution* service. These two facilities make use of the CORBA Object Transaction Service (OTS).

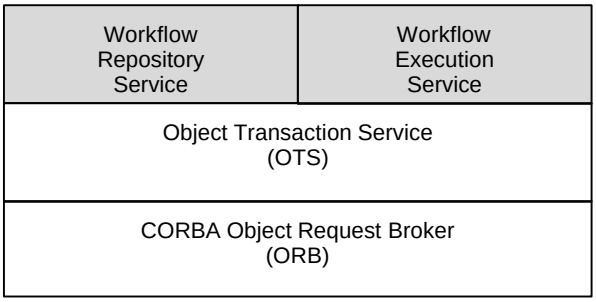


Fig. 1. Workflow management system structure

*Workflow Repository Service*: The repository service stores workflow schemas and provides operations for initialising, modifying and inspecting schemas. A schema is represented according to the model briefly discussed below in terms of tasks, compound tasks, genesis tasks and dependencies. Individual tasks that make up an application can be *atomic* ('all or nothing'ACID transactions, possibly containing nested transactions within, with properties of: Atomicity, Consistency, Isolation and Durability) or *non-atomic*. We have designed a scripting language that

provides high-level notations (textual as well graphic) for the specification of schemas [8]. We next describe the task model that enable flexible ways of composing an application

Tasks are 'wrapper' objects to real application tasks. A task is modelled as having a set of *input sets* and a set of *output sets*. In fig. 2 (a), task $t_i$ is represented as having two input sets $I_1$ and $I_2$, and two output sets $O_1$ and $O_2$. A task instance begins its life in a *wait* state, awaiting the availability of one of its input sets. The execution of a task is triggered (the state changes to *active*) by the availability of an input set, only the first available input set will trigger the task, the subsequent availability of other input sets will not trigger the task (if multiple input sets became available simultaneously, then the input set with the highest priority is chosen for processing). For an input set to be available it must have received all of its constituent input objects (i.e., indicating that all dataflow and notification dependencies have been satisfied). For example, in fig. 2(a), input set $I_1$ requires three dependencies to be satisfied: objects $i_1$ and $i_2$ must become available (dataflow dependencies) and one notification must be signalled (notifications are modelled as data-less input objects). A given input can be obtained from more than one source (e.g., three for $i_3$ in set $I_2$). If multiple sources of an input become available simultaneously, then the source with the highest priority is selected.

A task terminates (the state changes to *complete*) producing output objects belonging to exactly one of a set of output sets ($O_1$ or $O_2$ for task $t_i$). An output set consists of a (possibly empty) set of output objects ($o_2$ and $o_3$ for output set $O_2$). Task instances manipulate references to input and output objects. A task is associated with one or more implementations (application code); at run time, a task instance is bound to a specific implementation.

A schema indicates how the constituent tasks are 'connected.' We term a source of an input an *input alternative*. In fig. 2(b) all the input alternatives of a task $t_3$ are labelled $S_1$, $S_2$, ,.$S_8$. An example of an input having multiple input alternatives is $i_1$, this has two input alternatives $S_1$ and $S_2$. Note that the source of an input alternative could be from an output set (e.g., $S_4$) or from an input set (e.g., $S_7$); the latter represents the case when an input is consumed by more than one task.

The notification dependencies are represented by dotted lines, for example, $S_5$ is a notification alternative for notification dependency $n_1$.

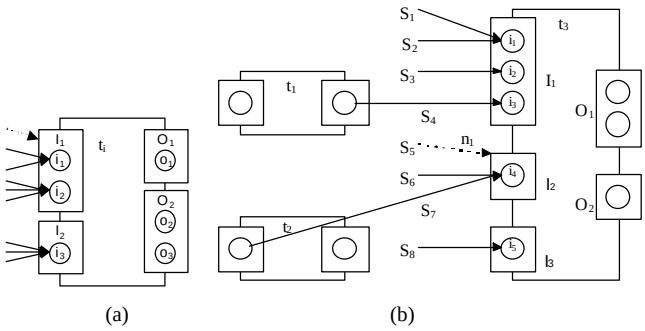


Fig. 2. Workflow schema: task and inter-task dependencies.

To allow applications to be composed from other applications, the task model allows a task to be realised as a collection of tasks, this task is called a *compound task*. A task can either be a *simple task* (primitive task) or a compound task composed from simple and compound tasks. The task model also supports a specialised form of compound task called a *genesis task* to specify workflow applications that contain recursive executions. Its main purpose is to enable dynamic (on demand) instantiation of schema; this provides an efficient way of managing a very large workflow, as only those parts that are strictly needed are instantiated. In a subsequent section we will describe how compound and genesis tasks can be used for structuring the trouble ticket process.

*Workflow Execution Service*: The workflow execution service coordinates the execution of a workflow instance: it records inter-task dependencies of a schema in persistent atomic objects and uses atomic transactions for propagating coordination information to ensure that tasks are scheduled to run respecting their dependencies. The dependency information is maintained and managed by *task controllers*. Each task within a workflow application has a single dedicated task controller. The purpose of a task controller is to receive notifications of outputs of other task controllers and use this information to determine when its associated task can be started. When an input set is ready, the task controller notifies its task. When a task finishes, it notifies its controller, together with the output set produced. In this way a task controller maintains accurate information on the status of its task (waiting, active, complete). The task controller is also responsible for propagating notifications of outputs of its task to other interested task controllers. Each task controller maintains a persistent, atomic object mentioned earlier (called TaskControl) that is used for recording task dependencies. The structure is shown in fig. 3. For example, task controller $tc_3$ will co-ordinate with $tc_1$ and $tc_2$ to determine when $t_3$ can be started and propagate to $tc_4$ and $tc_5$ the results of $t_3$.

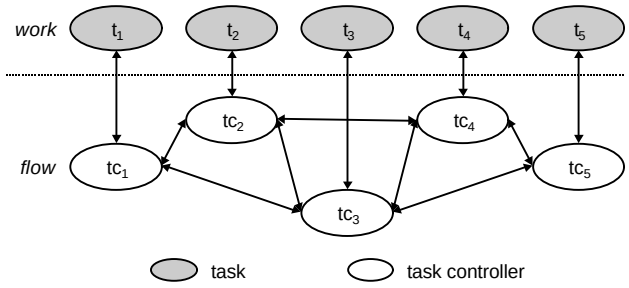


Fig. 3. Tasks and task controllers.

Our system provides a very flexible and dependable task coordination facility that need not have any centralised control [4]. Because the system is built using the underlying transactional layer, no additional recovery facilities are required for reliable task scheduling: provided failed nodes eventually recover and network partitions eventually heal, task notifications will be eventually completed. In addition, the system automatically records the workflow's execution history (audit trail): this is maintained in the committed states of TaskControl objects of the workflow.

Both the repository and execution services provide operations to examine and modify the structure of schemas and instances respectively. These operations can be used for constructing high-level GUI tools [4]. The GUI can be used for observing the execution of a workflow because the GUI can access the TaskControl objects of the workflow execution service and hence can display the starting and completion states of tasks (recall that TaskControl objects store information on dependencies and task states). In addition to this simple monitoring, the GUI can also be used for driving workflow administrative applications for dynamically modifying the execution of a workflow by forcing certain tasks to abort (when possible) or even by adding/removing tasks and dependencies [8].

The workflow execution service specifies an interface that allows the creation of task objects; this interface is called the *TaskFactory* interface. The interface has a single operation *create* that takes a single parameter, *create_criteria*, a sequence of name value pairs (containing information such as where to create, what to create, initialisation data) and returns an object reference to the created task object. Factories for TaskControl objects are also available in a similar fashion. Task factories (and TaskControl factories) provide the basic mechanism available for instantiation of a workflow: the user (or an agent acting on the user's behalf) needs to instantiate all the tasks and corresponding task controllers by invoking create operations on the task (and TaskControl) factories.

Earlier we had stated that any organisation dependent information relating to workflow instantiation and execution should be decoupled from corresponding workflow schemas. Our way of achieving this is very simple, and consists of attaching a create_criteria with each task definition within a workflow schema. It is left to TaskFactories to interpret the sequence of name value pairs of a particular criteria and to take specific set of actions (such as search organisation directory to find a role player and look for location information etc.) concerned with the creation of that task. Many different implementations of the TaskFactory interface can be provided. These task factory objects may be responsible for creating object that represent particular types of tasks or redirect creation requests to other task factory objects. In a subsequent section we describe the specific approach used here.

We conclude our discussion on the workflow system by describing how the trouble ticket process (section 3.1) can be represented by the task model.

### *B Trouble ticket workflow*

Fig. 4 below depicts the structure of the "Trouble Ticket" workflow schema. It shows that the entire process is represented by the "Trouble Ticket"compound task that is composed of five tasks (dotted boxes represent genesis tasks). The "Reproduce Problem and Correct Report"and "Identify and Verify Resolution"tasks, are specified to be genesis tasks which means that the task structure associated those tasks will only be instantiated when the task is started. This means that the instantiation of the trouble ticket task can be performed dynamically when needed. As can be seen, the task model supported by our system, with multiple input and output sets and genesis tasks provides intuitively simple way of modelling the process.

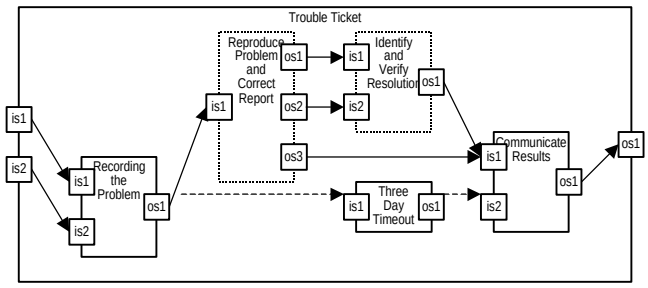


Fig. 4. Trouble ticket workflow

The "Trouble Ticket"task has two input sets (labelled *is1*; which corresponds to starting the task with a "problem from an internal person"and labelled *is2*; which corresponds to starting the task with a "problem from an external person") and a single output set (labelled *os1*; which corresponds to the "outcome"of the task, Produce report).

For the "Reproduce Problem and Correct Report"task, input set *is1* is Obtain report, output sets *os1*, *os2* and *os3* are respectively: Unknown resolution, Probable known resolution and Known resolution. For the "Identify and Verify Resolution" task, output set *os1* is Produce resolution. For the "Communicate Results"task, input sets *is1* and *is2* are respectively, Resolution and No Resolution and output set *os1* is Outcome.

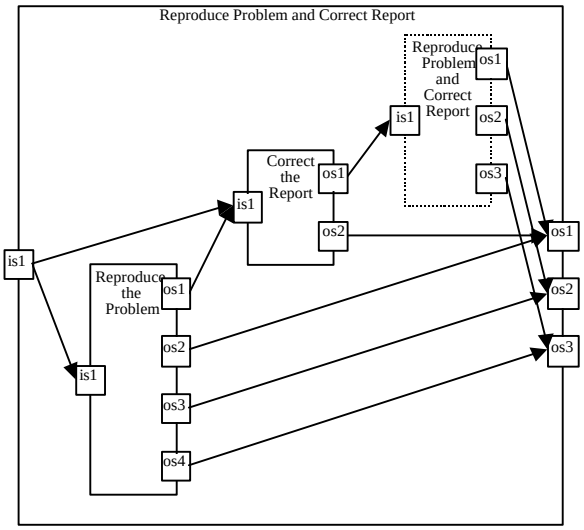


Fig. 5. Reproduce problem and correct report

The "Reproduce Problem and Correct Report" task is actually a compound task, whose internal structure is depicted in fig. 5. The task structure is recursive. Recursion is used here to perform the iteration between step 2 and step 3. The "Identify and Verify Resolution" task is similar in structure to the "Reproduce Problem and Correct Report" task discussed above and its details are not discussed for the sake of brevity.

## V. WORK ACTIVITY COORDINATION SUPPORT

We assume that associated with every task definition is a role that is responsible for carrying out that task; a role may be responsible for carrying out several types of tasks. There could be several workers capable of carrying out a role and each worker could take on several roles. Associated with a role is a task list and a task factory. The associations between all these entities are depicted in fig. 6.

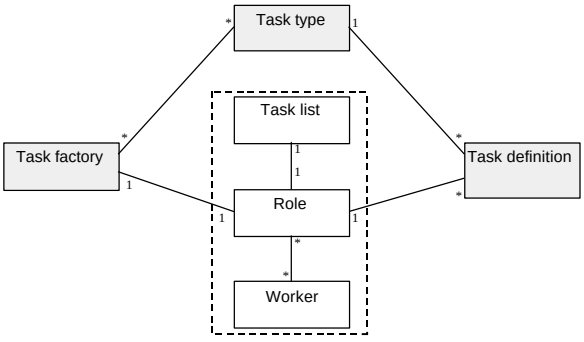


Fig. 6. Entity relationships

The interfaces of the shaded entities are provided by the workflow system, the remaining entities (inside the dotted box) are organisation specific and concerned with work activity coordination. Note that the entities and their relationships shown within the dotted box represent just one specific model that we have implemented; other models are possible. All the physical information required for instantiating the task objects of a workflow is provided by the (organisation specific) implementations of task factories. We describe one such scheme that we have implemented.

We have implemented a two stage workflow instantiation process. The *create_criteria* associated with a task definition has two attribute-value items: role name and task type. This information is passed to the first (initial) task factory as its create operation is invoked. This factory queries the organisation model held in a database (see fig. 7). The database contains, for every role in that organisation, the name of the role's task factory. A role's task factory is capable of creating all types of task objects that role is responsible for. The initial task factory then invokes the create operation of the specific task factory associated with the role, passing the task type. The role's task factory creates the specific task, obtaining all the location specific information from the database; an entry in the tasklist of the role is also made for the task just created (this entry is in the form of a CORBA IOR string). This process is repeated for every task definition in the workflow definition. (Note: we have only discussed the organisational aspects of workflow instantiation, glossing over other aspects, concerned with creation of task controllers and their initialisation with inter-task dependencies, as these are not directly relevant here; details are given in [4]).

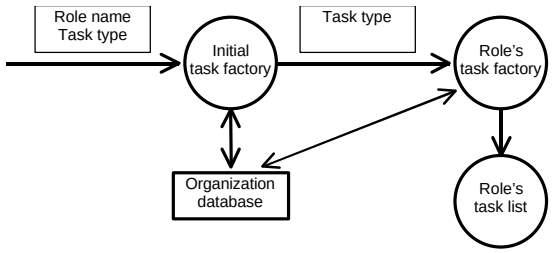


Fig. 7.Task object creation process.

Now a few words on the implementation of tasklists and the user interface. Users are provided a visual interface for tasklist manipulation similar to a mail client. This interface may take the form of an application (e.g., Java application), an applet, or a series of HTML pages (see Fig. 8). Our current implementation in the form of a Java application (shown shaded in the fig.). Tasklists are CORBA transactional objects and are distributed over a variety of server machines in an application specific manner. The organisation database is accessed via an LDAP enabled directory server.

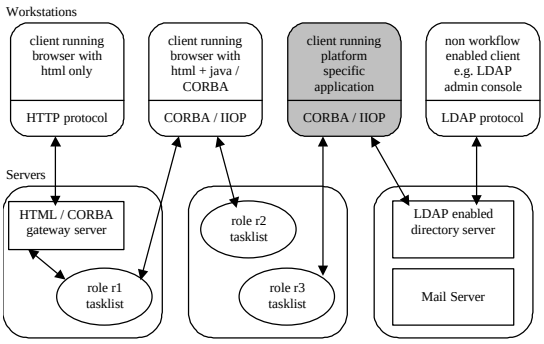


Fig. 8. System components.

The user work activity interface provides two main display areas - the summary list area (left hand side of the window) and the current task area; see a sample screen shot depicted in fig. 9. The summary list contains a listing of all tasks currently assigned to the user. This resembles a list of mail messages, with information such as task subject and priority being displayed. Selecting one of these summary lines causes the task to become current. An application specific representation of the current task occupies the current task area. This may be thought of as similar to displaying the content of a mail message. However, in contrast to mail messages, tasks can implement complex interactive behaviour. For example, a data entry task may be represented as a fill out form, complete with data integrity constraints. The graphical user interface window can be used to interact with the system in a number of ways. When used to access task lists it allows access to number of 'folders', including an 'in tray' (active) of newly assigned tasks, a 'incoming tray' (waiting) of task which may be become active in the future and an 'out tray' (completed), allowing users to maintain links to processes whose tasks they have completed but whose onward progress they may wish to check on periodically. The client interface may also provide facilities for creation of new workflow processes, searching for existing tasks not assigned to the current users, and management processes such as reassignment of tasks to other users.

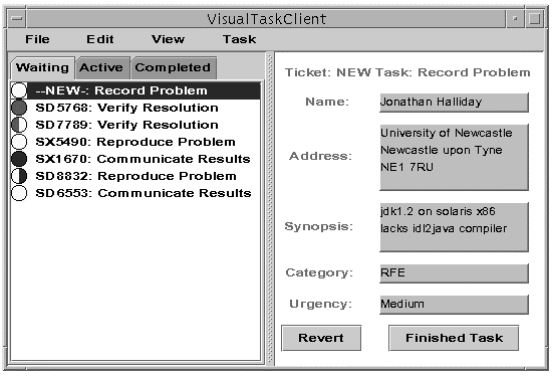


Fig. 9. Tasklist management GUI

A user's work activity interface is initialised as follows. The client software contacts the organisational directory server to obtain the list of roles currently assigned to the user. The client then contacts the directory server again, this time supplying the list of roles, in return for which it receives back a list of the corresponding tasklist object references. Like an email client, the client software is configured to periodically poll the directory server for updates in role information or other changes during the lifetime of the session. The client now iterates over the set of tasklist references to obtain references to the task objects (the CORBA interface provided by the tasklist objects provides a function for the retrieval of references to the set of tasks in the list). This process is repeated at periodic, user configurable intervals during the lifetime of the client (polling), and on the occurrence of certain events (e.g. user request for update, user creation of a new task in one of the tasklists). As the final phase of initialisation, the client conducts a further iteration, this time over the set of all task references obtained previously. From each task it retrieves the information necessary for display of the summary information, such as task type, subject and priority. The tasks are then arranged according to user preference and displayed in the interface.

When the user selects a particular task in the summary list, the task is contacted again and its visual representation retrieved. This is then displayed in the client interface, allowing interaction with state associated with the task. When the user indicates that the task is complete, it is moved from the active tasklist to the completed tasklist.

Having described our approach for work activity coordination we will now revisit the trouble ticket interaction scenario (section 3.2) and describe how we would support such interactions.

*Day 1*: When person A (person with role A) receives the phone call they decide that this call requires a "trouble ticket" workflow application to be initiated. This is done through the GUI which has been tailored for role A. It will allow the specification of the "trouble ticket" workflow application to be selected from the list of all the workflow applications which role A can initiate. The "trouble ticket" workflow is then instantiated and the workflow's initial input, if any, are specified. In the case of the "trouble ticket" this initial input would consist of basic information such as the phone number of the caller and if the call was internal or external. The workflow's instantiation will result in a "Recording the Problem" task appearing in person A's task list. Through the GUI the task can be selected by A and the details of the problem entered; when completed, the task will be removed for person A's task list. When person A instantiated the "trouble ticket" workflow, the GUI will retain a reference to the workflow instance (reference to a task control interface) and this can be used to monitor the progress of the workflow application through the GUI. In this interaction scenario person A could use the GUI to discover that "Reproduce the Problem" task has been assigned to person B (person with role B) and has appeared in his task list (this would be done by the GUI making calls on the object which control and represent task of the workflow application instance).

*Day 2*: Through the GUI person A can check the status of the "Reproduce the Problem" task, and discovers that person B has not completed the task (i.e, the status of the task is active); an e-mail can be sent to person B containing an encoded reference to the task which person A would like completed. Person B could then select and perform the indicated task, in so doing reach the conclusion that Person with role C would be best person to perform the subsequent "Identify Resolution" task. Person B thus has to change the role associated with the "Identify Resolution" task. This is easily achieved because this task, being a genesis task, has not been instantiated yet, so the associated role can be changed. The completion of the "Reproduce the Problem" task would result in an "Identify Resolution" task appearing in person C's task list.

*Day 3*: As before, A uses the GUI to check the status of the "trouble ticket" workflow, and discovers that the "Reproduce the Problem and Correct Report" task has completed but the "Identify Resolution" sub-task of the "Identify and Verify Resolution" task, which is assigned to person C, has not been completed. Through the GUI the priority of this task can be increased, this will be indicated on person C's task list. Person C sees the increase in priority and starts to seek a resolution to the problem, when found the "Identify Resolution" task is completed and then the "Verify Resolution" task appears in person B's task list.

*Day 9*: Person A can use the GUI to examine the execution details of the process which he had initiated. This is possible because (i) A person's GUI holds references to completed tasks (in this case, this would be a reference to the TaskControl object for the "trouble ticket" task; and (ii) the workflow system automatically records the workflow's execution history (audit trail) as maintained in the committed states of TaskControl objects of the workflow, and Person A will be able to "navigate" through the workflow structure and examine for example the states in which each constituent task completed.

## VI. CONCLUDING REMARKS

The paper has discussed design issues of work activity coordination systems for distributed workflow systems and presented a specific implementation and illustrated its features with the help of a detailed example, a customer complaint handling process within an organisation. The system serves as an example of an activity coordination system intended for a distributed organisation, as the entire system architecture - the workflow system as well as the work activity coordination system - is decentralised and open. Further, the workflow and the work activity coordination systems are fault tolerant as they make use of transactions where appropriate. At the same time, the coupling between the workflow system and the work activity coordination system has been kept quite narrow to minimise change dependency between workflow definitions and organisational aspects of workflow executions. In our system, this decoupling has been achieved by using the "familiar" concept of a role, and providing role specific task factories that can interact with organisation database in application specific manner.

## ACKNOWLEDGEMENTS

This work has been supported in part by grants from Nortel Networks, ESPRIT LTR Project C3DS (Project No. 24962) and ESPRIT Project MultiPLECX (Project No. 26810).

## REFERENCES

[1] P. Lawrence (ed.), "WfMC Workflow Handbook," John Wiley & Sons Ltd., 1997.

[2] S. Paul, E. Park and J. Chaar, "RainMan: A Workflow System for the Internet," Proc. of USENIX Symp. on Internet Technologies and Systems, November 1997.

[3] S. Jablonski and C. Bussler, "Workflow management: modelling concepts, architecture and implementation," Intl. Thomson Computer Press, 1996, ISBN 1-85032-222-8.

[4] S.M. Wheater, S.K. Shrivastava and F. Ranno "A CORBA Compliant Transactional Workflow System for Internet Applications ", Proc. Of IFIP Intl. Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware 98, (N. Davies, K. Raymond, J. Seitz, eds.), Springer-Verlag, London, 1998, ISBN 1-85233-088-0, pp. 3-18.

[5] H. Schuster, J. Neeb and R. Schamburger, "A configuration management approach to large workflow management systems," Proc. Joint Intl. Conf. on Work Activity Coordination and Collaboration, WACC'99, San Francisco, Feb. 1999, ACM Software Eng. Notes, March 1999, pp. 177-186.

[6] H. Ludwig and K. Whittingham, "Virtual enterprise co-ordinator - agreement-driven gateways for cross-organisational workflow management," Proc. Joint Intl. Conf. on Work Activity Coordination and Collaboration, WACC'99, San Francisco, Feb. 1999, ACM Software Eng. Notes, March 1999, pp. 29-38.

[7] K. D. Swenson, "Trouble Ticket Workflow Scenario (Version 2)," OMG Document bom/98-02-09.

[8] F. Ranno, S.K Shrivastava and S.M. Wheater, "A Language for Specifying the Composition of Reliable Distributed Applications," 18th IEEE Intl. Conf. on Distributed Computing Systems, ICDCS'98, Amsterdam, May 1998, pp. 534-543.

[9] S K Shrivastava and S M Wheater, "Architectural Support for Dynamic Reconfiguration of distributed workflow Applications," IEE Proceedings -Software, Vol. 145, No. 5, October 1998, pp. 155-162.

[10] G. Hillebrand, P. Krakowski, P. C. Lockemann and D. Posselt, "Integration-based cooperation in concurrent engineering," Proc. of second Enterprise Distributed Object Computing Workshop, EDOC'98, La Jolla, November 1998, pp. 344-355