# A Component-based Workflow System with Dynamic Modifications *

Pinar Koksal Ibrahim Cingil Asuman Dogac

Software Research and Development Center
Department of Computer Engineering
Middle East Technical University (METU)
06531 Ankara Turkiye
asuman@srdc.metu.edu.tr

**Abstract.** Adapting to changes in its environment dynamically is a very important aspect of workflow systems. In this paper, we propose a component-based workflow system architecture specifically designed for this purpose. To allow for easy modification of workflow instances, an instance is designed as an object that contains all the necessary data and control information as well as its execution history. This feature facilitates to dynamically modify the process definition on instance basis at run time. The system is designed to consist of functional components like, Basic Enactment Service, History Manager, Workflow Monitoring Tool, Dynamic Modification Tool, etc. The clients of the system are coded as network-transportable applets written in Java so that the end user can activate workflow system components by connecting to the Workflow Domain Manager over the Internet. In this paper we also present a workflow process definition language $\text{FLOW}_{DL}$, its graphical representation $\text{FLOW}_{GRAPH}$ and a workflow process modification language $\text{FLOW}_{ML}$ and illustrate how the modification process is handled.

## 1 Introduction

A workflow is defined as a collection of processing steps (activities) organized to accomplish some business processes. An activity can be performed by one or more software systems or machines, by a person or a team, or a combination of these. In addition to collection of activities, a workflow defines the order of activity invocations or condition(s) under which activities must be invoked (i.e. control flow) and data-flow between these activities. Activities within a workflow can themselves again be a workflow.

It is widely recognized that one of the basic characteristics that workflow system should provide is flexibility. In a fast-changing environment, companies need to constantly refine their processes in order to effectively meet the constraints

* This work is partially being supported by Middle East Technical University, the Graduate School of Natural and Applied Sciences, Project Number: AFP-97-07.02.08 and by the Scientific and Technical Research Council of Turkey, Project Number: 197E038.

and opportunities proposed by new technology, new market requirements, and new laws. Furthermore, in particular in the first execution of a process, unplanned situations not considered in the design could urge for a modification of the workflow definition [4].

Change in business processes can arise due to three main reasons [22]: *Process Improvement,* which involves performing the same business process with increased efficiency, e.g., organizational restructuring. *Process Innovation,* which involves performing the business process in a radically different way. *Process Adaptation,* which involves adapting the process for unforeseen change, e.g. passing of a new law or handling a special case in student admission.

One of the most challenging issues in the modification of workflows is the management of executions started with the old workflow model. Simple solutions, such as letting the processes finish according to the old model or aborting them, are often inconvenient or impossible to be applied, depending on the notification of the change and the nature of the workflow.

In this paper, we propose a component-based workflow system architecture specifically designed for adapting the business processes to changes in its environment dynamically. We also present a workflow process definition language $\text{FLOW}_{DL}$, its graphical representation $\text{FLOW}_{GRAPH}$ and a workflow process modification language $\text{FLOW}_{ML}$. Afterwards we illustrate how the modification process is handled.

The paper is organized as follows: In Section 2, related work is presented. Section 3 provides the system architecture, namely $\text{FLOW}_{DL}$, $\text{FLOW}_{GRAPH}$ and component-based workflow architecture. Handling dynamic modifications is described in Section 4. The syntax of $\text{FLOW}_{ML}$ and an example are also provided in this section. Finally, the paper is concluded with Section 5.

## 2 Related Work

[4] focuses on workflow modifications involving the flow structure, i.e., the definition of the sequence in which activities should be executed within a process. They propose a complete, minimal and consistent set of primitives that allow generic modification of a workflow, preserving syntactical correctness criteria both when they are applied to a static workflow description and to dynamic workflow instances. Then a taxonomy of policies to manage evolution of running instances when the corresponding workflow schema is modified, is introduced.

Three main policies have been devised to manage workflow instance evolution:

- *Abort:* All workflow instances of old schema are aborted.
- *Flush:* All existing instances terminate following the old schema. When all instances are finished, new instances can start following the new schema.
- *Progressive:* Different decisions for different instances are taken, according to instance's state or its history. Multiple schema versions may exist at the same time. It is the workflow administrator that should analyze running

instances of old workflow schema, and for each of them, define which policy should be applied.

In [17], Liu et.al. propose a handover policy specification language. A handover policy is specified to migrate current running instances of a workflow model to the new workflow model. When a handover policy is applied to an evolution of a workflow model, the running instances may be executing at any task of the old specification. Therefore, different instances may require different handover strategies. A handover policy is defined by a set of handover statements. Three handover aspects of a running instance are described in each handover statement: current position, history and action to be taken. Three actions are supported: rollback, change-over and go-ahead.

In [19], [20], a formal foundation for the support of dynamic structural changes of running workflow instances is presented. Based upon a formal workflow model, ADEPT, a complete and minimal set of change operations, ADEPT-$_{flex}$ is defined. ADEPT$_{flex}$ comprises operations for inserting tasks as well as whole task blocks into a workflow graph, for deleting them, for fast forwarding the progress of a workflow by skipping tasks, for jumping to currently inactive parts of a workflow graph, for serializing tasks that were previously allowed to run in parallel, and for the dynamic iteration and the dynamic rollback of a workflow respectively of a workflow region.

The structural changes are managed differently according to whether an applied change must be preserved until the completion of the workflow (permanent change), or whether it is only of temporary nature (temporary change). If it is a temporary change, then the change should be undone at the next iteration.

In [21], the authors use the clinical application domain to explain and to elaborate the functionality needed to support dynamic workflow changes in an advanced application environment using ADEPT$_{flex}$. In [19] and [20], they have only considered the adhoc changes, that do not affect the original workflow template. However in [21], issues related to the adaptations in the definition of a workflow type are also addressed and migrating the running workflow instances from the old template to the new one is discussed.

[10] presents a formal definition of a dynamic change, and a mathematical approach to its analysis. They use a Petri net formalism to analyze structural change within workflow procedures. Two types of dynamic changes are defined: *immediate*, i.e., changes done on a region take effect immediately, and *quasi-immediate*, i.e., both the old and the new change regions are maintained in the new region. Quasi-immediate change ensures that tokens already in the old change region will finish their progression in the old region.

In [12], changes are differentiated at four different levels: structure level, task level, resource level and system level. Structure level changes affect the interdependencies and sequences of tasks, task level changes are concerned with modifications of individual tasks, resource level changes are concerned with changes of workflow resources, and system level changes refer to adjustments of a concrete execution environment. The authors claim that this separation is very useful for allocating responsibility and controlling change right.

The authors also mention about two popular approaches concerning the adaptation of workflow models; meta-model approach and open-point approach. Meta-model approaches utilize meta-models to determine the structures and types of constituent components of workflow models. A set of primitives is usually defined with which change operations can be performed to a workflow model or even a certain model instance. Open-point approaches set up special points in a workflow model, where adaptation can be made. The concept of adaptation is often generalized, including provision of multiple choices for users, binding of certain resources at runtime, or provision of an open interface through which the late-modeling can be made. A major deficiency of open-point approaches is that they have difficulties to deal with certain structural changes. The approach that have been discussed in [12], supports both the meta-model and open-point approaches.

In [22], the following classes of change for workflows are identified:

- *Flush:* All current instances are allowed to complete according to the old process model.
- *Abort:* An ongoing workflow could be deliberately aborted when the process model is changed.
- *Migrate:* The change affects all current and new instances.
- *Adapt:* This class of change includes cases of errors and exceptions, where the process model does not change, but some instances have to be treated differently because of some exceptional and unforeseen circumstances.
- *Build:* Building of a new process is also a class of process change. The difference is that the starting point is not a detailed pre-existing model, but an elementary description.

The authors in [22], differentiate between two aspects of the workflow model: The *build time* aspect relates to the semantics of the process, and is captured by the process model. The *run time* aspect relates to process instances, and is handled by the process execution model. Then a simple formalization of a workflow, as a directed acyclic graph, is introduced by giving the necessary definitions formally.

After the workflow model is described, a three-phase methodology for dynamic modification is proposed which consists of defining, conforming to and effectuating the modification.

In [18], a family of activity-split and activity-join operations with a notion of validity are described. The Transactional Activity composition Model (TAM) as a concrete underlying environment for the specification of workflows with well defined semantics, is adopted, since TAM has a simple and effective facility feature to allow activity designers to specify the behavioral composition of complex activities and a wide variety of activity interaction dependencies declaratively and incrementally. In the paper, first, basics for activity restructuring operations are described on the TAM. Afterwards, two groups of activity restructuring operations, namely *activity-split* and *activity-join* operations, to allow users or applications to dynamically modify the set of concurrent activities while they are in progress are introduced.

In [14], first, the requirements of workflow evolution are identified. The different propagation strategies of workflow schema changes to their workflow instances that have to be provided by a WFMS are given:

- *Lazy propagation:* A workflow schema is changed without any impact on currently enacting instances. The new workflow schema version becomes only relevant for all new workflow instances.
- *Eager propagation:* Workflow schema changes are propagated immediately to all workflow instances of the changed workflow definition.
- *Selective propagation:* Workflow schema changes are propagated immediately to a selected set of workflow instances of the changed workflow definition.
- *Local modifications and upward propagation:* The propagation is applied to exactly one workflow instance in order to locally customize the workflow structure for a special case or to locally adjust it. This strategy is also useful in the case of processes which cannot be planned completely in advance.
- *Merging:* When changes have to be applied to different workflow variants, some mechanisms are required which support merging of different workflow specifications.

The process modeling, described in [14], is based on object-oriented modeling techniques. Workflow schema and workflow instance elements are modeled as first level objects and their relationships are explicitly maintained. The workflow schema and instance elements are tightly integrated. Workflow schema changes immediately affect all instances since the workflow engine will schedule the task according to the changed schema. To support lazy and selective propagation as well as local modifications of a workflow instance, the schema versioning is used.

## 3 Component-based Workflow System Architecture: METUFlow$_2$

### 3.1 METUFlow$_2$ Process Definition Language: FLOW$_{DL}$

METUFlow$_2$ has a block structured specification language, namely METUFlow$_2$ Process Definition Language (FLOW$_{DL}$). FLOW$_{DL}$ describes the tasks involved in a business process and the execution and data dependencies between these tasks. FLOW$_{DL}$ has also a graphical user interface developed through Java which allows defining a workflow process by accessing METUFlow$_2$ from any computer that has a Web browser [25]. This feature of METUFlow$_2$ makes it possible to support mobile users.

The WfMC have identified a set of six primitives with which to describe flows and hence construct a workflow specification [13]. With these primitives it is possible to model any workflow that is likely to occur. These primitives are: sequential, AND-split, AND-join, OR-split, OR-join and repeatable task. These primitives are all supported by FLOW$_{DL}$ through its block types. FLOW$_{DL}$ contains eight types of blocks, namely, serial, and_parallel, or_parallel, xor_parallel, for_each, contingency, conditional and iterative blocks. Of the above block types,

```
DEFINE_PROCESS OrderProcessing()
  ...
  GetOrder(OUT productNo, OUT quantity, OUT dueDate, OUT orderNo,
           OUT customerInfo)
  EnterOrderInfo(IN productNo, IN quantity, IN dueDate, IN orderNo)
  CheckBillofMaterial(IN productNo, OUT partList)
  PAR_AND (part = FOR EACH partList)
    SERIAL
        DetermineRawMaterial(IN part.No, IN part.Quantity, OUT rawMaterial,
                             OUT required)
        CheckStock(IN rawMaterial, IN required, OUT missing)
        IF (missing > 0) THEN
           VendorOrder(IN rawMaterial, IN missing)
        WithdrawFromStock(IN rawMaterial, IN required)
        GetProcessPlan(IN part.No, OUT processPlan, OUT noofSteps)
        i:=0
        WHILE (i < noofSteps)
           Assign(IN processPlan[i].cellId, IN orderNo, IN part.No,
                  IN part.Quantity, IN rawMaterial, IN required)
        END_WHILE
    END_SERIAL
  END_PAR_AND
  AssembleProduct(IN productNo)
  ...
  Billing(IN orderNo, IN productNo, IN quantity, IN customerInfo)
  ...
END_PROCESS
```

**Fig. 1.** Order Processing Example

serial block implements the sequential primitive. And_parallel block models the AND-split and AND-join primitives. AND-split, OR-join pair is modeled by or_parallel block. Conditional block corresponds to OR-split and OR-join primitives. Finally, repeatable task primitive is supported by the iterative block.

A workflow process is defined as a collection of blocks, tasks and subprocesses. A task is the simplest unit of execution. Processes and tasks have input and output parameters corresponding to workflow relevant data to communicate with other processes and tasks. The term *activity* is used to refer to a block, a task or a (sub)process. Blocks differ from tasks and processes in that they are conceptual activities which are used only to specify the ordering and the dependencies between activities.

An order processing example in a highly automated manufacturing enterprise is provided using FLOW$_{DL}$ [3], [8], [11], [15], [16]. An incoming customer request causes a product order to be created and inserted into an order entry database by *GetOrder* and *EnterOrderInfo* activities respectively (Figure 1). The next step is to determine required parts to assemble the ordered product by *CheckBillofMaterial* activity. A part is the physical object which is fabricated in the manufacturing system. For each part, $DetermineRawMaterial$ activity is executed to find out the raw materials required to manufacture that part,

and a *CheckStock* activity is initiated afterwards to check stock database for the availability of these raw materials. If the required amounts of these raw materials do not exist in the stock, they should be ordered from the external vendors through *VendorOrder*. After all missing raw materials are obtained, required raw materials to fabricate the part is withdrawn from the stock to be sent to the manufacturing cells. This is accomplished by *WithdrawFromStock* activity by decrementing the available amount of the withdrawn raw material (i.e., $quantity(m)$) in the stock database. The required steps to manufacture a part, and the manufacturing cells where these steps are performed are obtained as a result of *GetProcessPlan*. Actual manufacturing activity is initiated by assigning the work to the corresponding cells for each step in *Assign*. Finally, manufactured parts are assembled to form the product that the customer had ordered by the activity *AssembleProduct*. Further downstream activities include a billing activity. *Billing* is responsible for collecting bills of ordered products. *VendorOrder*, *GetProcessPlan* and *Billing* are also workflow processes which should be defined in the same manner as *OrderProcessing*.

In METUFlow$_2$, there are five types of tasks. These are TRANSACTIONAL, NON_TRANSACTIONAL, NON_TRANSACTIONAL with CHECKPOINT, USER and 2PC_TRANSACTIONAL activities. USER activities are in fact NON_TRANSACTIONAL activities. They are specified separately in order to be used by the worklist manager which handles the user-involved activities.

These activity types may have some attributes such as CRITICAL, NON_VITAL and CRITICAL_NON_VITAL. Critical activities can not be compensated and the failure of a non_vital activity is ignored [7], [5]. Besides these attributes, activities can also have some properties like retriable, compensatable, and undoable. A retriable activity restarts execution depending on some condition when it fails. Compensation is used in undoing the visible effects of activities after they are committed. Effects of an undoable activity can be removed depending on some condition in case of failures.

The block structured nature of FLOW$_{DL}$ prevents cyclic definitions and unreachable states. The further advantages brought by this language are summarized in [8].

### 3.2 Graphical Representation of the FLOW$_{\mathrm{DL}}$ : FLOW$_{\mathrm{GRAPH}}$

METUFlow$_2$ system has graphical tools to define a new process definition, to modify the definition dynamically and to monitor the state of the instances, described in detail in the next section. The same graphical representation, called FLOW$_{GRAPH}$, is used at these tools. In FLOW$_{GRAPH}$, each block has a *begin* and *end nodes*. For the AND_PARALLEL, OR_PARALLEL, XOR_PARALLEL and IF blocks, the join node is the *end node*. However SERIAL, CONTINGENCY, WHILE, FOR_EACH blocks have their own *end nodes*. The representation of the blocks in FLOW$_{GRAPH}$ can be seen in Figure 2.

In Figure 2, circles represent the activities. If the activity is a subprocess, it is shown with a thicker circle. Also note that, since a process definition has

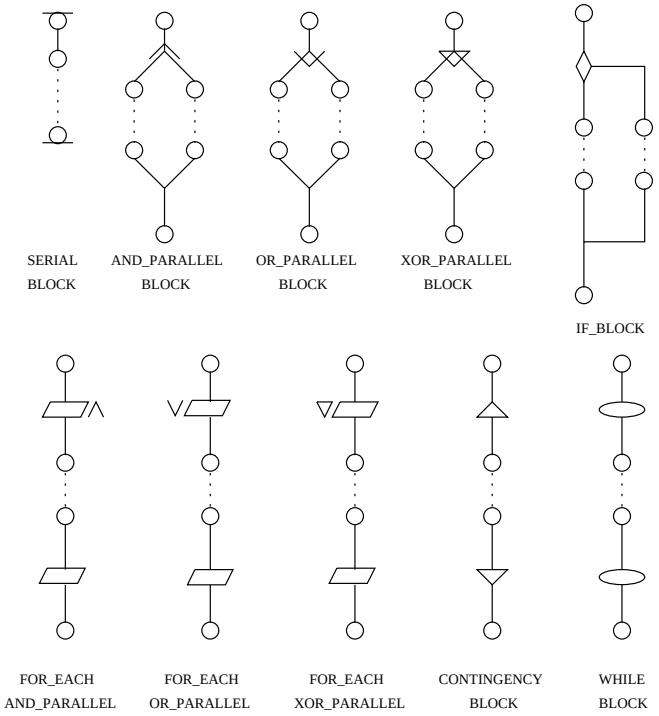


**Fig. 2.** The representation of the blocks in FLOW$_{GRAPH}$

SERIAL_BLOCK characteristics although not defined explicitly, the begin and end of a process are shown similar to that of SERIAL_BLOCK.

The graphical representation of the order processing example, described in Section 3.1 is given in Figure 3.

### 3.3 Component-based Architecture

We have designed a workflow system architecture based on Internet and CORBA with the following features:

- Each process instance is a CORBA object that contains all the necessary data and control information as well as its execution history. This feature makes it possible to dynamically modify the process definition on the instance basis at run time, and to migrate the object in the network to provide load balancing. It should be noted that with this architecture, a site failure affects only the process instances running on that site.
- The system is designed to consist of functional components containing but not restricted to: Basic Enactment Service, User Worklist Manager, Workflow Monitor, Workflow History Manager, Dynamic Modification Tool, Process Definitions Library Manager, Reliable Message Queue Manager, and

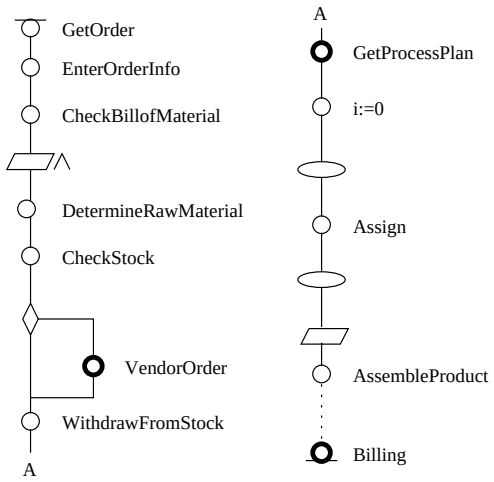


**Fig. 3.** The representation of the order processing example in FLOW$_{GRAPH}$

Workflow Domain Manager. This component-based architecture makes it possible to incorporate the functionality and thus the complexity only when it is actually needed at run time by a process instance by downloading only the necessary components which results in effective usage of system and network resources. It is also possible to add new components or maintain and upgrade the existing components of the system incrementally without effecting the other parts of the system. The component-based architecture facilitates the replication to a great extent. Each site can download its own copy of component server; also the Workflow Domain Manager can be replicated at each site as a Site Manager. This provides for availability and prevents network overhead.
The clients of the system are coded as network-transportable applets written in Java so that the end user can acquire workflow components from the Workflow Domain Manager over the network. Thus it is not necessary to have the software pre-installed on the user machine. This promotes user mobility further as well as easy maintenance of the system components which can be upgraded transparently on the server side.

There are four basic components of the METUFlow$_2$ system architecture shown in Figure 4, as presented in the following:

1. *Component-Server Repository:* The components of the system are implemented as CORBA objects that are invoked by Java applets. The Component-Server Repository contains these applets. The Java applets are downloaded to the client machine when a user through a Web browser accesses the Workflow Domain Manager and asks for a specific service. Thereon the Java applets interact with the user and direct the user requests to the appropriate

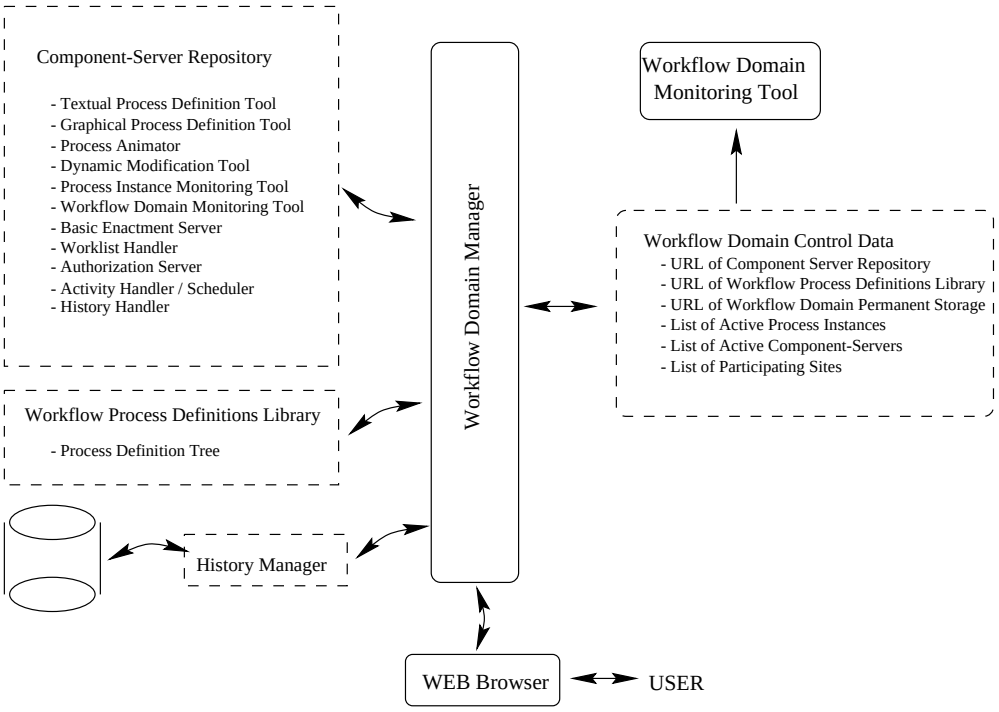


**Fig. 4.** Basic components of the METUFlow$_2$ Architecture

CORBA objects. Some of the components of our system are listed in the following:

- *Workflow Process Definition Tool;* to define new workflow processes.
- *Workflow Dynamic Modification Tool;* to modify previously defined workflow processes that are stored in the Workflow Process Definition Library and/or a particular workflow process instance.
- *Workflow Process Instance (WPI) Monitoring Tool;* to trace workflow process instances that have been initiated and extract run-time information about the current execution status of an instance.
- *Monitoring and Measurement Tool;* to collect and measure process enactment data needed to improve subsequent process enactment iterations as well as documenting what actions actually occurred in what order.
- *Enactment History Capture and Replay;* to simulate the re-enactment of a process graphically in order to more readily observe process state transitions or to intuitively detect possible process enactment anomalies.

2. *Workflow Process Definitions Library:* Workflow definitions (i.e. process templates), organizational role definitions, participant-role assignments are durably stored in this library. Only Workflow Specification Tool and Dynamic

Modification Tool inserts or updates workflow process templates in this library. This library is maintained by the WFMS Library Manager.
Different workflow schema versions have to be managed and different propagation strategies of workflow schema changes to their workflow instances have to be provided by a WFMS in order to flexibly support the migration from one business process to an improved one, to support alternative workflows for process variants, and to support adhoc changes of a workflow [14]. When the workflow definition is modified permanently, the versions of workflow definitions are stored, since:

- In some cases, it may be necessary to recover to the old workflow definition. For example, when it is observed that the new definition performs worse than the old definition.
- It may be desired that more than one version of definitions are active at the same time. That is, some instances are created from one version, and some others from a different version of the definition.

In the METUFlow$_2$ architecture, to handle the versioning of definitions, a *definition tree* is kept to provide the administrator the flexibility of modifying a definition several times. During the modification, the administrator selects one version, default being the last one. Thus new instances are created from the default definition, if the version number of workflow definition is not identified explicitly during the instance creation.

3. *History Manager:* The History Manager handles the database that stores the information about workflow process instances which have been enacted to completion to provide history related information to its clients (e.g. for data mining purposes). It should be noted that the history of active process intances are stored in the process instance object.
4. *Workflow Domain Manager:* The Domain Manager is the Web server of the system. All clients access to the Domain Manager via their Web browsers and in response to their authorized service requests, the Domain Manager downloads appropriate Java applets to the client which then handles subsequent requests of the same client for that particular service which is provided by a component server. If the client needs a different WFMS service, the Domain Manager is then accessed again via the Web browser and another Java applet is downloaded. The Domain Manager keeps runtime information such as list of active process instances, active component servers, list of participating sites, etc. for domain monitoring purposes.

The run-time system despite having a central control on a process instance basis, brings out all the benefits of highly distributed environments. Each WPI may execute at a different site. Component-Server Repository, Workflow Definition Library, Workflow Domain Control Data and Workflow Domain Manager may all be replicated for better performance and availability. Each participating site may have its own replication of Workflow Domain Manager as the Site Manager. Since no prior installation of any WFMS software is required on the client side the system is highly dynamic and thus any component-server implementation may be upgraded at the server side without needing any changes on

the client side. In addition a site failure can be overcome simply by migrating the instances to be executed on that site to another site/other sites. Detailed work on the component-based workflow system architecture can be found in [6].

## 4 Handling Dynamic Modifications in METUFlow$_2$

The set of running instances of a workflow definition can be called as *instance domain*. The modifications can be applied to none of the instances, to a single instance, to a set of instances, or to all of the instances of the *instance domain* depending on the modification that has been done and what the modification administrator, who has granted to make modifications on workflow definitions, defines as the domain that the modifications are applied. For example, a modification can be applied on the instances which have passed a particular point on the execution flow or a modification can not be applied to some of the instances since they have passed the critical point. The administrator can indicate the domain on which the modifications are applied. If the domain is not given, the modification is applied to all of the instances.

The changes can be classified in two groups, as permanent and temporary changes:

- For permanent changes, the workflow definition is changed permanently, so that the new instances are created from the new definition by default. The running instances may also be selectively migrated to the new definition.
- For temporary changes, the modification is only applied to the running instances, but not to the workflow definition. For example, there may be some user activities which are assigned to the users by adding the activity to their worklists, in the workflow definition. If a user is absent temporarily, because of illness for example, her/his activities can be assigned to another user who takes the responsibility of the activities of the absent user.

In our system, dynamic modification of an instance and/or a workflow definition template can be initiated in two ways: either by a user or by means of a special activity specified in the process definition as explained in the following:

- A user via her/his Web browser may access the Workflow Domain Manager and download the Dynamic Modification Tool which helps the administrator make necessary changes on the workflow definition and/or the running instances. Modifications on the workflow definition can only be done by authorized users.
  Dynamic Modification Tool asks the Authorization Server about the modification grant of the user whether the user can modify the definition, or not. Three different grants can be given to the users according to their roles by the Authorization Server:
  - *modify-permanently;* given to the users, like system administrator, to modify the workflow definition template and/or some/all of the process instances in the instance domain.

- *modify-temporarily-all;* given to the users to modify some/all of the instances in the instance domain temporarily. These users, who have this type of grant, can not modify the workflow definition template.
- *modify-temporarily-own;* given to the users to modify only the instances that they are the owners. These users also can not modify the workflow definition template.

If the user has taken any one of the modification grants, s/he chooses a workflow definition to update. The information about definitions can be obtained from Workflow Process Definition Library through Workflow Domain Manager and the set of running instances can be obtained from the Workflow Domain Control Data of the Workflow Domain Manager.

– Workflow process definition may contain a special activity called Workflow Process Modification Activity (WPMA) that (when executed) automatically invokes the WPI Dynamic Modification Tool on behalf of a user so that the user can modify the process instance. The WPMA handles instance-specific differences of the process definition when necessary. Each specification of the WPMA activity results in a separate modification of the instance. A WPMA initiated modification may not affect other instances of the same workflow process or the workflow definition template.

After the modification process is initiated by any one of the ways described above, the workflow definition is represented graphically using the FLOW$_{GRAPH}$.

The user can make the following modifications on this graphical definition using the Dynamic Modification Tool:

– A new activity can be defined, and inserted in the workflow definition.
– New control dependencies can be given, or they can be changed.
– Conditions can be updated or a new one can be given.
– The values of workflow relevant data can be modified.
– Block types can be updated.
– A user or a role, assigned to a user activity, can be changed.
– Activities can be deleted.

In addition to these modifications and augmentations, the domain can be specified to identify the instances that the modifications are to be applied, along with the type of modification, whether permanent or temporary.

After all of the necessary information are gathered from the user, by going through both the old and the new workflow definitions, the modification region is determined. A *modification region* contains the minimum part of the definition that includes all the modifications, that is, starts with the first modified activity and ends with the last one. An example is given in Figure 5.

If the modification is to be applied to the running instances, the modification region is checked for the critical points, if there are any in the workflow definition. If the modification region is after the critical points in the execution flow, then all the instances of this definition can be adapted to the new definition. However if a critical point is after the modification region, execution states of the instances

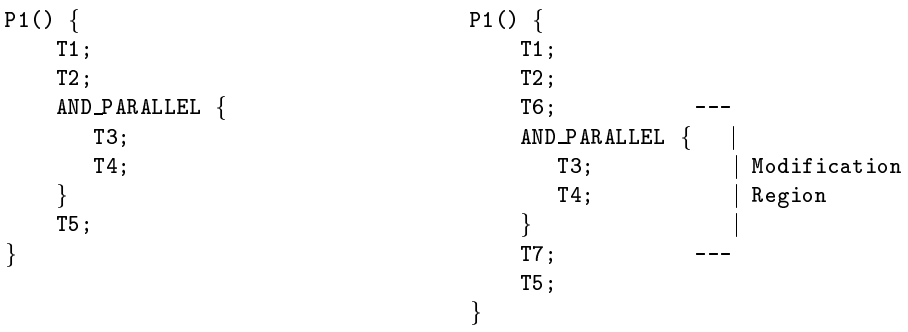


**Fig. 5.** A Modification Region Example

should be checked. If their executions have passed the critical point and the critical activity needs to be compensated to migrate the running instance to the new definition, then the modification should be rejected for these instances. If the critical point has not been executed yet, then the modification can be applied to these instances.

The instances that the modification can be applied, are grouped according to their execution states:

- The instances whose execution states has not reached to the modification region yet, are directly adapted to the new definition.
- If the first activity of the modification region is running then this activity is aborted, and these instances can continue their executions from the new workflow definition.
- If the execution is either running in the modification region or has passed the region, then the execution of these instances are held on. The activities until the beginning of the modification region are rolled back according to a compensation strategy. Afterwards their execution can continue using the new schema. For an activity (if it is not a critical activity), if a compensation activity is not given, this means that there is no need to compensate this activity during recovery. Also note that, critical activities can not be recovered, therefore they do not have compensation activities.

Dynamic Workflows which have no pre-specified process definition can be handled with another special activity called Dynamic Workflow Special Activity (DWSA) that automatically invokes the Dynamic Modification Tool on behalf of a user so that the user can specify the next activity to be executed. A dynamic workflow process definition initially includes only one activity, the DWSA. When this process is initiated, the DWSA invokes the Dynamic Modification Tool and awaits the user to specify activities to be executed. When the user specifies the next activity or activities, another DWSA is appended automatically such that after the user-specified next activity(s) is executed, the DWSA will be invoked again. The DWSA will not be appended only if the user explicitly indicates that

no more activities are to be specified in which case the termination of DWSA will indicate the termination of the process instance. In this way a workflow process can interactively be defined on-the-fly by a user and it is saved in the Workflow Process Definition Library if the user specifies so at the terminating DWSA.

### 4.1 METUFlow$_2$ Modification Language: FLOW$_{ML}$

The user who has a grant to modify a workflow definition or its running instances, should provide:

- an action, the modification that should be made. The user can provide this information using our modification language, FLOW$_{ML}$ as:

```
{ ADD | MODIFY | DELETE } { PROCESS <processname> |
                            TASK <taskname> |
                            BLOCK <blkname> |
                            CONDITION AT <activityname> |
                            WRD <wrdname> | USER AT <activityname> |
                            ROLE AT <activityname> } [AS <new defn> ]
```

- a place, where the modification is applied, can be given using FLOW$_{ML}$ as:

```
[ AFTER { <activityname> | <blkname> } | BEFORE { <activityname> |
  <blkname> } | IN { <activityname> | <blkname> } ]
```

- a domain that includes the instances to which the modification is applied, by providing the object references or the execution states of the instances. This information can be given using FLOW$_{ML}$ as:

```
DOMAIN [ALL | NONE] <processname>
[ WHICH OBJ_REF <comparison_op> objref |
  BEFORE { <activityname> | <blkname> } STARTS |
  AFTER { <activityname> | <blkname> } COMMITS |
  AT { <activityname> | <blkname> } EXECUTING ]
```

- the type of the modification, permanent or temporary, can be given as:

```
[PERMANENTLY | TEMPORARILY]
```

More than one modification statements can be combined with **AND** connector.

The user can use either our modification language, FLOW$_{ML}$, or graphical dynamic modification tool to specify the modifications or additions.

After the modification of the processes, the modified process definition is checked for the following:

– If a new activity is defined, its input parameters are checked whether they have been defined or not, before the activity.
– Task, block and process names that appear in the "place" or "domain" part of the FLOW$_{ML}$ are checked whether they exist in the old definition or not.
– For DELETE and MODIFY statements, the validity of task names, block names, conditions, role names, user names and wrd names are checked.
– For DELETE statements, it is checked that whether the deletion affects the input and output parameters of other activities.

### 4.2 An Example

The manufacturer may decide to modify their billing process as requesting some percentage of the total payment in advance before the manufacturing steps have started. Therefore a new *"RequestPayment"* activity may be added after the activity *"EnterOrderInfo"*. Additional changes should be handled at the *Billing* subprocess also. In METUFlow$_2$, these modifications can be defined either graphically by using the Dynamic Modification Tool, or textually by FLOW$_{ML}$. The FLOW$_{ML}$ statements for these modifications are as follows:

```
ADD TASK RequestPayment (IN int orderNo, IN int productNo,
                         IN int quantity, IN custumerStruct customerInfo,
                         OUT double amountPaid)
AFTER EnterOrderInfo
AND
MODIFY PROCESS Billing
AS Billing (IN int orderNo, IN int productNo, IN int quantity,
             IN custumerStruct customerInfo, IN double amountPaid)
AND
MODIFY TASK Payment
AS Payment (IN int orderNo, IN int productNo, IN int quantity,
             IN custumerStruct customerInfo, OUT double amount,
             OUT int paymentStatus, IN double amountPaid)
DOMAIN ALL OrderProcessing
PERMANENTLY;
```

First FLOW$_{ML}$ statement adds a new activity *"RequestPayment"* after the activity *"EnterOrderInfo"*. Second and third statements add a new *IN* parameter to the *"Billing"* subprocess and the *"Payment"* task respectively. The *"RequestPayment"* task should be written and the operation logic of the *"Payment"* task should also be changed accordingly. However from a workflow point of view, a WFMS does not have the responsibility of providing these changes. This modification is applied to all of the instances of the process *"OrderProcessing"* and the definition of the process is also modified permanently. This means that a new version of the definition is created and stored in the Process Definitions Library.

## 5 Conclusion and Future Work

Business processes need to be constantly refined in order to effectively meet the constraints and opportunities proposed by new technology, new market requirements, and new laws. Workflow Management Systems, which are used for the development of business applications, should provide the facilities to manage the dynamic modification of running instances to the modified definition. The component-based architecture that we propose in this paper facilitates dynamic modification on an instance basis and avoids process template modification problems by keeping the process definition for each instance separately. After The user provides the modifications to the process definition either by using $\text{FLOW}_{ML}$ or by using graphical modification tool, the Dynamic Modification Tool determines on instance basis how the migration of instances to the new definition can be handled, and without any further user interaction, the instances are migrated.

During the migration of the running instances to the new process definition, sometimes the need may arise to rollback some of the committed tasks using compensation tasks. In many situations there is no need to compensate all of the tasks, since the modification region has not affected all of them. Therefore during roll-back operation, the Modification Tool determines which tasks to be compensated according to the modification region. To make this automatic, the dependence between the tasks should be determined automatically. Our work on determining task interdependencies according to the data and control flow between them still continues.

## References


1. N. Adam, V. Atluri, W. K. Huang; "Modeling and Analysis of Workflows Using Petri Nets", Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Volume 10, Issue 2, March 1998.
2. G. Alonso, and H. J. Schek; "Research Issues in Large Workflow Management Systems", Proc. of NFS Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, Edited-by A. Sheth, Athens, Georgia, May 1996.
3. I. B. Arpinar, S. (Nural) Arpinar, U. Halici, and A. Dogac; "Correctness of Workflows in the Presence of Concurrency", Intl. Conf. on Next Generation Info. Tech. and Sys., Israel, July 1997.
4. F. Casati, S. Ceri, B. Pernici, G. Pozzi, "Workflow Evolution", Data and Knowledge Engineering, Volume 24, Issue 3, pp. 211-238, January 1998.
5. Q. Chen, U. Dayal, "A Transactional Nested Process Management System", Proc. of the 12th Intl. Conf. on Data Engineering, New Orleans, Louisiana, USA, February 1996.
6. I. Cingil, A. Dogac, "A Component-based System Architecture for Adaptable Workflow Systems", Technical Report 98-2, Software Research and Development Center, Dept. of Computer Engineering, Middle East Technical University, 1998.
7. U. Dayal, M. Hsu, R. Ladin, "A Transaction Model for Long-running Activities", Proc. of the 17th Intl. Conf. on Very Large Databases, pages 113-122, September 1991.

8. A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, I. B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, M. Altinel, "Design and Implementation of a Distributed Workflow Management System: METUFlow", In: [9].
9. A. Dogac, L. Kalinichenko, M. T. Ozsu, and A. Sheth (eds.), "Advances in Workflow Management Systems and Interoperability", Springer Verlag, 1998.
10. C. Ellis, K. Keddara, and G. Rozenberg, "Dynamic Change Within Workflow Systems", Proc. of the ACM Conf. on Organizational Computing Systems, 1995.
11. E. Gokkoca, M. Altinel, I. Cingil, N. Tatbul, P. Koksal, A. Dogac, "Design and Implementation of a Distributed Workflow Enactment Service", Proc. of Intl. Conf. on Cooperative Information Systems, Charleston, USA, June 1997.
12. Y. Han, A. Sheth, "On Adaptive Workflow Modeling", 4th Intl. Conf. on Information Systems Analysis and Synthesis, Orlando, Florida, July 1998.
13. D. Hollinsworth, "The Workflow Reference Model", Technical Report TC00-1003, Workflow Management Coalition, December 1996. Accessible via: http://www.aiai.ed.ac.uk/WfMC/.
14. G. Joeris, O. Herzog, "Managing Evolving Workflow Specifications", 3rd Intl. Conf. on Cooperative Information Systems, COOPIS'98, New York, August 1998.
15. P. Karagoz, S. Arpinar, P. Koksal, N. Tatbul, E. Gokkoca, and A. Dogac, "Task Handling in Workflow Management Systems", Intl. Workshop on Issues and Applications of Database Technology, Berlin, June 1998.
16. P. Koksal, S. Arpinar, and A. Dogac, "Workflow History Management", ACM Sigmod Record, Vol. 27, No. 1, March 1998.
17. C. Liu, M. E. Orlowska, H. Li, "Automating Handover in Dynamic Workflow Environments", CAiSE 1998, pp. 139-157.
18. L. Liu, C. Pu, "Methodical Restructuring of Complex Workflow Activities", Intl. Conf. on Data Engineering, ICDE '98, 1998.
19. M. Reichert, P. Dadam, "A Framework for Dynamic Changes in Workflow Management Systems", in Proc. of DEXA'97, September 1997, Toulouse, France.
20. M. Reichert, P. Dadam, "ADEPT_flex-Supporting Dynamic Changes of Workflows Without Loosing Control", in Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management, Volume 10, Issue 2, March 1998.
21. M. Reichert, C. Hensinger, P. Dadam, "Dynamic Workflow Changes in Clinical Application Environments", in Proc. of EDBT-Workshop, 1998.
22. S. W. Sadiq, M. E. Orlowska, "On Dynamic Modification of Workflows", Technical Report, July 1998.
23. A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, A. Wolf, Report from the NSF workshop on workflow and process automation in information systems, in ACM SIGMOD Record, 25(3):55-67, December 1996.
24. A. Sheth, K. Kochut, "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems", in [9].
25. E. Turanalp, "Design and Implementation of a Graphical Workflow Definition Tool", Master Thesis, Department of Computer Engineering, Middle East Technical University, Ankara, Turkiye, 1997.