IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITJ2507

A Systematic Approach to Flexible Specification, Composition, and Restructuring of Workflow Activities

Ling Liu and Calton Pu, Georgia Institute of Technology, USA
Duncan Dubugras Ruiz[1], Pontifical Catholic University of RS, Brazil

ABSTRACT

We introduce the ActivityFlow specification language for flexible specification, composition, and coordination of workflow activities. The most interesting features of the ActivityFlow specification language include: (1) a collection of specification mechanisms, allowing workflow designers to use a uniform workflow specification interface to describe different types (i.e., ad-hoc, administrative, or production) of workflows involved in their organizational processes– this feature helps to increase the flexibility of workflow processes in accommodating various types of changes; (2) a set of activity modeling facilities, enabling workflow designers to describe the flow of work declaratively and incrementally, allowing to reason about correctness and security of complex workflow activities independently from their underlying implementation mechanisms; (3) an open architecture that supports user interaction as well as collaboration of workflow systems of different organizations, and a set of workflow activity restructuring operators to respond to dynamic changes of workflow activities. We end the paper with a series of simulation-based experiments that demonstrate the effectiveness of these restructuring operators and the implementation architecture of the ActivityFlow system.

Keywords: *business process; complex workflow activities; workflow evolution; extended transactions; information system engineering; workflow management.*

INTRODUCTION

The focus of office computing today has shifted from automating individual work activities to supporting the automation of organizational business processes. Examples of such business processes include handling bank loan applications, processing insurance claims, and providing telephone services. Such requirement shift, pushed by the technology trends, has promoted the emergence of a new computing infrastructure, workflow management systems (WFMSs), which provides a model of business processes, and a foundation on which to build solutions supporting the coordination, execution, and management of business processes (Hsu and Kleissner, 1996). One of the main challenges in today's WFMSs is to provide tools to sup-

port organizations to coordinate and automate the flow of work activities between people and groups within an organization, and to streamline and manage business processes that depend on both information systems and human resources.

Workflow systems have gone through three stages over the last decade (McCarthy and Bluestein, 1991; Gawlick, Hsu and Obermarck, 1994). First, homegrown workflow systems were monolithic in the sense that all control flows and data flows were hard-coded into applications, thus they are difficult to maintain and evolve. The second generation of workflow systems was driven by imaging/document management systems or desktop object managements. The workflow components of these products are usually tightly coupled with the production systems. Typical examples are smart form systems (e.g., expense report handling), and case folder systems (e.g., insurance claims handling). The third generation workflow systems have an open infrastructure, a generic workflow engine, a database or repository for sharing information, and use middleware technology for distributed object management. Several research projects are contributing towards building the third generation workflow systems (Sheth, 1995; Sheth et al., 1996; Mohan, 1994). Examples include Exotica (Mohan, Alonso, Gunthor and Kamath, 1995) from IBM, InConcert from Xerox, ObjectFlow from DEC (Hsu and Kleissner, 1996), and WorkManager from HP. For an extensive survey of the workflow automation software products and prototypes, see Georgakopoulos, Hornick and Sheth (1995).

Although there are more and more successes in the workflow research and development, it is widely recognized (Mohan, 1994; Sheth et al., 1996) that there are still technical problems, ranging from inflexible and rigid process specification and execution mechanisms, and insufficient possibilities to handle exceptions, to the need for uniform interface support for various types of workflows (i.e., ad-hoc, administrative, or production workflows), for dynamic restructuring of business processes, process status monitoring, automatic enforcement of consistency and concurrency control, and recovery from failure, and for improved interoperability between different workflow servers. As pointed out by Sheth et al. (1996), many existing workflow management systems use a petri-net based tool for process specification. The available design tools typically support definition of control flows and data flows between activities by connecting the activity icons with specialized arrows, specifying the activity precedence order and their data dependencies. In addition to graphical specification languages, many workflow systems provide rule-based specification languages (Dayal et al., 1990; Georgakopoulos et al., 1995). Although these existing workflow specification languages are powerful in expressiveness, one of the common problems (even those based on graphical "node and arc" programming models) is that they are not "well-structured". Concretely, when used for modeling complex workflow processes without discipline, these languages may result in schemas with intertwined precedence relationships. This makes debugging, modifying, and reasoning of complex workflow processes difficult (Liu and Meersman, 1996).

In this paper, we concentrate our discussion on the problem of flexibility and extensibility of process specification and execution mechanisms. We introduce the ActivityFlow specification language for structured specification and flexible coordination of workflow activities. The most interesting features of the ActivityFlow

specification language include:

- A collection of specification mechanisms, which allows the workflow designer to use a uniform workflow specification interface to describe different types (i.e., ad-hoc, administrative, or production) of workflows involved in their organizational processes, and helps to increase the flexibility of workflow processes in accommodating changes;
- A set of activity modeling facilities, which enable the workflow designer to describe the flow of work declaratively and incrementally, allowing reasoning about correctness and security of complex workflow activities independently from their underlying implementation mechanisms; and
- An open architecture, which supports user interaction as well as collaboration of workflow systems of different organizations.

The rest of this paper proceeds as follows. In the next section we describe the basic concepts of ActivityFlow and highlight some of the important features. Then we present our ActivityFlow specification language and illustrate the main features of the language using the telephone service provisioning workflow application as the running example. We describe a set of workflow activity restructuring operators and how they can be used in response to dynamic change of ActivityFlow models, including a series of simulation-based experiments to demonstrate the effectiveness of these restructuring operators. We discuss the implementation architecture of ActivityFlow and implementation-related issues, and conclude the paper with a discussion on related works and a summary.

BASIC CONCEPTS OF ACTIVITYFLOW

Business Process vs Workflow Process

Business processes are collection of activities that support critical organizational and business functions. The activities within a business process have a common business or organizational objective, and are often tied together by a set of precedence dependency relationships. One of the important problems in managing business processes (by organization or human) is how to effectively capture the dependencies among activities and utilize the dependencies for scheduling, distributing, and coordinating work activities among human and information system resources efficiently.

A workflow process is an abstraction of a business process, and it consists of *activities*, which correspond to individual process steps, and *actors*, which execute these activities. An actor may be a human (e.g., a customer representative), an information system, or any combinations of the two. A notable difference between business process and workflow process is that a workflow process is an automated business process, namely the coordination, control and communication of activities are automated, although the activities themselves can be either automated or performed by people (Sheth et al., 1996).

A workflow management system is a software system which offers a set of workflow enactment services to carry out a workflow process through automated coordination, control and communication of work activities performed by both human and computers. An execution of a workflow process is called a workflow case

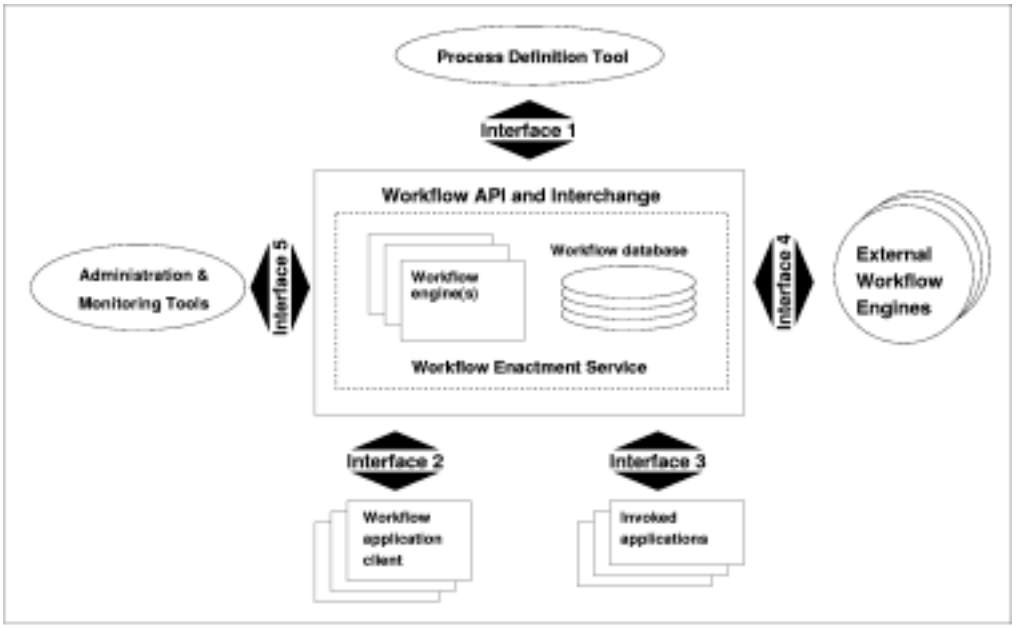
(Hollingsworth and WfMC, 1995; WfMC, 2003). Users communicate with workflow enactment services by means of workflow clients, programs that provide an integrated user interface to all processes and tools supported by the system.

Reference Architecture

Figure 1 shows the WFMS reference architecture provided by the Workflow Management Coalition (WfMC) (Hollingsworth and WfMC, 1995). A WFMS consists of an engine, a process definition tool, workflow application clients, invoked applications, and administration and monitoring tools. The process definition tool is a visual editor used to define the specification of a workflow process, and we call it workflow process schema in ActivityFlow. The same schema can be used later for creating multiple instances of the same business process (i.e., each execution of the schema produces an instance of the same business process). The workflow engine and the surrounding tools communicate with the workflow database to store, access, and update workflow process control data (used by the WFMS only), and workflow process-specific data (used by both application and WFMS). Examples of such data are workflow activity schemas, statistical information, and control information required to execute and monitor the active process instances. Existing WFMSs maintain audit logs that keep track of information about the status of the various system components, changes to the status of workflow processes, and various statistics about past process executions. This information can be used to provide real-time status reports about the state of the system and the state of the active workflow process instances, as well as various statistical measurements, such as the average execution time of an activity belonging to a particular process schema, and the timing characteristics of the active workflow process instances.

ActivityFlow discussed in this paper

Figure 1: Reference Architecture of Workflow Management Coalition



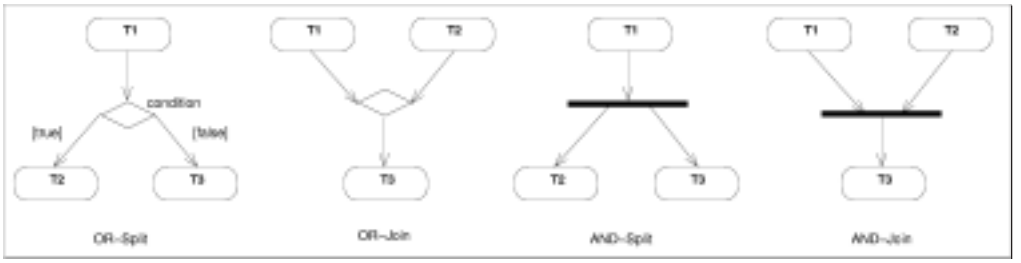
can be seen as a concrete instance of the WfMC reference architecture in the sense that in ActivityFlow, concrete solutions are introduced for process definitions, workflow activity enactment services, and interoperability with external workflow management systems. Our focus is on the ActivityFlow process definition facilities, including the ActivityFlow meta-model , the ActivityFlow workflow specification language and graphical notation for ActivityFlow process definition based on UML Activity diagrams.

ActivityFlow Meta Model

The ActivityFlow meta-model describes the basic elements that are used to define a workflow process schema, which describes the pattern of a workflow process and its coordination agreements. In ActivityFlow, a workflow process schema specifies activities that constitute the workflow process and dependencies between these constituent activities. Activities represent steps required to complete a business process. A step is a unit of processing and can be simple (primitive) or complex (nested). Activity dependencies determine the execution order of activities and the data flow between these activities. Activities can be executed sequentially or in parallel. Parallel executions may be unconditional, i.e., all activities are executed, or conditional, i.e., only activities that satisfy the given condition are executed. In addition, activities may be executed repeatedly, and the number of iterations may be determined at run-time.

A workflow process schema can be executed many times. Each execution is called a workflow process instance (or a workflow process for short), which is a partial order of activities and connectors. The set of activity-precedence-dependency relationships defines a partial order over the given set of activities. The connectors represent the points where the control flow changes. For instance, the point where control splits into multiple parallel activities is referred to as *split point* and is specified using a split connector. The point where control merges into one activity is referred to as *join point*, and is specified using a split connector. A join point is called AND-join if the activity immediately following this point starts execution only when all the activities preceding the join point finish execution. A join point is called OR-join when the activity immediately following this point starts execution as soon as one of the activities preceding the join point finishes execution. A split point that can be statically determined (before execution) in which all branches are taken is called AND-split. A split point which can be statically determined in which exactly one of the branches will be taken is called OR-split. Figure 2 lists the typical graphical representation of AND-split, OR-split, AND-join, and OR-

Figure 2: UML Graphical representation of AND-split, OR-split, AND-join, and OR-join



join by the use of UML activity diagram constructs (Rumbaugh, Jacobson and Booch, 1999; Fowler and Scott, 2000).

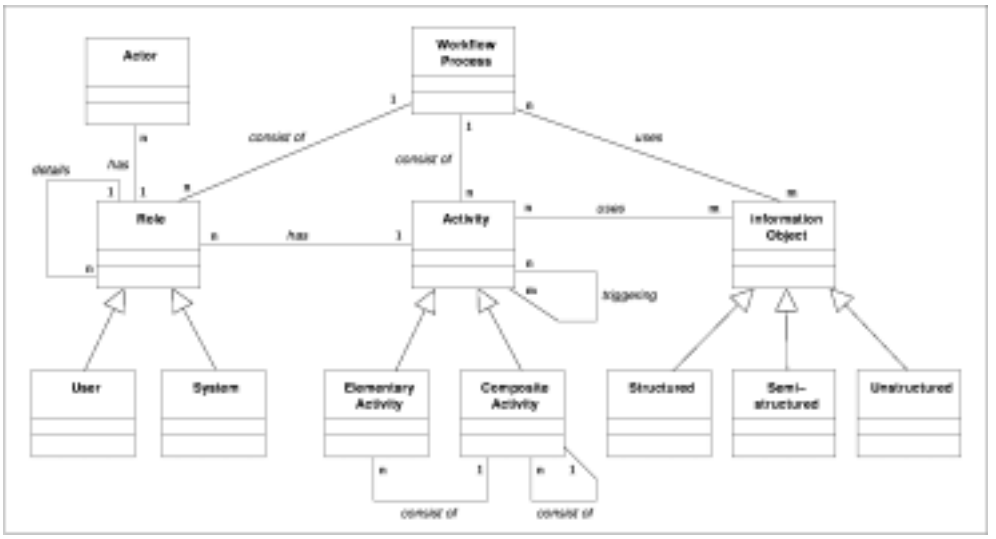
The workflow process schema also specifies which actors can execute each workflow activity. Such specification is normally done by associating *roles* with activities. A role serves as a "description" or a "place holder" for a person, a group, an information system, or any of the combinations required for the enactment of an activity. Formally, a role is a set of actors. Each activity has an associated role that determines which actors can execute this activity. Each actor has an activity queue associated with it. Activities submitted for execution are inserted into the activity queue when the actor is busy. The actor follows its own local policy for selecting from its queue for next activity to execute. The most common scheduling policies are priority-based and FIFO. The notion of a role facilitates load balancing among actors and can flexibly accommodate changes in the workforce and in the computing infrastructure of an organization, by changing the set of actors associated with roles.

Figure 3 shows a sketch of the ActivityFlow meta-model using the UML class diagram constructs (Rumbaugh et al., 1999; Fowler and Scott, 2000). The following concepts are the basics of the activity-based process model:

- *A workflow process* consists of a set of activities and roles, and a collection of information objects to be accessed from different information resources.
- *An activity* is either an *elementary* activity or a *composite* activity. The execution of an activity consists of a sequence of interactions (called events) between the performer and the workflow management system, and a sequence of actions that change the state of the system.
- *An elementary activity* represents a unit of work that an individual, a machine, or a group can perform in an uninterrupted span of time. In other words, it is not decomposed any further in the given domain context.
- *A composite activity* consists of several other activities, either elementary or composite. The nesting of activities provides higher levels of abstraction that help to capture the various structures of organizational units involved in a workflow process.
- *A role* is a place holder or description for a set of actors, who are the authorized performers that can execute the activity. The concept of associating roles with activities not only allows us to establish the rules for association of activities or processes with organizational responsibilities, but also provides a flexible and elegant way to grant the privilege of execution of an activity to individuals or systems that are authorized to assume the associated role.
- *An actor* can be a person, a group of people, or an information system, that is granted memberships into roles and that interacts with other actors while performing activities in a particular workflow process instance.
- *Information objects* are the data resources accessed by a workflow process. These objects can be structured (e.g., relational databases), semi-structured (e.g., HTML forms), or unstructured (e.g., text documents). Structured or semi-structured data can be accessed and interpreted automatically by the system, while unstructured data cannot and thus often requires human involvement through manual activities.

Important to note is that activities in

Figure 3: ActivityFlow meta-model



ActivityFlow can be (1) manual activities, performed by users without further support from the system; (2) automatic activities, carried out by the system without human intervention, or (3) semi-automatic activities, using specific interactive programs for performing an activity.

The Running Example

To illustrate the ActivityFlow meta-model, we use a telephone service provisioning process in a telecommunication company. A synopsis of the example is described below.

Consider a business process TeleConnect that performs telephone-service-provision task by installing and billing telephone connections between the telecomm company and its clients (Ansari, Ness, Rusinkiewicz and Sheth, 1992, Georgakopoulos et al., 1995). Suppose the workflow process *A*:TELECONNECT consists of five activities A_1:CLIENTREGISTER, A_2:CREDIT CHECK, A_3:CHECKRESOURCE, A_{11}:INSTALLNEWCIRCUIT and *B*:ALLOCATECIRCUIT (see Figure *4*, (A)). *A*: TELECONNECT is executed when an enterprise's client requests telephone service installation. Activity A_1:CLIENTREGISTER registers the client information and activity A_2:CREDITCHECK evaluates the credit history of the client by accessing financial data repositories. Activity A_3:CHECK RESOURCE consults the facility database to determine whether existing facilities can be used, and *B*: ALLOCATECIRCUIT attempts to provide a connection by allocating existing resources, such as allocating lines (*C*: ALLOCATELINES), allocating slots in switches (A_8:ALLOCATESWITCH, A_9:ALLOCATESWITCH), and preparing a bill to establish the connection (A_{10}:PREPAREBILL) (see Figure 4, (B)). The activity of allocating lines (*C*:ALLOCATELINES) in turn has a number of subtasks such as selecting nearest central offices (A_4:SELECTCENTRAL OFFICES), and relocating existing lines (A_5: ALLOCATELINE, A_6:ALLOCATELINE) and spans (trunk connection) between two allocated lines

(A_7:ALLOCATESPAN) (see Figure 4, (C)). If A_3:CHECKRESOURCE succeeds, the costs of connection are minimal. The activity A_{11}:INSTALLNEWCIRCUIT is designed to perform an alternative task that involves physical installation of new facilities in the event of failure of activity A_3:CHECKRESOURCE. The roles involved with these activities are the CreditCheck-GW, the Telecommunication Company, and the Telecomm Contractor. In addition, the Telecommunication Company is detailed into three roles: Telecomm-HQ, T-central 1 and T-central 2. We use the swimlane feature on UML activity diagrams to depict such different roles of actors as involved on performing activity instances.

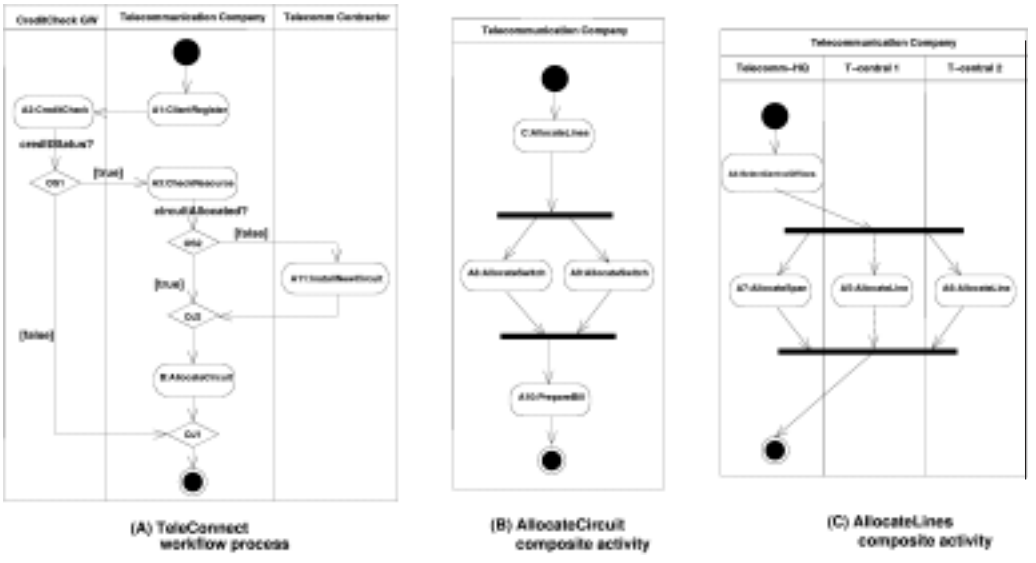
Advanced Concepts

ActivityFlow provides a number of facilities to support advanced concepts such as a variety of possibilities for handling errors and exceptions. For example, at the activity specification stage, we allow the workflow designers to specify valid processes and the compensation activities. At run-time additional possibilities are offered to support recovery from errors or crashes by triggering alternative executions defined in terms of user-defined compensation activities or system-supplied recovery routines.

Time dimension is very important for the deadline control of workflow processes. In ActivityFlow, we provide a construct to allow the workflow designer to specify the maximum allowable execution durations for both the activities (i.e., subactivities or component activities) and the process (i.e., top activity). This time information can be used to compute deadlines for all activities in order to meet an overall deadline of the whole workflow process. When an activity misses its deadline, special actions may be triggered. Furthermore, this time information plays an essential role in decisions about priorities, and in monitoring deadlines and generating time errors in the case that deadlines are missed. It also provides the possibility to delay some activities for a certain amount of time or to a specific date.

The third additional feature is the con-

Figure 4: Telephone Service Provisioning Workflow



(A) TeleConnect workflow process

(B) AllocateCircuit composite activity

(C) AllocateLines composite activity

cept of workflow administrator (WFA). Modern business organizations build the whole enterprise around their key business processes. It is very important for the success of process-centered organizations that each process has a WFA who is responsible for monitoring the workflow process according to deadlines, handling exceptions and failures that cannot be resolved automatically. More specifically, he/she is able to analyze the current status of a workflow process, make decisions about priorities, stop and resume a workflow process, abort a workflow process, dynamically restructure a workflow process, or change a workflow specification, etc. A special workflow client interface is needed which offers functionality to enable a workflow process administrator to achieve all these goals.

ACTIVITYFLOW PROCESS DEFINITION LANGUAGE

Design Principles

Most workflow management systems provide graphical specification of workflow processes. The available design tools typically support iconic representation of activities. Definition of control flows and data flows between activities is accomplished by connecting the activity icons with specialized arrows specifying the activity precedence order and their data dependencies. In addition to graphical specification languages, many WFMSs provide rule-based specification languages (Dayal, Hsu and Ladin, 1990). One of the problems with existing workflow specification languages (even those based on graphical "node and arc" programming models) is that they are not well-structured languages, in the sense that, when used without a discipline, these languages may result in schemas with a "spaghetti" of intertwined precedence relationships, which makes debugging, modifying, and reasoning of complex workflow processes difficult (Liu and Meersman, 1996). As recognized by Sheth et al. (1996), there is a need for finding a more structured way of defining the wide spectrum of activity dependencies.

Thus, the first and most important design principle in ActivityFlow is to develop a well-structured approach to specification of workflow processes, by providing a small set of constructs and a collection of mechanisms to allow workflow designers to specify the nested process structure and the variety of activity dependencies declaratively and incrementally.

The second design principle is to support the specification of basic requirements that are not only critical in most of the workflow applications (Sheth et al., 1996) but also essential for correct coordination among activities in accomplishing a workflow process. These basic requirements include:

- activity structure (control flow) and information exchange between actors (data flows) in a workflow process.
- exception handling, specifying what actions are necessary if an activity fails or a workflow cannot be completed.
- activity duration, specifying the estimated or designated maximum allowable execution time for both the workflow process (top activity) and its constituent activities. This time information is critical for monitoring deadlines of activities and for providing priority attributes, specifying priorities for activity scheduling.

Main Components of a Workflow Specification

In ActivityFlow, a workflow process is described in terms of a set of activities and the dependencies between them. For presentation convenience, in the rest of paper we refer to a workflow process as top activity and workflow component activities as subactivities. We use activities to refer to both the process and its component activities when no distinction needs to be made.

Activities are specified by activity templates or so called parameterized *activity patterns*. An activity pattern describes concrete activities occurring in a particular organization, which have similar communication behavior. An execution of the activity pattern is called an instantiation (or an activity instance) of the activity pattern. Informally, an activity pattern consists of objects, messages, message exchange constraints, preconditions, postconditions, and triggering conditions (Liu and Meersman, 1996).

Activities can be composed of other activities. The tree organization of an activity pattern α is called the *activity hierarchy* of α. The set of activity dependencies specified in the pattern α can be seen as the cooperation agreements among activities that collaborate in accomplishing a complex task. The activity at the root of the tree is called *root activity* or *workflow process*; the others are subactivities. An activity's predecessor in the tree is called a *parent*; a subactivity at the next lower level is called a *child*. Activities at leaf nodes are elementary activities in the context of the workflow application domain. Nonleaf node activities are composite activities. In ActivityFlow we allow arbitrary nesting of activities since it is generally not possible to determine a priori the maximum nesting an application task may need.

A typical workflow specification consists of the following five units:

- *Header*: The header of an activity specification describes the signature of the activity, which consists of a name, a set of input and output parameters, and the access type (i.e., Read or Write). Parameters can be objects of any kind, including forms. We use keyword *In* to describe parameters that are inputs to the activity and *Out* to describe parameters that are outputs of the activity. Parameters that are used for both input and output are specified using keyword *InOut*.
- *Activity Declaration*: The activity declaration unit captures the general information about the activity such as the synopsis (description) of the task, the maximum allowable execution time, the administrator of the activity (i.e., the user identifier (UID) of the responsible person), and the set of compensation activities that are used for handling errors and exceptions and their triggering conditions.
- *Role Association*: This unit specifies the set of roles associated with the activity. Each role is defined by a role name, a role type, and a set of actors that are granted membership into the role based on their responsibility in the business process or in the organization. Each actor is described by actor ID and role name. We distinguish two types of roles in the first prototype implementation of ActivityFlow: user and system, denoted as USER and SYS respectively.
- *Data Declaration*: The data declaration unit consists of the declaration of the classes to which the parameters of the activity belong and the set of messages (or methods) needed to manipu-

late the actual arguments. Constraints between these messages are also specified in this unit (Liu and Meersman, 1996).
- *Procedure*: The procedure unit is defined within a begin and end bracket. It describes the composition of the activity, the control flow and data flow of the activity, and the pre- and post-condition of the activity. The main component of the control flow includes activity-execution-dependency specification, describing the execution precedence dependencies between children activities of the specified activity and the interleaving dependencies between a child activity and children of its siblings or between children activities of two different sibling activities. The main component of the data flow specification is defined through the activity state-transition dependencies.

Dynamic Assignments of Actors

The assignment of actors (humans or information systems) to activities according to the role specification is a fundamental concept in WFMSs. At run time, flexible and dynamic assignment resolution techniques are necessary to react adequately to the resource allocation needs and organizational changes. ActivityFlow uses the following techniques to fulfill this requirement:

- When the set of actors is empty, the assignment of actors can be any users or systems that belong to the roles associated with the specified activity. When the set of actors is not empty, only those actors listed in the associated actor set can have the privilege to execute the activity.
- The assignment of actors can also be done dynamically at run time. The activity-enactment service engine will grant the assignment if the run time assignment meets the role specification.
- The assignment of actors can be the administrator of the workflow process to which the activity belongs, as the workflow administrator is a default role for all its constituent activities.

The role-based assignment of actors provides great flexibility and breadth of application. By statically and dynamically establishing and defining roles and assigning actors to activities in terms of roles, workflow administrators can control access at a level of abstraction that is natural to the way that enterprises typically conduct business.

Control Flow Specification: Activity Dependencies

In ActivityFlow a number of facilities are provided to promote the use of declarative and incremental approach to specification of activities and their dependencies. For example, to make the specification of activity execution dependencies easier and more user friendly for the activity model designers, we classify activity dependencies into three categories: activity execution dependencies, activity interleaving dependencies, and activity state transition dependencies. We also regulate the specification scope of the set of activity dependencies associated with each activity pattern to encourage incremental specifications of hierarchically complex activities. For instance, to define an activity pattern *T*, we require the workflow designer to specify only the activity execution dependencies between activities that are children of a *T* activity, and restrict the activity interleaving dependencies specified in *T* to

be only the interaction dependencies between (immediate) subactivities of different child activities of *T*, or between a *T*'s child activity and (immediate) subactivities of its siblings. As a result, the workflow designers may specify the workflow process and the activities declaratively and incrementally, allowing reasoning about correctness and security of complex workflow activities independently from their underlying implementation mechanisms.

In addition, we provide four constructs to model various dependencies between activities. They are **precede, enable, disable**, and **compatible**. The semantics of each construct are formally described in Figure 5. The construct **precede** is designed to capture the temporary precedence dependencies and the existence dependencies between two activities. For example, "*A* **precede** *B*" specifies a *begin-on-commit* execution dependency between the two activities: ``*B* cannot begin before *A* commits". The constructs **enable** and **disable** are utilized to specify the enabling and disabling dependencies between activities. One of the critical differences between the construct **enable** or **disable** and the construct **precede** is that **enable** or **disable** specifies a triggering condition and an action being triggered, whereas **precede** only specifies an execution precedence dependency as a precondition that needs to be verified before an action can be activated, and it is not an enabling condition that, once satisfied, triggers the action. The construct **compatible** declares the compatibility of activities A_1 and A_2. It is provided solely for specification convenience since two activities are compatible when there is no execution precedence dependency between them.

Recall the telephone service provisioning workflow example given earlier. After having entered the service request in the client and service order databases, the activity A_3:CHECKRESOURCE tries to determine which facilities can be used when establishing the service. If A_3:CHECKRESOURCE commits, it means that the client's request can be met. In case of failing on the allocation of the service with existing lines and spans, but being viable the installation of such new circuit elements, a human field engineer is selected to execute the activity A_{11}:INSTALLNEWCIRCUIT, which may involve manual changes to some switch and the installation of a new telephone line. We have adopted the Eder and Liebhart (1995) approach and model in ActivityFlow diagrams to represent only **expected exceptions**. Such cooperation dependencies among A_3:CHECKRESOURCE, *B*:ALLOCATECIRCUIT and A_{11}:INSTALLNEWCIRCUIT can be specified as follows:

1. $A_3 \wedge \neg$ *circuitAllocated* **precede** A_{11}. ("*circuitAllocated* =false" is a precon-

Figure 5: Constructs for activity dependency specification

Construct	Usage	Synopsis
precede	A_1 **precede** A_2 *condition*(A_1) **precede** A_2 *condition*(A_1) **precede** *condition*(A_2)	A_2 can *begin* if A_1 *commits* A_2 can *begin* if *condition*(A_1) = 'true' holds. If *condition*(A_1) = 'true' then *condition*(A_2) can be 'true'
enable	*condition*(A_1) **enable** A_2 *condition*(A_1) **enable** *condition*(A_2)	*condition*(A_1) = 'true' → *begin*(A_2) If *condition*(A_1) = 'true' then *condition*(A_2) will be 'true'
disable	*condition*(A_1) **disable** A_2 *condition*(A_1) **disable** *condition*(A_2)	*condition*(A_1) = 'true' → *abort*(A_2) If *condition*(A_1) = 'true' then *condition*(A_2) cannot be 'true'
compatible	**compatible**(A_1, A_2)	'true' if A_1 and A_2 can be executed in parallel, 'false' if the order of A_1 and A_2 is important

dition for a human engineer to execute A_{11} after A_3 commits.)

2. ($A_3 \wedge$ *circuitAllocated*) $\vee A_{11}$ **enable** *B*. (if A_3 commits and returns the true value in its *circuitAllocated* output parameter, or a human field engineer succeeds on installing the line/span needed, then *B* is triggered).

The first dependency states that the commit of A_3: CHECKRESOURCE and the false value of the *circuitAllocated* output parameter are preconditions for A_{11}: INSTALLNEWCIRCUIT. The second dependency amounts to saying that if A_3: CHECKRESOURCE is successful on defining existing facilities that satisfy the request, (*circuitAllocated* = true) or A_{11}: INSTALLNEWCIRCUIT have installed the needed new facility, then *B*: ALLOCATECIRCUIT is triggered. The reason that we use the construct **precede**, rather than **enable**, for specifying the first dependency is because A_{11}: INSTALLNEWCIRCUIT involves some manual work and thus must be executed by a human field engineer. ActivityFlow also allows the users to specify conditional execution dependencies to support activities triggered by external events (e.g., *Occurs*(E_1) **enable** A_1).

Activity Specification: An Example

To illustrate the use of ActivityFlow workflow specification language in describing activities of a nested structure, we recast the telephone-service-provisioning workflow, given previously. Figure 4 shows the hierarchical organization of the workflow process TELECONNECT. The top activity TELECONNECT (see Figure *4*, (A)) is defined as a composite activity, consisting of the following five activities: A_1: CLIENTREGISTER, A_2: CREDITCHECK, A_3: CHECK RESOURCE, *B*:ALLOCATECIRCUIT, and A_{11}: INSTALLNEWCIRCUIT. The activity *B*: ALLOCATECIRCUIT (see Figure *4*, (B)) is again a composite activity, composed of four subactivities:

Figure 6: Example specification of the top activity TELECONNECT

Activity TELECONNECT(*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
  Access Type: Write
  Synopsis: Telephone service provisioning
  Max Allowable Time: 2 weeks
  Administrator: UID: 0.0.0.337123545
  Exception Handler: none
  Role Association:
    Role name: Telecommunication Company
    Role type: System
  Data Declaration:
    import class CLIENT,
    import class POINT,
    import class CIRCUIT;
begin Behavioral Aggregation of component Activities:
    A_1: CLIENTREGISTER (*In:* ClientId:CLIENT, Start:POINT, End:POINT)
    A_2: CREDITCHECK (*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* creditStatus:Boolean)
    A_3: CHECKRESOURCE (*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* circuitAllocated:Boolean)
    A_{11}: INSTALLNEWCIRCUIT(*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
    B: ALLOCATECIRCUIT (*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
  Execution Dependencies:
    $ExeR_1$: A_1 **precede** {A_2, A_3}
    $ExeR_2$: $A_3 \wedge \neg$ *circuitAllocated* **precede** A_{11}
    $ExeR_3$: ($A_3 \wedge$ *circuitAllocated*) $\vee A_{11}$ **enable** B
  Interleaving Dependencies:
    ILR_1: $A_2 \wedge$ *creditStatus* **precede** A_{10}
  State Transition Dependencies:
    STR_1: *abort*(B) **enable** *abort*(*self*)
end Activity

C:ALLOCATELINES, A_8: ALLOCATE SWITCH, A_9: ALLOCATESWITCH and A_{10}: PREPAREBILL. The activity *C*:ALLOCATELINES (see Figure *4*, (C)) is also a composite activity, with four subactivities: A_4: SELECTCENTRAL OFFICES, A_5: ALLOCATELINE, A_6: ALLOCATELINE, and A_7: ALLOCATE SPAN. Based on the structure of a workflow process definition discussed previously, we provide an example specification for the telephone service provisioning workflow (top activity) in Figure *6*, the composite activities *B*: ALLOCATECIRCUIT in Figure *7* and *C*: ALLOCATELINES in Figure *8*, and the elementary activity A_{11}: INSTALLNEWCIRCUIT in Figure *9*.

A Formal Model for Flow Procedure Definition

In this section, we provide a graph-based model to formally describe the procedure unit of a workflow specification in ActivityFlow. This graph-based flow procedure model provides a formal foundation for ActivityFlow graphical user interface, which allows the end-users to model office procedures in a workflow process using iconic representation.

In ActivityFlow, we describe an activity procedure in terms of (1) a set of *nodes*, representing individual activities or connectors between these activities (e.g., split and join connectors), and (2) a set of *edges*, representing signals among the nodes. Each node in the activity flow procedure is annotated with a trigger. A trigger defines the condition required to fire the node upon receiving signals from other nodes. The trigger condition is defined using the four constructs described earlier. Each flow procedure has exactly one begin node and one end node. When the begin node is fired, an activity flow instance is created. When the end node is triggered,

Figure 7: Example specification of the composite activity ALLOCATECIRCUIT

Activity ALLOCATECIRCUIT(*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
Access Type: Write
Synopsis: Circuit allocation
Max Allowable Time: 3 days
Administrator: UID: 0.0.0.337123545
Exception Handler: none
Role Association:
Role name: Telecommunication Company
Role type: System
Data Declaration:
import class CLIENT,
import class POINT,
import class CIRCUIT,
import class LINE,
import class SPAN;
begin Behavioral Aggregation of component Activities:
C: ALLOCATELINES (*In:* Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
A_8: ALLOCATESWITCH (*In:* Line1:LINE, *Out:* Span:SPAN)
A_9: ALLOCATESWITCH (*In:* Line2:LINE, *Out:* Span:SPAN)
A_{10}: PREPAREBILL(*In:* ClientId:CLIENT, Line1:LINE, Line2:LINE, Span:SPAN, *Out:* CircuitId:CIRCUIT)
Execution Dependencies:
$ExeR_4$: $C \wedge A_8 \wedge A_9$ **precede** A_{10}
Interleaving Dependencies:
ILR_2: $A_5 \wedge A_7$ **precede** A_8
ILR_3: $A_6 \wedge A_7$ **precede** A_9
State Transition Dependencies:
STR_2: *abort*(C) $\vee$ *abort*(A_8) $\vee$ *abort*(A_9) **enable** *abort*(*self*)
end Activity

Figure 8: Example specification of composite activity ALLOCATELINES

Activity ALLOCATELINES(*In:* Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
Access Type: Write
Synopsis: Line allocation
Max Allowable Time: 1 days
Administrator: UID: 0.0.0.337123545
Exception Handler: none
Role Association:
Role name: Telecommunication Company
Role type: System
Data Declaration:
import class POINT,
import class LINE,
import class SPAN,
import class CentralOff;
begin Behavioral Aggregation of component Activities:
A_4: SELECTCENTRALOFFICES (*In:* Start:POINT, End:POINT, *Out:* Off1:CentralOff, Off2:CentralOff)
A_5: ALLOCATELINE (*In:* Start:POINT, Off1:CentralOff, *Out:* Line1:LINE)
A_6: ALLOCATELINE (*In:* End:POINT, Off2:CentralOff, *Out:* Line2:LINE)
A_7: ALLOCATESPAN(*In:* Off1:CentralOff, Off2:CentralOff, *Out:* Span:SPAN)
Execution Dependencies:
$ExeR_5$: A_4 **precede** {A_5, A_6, A_7}
State Transition Dependencies:
STR_3: *abort*(A_4) ∨ *abort*(A_5) ∨ *abort*(A_6) ∨ *abort*(A_7) **enable** *abort*(*self*)
end Activity

Figure 9: Example specification of elementary activity INSTALLNEWCIRCUIT

Activity INSTALLNEWCIRCUIT(*In:* ClientId:CLIENT, Start:POINT, End:POINT, *Out:* CircuitId:CIRCUIT)
Access Type: Write
Synopsis: New line/span installation
Max Allowable Time: 1 week
Administrator: UID: 0.0.0.337123545
Exception Handler: none
Role Association:
Role name: Telecomm Contractor
Role type: User
end Activity

the activity flow instance terminates.

Definition 1 (activity flow graph)
An activity flow graph is described by a binary tuple < *N, E* >, where *N* is a finite set of activity nodes and connector nodes. $N = AN \cup CN \cup \{bn, en\}$, where $AN = \{nd_1, nd_2,..., nd_n\}$ is a set of activity nodes, $CN = \{cn_1, cn_2,..., cn_n\}$ is a set of connector nodes, *bn* denotes the begin node and *en* denotes the end node. Each node $n_i \in N$ ($i = 1,.., n$) is described by a quadruple (*NN, TC, NS, NT*), where *NN* denotes the node name. *TC* is the trigger condition of the node. *NS* is one of the two states of the node: *fired* or *not fired*. *NT* is the node type.

- If $n_i \in AN \rightarrow NT = \{simple, compound, iteration\}$
- If $n_i \in CN \rightarrow NT = \{$AND-split, OR-Split, AND-join, OR-join$\}$

$E = \{e_1, e_2,..., e_m\}$ is a set of edges. Each edge is of the form *ndi* → *ndj*. An edge e_{ij}: $nd_i \rightarrow nd_j$ is described by a quadruple (*EN, DPnd, AVnd, ES*), where *EN* is the edge name, *DPnd* is the departure node, *AVnd* is the arrival node, and *ES* is one of the two states of the node: *signaled* and *not signaled*. We call e_{ij} an outgoing edge

of node nd_i and incoming edge of node nd_j.❑

For each node nd_i, there is a path from the begin node *bn* to nd_i. We say that a node nd_i is reachable from another node nd_j if there is a path from nd_i to nd_j.

Definition 2 (reachability)
Let $G = < N, E >$ be an activity flow graph. For any two nodes nd_i, $nd_j \in N$, nd_j is reachable from nd_i, denoted by $nd_i \overset{*}{\rightarrow} nd_j$, if and only if one of the following conditions is verified:

(1) $nd_i = nd_j$.
(2) $nd_i \rightarrow nd_j \in E$.
(3) $\exists\ nd_k \in N$, $nd_k \neq nd_i$ and $nd_k \neq nd_j$ such that $nd_i \overset{*}{\rightarrow} nd_k$ and $nd_k \overset{*}{\rightarrow} nd_j$. ❑

A node nd_j is said to be directly reachable from a node nd_i if the condition (2) in Definition *2* is satisfied.

To guarantee that the graph $G = < N, E >$ is acyclic, the following restrictions are placed:

1) $\forall\ nd_i, nd_j \in N$, if $nd_i \rightarrow nd_j \in E$ then $nd_j \rightarrow nd_i \notin E$.
2) $\forall\ nd_i, nd_j \in N$, if $nd_i \overset{*}{\rightarrow} nd_j$ then $nd_j \overset{*}{\rightarrow} nd_i$ does not hold.

To illustrate the definition, let us recast the telephone service provisioning workflow procedure depicted in the Figure 4, (A) diagram, and described in Figure 6 in terms of the above definition as follows:
N = {(*Begin*, NeedService, *notfired*, *simple*), (A_1, NeedService, *notfired*, *simple*), (A_2, *commit*(A_1), *notfired*, *simple*), (OS_1, *commit*(A_2), *notfired*, OR-Split), (A_3, *creditStatus* = true, *notfired*, *simple*), (OS_2, *commit*(A_3), *notfired*, OR-Split), (A_{11}, *circuitAllocated* = false, *notfired*, *simple*), (OJ_2, *circuitAllocated* = true $\vee$ *commit*(A_{11}), *notfired*, OR-Join), (*B*, *terminate*(OJ_2), *notfired*, *compound*), (OJ_1, *creditStatus* = false $\vee$ *commit*(*B*), *notfired*, OR-Join), (*End*, *terminate*(OJ_1), *notfired*, *simple*)}
E = {*Begin* $\rightarrow A_1$, $A_1 \rightarrow A_2$, $A_2 \rightarrow OS_1$, $OS_1 \rightarrow A_3$, $OS_1 \rightarrow OJ_1$, $A_3 \rightarrow OS_2$, $OS_2 \rightarrow OJ_2$, $OS_2 \rightarrow A_{11}$, $A_{11} \rightarrow OJ_2$, $OJ_2 \rightarrow B$, $B \rightarrow OJ_1$, $OJ_1 \rightarrow$ *End*}

Note that *NeedService* is a Boolean variable from the ActivityFlow runtime environment. When a new telephone service request arrives, *NeedService* is true. Figure *4* (A) shows the use of the UML-based ActivityFlow graphical notations to specify this activity flow procedure. When a node is clicked, the node information will be displayed in a quadruplet, including node type, name, its trigger, and its current state. When an edge is clicked, the edge information, such as the edge name, its departure and arrival nodes, and its current state, will be displayed. From Figure 4 (A), it is obvious that activity node *B* is reachable from nodes A_1, A_2, A_3 and A_{11}.

An activity flow graph *G* is instantiated by an instantiation request issued by an actor. The instantiation request provides the initial values of the data items (actual arguments) required by the parameter list of the flow. An activity flow instantiation is valid if the actor who issued the firing satisfies the defined role specification.

Definition 3 (valid flow instantiation)
Let $G = < N, E >$ be the activity flow and u = *(actor_oid, role_name)* be an actor requesting the activity flow instantiation *T* of *G*. The flow instantiation *T* is *valid* if and only if $\exists \rho \in Role(G)$ such that *role_name (u)* = ρ. ❑

When the actor who initiates a flow instantiation request is not authorized, the instantiation request is rejected, and the flow instantiation is not created.

When a flow instantiation request is valid, a flow instantiation, say *T*, is created by firing the begin node of *T*.

Definition 4 (activity flow instantiation) Let $G = < N, E >$ be the activity flow and T denote a *valid* flow instantiation of G. T is created by assigning a flow instance identifier and carrying out the following steps to fire the begin node *bn(T)*: set the state of node *bn*(T) to be *fired*; set all the outgoing edges of *bn*(T) to be *signaled*; perform a node instantiation for each node that is directly reachable from the begin node *bn*(T).

A node can be instantiated or triggered when all the incoming edges of the node are signaled, its trigger condition is evaluated to be true. When a node is triggered, a unique activity instance identifier is assigned, and the node state is set to *fired*. In ActivityFlow, all the nodes are initialized to *not_fired* and all the edges are initialized to *not_signaled*.

Definition 5 (node instantiation) Let $G = < N, E >$ be the activity flow and T denote a valid flow instantiation of G. A node $nd_k \in N$ can be instantiated if $\forall nd_i \in N$ such that $nd_i \neq nd_k$ and nd_k is directly reachable from nd_i, we have nd_i is in the state *fired*, the instance identifier of T is identified, the trigger of nd_k can be evaluated.

A node nd_k is instantiated if the following steps are performed:

- updates to data items are applied in all the nodes nd_i from which nd_k is directly reachable.
- all the incoming edges of nd_k are set to be signaled.
- nd_k is fired if (1) its trigger condition is evaluated to be true and (2) it is currently not fired or it is an iteration activity node and its iteration condition is evaluated to be true. ❑

In ActivityFlow, we use the term conditional rollback to refer to the situations that require revisiting the nodes previously terminated or not fired. Conditional rollbacks are a desirable functionality and encountered frequently in some business processes. We provide the UML activity iterator symbol ("*" into a compound activity-node construct) for the realization of conditional rollbacks. The use of iterating activities has a number of interesting features. First, by defining an activity with the iterator symbol, being such activity a composite activity, we identify the nodes that can be or allowed to be revisited by the subsequent activities in the same subflow instance. Second, when using iteration rather than explicitly backward edges, the conditional rollback may be considered as a continuation of the workflow instance execution. We believe that the use of iteration provides a much cleaner graphical notation to model cyclic activity workflows.

To reduce the complexity and facilitate the management of conditional rollbacks, the only restriction we place on the conditional rollback is the following: A call to rollback to an activity node nd_k can only be accepted if it comes from subactivity nodes or sibling activity nodes of nd_k.

Figure *10* shows an example which recasts the composite activity *C*:ALLOCATELINES discussed earlier by allowing a conditional rollback of some allocation line activities (A_5:ALLOCATE LINE, A_6:ALLOCATELINE and A_7:ALLOCATESPAN). It permits the execution of a set of *C*:ALLOCATELINES and evaluates which instance is more profitable. The others are rollbacked. We model this requirement using iteration (see Figure 10).

By clicking the iteration-type activity node, the information about its subflow will be displayed. The rollback condition is also displayed. In this case, it says that if *C*:ALLOCATELINES is successful then the AND-Split type connection node fol-

lowing *C*:ALLOCATELINES is fired. Then, activities A_8:ALLOCATESWITCH and A_9:ALLOCATESWITCH are fired. Otherwise, a new *C*:ALLOCATELINES instance is fired until the profit level required may be reached.

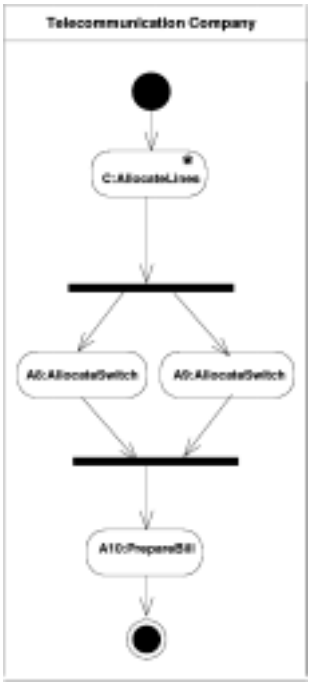
Definition 6 (termination property) An activity flow instance terminates if its end node is triggered. A flow instance is said to satisfy the termination property if the end node will eventually be fired. ❑

The termination property guarantees that the flow procedure instantiation will not "hang".

Definition 7 (precedence preserving) Let $G = < N, E >$ be the activity flow. An activity flow instance of *G* is said to satisfy precedence preserving property if the node firing sequence is compatible with the partial order defined by the activity precedence dependencies in *G*. ❑

In ActivityFlow, these two latter properties are considered as correctness properties, among others, for concurrent executions of activities. For a detailed discussion on preservation of the correctness properties of workflow activities, see Liu and Pu (1998a).

Figure 10: An example using iterator connectors



DYNAMIC WORKFLOW RESTRUCTURING OF ACTIVITYFLOW MODELS

To maintain the competitiveness in a business-oriented world, enterprises must offer high-quality products and services. A key factor in successful quality control is to ensure the quality of all corporate business processes, which include clearly defined routing among activities, association among business functions (e.g. programs) and automated activities, execution dependency constraints and deadline control, at both activity level and whole workflow process level. Besides the workflow characteristics, most workflow applications are expected to have 100% uptime (24 hours per day and 7 days per week). Production workflow (Leymann and Roller, 2000) is a class of workflow that presents such characteristics and the workflow processes have a high business value for the organizations.

The enterprise commitment with a deadline for each of its workflow process execution becomes one of the design and operation objectives for workflow management systems. However, deadline control of workflow instances have led to a growing problem that conventional workflow management systems do not address, namely how to reorganize existing workflow activities in order to meet deadlines in the presence of unexpected delays. Besides, having long-lived business-process in-

stances, workflow designs must deal with schema evolution with the proper handling of ongoing instances. These problems are known as the workflow-restructuring problem.

This section describes the notation and issues of workflow restructuring, and discusses how a set of workflow activity restructuring operators can be employed to tackle the workflow-restructuring problem on ActivityFlow modeling. We restrict our discussion in the context of how to handle unexpected delays. A deeper study on such context can be found in Ruiz, Liu and Pu (2002).

Basic Notions

Activity restructuring operators are used to reorganize the hierarchical structure of activity patterns with their activity dependencies remaining valid. Two types of activity restructuring operators are proposed by Liu and Pu (1998): Activity-Split and Activity-Join. *Activity-Split* operators allow releasing committed resources that were updated earlier, enabling adaptive recovery and added concurrency (Liu and Pu, 1998a). *Activity-Join* operators, the inverse of activity-split, combine results from sub-activities together and release them atomically. The restructuring operators can be applied to both simple and composite activity patterns and can be combined in any formation. Zhou, Pu and Liu (1998) present a practical method to implement these restructuring operators in the context of the Transaction Activity Model (TAM) (Liu and Pu, 1998a).

In TAM, activities are specified in terms of *activity patterns*. An *activity pattern* describes the communication protocol of a group of cooperating objects in accomplishing a task (Liu and Meersman, 1996). We distinguish two types of activities: *simple activity pattern* or *composite activity pattern*. A *simple activity pattern* is a program that issues a stream of messages to access the underlying database (Liu and Pu, 1998a). A *composite activity pattern* consists of a tree of composite or simple activity patterns and a set of user-defined activity dependencies: (a) activity execution and interleaving dependencies, and (b) activity state-transition dependencies. The activity at the root of the tree is called *root activity*; the others are called *sub-activities*. An activity's predecessor in the tree is called *parent*; a subactivity at the next lower level is called a *child*. *Activity hierarchy* is the hierarchical organization of activities (see Figure 4 for an example).

A TAM activity has a set of observable states S and a set of possible state transitions $\varphi: S \rightarrow S$, where S = {*begin, commit, abort, done, compensate*} (Liu and Pu, 1998) (see Figure 11). When an activity T is activated, it enters in the state *begin* and becomes active. The state of T changes from *begin* to *commit* if T *commits*, and to *abort* if T or its parent *aborts*. If T's root activity *commits*, then its state becomes *done*. When T is a *composite activity*, T enters the *commit* state if all its component activities legally terminate, i.e., *commit* or *abort*. If an activity *aborts*, then all its children that are in *begin* state are *aborted* and its *committed* children, however, are *compensated* for. We call this property **termination-sensitive** *dependency* (Liu and Pu, 1998a) between an activity A_C and its parent activity A_p, denoted by $A_p \sim> A_C$. This termination-sensitive dependency, inherent in an activity hierarchy, prohibits a child activity instance from having more than one parent, ensuring the hierarchically nested structure of active activities. When the abort of all active sub-activities of an activity is com-

pleted, the compensation for committed subactivities is performed by executing the corresponding compensations in an order that is the reverse of the original order.

Definition 8 (TAM activity)
Let α denote an activity pattern and S denote a set of activity patterns. Let AD(α) denote a set of activity dependencies specified in α, children(α) denote the set of child activity patterns of α, and Pattern(T) denote the activity pattern of activity T. An activity T is said to be a TAM activity if and only if it satisfies the following conditions:

- $\exists \alpha \in \Sigma$, *Pattern*($T$) = α.
- $\forall P \in AD(\alpha)$, $P(T) = true$.
- $\forall S \in children(T)$, $T \sim> S$ *and S is* α TAM *activity.* ❑

Another property of an *activity hierarchy* is the **visibility** of objects between activities. The *visibility* of an activity refers to its ability to see the results of other activities while it is executing. A *child* activity A_C has access to all objects that its *parent activity* A_p can access, i.e., it can read objects that A_p has modified (Liu and Pu, 1998a). TAM uses the *multiple object version schemes* (Nodine and Zdonik, 1990) to support the notion of *visibility* in the presence of concurrent execution of activities. The *Root activity* at the top of the activity hierarchy contains the most stable version of each object, and guarantees the possibility to recover its copies of objects in the event of a system failure.

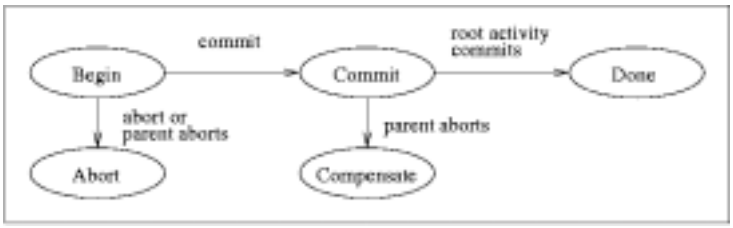
Workflow Restructuring Operators

There are three types of activity-split operators: *serial activity-split* (*s-Split*), *parallel activity-split* (*p-Split*), and *unnesting activity-split* (*u-Split*).

- The *s-Split* operator splits an activity into two or more activities that can be performed and committed sequentially. It establishes a linear execution dependency among the resulting activities which is captured by using the **precede** construct.
- The *p-Split* splits an activity into two or more activities that can be submitted and committed independently of each other. The only dependency established between them is the compatibility among all split activities and can be represented by **compatible** construct.
- The *u-Split* splits C activity by unnesting the activity hierarchy anchored at C. *U-Split* operators are effective only on composite activity patterns.

A series of specializations are introduced for activity split, including *s-Split* - serial activity-split, (*sa-Split* - serial-alternative activity-split), and *p-Split* - parallel activity-split, (*pa-Split* - parallel-alternative activity-split, *cc-Split* - commit-on-commit activity-split, and *ca-Split* - commit-on-abort activity-split). These specializations

Figure 11: TAM activity state transition graph



tackle situations where it is necessary to synchronize concurrent split activities and when certain activities can be performed only if another aborts.

An activity-split operation is said to be *valid* if and only if the resulting activities: (1) satisfy the implicit dependencies implied in the activity composition hierarchy such as the termination-sensitive dependency, i.e., TAM activities; (2) all existing activity dependencies are semantically preserved after the split; and (3) do not introduce any conflicting activity dependencies (Liu and Pu, 1998).

Similarly, activity-Join has two specialized versions: *join-by-group* (*g-Join*) and *join-by-merge* (*m-Join*).

- The *g-Join* operator groups two or more activities by creating a new activity as their parent activity, while preserving the activity composition hierarchy of each input activity. A *g-Join* is considered legal if the input activities are all sibling activities or independently ongoing activities, i.e., they do not have common parent activity.
- The *m-Join* operator physically merges two or more activities into a single activity. An *m-Join* is considered legal, if for each pair of input activities (C1, C2), C1 and C2 are sibling activities, or one is a parent activity of another, or they are independently ongoing activities.

Restructuring Possibilities on TeleConnect Workflow

Most workflow designs take into account the organizational structure, the computational infrastructure, the collection of applications provided by the corporate enterprises, and the cost involved. Such designs are based on the assumptions that the organizational structure is an efficient way to organize business processes (workflows) and the computational infrastructure has the optimal configuration within the enterprise. However, such assumptions may not hold when unexpected delays happen and when such delays severely hinder the progress of ongoing workflow executions.

Typical delays in execution of business workflows are due to various types of failures or disturbances in computational infrastructure, including instabilities in network bandwidth, and replacement of low power computing infrastructure in coping with server failures. Such disturbances can be transient or perennial, unexpected or intentional, and can affect an expressive number of processes.

Figure 12 shows the typical implementation architecture of the Telecomm computational infrastructure, which is used in our experimental study. Each telecommunications central T-central has a computer server to support its activities and to manage its controlled lines and switches. In the Telecomm Headquarters, Telecomm-HQ, a computer server supports all the management activities and controls the information with respect to communication among its branches (spans), centralizes the billing, etc. The credit check gateway CeditCheck-GW executes the dialog between Telecomm and credit operators and banks to check the current financial situation of the clients. Figure 12 describes a typical computational capacity of computing systems as well as the network connection speeds assumed in the experiments reported earlier. We have adopted TPC-W (T.-W. Subcommittee, 2001) to show the power of computing systems because we have assumed all Telecomm information systems are web-based e-commerce applications.

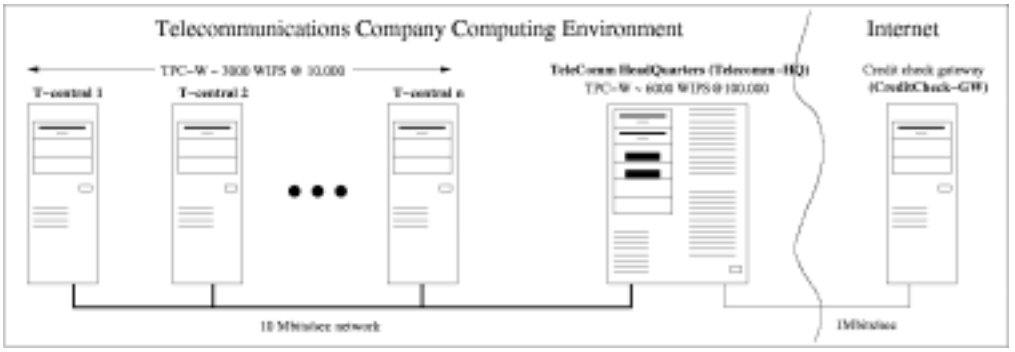
Recall the telephone-service-provision introduced earlier, and assume that this

workflow process was designed to match the organizational structure with respect to its administrative structure and corresponding responsibilities. From the activity hierarchy shown in Figure *4*, the activity *A*:TELECONNECT consists of two composite activities: *B*:ALLOCATECIRCUIT and *C*:ALLOCATELINES. The execution dependencies of these compound activities (A, B and C) are given in Figures 6, 7 and 8. We can conclude that A_2:CREDITCHECK must be completed before *B*:ALLOCATE CIRCUIT because A_{10}:PREPAREBILL depends on A_2:CREDITCHECK and A_{10}:PREPARE BILL is a sub-activity of *B*:ALLOCATE CIRCUIT. By combining the hierarchical structure of those composite activities and their corresponding execution dependencies, we present the workflow design, without compound activities, in Figure 13.

In the presence of delays, restructuring operators can be applied to rearrange the activity hierarchy anchored by *A*:TELECONNECT. The goal is to add concurrency during execution of their instances. Such *added concurrency* means the earlier release of committed resources to allow access by other concurrent activities (Liu and Pu, 1998). The TAM operators that permit increase of concurrency among TAM activities are *p-Split* and *u-Split*. For simplicity, we discuss only the use of the *u-Split* operator because it does not demand previous knowledge of the internal structure and behavior of the target activity.

By applying *u-Split* on *A*:TELECONNECT, it is possible to unnest its compound activities *B*: ALLOCATECIRCUIT or *C*: ALLOCATE LINES. Then, two different restructured workflows with added concurrency are obtained: unnesting *C*: ALLOCATELINES (Figure 14) and unnesting *B*: ALLOCATE CIRCUIT (Figure 15). When compared with the initial workflow designs shown in Figure 4, unnesting *C*: ALLOCATELINES permits the start of activity A_8: ALLOCATESWITCH or A_9: ALLOCATE SWITCH in case of delay in execution of A_6: ALLOCATELINE or A_5: ALLOCATE LINE respectively. Similarly, unnesting *B*: ALLOCATECIRCUIT allows the start of composite activity *C*: ALLOCATELINES before the credit check activity A_2: CREDITCHECK commits. We have chosen to control instances of activity A_2: CREDITCHECK to decide if *B*: ALLOCATECIRCUIT needs restructuring. In addition, A_6: ALLOCATELINE is the chosen activity to be controlled when examining the need for *C*: ALLOCATELINES restructuring because both A_5 and A_6 show the same behavior in the workflow model. The results on restruc-

Figure 12: A typical computing environment for the Telecomm Company



turing C by controlling A_6 are similar to those obtained by controlling A_5.

Simulation Environment

To study the effectiveness of activity restructuring operators, we built a simulator using CSIM18 (Mesquite Software, Inc., 1994) that performs workflow models. These models consist of simple and composite workflow activities such as TAM (Zhou et al., 1998). The typical computing environment depicted in Figure 12 is used to quantify the disturbance effects and to tune the simulator. We discussed further experiments with a range of parameter settings that expand and support the results outlined here. Here we briefly describe the simulator, focusing on the aspects that are pertinent to our experiments.

To simulate the TELECONNECT workflow activity, we assume 60 seconds being the upper average limit for the elapsed time of one workflow instance execution. In other words, a TeleConnect workflow

Figure 13: Plane graphical representation of the flow procedure of activity TELECONNECT

instance carries out by 60-second upper limit when the installation of new facilities (execution of activity A_{11}: INSTALLNEW CIRCUIT) is not required. We represent the elapsed time of activity instances using the uniform statistical distribution, since these activities involve a combination of computer and human activities of unpredictable duration within a known range of reasonable values. Figure *16* shows the type of computer system where each activity executes the corresponding activities in the simulation, and the minimum and maximum elapsed time values taken.

For the sake of simplicity, we assume only three different time intervals for the elapsed time of activity instances. For each time interval corresponds to one computing system type. Activity instances executing at Telecomm-HQ (A_1, A_3, A_4, A_7 and A_{10}) show elapsed time between *3.2 seconds* and *5.2 seconds. 6.4 seconds - 10.4 seconds* is the elapsed time interval for activity instances executed on any T-central systems (A_5, A_6, A_8 and A_9) and A_2 instances present elapsed time between *2.0 seconds* and *22.0 seconds* when executing on CreditCheck-GW system. We adopt these time intervals because: (1) Telecomm-HQ is the most powerful system in the computing environment and hosts the workflow management system (WfMS); (2) as regards Telecomm-HQ, the elapsed time of each activity instance (executed at T-central, or at CreditCheck-GW) considers also the time to flow data and commands into network connections; (3) CreditCheck-GW represents a computing system beyond the responsibilities of the Telecomm Company technical staff and with a quite variable response time.

We adopted the 90% percentile principle from TPC-C (T.-C. subcommittee, 2001) to define the 60-second limit. TPC-

Figure 14: TELECONNECT workflow design after u-Split of C

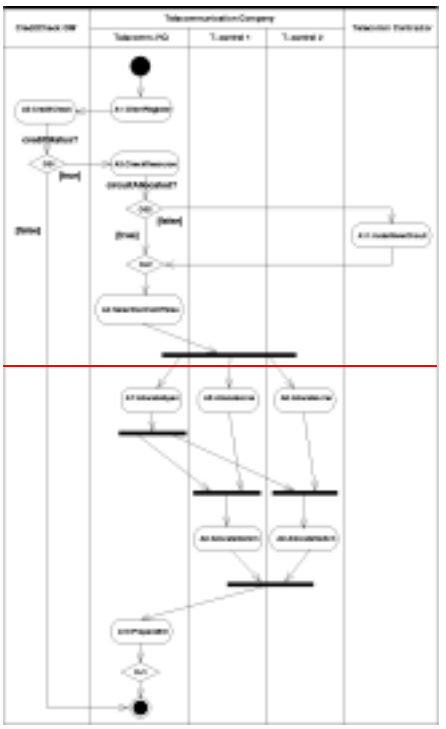


Figure 15: TELECONNECT workflow design after u-Split of B

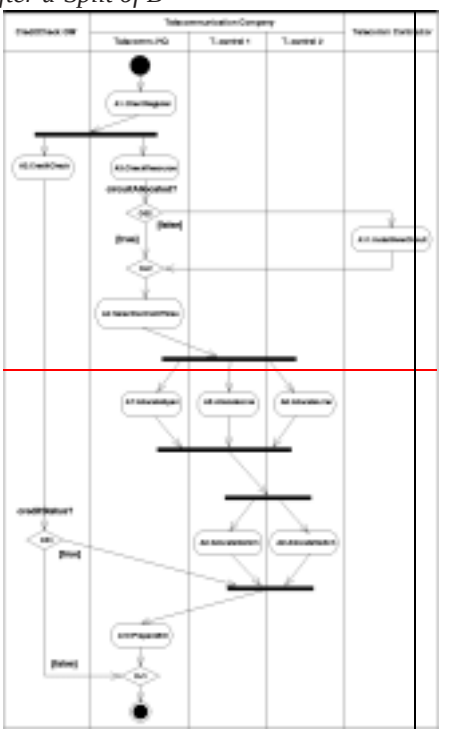


Figure 16: Parameter values for the uniform statistical function

Activity(ies)	Computer System	Min.	Max.
A_1, A_3, A_4, A_7, A_{10}	Telecomm-HQ	3.2 sec.	5.2 sec
A_5, A_6, A_8, A_9	T-central	6.4 sec.	10.4 sec.
A_2	CreditCheck-GW	2.0 sec.	22.0 sec.

C defines 90% percentile as the upper limit for response time on benchmarking complex OLTP application environments. Thus, 90% percent of the activity instances executed in Telecomm-HQ must show an elapsed time not greater than *5.0 seconds*. Analogously, *10.0 seconds* and *20.0 seconds* correspond to T-central and CreditCheck-GW activity instances, respectively. The exponential statistical distribution, describing jobs that arrive independently, has been used to define the time interval between the start of each workflow instance. These values considered, the simulation environment has been calibrated to execute 165 workflow instances in parallel. Such fine-tuning has been obtained by using 0.2 second as the input for the exponential statistic function.

We assess the effectiveness of workflow restructuring by comparing the execution of TELECONNECT workflow with and without restructuring of its composite activities *B*: ALLOCATECIRCUIT and *C*: ALLOCATELINES. As defined earlier, the sub-activities A_2: CREDITCHECK and A_6: ALLOCATE LINE are the chosen activities to be controlled, namely, the latency of these activities will be increased in the presence of disturbance. The population of the set of workflow instances (cases executed in parallel) varies from 1 to 300. Then, the same populated set of workflow instances is executed for each variation of TELECONNECT workflow. Figure 17 shows the activities being controlled at the simulation, the type of disturbances considered and the amount of delays occurred. To simulate a controlled activity instance facing a disturbance, the elapsed time obtained from the uniform statistical function is increased by the value stated in Figure *17*. For example, if uniform function returns 7.0 seconds for an A_2 instance, and CreditCheck-GW faces *very low spare gateway computer* disturbance, the simulator increases this elapsed time by 100%. Hence, such A_2 instance is simulated considering 14.0 seconds as its elapsed time.

There are two types of disturbances stated in Figure 17. The first type is the disturbance caused by a computing system with lower computational power. Three different computational powers are chosen: slightly low spare computer, low spare computer and very low spare computer. We consider that a *slightly low* computer causes a typical delay of 20% on the average elapsed time of the controlled activity and represents a computing system with a similar performance to the original one. Analogously, *low* computers and *very low* computers cause typical delays of 50% and 100%, respectively. The last two represent significantly slower computing systems. We adopt these three computing system disturbances for both controlled activities as being typical of real situations. Different elapsed time increases could be assumed to perform the simulation experiments. Such assumptions are reasonable because the typical elapsed time of an activity execution is the sum of its CPU-time, I/O-time and network-transfer-time. Hence, a loss in performance is expected when replac-

Figure 17: Effects of disturbances on the elapsed time of controlled activities A_2 *and* A_6

Activity	Type of disturbance	Elapsed time increase
A_6	*Slightly low spare computer*	*20%*
A_6	*Low spare computer*	*50%*
A_6	*Very low spare computer*	*100%*
A_6	*Low 1Mbits/sec network connection*	*10%*
A_6	*Low 256kbits/sec network connection*	*40%*
A_2	*Slightly low spare computer*	*20%*
A_2	*Low spare computer*	*50%*
A_2	*Very low spare computer*	*100%*
A_2	*Low 256kbits/sec network connection*	*36%*
A_2	*Low 56kbits/sec network connection*	*80%*

ing a computer system by one of lower power. However, such loss hardly matches the same reducing degree of computational power. At least, the network connection remains with the same transfer speed.

The second type is the disturbance caused by a network delay. In this case, the disturbances stated in Figure *17* are peculiar to each controlled activity because the network connection speeds are rather different (10Mbits/sec among Telecomm-HQ and T-centrals, and 1Mbits/sec between Telecomm-HQ and CreditCheck-GW). However, the network speeds adopted depict just typical transfer rates found in the real world. Different network speeds could be assumed to perform the simulation experiments. For the controlled activity A_6, we assume a slight delay of 10% caused by network speed falling to *1Mbit/sec* and an average delay of 40% by *256kbits/sec* network speed. In the same way for the controlled activity A_2, we assume an average delay of 36% caused by a low *256kbits/sec* network connection and a high delay of 80% due to a network connection with *56kbits/sec*. As a result, we have used a wide range of delays to test workflow restructuring in different situations.

In this section, the CSIM-based simulator is used primarily to demonstrate the properties of the restructuring operation rather than carry out a detailed analysis of the algorithm for execution restructuring operators. To study the behavior of the restructuring operators in the experiments, various delays were generated by simply applying a uniform probabilistic function provided in CSIM, rather than stochastically generating delays. Consider the values in Figure 16 as an example. The elapsed time for each activity instance can be estimated using the average of the Min and Max values, or more realistically, the elapsed time of activity instances should be measured as random values within a time interval because two instances of the same activity can perform a different number of I/O operations, demand a different amount of data transfer across the network and execute different sets of CPU operations. Taking into account all typical elapsed time for activity instances, the expected elapsed time for a TeleConnect workflow instance is 45.6 seconds (without A_{11}: INSTALLNEWCIRCUIT execution). By applying u-Split of *B*: ALLOCATE CIRCUIT, such elapsed time becomes 33.6 seconds. When an activity instance of A_2: CREDITCHECK suffers the effects of one disturbance listed in Figure 17, the elapsed time of a TELECONNECT workflow instance grows linearly while the elapsed time of the restructured version remains the same up to a 110% delay amount. Only when the delay amount exceeds 110% of the elapsed time of a restructured

TELECONNECT workflow, does the elapsed time also start to grow linearly. However, these results take into consideration none of the effects of the disturbances in the other computational components. As demonstrated before, such disturbances overload the environment and deteriorate the performance of its components, and the average elapsed time of TELECONNECT workflow instances presents different behavior.

Experimental Results

The goal of our experimental study is to show the benefits and costs of dynamic activity restructuring. Concretely, the experiments are set to maximize parallel execution of ongoing workflow instances (WI) by reorganizing the hierarchical structure of the root activity. Our experiments examine and compare the workflow execution with and without restructuring in the following two situations: (1) a temporarily non-optimal runtime environment, and (2) an unexpected malfunction in some infrastructure component. The types of disturbances considered are listed in Figure 17.

To properly evaluate the effectiveness on restructuring workflow instances, a simulation for TeleConnect workflows without restructuring is performed in an environment without disturbances. The goal of this simulation is to determine the population of ongoing WI executed in parallel that presents the highest average elapsed time satisfying the 60-second company goal. The resulting population becomes the reference to understand the effectiveness of workflow restructuring in a runtime environment with disturbances. To authenticate this population, sets of TeleConnect WI with different populations (from 1 to 300 cases in parallel) are executed considering the environment specified earlier. Figure 18 plots the simulation results for each set of WI.

In Figure 18, the *x-axis* shows the population of each set of WI. In other words, it shows how many WI are executed in parallel and concurrently have used the limited resources of the computing environment. The *y-axis* presents the corresponding average elapsed time of a set of WI. The line shows the results for the TeleConnect workflow. As expected, a higher number of WI executed in parallel raises their average elapsed time. The special point marked in Figure 18, (165, 59.8), shows the desired population: 165 is the number of workflow instances, executing in parallel, that presents the highest average elapsed time and satisfies the 60-second upper limit.

We adopted only one type of graph to present the experimental results already discussed. All graphs plot the average elapsed time of 165 WI executed in an environment where instances of a controlled activity (A2 or A6) face the delays defined in Figure 17. The dashed line depicts results for WI without restructuring and the continuous line plots result, considering a restructuring criterion. The *x-axis* shows the percent values of delays the controlled activity faces and the *y-axis* shows the average elapsed time of WI. Each asterisk in the continuous line corresponds to one of the percent values listed in the column *elapsed time increase* of Figure 17. In the graphs related to controlled activity A2 (Figures 19 and 21), the asterisks correspond to average WI elapsed time for 20%, 36%, 50%, 80% and 100% of percent delays. Similarly, the asterisks in the graphs related to controlled activity A6 (Figures 20 and 22) correspond to 10%, 20%, 40%, 50% and 100% of percent delays. The uniformity of this graph permits us to directly

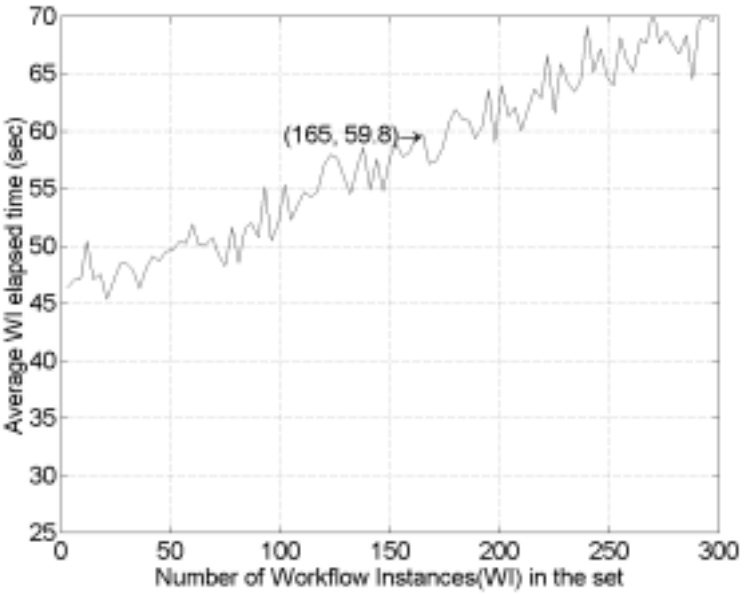
compare simulation results for different restructuring criteria (at start and checkpoint 25%) and the control of different activities (A2 and A6).

Experiment 1: Temporarily Non-Optimal Environment

The goal of this experiment is to comprehend the advantages and limitations of workflow restructuring, when the runtime environment presents one of the disturbances stated in Figure 17. The restructuring of activities takes place *before* the start of each WI. We compare cases of running TeleConnect workflows with and without restructuring for each disturbance listed. Each case has 165 WI in the set. The issue that must be answered by this experiment is: which disturbances in the computing environment can be properly managed if workflow restructuring takes place at the start of controlled activities? To answer this question, it is necessary to check the average WI elapsed time of the WI set for each percent value of delay on executing controlled activity instances. A particular disturbance can be properly managed by workflow restructuring if the resulting average WI elapsed time is less or equal to 60 seconds. Figure 19 show results for the controlled activity A2 and Figure 20 shows results for A6, the other controlled activity.

The dashed line in Figure 19 plots the average WI elapsed time of TeleConnect workflow without restructuring for the different delay instances of A2 face. The average WI elapsed time grows linearly as A2 delays increase. For example, 50% of average A2 delay increase corresponds to 74.9 seconds for the average WI elapsed time. Similarly, 100% corresponds to 91.6 seconds. Taking into account 50% and 100% of average A2 delay means about 18 seconds and 24 seconds, respectively, for the average elapsed time of A2 instances; an increase of 6 seconds in the A2 average elapsed time then implies an increase of 16.7 seconds to the average WI elapsed time. In other words, for each additional second of delay for A2 instances,

Figure 18: Results on simulating TeleConnect WI without delays and disturbances



2.78 extra seconds for the average WI elapsed time will result, a 2.78 growth factor. This result shows the overload caused by disturbances into the computing environment. This dashed line is present also in Figure *21* with exactly the same results and meaning. It is the reference to compare results from different restructuring criteria.

The continuous line in Figure 19 plots average WI elapsed times with B restructuring at the start. In this simulation, all 165 WI are restructured before the start of their A2 instances. The average WI elapsed time grows as A2 delays increase. But its growth factor also increases. For example, in the segment 0% to 20% (average A2 elapsed times 12 seconds and 14.4 seconds, respectively) the average WI elapsed time grows from 49.5 seconds to 49.6 seconds. Hence, the growth factor is 0.04 (average WI elapsed time grows 0.04 seconds for each second of delay in the average elapsed time of A2 instances). But considering the segment 50% to 100% (18 seconds and 24 seconds, respectively), the average WI elapsed time grows from 53.5 seconds to 67.4 seconds, and the growth factor is 2.3. The transition between the two segments above presents 1.08 as the growth factor. A growth factor of less than 1.0 means that the computing environment still presents availability to perform more WI. On the other hand, growth factors greater than 1.0 show an overloaded environment. Moreover, the point where the line shows 60 seconds for average WI elapsed time is 73%. Hence, disturbances that cause delays on A2 instances up to 73% are properly managed if B workflow restructuring takes place at start.

Similar to Figure 19, the dashed line in Figure 20 plots the average WI elapsed time of TeleConnect WI without restructuring for delays in A6 instances. The average WI elapsed time grows as A6 delays increase. For delays over 20%, the growth factor is virtually constant. In fact, the delays 20%, 40%, 50% and 100% (average A6 elapsed times 10.1 seconds, 11.8 seconds, 12.6 seconds and 16.8 seconds, respectively) correspond to 64.5 seconds, 71.1 seconds, 74.4 seconds and 91.3 seconds, and the growth factor increases from 3.9 to 4.0. For the two first segments, (0%, 59.8 seconds) - (10%, 61.4 seconds) and (10%, 61.4 seconds) - (20%, 64.5 seconds), the growth factors are 1.9 and 3.7, respectively. These results confirm the assumption that delays on activity instances overload the computing environment, as observed in the dashed line of the Figure 19.

Figure 19: B restructuring at start

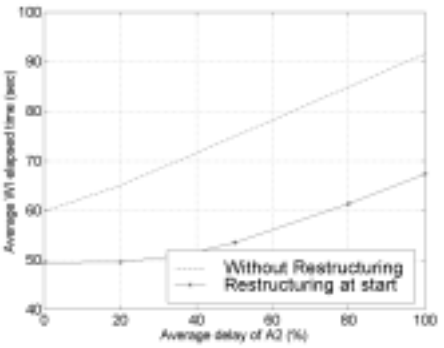
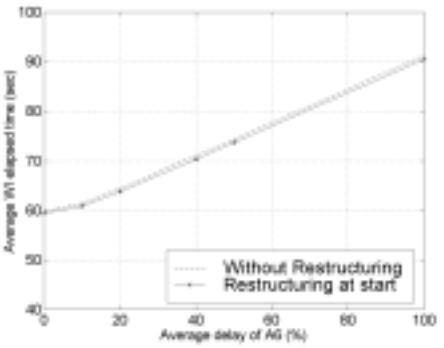


Figure 20: C restructuring at start



This dashed line is also used as the reference to compare results from different restructuring criteria in Figure 22, with exactly the same results and meaning.

The continuous line in Figure 20 plots average WI elapsed times with C restructuring at the start. All 165 WI are restructured before the start of A6 instances. This line shows virtually the same shape depicted by the dashed line, with *y-values* about 0.7 seconds shorter. For example, 63.9 seconds corresponds to 20% of delay and 90.5 seconds corresponds to delay of 100%. It means a very narrow gain on C restructuring and only delays not greater than 4% are properly managed with C restructuring at the start.

Experiment 2: Unexpected Malfunction of Infrastructure Component

The goal of this experiment is to examine the pros and cons of dynamic workflow restructuring, when the runtime environment presents some disturbance and the restructuring of activities takes place *during* the execution of WI. The disturbances are detected at 25% checkpoint. We compare cases of running TeleConnect workflows with and without restructuring for each disturbance listed in Figure 17. Each case has 165 WI in the set.

The choice of 25% as a checkpoint to verify whether a particular WI should be restructured represents the ability of the WfMS to monitor the computational environment and to react early when it detects disturbances. Considering the Min and Max values defined in Figure 16 for each activity, the expected elapsed time (E-ET) for an A2 instance is 12 seconds and, for an A6 instance, is 8.4 seconds. When an ongoing instance of A2 (or A6) is running, at 25% for its E-ET the simulator estimates which will be its real elapsed time. If the estimated elapsed time overcomes its E-ET then the workflow restructuring takes place. For a simulation controlling A2 instances, the simulator estimates the elapsed time of an ongoing A2 instance at 3 seconds of its starting. If the estimated value is greater than 12 seconds then the corresponding WI is restructured by *u-Split* of B. Similarly, if A6 instances are being controlled, the checkpoint occurs at 2.1 seconds of an ongoing A6 instance execution and the restructuring takes place if the estimated elapsed time value overcomes 8.4 seconds. Consequently, the simulator only restructures an ongoing WI if it estimates the elapsed time of the activity instance being controlled are greater than its E-ET. Moreover, each set of simulated WI probably has restructured and not restructured instances. Hence, the issue that must be answered by this experiment is: which disturbances in the computing environment can be properly managed if dynamic workflow restructuring is checked at 25% checkpoint on controlled activities? To answer this question it is necessary to check the average WI elapsed time of the WI set, with and without restructuring, for each delay value on executing controlled activity instances. A particular disturbance can be properly managed if the resulted average WI elapsed time is less or equal to 60 seconds. Figure 21 shows results for the controlled activity A2 and Figure *22* shows results for A6. As stated before the dashed line in Figures 21 and 22 are exactly the same as those depicted in Figures 19 and 20, respectively.

The continuous line in Figure 21 plots average WI elapsed time where B restructuring affects only WI with delayed A2 instances. It shows a different behavior when compared with a previous restructuring criterion, restructuring at the start, depicted in Figure 19. In fact, the growth factor of

this line starts near 0.0 and begins to grow after 20% of average delay for A2 instances. In the graph, the asterisks plot the following points: (0%, 56.6 seconds), (20%, 56.7 seconds), (36%, 57.5 seconds), (50%, 59.0 seconds), (80%, 64.8 seconds) and (100%, 69.8 seconds). The percent values for average delays of A2 instances correspond, respectively, to 12.0 seconds, 14.4 seconds, 16.3 seconds, 18.0 seconds, 21.6 seconds and 24.0 seconds. Then, the growth factors for each segment are: 0.04, 0.4, 0.9, 1.6 and 2.1. The growth factor close to 0.0 in the first segment (between 0% and 20%) means that workflow restructuring can properly manage delays on A2 instances up to 20% without increasing the load over the computing environment and, consequently, without perturbing other running applications. On the other hand, only the disturbances that cause delays of up to 54% are properly managed by B workflow restructuring with 25% checkpoint. By comparing with the results from earlier, B workflow restructuring at 25% checkpoint supports lower delays on A2 instances considering the 60-second company goal.

Figure 22 plots average WI elapsed time for different delays affecting A6 instances. The continuous line shows results where C restructuring takes place on WI with delayed A6 instances. For delays over 20% the behavior of this line is the same as that presented in Figure *20* by the continuous line. The slight difference is at the start of the line. At 0%, the average WI elapsed time is 59.7 seconds while the same point, in the dashed line, is 59.8 seconds. These times at 10% are, respectively, 61.1 seconds and 61.4 seconds. It means a lower gain on C restructuring than that depicted in Figure 20 and only delays under 3% are properly managed with C restructuring at 25% checkpoint.

Experimental Observations and Discussion

The experiments presented in the last section show the effectiveness of the *u-Split* operator on restructuring workflow instances facing disturbances in the operational environment, under certain conditions. Experiments 1 and 2 demonstrate B restructuring of workflow instances (WI) before start or, at least, at 25% of expected elapsed time of controlled activity, permitting the achievement of the 60-second upper limit for three types of disturbance: *slightly low spare gateway computer*, *low spare gateway computer* and *low 256kbits/sec network connection* (Figure *17*). However, workflow-restructuring

Figure 21: B restructuring at 25%

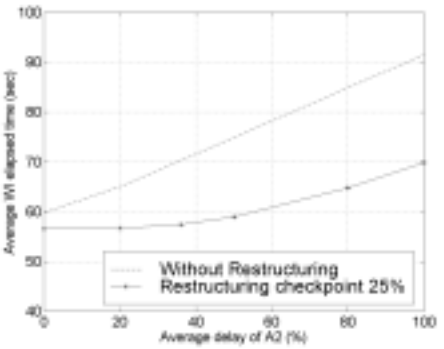
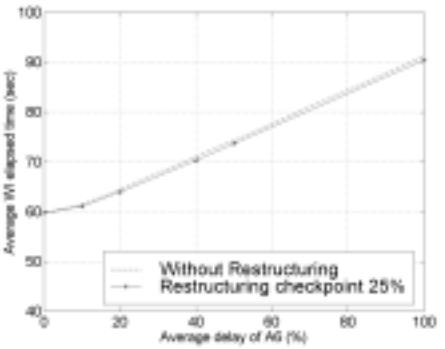


Figure 22: C restructuring at 25%



instances of activity A6 achieve the 60-second upper limit for none of the types of disturbances in Figure 17. Hence, only one of the two restructuring possibilities presented earlier is effective on satisfying the company goal when delays happen, and only part of the disturbances are properly managed.

To better evaluate the effectiveness of workflow restructuring in the experiments presented earlier, the experimental results are consolidated in Figures 23 and 24 for the controlled activities A2 and A6, respectively. The idea is to put in one graph the results of the workflow restructuring experiments. All Figures consider 165 as the population of simultaneous WI under execution.

Figure 23 shows the gains on restructuring A2 instances when occasionally facing delays. The *x-axis* depicts the percent values of average delays for A2 instances when facing disturbances. The *y-axis* depicts the gain on restructuring A2 by the difference between the average elapsed time of the set of workflow instances without restructuring and the average elapsed time of the same set with restructuring. The continuous line plots result for B restructuring is executed before the start of A2 instances. The dotted-dashed line plots results for B restructuring is at the 25% checkpoint . In a similar way, Figure 24 shows the gains on restructuring A6 instances. The *x-axis* depicts the percent values of average delays for A6 instances when facing disturbances, and the *y-axis* depicts the same difference of Figure 23 *y-axis*.

Figure 23 shows that the gains resulting from restructuring B increase independent of the moment the restructuring takes place. Moreover, restructuring is more effective if it takes place earlier. The same result is possible to be observed in Figure 24 but the values are too small. Figure 23 also presents the difference between average WI elapsed time growing faster for dynamic B restructuring at 25% checkpoint and considering lower delays (until 40%). Figure 25 shows the values used to plot the lines in Figure 23 and permits to observe better the behavior of its graphs. Considering Max and Min values in Figure 16, the percent values presented in Figure 17 correspond to 14.4 seconds (20%), 16.3 seconds (36%), 18.0 seconds (50%), 21.6 seconds (80%) and 24.0 seconds (100%). For 0%, the related average elapsed time is 12.0 seconds. For example, at 20% delay, the y-value in the dotted-dashed line (restructuring at 25% checkpoint) is 11.2 seconds. In the same curve, 4.4 seconds correspond to 0% of A2 delay. Hence, for 2.4 seconds of delay increase, B restructuring at 25% checkpoint grows 6.8 seconds in y-value.

Figure 23: Gains on B restructuring

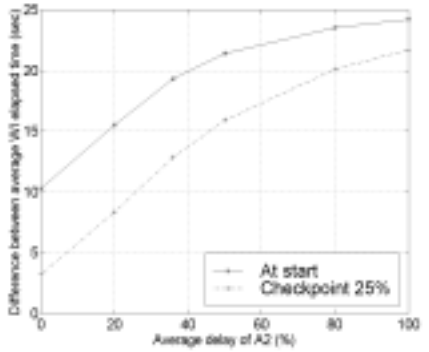


Figure 24: Gains on C restructuring

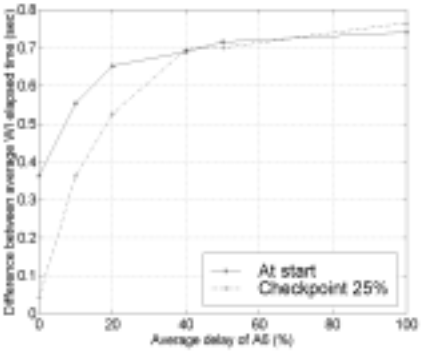


Figure 25: Differences between average WI elapsed times for WI with and without B restructuring

Average delay of A_2 (%)	Restructuring at start	Restructuring at 25% checkpoint
0%	10,3	3,2
20%	15,5	8,4
36%	19,4	12,9
50%	21,4	15,9
80%	23,5	20,1
100%	24,2	21,7

In other words, for each second of A2 delay, B restructuring at 25% checkpoint increases the difference between the average WI elapsed time without restructuring and with restructuring by 2.8 seconds. Then, 2.8 is the growth factor in this segment. In fact, B restructuring at 25% presents the higher growth factor in the interval 0% - 50%. Consequently, such restructuring criterion is the most effective to dynamically manage delays during execution of WI because it permits us to achieve the company goal for expressive delays on controlled activity A2, up to 54%, and it does not imply a significant increase of the load over the computing environment (only affected WI are restructured).

It is not possible to predict the exact amount of elapsed time saved after restructuring a workflow instance. The delay caused by a disturbance depends on the configuration and current workload of the computing infrastructure, and these factors are changing constantly. However, it is possible to characterize scenarios where the chances to save elapsed time are great. Each scenario corresponds to a possible workflow modeling based on the corresponding hierarchy of activity patterns. For the Telecomm Company, the TELECONNECT workflow is the base scenario (Figure 4) and the workflow model variants obtained by restructuring composite activities are the others (Figures 14 and 15). The analysis of the repercussions caused by a disturbance on the base and variant scenarios permit the evaluation of the possible benefits on restructuring the base workflow model. According to the simulation experiments, it is possible to state the restructuring of AllocateCircuit as being highly beneficial while the restructuring of AllocateLines is not. In fact, the restructuring of AllocateLines would not result in effective elapsed time saving of activity instances. Consequently, the modeling of a business process by a hierarchy of activity patterns must present restructuring opportunities. Nevertheless, these restructuring opportunities must enable a considerable amount of saved elapsed time when activities are running on a disturbed environment. It is important that the restructuring process take place only when the restructured workflow instance saves a considerable amount of time, because such process restructuring takes time when performed.

For a business process, such scenario analysis suggests that restructuring is beneficial for workflow instances facing some types of delays. Moreover, it suggests also that it is beneficial even in situations without facing delays. Although B workflow restructuring at start is not the most effective criterion on dynamically managing disturbances, such criterion presents the highest difference between average WI elapsed times: 10.3 seconds for A2 without delays (0% in Figure 23). Hence, workflow restructuring can be a way to improve performance of workflow instances when their

workflow models do not explore all possible concurrency among activities. When considering the workflow models to reflect the organizational structure, among other aspects, the restructuring approach presented in this paper enables the optimization of a company business process without necessarily the reengineering of the enterprise.

IMPLEMENTATION ISSUES

The implementation architecture for the first prototype of ActivityFlow is based on the World Wide Web (WWW) technologies. We use the HTML (HyperText Markup Language) to represent information objects required for workflow processes and to integrate different media types into a document. We access information from multiple remote information sources available on the Internet by using the uniform addressing of information via URLs (Uniform Resource Locators) and the transmission of data via HTTP (Hypertext Transfer Protocol), The HTML fill-in forms are the main interaction media between users and a server.

Figure 26 shows the implementation architecture of ActivityFlow. The HTTP server translates the requests from the users in the HTML forms to calls of the corresponding procedures of the prototype system of ActivityFlow using a CGI interface or a Java interface. The prototype implementation consists of three main components:

1. ***The workflow actor interface toolkit*:** It includes Web-based workflow process definition tool, administration and monitor tool, and workflow-application client-interface program.
   - W*orkflow process definition tool*
     We provide two interfaces for process definition: one is script-based language and the other is graphical interface that use the graph-based concepts such as nodes and edges between nodes. When a script language is used, the process definition tool will compile and load the workflow schema into the workflow database. We also provide a facility to map the script-based specification to iconic representation that can be displayed using a Web-browser. When a graphical interface is used to define the workflow procedure unit, a form will also be provided to capture the information required in the units such as header, activity declaration, role association, and data declaration. A script-based specification can also be generated upon request.
   - *Administration and monitoring tool*
     This module contains functions for creating, updating and assigning users, roles and actors, for the inspection and modification of the running process according to deadlines and priorities, including terminating undesired flow instances, and re-structuring on-going activities. The interactions with the users are supported primarily by creating and receiving HTML pages and forms.
   - *Workflow client interface*
     This module provides a number of convenient services for workflow clients, such as viewing the process state information, the worklist of an on-going process, and linking to the other relevant information, i.e., process description, process history, process deadlines, etc.. It interacts with the users via HTML pages and forms.

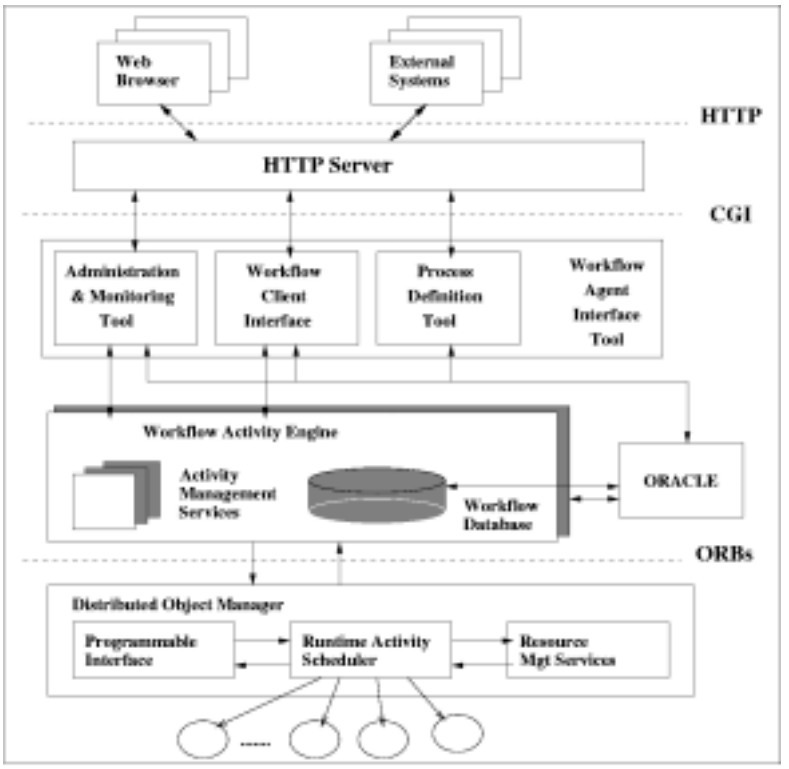
We are currently exploring the possibility of using or adapting production software, such as Caprera from Tactica (see URL http://www.tactica.com/) for managing and maintaining the activity dependency

specifications.

2. ***The workflow activity engine*:** It provides basic workflow enactment services, such as creating, accessing, and updating workflow activity description, providing the correctness guarantee for concurrent execution of activities and application-specific coordination control, and using the deadlines and priorities for scheduling and re-scheduling activity flows. We have done some initial study on the correctness properties of concurrent activity executions, such as compatibility and mergeability, using user-defined activity dependencies (Liu and Pu, 1998a). We are exploring possibilities to build value-added adapters on top of existing on-line transaction processing (OLTP) monitors, for example, using some recent results in open implementation (Barga and Pu, 1995) of extended transaction models (Elmagarmid, 1992) and the micro-protocols (Zhou, Pu and Liu, 1996) built on top of the Transarc Encina.
3. ***The distributed object manager*:** It provides consistent access to information objects from multiple and possibly remote information sources. The main services include resource manager, transaction router, and run-time supervisor. This component is built on top of the DIOM prototype system, an adaptive query mediation system for querying heterogeneous information sources (Lee, 1996).

To adapt our implementation architecture to the open system environment, we allow a flexible configuration of the actor interface, the workflow engine base, and the distributed object manager. For

Figure 26: Implementation Architecture of ActivityFlow



example, one scenario is to let the actor interface, the engine base and the distributed object manager all exist on different servers. Another scenario is to let the actor interface toolkit exist on one server and the activity engine base the distributed object manager exist on the same server but different from the actor interface server. We may also take the scenario that the actor interface toolkit and the activity engine base exist on the same server and the distributed object manager exists on a different server.

RELATED WORK AND CONCLUSION

In this paper, we have described the ActivityFlow approach to workflow process definition. Interesting features of ActivityFlow are the following. First, we use a small set of constructs and a collection of mechanisms to allow workflow designers to specify the nested process structure and the variety of activity dependencies declaratively and incrementally. The ActivityFlow framework is intuitive and flexible. Additional business rules can be added into the system simply through plug-in actors. The associated graphical notations bring workflow design and automation closer to users. And the restructuring operators can change an ActivityFlow diagram preserving their business process dependencies. Second, ActivityFlow supports a uniform workflow specification interface to describe different types (i.e., ad-hoc, administrative, or production) of workflows involved in their organizational processes, and to increase the flexibility of workflow processes in accommodating changes.

Several recent efforts have shared similar motivation as TAM. Eder and Liebhart (1995) present the WAMO formalism to describe workflow activities as a composition hierarchy and a set of execution dependencies. The expected exceptions are specified with activity hierarchies. Although both TAM and WAMO have their origins on extended transaction models and organizing activity descriptions as trees of activities, only TAM offers a set of restructuring operators capable of restructuring hierarchically organized activities. Kumar and Zhao (1999) present a similar approach of TAM (and WAMO) to describe the dependencies among activities. The properties of a business process are specified through sequence constraints and workflow management rules. However, the absence of a diagrammatic representation of a modeled business process makes the communication among designers and administrators difficult.

Our research and development for ActivityFlow continue along several dimensions. On the theoretical side, we are investigating workflow correctness properties and the correctness assurance in the concurrent execution of activities. On the practical side, we are building value-added adapters on top of existing transaction processing systems (Barga and Pu, 1995) to support extended transaction models and ActivityFlow specifications. In addition, we are exploring the enhancement of process design tools to interoperate with various application development environments.

In this paper, we propose a structured framework and a set of mechanisms for workflow process specification, not a new workflow model. The ActivityFlow framework is targeted towards advanced collaborative application domains, such as computer-aided design, office automation, and CASE tools, all of which require support for complex activities that have sophisticated activity interactions in addition to the hierarchically nested composition structure.

Furthermore, like most of the modeling concepts and specification languages, the proposed framework is based on pragmatic ground and hence no rigorous proof of its completeness can be given. Rather, its usefulness is demonstrated by concrete examples of situations that could not be handled adequately within other existing formalisms for organizing workflow activities.

ACKNOWLEDGMENT

We would like to thank the editor in chief, the area editor, and the reviewers for their helpful comments. Our thanks are also due to the ActivityFlow implementation team, especially to Roger Barga and Tong Zhou for their implementation endeavor on various components that form the basis for the ActivityFlow project, and to Jesal Pandya and Iffath Zofishan for their implementation effort on the ActivityFlow specification language and its graphical user interface. This project was supported partially by an NSF CCR grant, a DARPA ITO grant, and a DOE SciDAC grant.

ENDNOTES

[1] On post-doctoral internship at College of Computing, Georgia Institute of Technology, supported partially by grant from CAPES-Brazil .

REFERENCES

Ansari, M., Ness, L., Rusinkiewicz, M., & Sheth, A. P. (1992). Using flexible transactions to support multi-system telecommunication applications. In *Proceedings of the 18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada,* pages 65-76. Morgan Kauffman, August 1992.

Barga, R. S., & Pu, C. (1995). A practical and modular implementation of extended transaction models. In *Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland,* pages 206-217. VLDB Endowment, Morgan Kaufmann, September 1995.

Dayal, U., Hsu, M., & Ladin, R. (1990). Organizing long-running activities with triggers and transactions. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data,* Atlantic City, NJ, May 23-25, 1990, pages 204-214. ACM Press.

Eder, J. & Liebhart, W. (1995). The workflow activity model WAMO. In *Proceedings of the 3rd International Conference on Cooperative Information Systems* (CoopIS), Vienna, Austria, May 1995, 87-98.

Elmagarmid, A. K., editor. (1992). *Database Transaction Models for Advanced Applications.* Morgan Kaufmann.

Fowler, M. & Scott, K. (2000). *UML Distilled: a brief guide to the standard object modeling language*. Addison Wesley.

Gawlick, D., Hsu, M., & Obermarck, R. (1994). Strategic issues in workflow management. In *Proc. of the IEEE Conference.* IEEE Computer Society Press.

Georgakopoulos, D., Hornick, M. F., & Sheth, A. P. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases,* 3(2), 119-153.

Hollingsworth, D. & WfMC. (1995).*The workflow reference model,* WfMC, 1995. Available at http://www.wfmc.org/standards/docs/tc003v11.pdf (last access: 04/2003)

Hsu, M. & Kleissner, C. (1996). Objectflow: Towards a process manage-

ment infrastructure. *Distributed and Parallel Databases*, 4(2):169-194.

Kumar, A. & Zhao, J. L. Dynamic routing and operational controls in Workflow Management Systems. *Management Science*, 45(2):253-272.

Lee, Y. (1996). *Rainbow: Prototyping the DIOM interoperable system*, Department of Computer Science, Univ. Alberta, Canada, 1996. Technical Report TR96-32.

Leymann, F. & Roller, D. (2000). *Production workflow: concepts and techniques*. Prentice Hall.

Liu, L. & Meersman, R. (1996). The building blocks for specifying communication behavior of complex objects: An activity-driven approach. *ACM Transactions on Database Systems*, 21(2):157-207, June 1996.

Liu, L. & Pu, C. (1998). Methodical restructuring of complex workflow activities. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 342-350. IEEE Computer Society Press, February 1998.

Liu, L. & Pu, C. (1998a). A transactional activity model for organizing open-ended cooperative activities. In *HICSS'31, Hawaii International Conference on System Sciences, January 6-9, 1998, Big Island, Hawaii, USA, volume VII*, pages 733-742. IEEE Press.

McCarthy, J. & Bluestein, W. (1991). The computing strategy report: Workflow's progress. Forrester Research, Inc., 1991.

Mesquite Software, Inc. (1994). *CSIM18 Simulation Engine (c++ version) Users Guide*.

Mohan, C. (1994). A survey and critique of advanced transaction models. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, page 521. ACM Press..

Mohan, C., Alonso, G., Gunthor, R., & Kamath, M. (1995). Exotica: A research perspective of workflow management systems. *Data Engineering Bulletin*, 18(1):19-26..

Nodine, M. H. & Zdonik, S. B. (1990). Cooperative transaction hierarchies: A transaction model to support design applications. In *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*. Morgan Kaufmann.

Ruiz, D. D., Liu, L., & Pu, C. (2002). Distributed workflow restructuring: an experimental study, College of Computing, GeorgiaTech, Atlanta, USA, 2002. Technical Report GIT-CC-02-36.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

Sheth, A. P. (1995). Workflow automation: applications, technology and research. In *Proceedings of the ACM SIGMOD*, San Jose, CA, May 1995, 469-469.

Sheth, A. P., Georgakopoulos, D., Joosten, S., Rusinkiewicz, M., Scacchi, W., Wileden, J. C., & Wolf. A. L. (1996). Report from the NSF workshop on workflow and process automation in information systems. *SIGMOD Record*, 25(4):55-67.

T.-C. subcommittee. (2001). *TPC Benchmark C: Revision 5.0*. Transaction Processing Performance Council.

T.-W. subcommittee. (2001). *TPC Benchmark W: Revision 1.7*. Transaction Processing Performance Council.

WfMC. (2003). Workflow management coalition homepage: http://www.wfmc.org.

Zhou, T., Pu, C., & Liu, L. (1996).

Adaptable, efficient, and modular coordination of distributed extended transactions. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems, December 18-20, 1996, Miami Beach, Florida, USA.* IEEE Computer Society.

Zhou, T., Pu, C., & Liu, L. (1998). Dynamic restructuring of transactional workflow activities: A practical implementation method. In *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management, Bethesda, Maryland, USA, November 3-7, 1998.* ACM Press, November 1998.

Ling Liu is an associate professor of the College of Computing at Georgia Tech. Her research involves both experimental and theoretical study of distributed systems in general and distributed data intensive systems in particular, including distributed middleware systems, advanced Internet systems, and Internet data management. Her current research interests include performance, scalability, reliability, and security of Internet services, pervasive computing applications, as well as data management issues in mobile and wireless systems. Her research group has produced a number of software systems that are either operational online or available as open source software, including Continual Query systems over the Internet such as WebCQ, OpenCQ, and PeerCQ, information extraction and code generation systems such as XWRAPOriginal, XWRAPElite, and Omini. She is currently on the editorial board of International Journal of Very Large Databases (VLDBJ), editor-in-chief of ACM SIGMOD Record, and a vice PC chair of IEEE International Conference on Data Engineering (ICDE 2004). She was the PC co-Chair of 2001 International Conference on Knowledge and Information Management (CIKM 2001), held in November 2001 in Atlanta, and the PC co-Chair of the 2002 International Conference on Ontology's, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE), held in October, Irvine, California. She is a member of the IEEE. Her research is currently funded by NSF, DARPA, DoE, and other industry sources.

Calton Pu received his PhD from the University of Washington in 1986 and served on the faculty of Columbia University and Oregon Graduate Institute. Currently, he is Professor and John P. Imlay, Jr. Chair in Software at the College of Computing, Georgia Institute of Technology. He is leading the Infosphere project, part of the DARPA Information Technology Expedition program. Infosphere builds on his

previous and ongoing research interests. First, he has been working on next-generation operating system kernels to achieve high performance, adaptiveness, security, and modularity, using program specialization, software feedback, and domain-specific languages. This area has included projects such as Synthetix, Immunix, Microlanguages, and Microfeedback, applied to distributed multimedia and system survivability. Second, he has been working on new data and transaction management by extending database technology. This area has included projects such as Epsilon Serializability, Reflective Transaction Framework, and Continual Queries over the Internet. He has published more than 30 journal papers and book chapters, 100 conference and refereed workshop papers, and served on more than 40 program committees, including the co-PC chair of SRDS'95, co-general chair of ICDE'97, co-PC chair of ICDE'99, general chair of CIKM'01, co-PC chair of COOPIS'02, and co-PC chair of SRDS'03. He is currently an associate editor of DAPD and IJODL. His research is currently funded by NSF, DARPA, Intel, and other sources.

Duncan Dubugras Ruiz is an associate professor of Computer Science at the Post-Graduate Program on Computer Science, Faculty of Informatics, Pontifical Catholic University of Rio Grande do Sul, Brazil. His research interests include information systems modeling and design, in particular workflow modeling and automation, and applications of database technology, in special temporal and active databases, and XML database schemes. He spent a sabbatical year at the College of Computing, Georgia Institute of Technology - USA, working on self-restructuring workflow systems. He received a PhD degree in Computer Science from the Federal University of Rio Grande do Sul - Brazil in 1995, and a MSc. degree in 1987 from the same university.