ActivityFlow: Towards Incremental Specification and Flexible Coordination of Workflow Activities

Ling Liu and Calton Pu

Department of Computer Science & Engineering
Oregon Graduate Institute
P.O.Box 91000 Portland, Oregon 97291-1000 USA
{lingliu,calton}@cse.ogi.edu

Abstract. We introduce the ActivityFlow specification language for incremental specification and flexible coordination of workflow activities. The most interesting features of the ActivityFlow specification language include (1) a collection of specification mechanisms, which provides a workflow designer with 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; (2) a set of activity modeling facilities, which enables 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; (3) an open architecture that supports user interaction as well as collaboration of workflow systems of different organizations.

1 Introduction

The focus of office computing today has shifted from automating individual work activities to supporting the automation of organizational business processes. 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. One of the main challenges in today's WFMSs is to provide tools to support 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.

Over the past few years, many workflow management systems have become available on the market, or developed in research labs world wide [8, 10, 3]. Although there are more and more successes in the workflow research and development, it is widely recognized [8, 10] 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)[1], 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.

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 enables 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.

2 Basic Concepts of ActivityFlow

2.1 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 *agents*, which execute these activities. An agent may be a human (e.g., a customer representative), an information system, or any of the combinations. A notable difference between business process and workflow process is that a workflow process is an

[1] Ad-hoc workflows are controlled by users at run time. Users can react to situations not considered at workflow design stage. Administrative workflows are those workflows where activities are mainly performed by humans. Production workflows are predefined and use a great deal of complex information structures, and involve application programs and automatic activities [3, 7].

automated business process, namely the coordination, control and communication of activities is automated, although the activities themselves can be either automated or performed by people [10].

A workflow management system (WFMS) 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.

2.2 Reference Architecture

Figure 1 shows the WFMS reference architecture provided by the Workflow Management Coalition (WfMC) (see `http://www.aiai.ed.ac.uk/WfMc/`). A

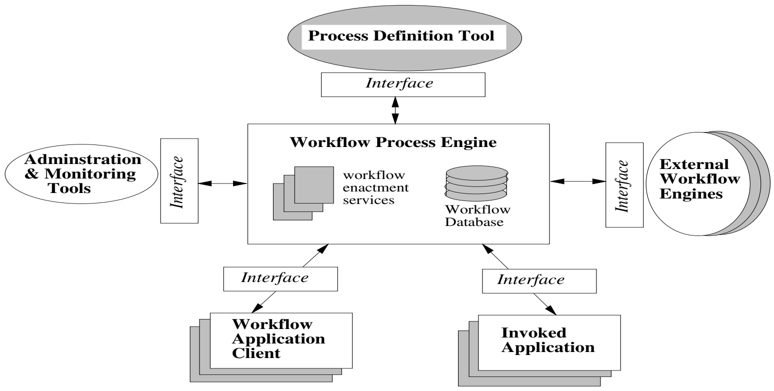

Fig. 1. Reference Architecture of Workflow Management Coalition

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 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. In this paper our main focus is on the ActivityFlow process definition facilities, including the ActivityFlow meta model (see Section 2.3), the ActivityFlow workflow specification language (see Section 3) and graphical notation for ActivityFlow process definition (see Section 3.4).

2.3 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 such that it can be statically (before execution) determined that all branches are taken is called AND-split. A split which can be statically determined that exactly one of the branches will be taken is called OR-split. Figure 2 lists the graphical representation of AND-split, OR-split, AND-join, and OR-join.

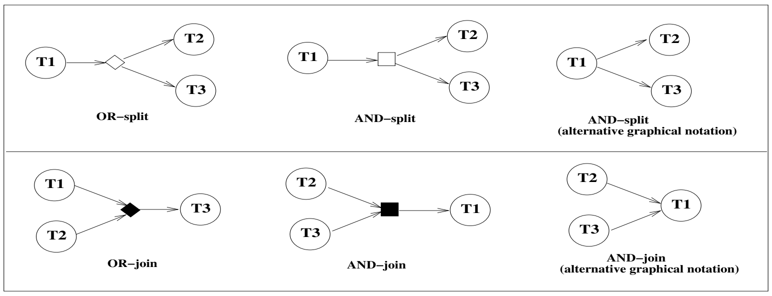

Fig. 2. Graphical representation of AND-split, OR-split, AND-join, and OR-join

The workflow process schema also specifies which agents 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 agents. Each activity has an associated role that determines which agents can execute this activity. Each agent has an activity queue associated with it. Activities submitted for execution are inserted into the activity queue when the agent is busy. The agent 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 agents and can flexibly accommodate changes in the workforce and in the computing infrastructure of an organization, by changing the set of agents associated with roles.

Figure 3 shows a sketch of the ActivityFlow meta model using extended ER diagram. The following concepts are basics of our activity-based process model:

- *a workflow process*: consists of a set of activities, a role 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 agents, who are the authorized performers that can execute the activity. The concept of associating roles with activities not only allow us to establish the rules for association of

activities or processes with organizational responsibilities, but also provide 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 agent*: can be a person, a group of people, or an information system, that are granted memberships into roles and that interacts with other agents 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.

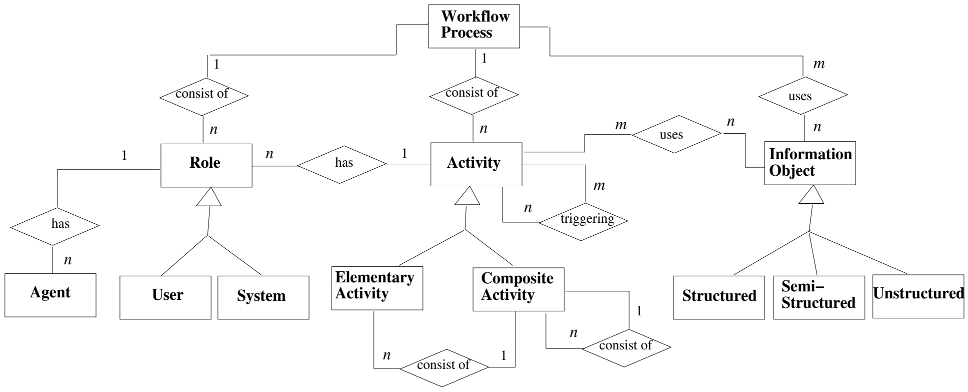

Fig. 3. ActivityFlow meta model

Important to note is that activities in ActivityFlow can be manual activities, performed by users without further support from the system; automatic activities, carried out by the system without human intervention, or semi-automatic activities, using specific interactive programs for performing an activity.

2.4 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.

Assume a telephone service provisioning workflow consists of the following eight activities, each corresponds to a step in the telephone service provisioning process. An instance of this telephone service provisioning workflow is created when an operator collects from a client the information needed to carry out the connection installation service. The client information required includes client name, address, kind of service and requested options. Figure 4 illustrates this example which we will use in the rest of the paper. The activity T_1:EnterRequest

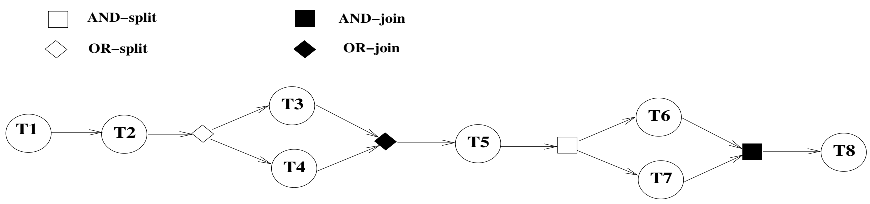

Fig. 4. Telephone Service Provisioning Workflow Process

verifies the service request and creat service order in the client database and service order database. The next activity T_2:CheckResource uses the service order created by T_1 and consults the facilities database to determine whether existing facilities can be used for establishing the service. If existing facilities can be used, the activity T_3:AllocateCircuit is executed. Otherwise, a human field engineer is selected to execute the activity T_4:InstallNewCircuit, which may involve manual changes to some switch and the installation of a new telephone line. Once T_3 or T_4 is completed, the activity T_5:VerifyInstallation is executed to verify if the installation was successful. Then the T_6:activity UpdateDirectory and the activity T_7:PrepareBill are executed in parallel to update the telephone directory database and generate billing information, respectively. Finally, the activity T_8:GenerateSummary is executed to generate some summary data about the provisioning of the new telephone service.

2.5 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 concept 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 which 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 such workflow process administrator to achieve all these goals.

3 ActivityFlow Process Definition Language

3.1 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. 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.

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. We use activities to refer to both the process and its component activities when no distinction needs to be made.

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 agents that are granted membership into the role based on their responsibility in the business process or in the organization. Each agent is described by agent 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 manipulate the actual arguments. Constraints between these messages are also specified in this unit [5].
- *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.

3.2 Dynamic Assignments of Agents

The assignment of agents (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: (1) When the set of agents is empty, the assignment of agents can be any users or systems that belong to the roles associated with the specified activity. When the set of agents is not empty, only those agents listed in the associated agent set can have the privilege to execute the activity. (2) The assignment of agents 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. (3) The assignment of agents 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 agents provides great flexibility and breath of application. By statically and dynamically establishing and defining roles and assigning agents 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.

```
ACTIVITY TeleServProv
PARAMETER
    In: ClientId:CLIENT,
    In: Start:POINT,
    In: End:POINT,
    Out: CircuitId:CIRCUIT
ACCESS_TYPE Write
SYNOPSIS
    Telephone service provisioning
MAX_ALLOWABLE_TIME 3 weeks
ADMINISTRATOR
    UID: 0.0.0.337123545
EXCEPTION_HANDLER
    none
ROLE_ASSOCIATION
    Role_name: Tele_Service Officer
    Role_type: User
    Agent_Set:
        ⟨UID: 0.0.0.135678221⟩,
        ⟨UID: 0.0.0.355983145⟩;
DATA_DECLARATION
    import class CLIENT,
    import class POINT,
    import class CIRCUIT;
    import class SummaryTable;
BEGIN
    COMPOSITION
        T1: EnterRequest(InOut: ClientId:CLIENT, In: Start:POINT, End:POINT)
        D: InstallCircuit(In: Start:POINT, End:POINT, Out: Status:Boolean)
        T6: UpdateDirectory(InOut: ClientId:CLIENT)
        T7: PrepareBill(InOut: ClientId:CLIENT, In: CircuitId:CIRCUIT)
        T8: GenerateSummary(In: ClientId:CLIENT, CircuitId:CIRCUIT, Out: Sum:SummaryTable)
    EXECUTION_DEPENDENCY
        ExeR1: T1 precede D
        ExeR1: D precede T6, T7
        ExeR2: T6, T7 precede T8
    INTERLEAVING_DEPENDENCY
        ILR1: T1 precede T2
        ILR2: T5 precede T6, T7
    STATE_TRANSITION_DEPENDENCY
        STR1: abort(D) enable abort(TeleServProv)
END
```

Fig. 5. An Example specification of the top activity TeleServProv

3.3 Control Flow Specification: Activity Dependencies

In ActivityFlow, we provide four constructs to model various dependencies between activities. They are **precede, enable, disable**, and **compatible**. 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.

Consider the telephone service provisioning workflow example, a sample specification of the top activity TeleServProv is presented in Figure 5.

3.4 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 connector, join connector described in Section 2.3), 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 in Section 3.3. 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, 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, $CN = \{$AND-split, OR-Split, AND-join, OR-join, Iterator$\}$, bn denotes the begin node and en denotes the end node.
  - Each node nd_i $(i = 1, .., n)$ is described by a quadruple (NT, NN, TC, NS), where
    NT is the node type. An activity node has two types: *simple* and *compound*. A connector node has three types: split, join and spawn.
    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*.
- $E = \{e_1, e_2, ..., e_m\}$ is a set of edges.
  - Each edge is of the form $nd_i \longrightarrow nd_j$.
  - An edge e_{ij}: $nd_i \longrightarrow nd_j$ is described by a triple $(EN, DPnd, AVnd, ES)$, 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 $nd_i \xrightarrow{*} nd_j$, if and only if one of the following conditions is verified:

(1) $nd_i = nd_j$.
(2) $nd_i \longrightarrow nd_j \in E$.
(3) $\exists nd_k \in N, nd_k \neq nd_i \wedge nd_k \neq nd_j$ s. t. $nd_i \stackrel{*}{\longrightarrow} nd_k \wedge nd_k \stackrel{*}{\longrightarrow} nd_j$. □

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 guarantees that the graph $G =< N, E >$ is acyclic, the following restrictions are placed:
(1) $\forall nd_i, nd_j \in N$, if $nd_i \longrightarrow nd_j \in E$ then $nd_j \longrightarrow nd_i \notin E$.
(2) $\forall nd_i, nd_j \in N$, if $nd_i \stackrel{*}{\longrightarrow} nd_j$ then $nd_j \stackrel{*}{\longrightarrow} nd_i$ does not hold.

To illustrate the definition, let us recast the telephone service provisioning workflow procedure described in Figure 5 in terms of the above definition:

- $N = \{Begin(T), T_1, D, T_6, T_7, T_8, End(T)\}$
- $E = \{Begin(T) \longrightarrow T_1, T_1 \longrightarrow D, D \longrightarrow T_6, D \longrightarrow T_7, T_6 \longrightarrow$ AND-join, $T_7 \longrightarrow$ AND-join, AND-join $\longrightarrow T_8, T_8 \longrightarrow End(T)\}$
- $TC(Begin)$: `NeedService`
  $TC(T_1)$: `NeedService`
  $TC(D)$: $commit(T_1)$
  $TC(T_6)$: $Status(D)$ = `true`
  $TC(T_7)$: $Status(D)$ = `true`
  TC(AND-join): $terminate(T_6) \wedge terminate(T_7)$
  $TC(T_8)$: $commit(T_6) \wedge commit(T_7)$
  $TC(End)$: $commit(T_6)$

Note that `NeedService` is a boolean variable. When a new telephone service request arrives, `NeedService` is true. Figure 6 shows the use of the ActivityFlow graphical notations to specify this activity flow procedure. When a node

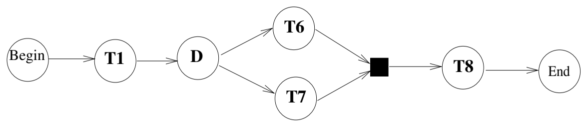

Fig. 6. Graphical representation of the flow procedure of top activity TeleServProv

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 6, it is obvious that activity node T_8 is reachable from nodes T_1, D, T_6 and T_7, and activity nodes T_6 and T_7 are reachable from nodes T_1 and D respectively.

An activity flow procedure G is instantiated by an instantiation request issued by an agent. 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 agent who issued the firing satisfies the defined role specification.

Definition 3. *(valid flow instantiation request)*
Let $G =< N, E >$ *be the activity flow and* $u = (agent\_oid, role\_name)$ *be an agent requesting the activity flow instantiation* T *of* α*. The flow instantiation* T *is* valid *if and only if* $\exists \rho \in Role(G)$ *such that* $role\_name(u) = \rho$*.* □

When the agent 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. 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`.

In ActivityFlow, we use the term conditional rollback to refer to the situations which require to revisit the nodes previously terminated or not fired. Conditional rollbacks are a desirable functionality and encountered frequently in some business processes. We provide an iterator connector for realization of conditional rollbacks. We also define the termination property and precedence preserving property to guarantee the correctness of workflow execution. Due to the space limitation, the details are omitted here. Readers may refer to [6] for further discussion.

4 Conclusion

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 agents. The associated graphical notations bring workflow design and automation closer to users. 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.

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 [1] 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.

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. 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: *the workflow agent interface toolkit, the workflow activity engine*, and *the distributed object manager*. Readers who are interested in detailed implementation issues may refer to [6, 1, 11].

Acknowledgement This work was partially carried out when the first author was with the University of Alberta. Our thanks are also due to Roger Barga, David Buttler, Yooshin Lee, Kirill Richine, Wei Tang, and Tong Zhou for their implementation endeavour on various components that form the basis for the ActivityFlow project.

References

1. R. Barga and C. Pu. A practical and modular implementation technique of extended transaction models. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, September 1995.
2. A. K. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.
3. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From processing modeling to workflow automation infrastructure. *Distributed and Parallel Database*, 3(2):119–152, 1995.
4. M. Hsu and C. Kleissner. Objectflow: Towards a process management infrastructure. *Distributed and Parallel Databases*, (4):169–194, 1996.
5. L. Liu and R. Meersman. The basic building blocks for modeling communication behavior of complex objects: an activity-driven approach. *ACM Transactions on Database Systems*, 21(3):157–207, June 1996.
6. L. Liu and C. Pu. ActivityFlow: A formalism for incremental specification of workflow activities. Technical Report TR97, University of Alberta.
7. J. McCarthy. There is more than one kind of workflow software. *Computerworld*, November 2 1992.
8. C. Mohan. *Advanced Transaction Models - Survey and Critique*. Tutorial presented at the ACM SIGMOD international conference, 1994.
9. A. Sheth. *Workflow Automation: Applications, Technology and Research*. Tutorial presented at the ACM SIGMOD international conference, 1995.
10. A. Sheth,G. Georgakopoulos, S.Joosten, M. Rusinkiewicz, W. Scacchi, J. Wildedn, and A. Wolf. Report from the nsf workshop on workflow and process automation in information systems. *ACM SIGMOD Record*, 25(4):55–67, December 1996.
11. T. Zhou, C. Pu, and L. Liu. Adaptable, efficient, and modular coordination of distributed extended transactions. In *Proceeding of the International Conference on Parallel and Distributed Databases*, 1996.