On Transactional Workflows

Amit Sheth* Marek Rusinkiewicz†
amit@ctt.bellcore.com *marek@cs.uh.edu*

The basic transaction model has evolved over time to incorporate more complex transactions structures and to take the advantage of semantics of higher-level operations that cannot be seen at the level of page reads and writes. Well known examples of such *extended transaction models* include nested and multi-level transactions. A number of *relaxed transaction models* have been defined in the last several years that permit a controlled relaxation of the transaction isolation and atomicity to better match the requirements of various database applications. Correctness criteria other than global serializability have also been proposed. Several examples of extended/relaxed transaction models are reported in [5].

Recently, transaction concepts have begun to be applied to support applications or activities that involve multiple tasks of possibly different types (including, but not limited to transactions) and executed over different types of entities (including, but not limited to DBMSs). The designer of such applications may specify inter-task dependencies to define task coordination requirements, and (sometimes) additional requirements for isolation, and failure atomicity of the application. We will refer to such applications as *multi-system transactional workflows*. While such workflows can be developed using *ad hoc* methods, it is desirable that they maintain at least some of the safeguards of transactions related to the correctness of computations and data integrity. Below, we discuss briefly the specification and execution issues in this evolving field, with emphasis on the role of database transaction concepts.

The idea of a workflow can be traced to Job Control Languages (JCL) of batch operating systems that allowed the user to specify a job as a collection of steps. Each step was an invocation of a program and the steps were executed as a sequence. Some steps could be executed conditionally. This simple idea was subsequently expanded in many products and research prototypes by allowing structuring of the activity, and providing control for concurrency and commitment. The extensions allow the designer of a multitask activity to specify the data and control flow among tasks and to selectively choose transactional characteristics of the activity, based on its semantics.

The work in this area has been influenced by the concept of long running activities [3]. Workflows discussed in this paper may be "long running" or not. Other related terms used in the database literature are task flow, multitransaction activities [7], multi-system applications [1], application multiactivities, and networked applications [4]. Some related issues are also addressed in various relaxed transaction models.

A fundamental problem with many extended and relaxed transaction models is that they provide a predefined set of properties that may or may be not required by the semantics of a particular activity. Another problem with adopting these models for designing and implementing workflows is that the systems involved in the processing of a workflow may not provide support for facilities implied by an extended/relaxed transaction model. Furthermore, the extended and relaxed transaction models are mainly geared towards processing entities that are DBMSs that provide transaction management features (often assumed to be of a particular restrictive type), with the focus on preserving data consistency, and not on coordinating independent tasks on different entities, including legacy systems.

*Bellcore, RRC-1J210, 444 Hoes Ln., Piscataway NJ 08854. Please send email for an extended version.
†University of Houston, Houston, TX 77204-3475.

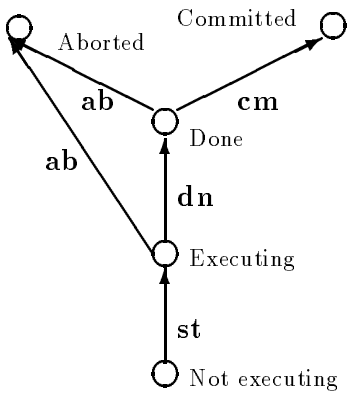
Specification of Tasks

A task in a workflow is a unit of work that is represented by sending a message, filling out a form, or executing a procedure, a contract or a transaction. A task can be processed by one or more entities, although we will limit our attention to the cases where a task is executed by only one entity, such as a DBMS or an application system.

An abstract model of a task is a state machine (automaton) whose behavior can be defined by providing a *state transition diagram* (task skeleton). As with the correctness of traditional transactions, on the workflow level we do not model internal operations of the task – we deal only with those aspects of a task that are externally visible or controllable. In general, each task (and the corresponding automaton) can have a different internal structure resulting in a different task skeleton. One example corresponding to a standard transaction with a visible prepared to commit state, is shown below (cf: [10, 2]).

A task specification may include:

- a set of (externally) visible execution states of a task including an initial state and one or more termination states,
- a set of significant events that lead to transitions between these states, with each event identified by an attribute such as forcible, rejectable, and delayable (these are required to enforce inter-task dependencies [2]).



A task can be specified independently of the entity that can execute it or by considering the capabilities and the behavior of the executing entity. In the former case, it may be necessary to determine which entity can execute the task or the workflow system should be able to adequately simulate the states not supported by the entity on which a task is executed. The latter case, in which a task is specified for execution by a specific entity or a specific type of entity is usually appropriate when dealing with existing (legacy) systems. The task skeleton then depends, to a large entent, on the characteristics of the system on which the task is executed. Some of the properties of the local system responsible for the execution of a task, like presence or absence of the two-phase commitment interface will directly affect the form of the task skeleton and thus, the definition of the activity. Other characteristics of an entity that executes a task may influence the properties of a task, without affecting its structure.

When the task is a transaction executed by a DBMS that provides a full range of transaction management functions, we need to take advantage of local concurrency control, commitment, recovery and access granting facilities. However, when the task is executed by an application system, we need to understand the application system semantics that affects its transactional behavior. Rather than

developing new "global" mechanisms that duplicate the functionality of local systems, we should build a model for managing multi-system workflows that utilizes the known task structures and semantics, coordination requirements of a collection of tasks, and execution semantics of systems that execute the tasks.

Workflow specification also consists of the conditions that affect the execution of tasks. These result from the specification of inter-task and inter-workflow execution requirements discussed next.

Dependencies and Correctness Criteria

Once the tasks constituting a workflow are specified, the internal structure of the workflow can be defined by specifying inter-task dependencies. Dependencies can be specified using a variety of software paradigms (e.g., rules, constraints, or programs). In general, dependencies can either be defined *á priori* (statically) or determined dynamically during its execution. In the first case, the tasks and dependencies among them are defined before the execution of the workflow starts. Some of the relaxed transaction models (e.g., [6],[13]) and [7]) use this approach.

A generalization of the static strategy is to have a precondition for execution of each task in the workflow or specific transitions of the tasks, so that all possible tasks in a workflow and their dependencies are known in advance, but only those tasks whose preconditions are satisfied, are executed [1]. Different initial parameters for the task may result in different executions of a task. The preconditions may be defined in terms of *execution states* of other (sibling) tasks, *output values* of other (sibling) tasks, and *external variables* including time and data states. The terms *execution dependencies, data or value dependencies and temporal dependencies* are used in the literature to refer to various scheduling preconditions. In the dynamic case, the task dependencies are created during the execution of a workflow, often by executing a set of rules. Examples of this kind of dependency specifications are found in long-running activities [3] and polytransactions [12].

The tasks of a workflow can communicate with each other through variables, local to the workflow and made persistent by the workflow system. These variables (including temporal variables) may also hold parameters for the task programs. The *data flow* between tasks is determined by assigning values to their input and output variables. In practice, there can be substantial difference in the format and representations of the data that is output by one task and input to another. The corresponding mapping and translation needs must be recognized but need not be an integral part of of the workflow model. The execution of a task has effects on the state of a database and the value of its output variable.

Additional aspects of intra- and inter-workflow specifications that are not captured using inter-task dependencies [1], include [11]:

- Failure atomicity requirements that can be defined using acceptable termination states of the workflow (committed or aborted).
- Execution atomicity requirements that define isolation properties of the workflow. Some of these requirements may be specified by providing the coupling modes between the tasks and requiring execution of tasks as atomic transactions,
- Dependencies that span across workflows. For example, it may be required that all tasks of one workflow must follow those of another at every execution entity.

[1] Some of these requirements are referred to as "correctness criteria" in [8].

Execution of Workflows

The correct execution of workflows involves enforcing all intertask dependencies, and assuring correctness of interleaved execution of multiple workflows. A scheduler (e.g., [2]) determines allowable transitions of each task based on different system and user events. These are then analyzed before allowing the corresponding transition(s) to take place or before terminating a workflow. By taking into account the semantics of tasks, workflows, and executing entities, we can significantly simplify the control needed to assure the correct concurrent execution of multiple workflows [9].

Two basic approaches to the implementation of a workflow management system can be identified: (a) An embedded approach that assumes that the executing entities support some active data management features. This approach is frequently used in dedicated systems developed to support a particular class of workflows and usually involves modification of the executing entities. (b) A layered approach that implements workflow control facilities on the top of uniform application-level interfaces to execution entities. A workflow manager based on such an approach is developed by the Carnot project at MCC. As a follow-on to the work reported in [1] and partly based on [2], we are currently working on a workflow management project that utilizes the latter approach.

Acknowledgements

Discussions with colleagues at Bellcore (N. Krishnakumar, L. Ness) and collaboration with members of MCC's Carnot project (P. Attie, P. Cannata, M. Singh, C. Tomlinson, D. Woelk) influenced this work.

References

[1] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth. Using Flexible Transactions to Support Multi-System Telecommunication Applications. In *Proceedings of the 18th VLDB*, August 1992.

[2] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and Enforcing Intertask Dependencies. In *Proceedings of the 19th VLDB Conference*, 1993.

[3] U. Dayal, M. Hsu, and R. Ladin. A Transactional Model for Long-Running Activities. In *Proceedings of the 17th VLDB Conference*, September 1991.

[4] E. Dyson. Workflow. In *Forbes*, November 1992, p. 192.

[5] A. Elmagarmid, editor. *Transaction Models for Advanced Database Applications.* Morgan-Kaufmann, February 1992.

[6] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on VLDB*, 1990.

[7] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating Multi-transaction Activities. Technical Report CS-TR-247-90, Princeton University, February 1990.

[8] D. Georgakopoulos, M. Hornick, and P. Krychniak. An Environment for Specification and Management of Extended Transactions in DOMS. Technical Report September, GTE Laboratories Inc., 1992.

[9] W. Jin, L. Ness, M. Rusinkiewicz, A. Sheth. Concurrency Control and Recovery of Multidatabase Workflows in Telecommunication Applications. In *Proceedings of the SIGMOD Conference*, May 1993.

[10] J. Klein. Advanced Rule Driven Transaction Management. In *Proceedings of IEEE COMPCON*, 1991.

[11] M. Rusinkiewicz, A. Cichocki, P. Krychniak. Towards a Model for Multidatabase Transactions, *International Journal of Intelligent and Cooperative Information Systems*, Vol 1, No. 3, 1992.

[12] A. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In [5].

[13] H. Wachter and A. Reuter. The ConTract Model. In [5].