A 3-level Atomicity Model for Decentralized Workflow Management Systems

Israel Z. Ben-Shaul
Technion-Israel Institute of Technology
Department of Electrical Engineering
Technion City, Haifa 32000
ISRAEL
issy@ee.technion.ac.il
phone +972-4-294689
fax +972-4-323041

George T. Heineman
Columbia University
Department of Computer Science
500 West 120th Street
New York, NY 10027
UNITED STATES
heineman@cs.columbia.edu
phone 212-939-7085

Abstract

Decentralized workflow management systems (WFMSs) provide an architecture for multiple, heterogeneous WFMSs to interoperate. Atomicity is a standard correctness model for guaranteeing that a set of operations occurs as an atomic unit, or none of them occur at all. Within a single WFMS, atomicity is the concern of its transaction manager. In a decentralized environment, however, the autonomous transaction managers must find ways to cooperate if an atomic unit is split between multiple WFMSs. This paper describes a flexible atomicity model that enables workflow administrators to specify the *scope* of multi-site atomicity based upon the desired semantics of multi-site tasks in the decentralized WFMS.

Keywords: transaction management, workflow interoperability, distributed systems, process modeling, software engineering environments, collaborative work.

©1996, Israel Z. Ben-Shaul and George T. Heineman

1 Introduction

In this paper we are concerned with the advanced atomicity support required by decentralized workflow management systems (WFMSs). In WFMSs, workflow specifications are explicitly defined, evolved, and carried out with the assistance of an *enactment engine*. Atomicity is the property that a group of workflow activities are all executed or none of them are. Conventional transaction managers ensure atomicity within the context of isolated transactions; transactions are rolled back as needed. We are concerned here mainly with recovery atomicity as opposed to concurrency atomicity (as defined in [6]).

WFMSs access and manipulate data in ways that cannot abide by such restrictions: (1) Activities may entail much work and require hours or days of operation. Simple rollback upon aborting a transaction might be undesirable. (2) Activities may lead to, or trigger, invocation of unforeseen related workflow activities, a common feature in WFMSs. This makes it hard to determine *a priori* the atomicity boundaries of transactions. (3) Activities may require cooperation with other activities on shared data, eliminating most locking-based concurrency-control mechanisms that enforce serializability, such as two-phase locking.

In a centralized WFMS, the enactment engine typically has full control over the activities and data it manages. *Decentralized* multi-site WFMSs (DWFMSs) are a loosely-coupled collection of distributed autonomous and heterogeneous WFMSs (i.e., a site) with interoperability mechanisms and minimal global control. Each member of the DWFMS can initiate activities that involve data from multiple sites (henceforth multi-site activities), and in doing so becomes the *coordinating site* that oversees the execution of (possibly) multiple remote *participating* sites. Such behavior, however, poses a complication. On one hand, a DWFMS must support the atomicity of multi-site activities, but on the other hand it may have to respect the autonomy of the local WFMSs regarding the management of access to their data. These are conflicting goals, and as shown in [7, 3], it is impossible to guarantee global atomicity on top of local autonomous transaction managers (TMs).

The conflict between local autonomy and global atomicity is particularly evident when a (long duration) global task consists of a combination of local and multi-site activities. The local activities may be privately encapsulated and only their interface may be known by the coordinating site, or even worse, even their interface may not be known (see Section 2). Thus, it might be prohibitive to allow local TMs to freely affect the global task and TMs at remote sites. (Consider, in the worst case, a "malicious" local activity that repeatedly aborts when invoked.) We would like to reduce the impact that local TMs can have on remote data and activities, and seek to minimize the effect of local aborts of subtransactions to a degree permissible by the global task.

This paper describes a flexible atomicity model for specifying the "scope" and "consistency" of atomicity based on the semantics of the global task being executed. In particular:

- Each workflow task decides whether it is atomic or not; atomicity is an optional property, not inherent. This decouples the notion of tasks as logical units of execution, from transactions.
- When global atomicity is required for correctness, local autonomy is compromised in favor of atomic commitment, and local transaction managers may be affected by the global task or by remote transaction managers.
- Most interestingly, when global atomicity can be compromised (at least for some segments of a task), local autonomy takes precedence over global atomicity.

In any case, all compromises of local autonomy or global atomicity are explicitly specified and agreed upon by participating sites on a per-task basis, as opposed to being imposed by a global authority and applied to all global tasks. Our atomicity model, presented in Section 3, assumes an underlying execution model and system architecture, which we first discuss in Section 2. This execution model has been implemented in the Oz DWFMS [1] on top of its rule-based workflow modeling language and the Pern transaction component [5], and is outside the scope of this paper.

2 System Architecture and Execution Model

A DWFMS consists of a set of local WFMSs that share no resources and communicate via message passing. Each WFMS consists of a local data manager (DM), local transaction manager (TM), and local workflow manager (WFM). We focus on operational autonomy (more specifically on execution and control autonomy as defined in [7]) and ignore issues of design autonomy, thus we assume that the TM components in all sites are structurally similar although sites may consist of heterogeneous workflow engines and/or workflow processes and employ different concurrency control policies.

Each WFM can access data items from its local DM or a remote DM. If the data is local, the WFM contacts its local TM and a *local transaction* is created to manage the data; otherwise, the WFM contacts the remote TM and that TM creates the necessary and that TM creates the necessary transaction. In contrast, distributed database systems [2] employ a global transaction manager that must process any request to access remote data. Although multi-site activities are "global", there is no centralized global TM. Instead, we assume that each local TM supports nested transactions and are extended to support multi-site transactions as explained below.

A workflow task consists of a partially-ordered set of activities — which typically involve invocation of external tools. Each activity in turn consists of a sequence of database accesses and updates. If an activity a_i only accesses data from its local data manager, then a single local transaction T_i is created to encapsulate the data requests for the activity. Each multi-site activity is created when two or more sites enter into an agreement, called *Treaty* [1], whereby the activity becomes "shared" between the multiple sites. A multi-site activity a_i involving n sites is associated with n transactions — $T_i^1, T_i^2, \ldots, T_i^n$ — one at each site.

From the perspective of transaction management, the execution of a global task interleaves multi-site "global" operations and single-site local operations. That is, a multi-site task (or, in Oz terminology, a Summit) alternates between global and local modes, where:

- *global mode* involves synchronous execution of multi-site activities at the coordinating site, involving data from multiple sites and possibly multiple users (e.g., using groupware tools).
- *local mode* involves execution of local (sub)tasks emanating from the global task at multiple sites. These local tasks can execute asynchronously and in parallel.

An interesting aspect of this execution model is that a multi-site task does not specify *which* local activities will be part of the task, and therefore the coordinating site requires no knowledge of the local activities or even of their interfaces. Instead, each participating WFMS only knows about the multi-site activities of a task, and the coordinating site asks each site to carry out local activities (in local mode). This model demands a high degree of freedom in balancing atomicity and autonomy, because it may be desirable in some cases to limit the impact that (unknown) activities may have on the (local and multi-site) work performed at other sites.

3 The Atomicity Model

Each WFMS determines the division of a task into activities, and as each activity completes, the enactment engine of each workflow manager may generate other activities that are to be carried out. For example, upon completion of an activity a_1, an enactment engine can require that activity a_2 be carried out and complete successfully, while activity a_3 need only be attempted (i.e., if a_3 fails to complete, a_1 is unaffected, but if a_2 fails, a_1 must be undone). This control flow determines the *atomic units* of a task; by default, each atomic unit is composed of exactly one activity. That is, the enactment engine must explicitly bind multiple activities together atomically to create larger atomic units; these atomic units can spread to cover multiple sites, as agreed upon by each site. We identify the following three levels of atomicity, each of which can be explicitly and separately specified on a per-task basis:

1. Level **G** (Global) — This provides atomicity for a single multi-site activity. It requires an atomic-commitment protocol such as two phase commit since each multi-site activity has a transaction acting on its behalf at the coordinating site and each participating site. If any of these transactions abort, for any reason, all transactions for the multi-site activity must abort, to preserve atomicity. This may be viewed as "horizontal" atomicity.

2. Level **GL** (Global to Local) — This "vertical" atomicity is an intermediate mode that preserves local autonomy and atomicity by compromising global atomicity. At a particular site, s, **GL** binds into an atomic unit the local transaction T_i^s, acting on behalf of the multi-site activity a_i, and the local transactions $\mathcal{L} = \{T_1, T_2, \ldots, T_k\}$ initiated for the emanating local activities at site s from a_i. Since each local activity is encapsulated by exactly one local transaction, **GL** only creates dependencies between transactions within the same site.

   Within a given site, s, the local transactions, $\mathcal{L}$, for the local emanating activities can commit independently from transactions at other sites, but they must be synchronized locally since they are part of an atomic unit. Once all the local activities emanating from a particular multi-site activity a_i have completed, the local transaction associated with the multi-site activity at T_i^s can commit as well as all the local transactions $\mathcal{L}$.

   If any of these local transactions aborts, then the entire set $\mathcal{L}$ must be rolled back as well as T_i^s, but other local transactions acting on behalf of, or emanating from a_i in other sites are not necessarily rolled back. Therefore the atomicity of activity a_i is compromised in favor of retaining atomicity within a given site s. Such inconsistencies are easily detected, and may either be tolerated, in cases where it is semantically allowable, or fixed by a compensating operation (as in [4]).

3. Level **GG** (Global to Global) — This "two-dimensional" mode atomically binds together several multi-site and local activities. It enforces global atomicity, i.e., local aborts imply full rollback of all updates at all sites, and therefore necessarily violates local autonomy. However, since it connects several multi-site activities which were explicitly specified and are known by all sites (by the definition of a global task), the autonomy is *voluntarily* compromised by the local sites.

Figure 1 summarizes the three atomicity levels and shows their different scopes when applied to the execution of a multi-site task. A multi-site activity, B_1, is initiated at site B, involving sites A and C. Upon completion, site B contacts the workflow engines for all sites, requesting that

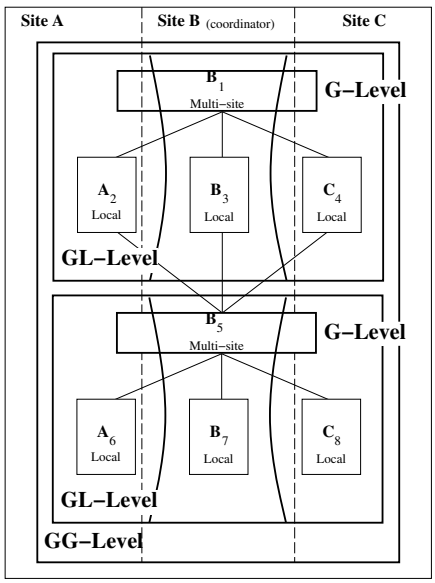

Figure 1: 3-level Atomicity Model

any generated local activities be executed, thus causing A_2, B_3, and C_4. Finally, once these all complete, a new round of multi-site activities is initiated at site B, causing the execution of local activities A_6, B_7, and C_8.

Figure 2 illustrates the various atomic units created by all eight combinations of modes, using the example from Figure 1. The failure of any (sub)transaction forces all transactions within the same atomic unit to abort. For simplicity, all sites in Figure 2 have the same atomicity mode; in practice, each site can select its own mode. Note how the entry for **G**-$\overline{\mathbf{GL}}$-**GG**, for example, protects the atomicity of the multi-site activities if any of the local emanating (sub)tasks fail.

3.1 Example

Consider a three-site DWFMS consisting of a travel agency (TA), a plane reservation system (P) and a car reservation system (C). Assume there exists a multi-site activity *book-plane-and-car* (*bpac*) which allows a travel agent in TA to produce a "travel package" consisting of plane tickets (booked from P) and a rental-car (reserved from C). *bpac* checks availability of tickets and cars at P and C, respectively, and checks the customer's credit (at TA); these actions are performed by local activities. At a later point, each site performs further local activities (e.g., site C may invoke a local *schedule-car-for-maintenance-check* activity). Further, assume that sites P and TA employ **G-GL** mode, and site C employs **GL**-only (For simplicity, we show only a single multi-site activity in this task, thus **GG** level is unnecessary). If site C aborts while issuing an emanating activity, then it rolls back the effects of *bpac* at C but not the effects that *bpac* has at TA and P, leaving each local site in a consistent state, but creating a global inconsistency with a plane reservation without a car reservation. This situation may be acceptable (i.e., the passenger may rent a car upon arrival), perhaps with a compensating action (invoked at TA only) that notifies the customer about the cancelation. In any case, no global abort is required (including the plane reservation), hence the **GL**-only mode. On the other hand, an abort in a local activity at P does lead to a global abort due to its **G-GL** mode, meaning that if the plane reservation is canceled, the car reservation

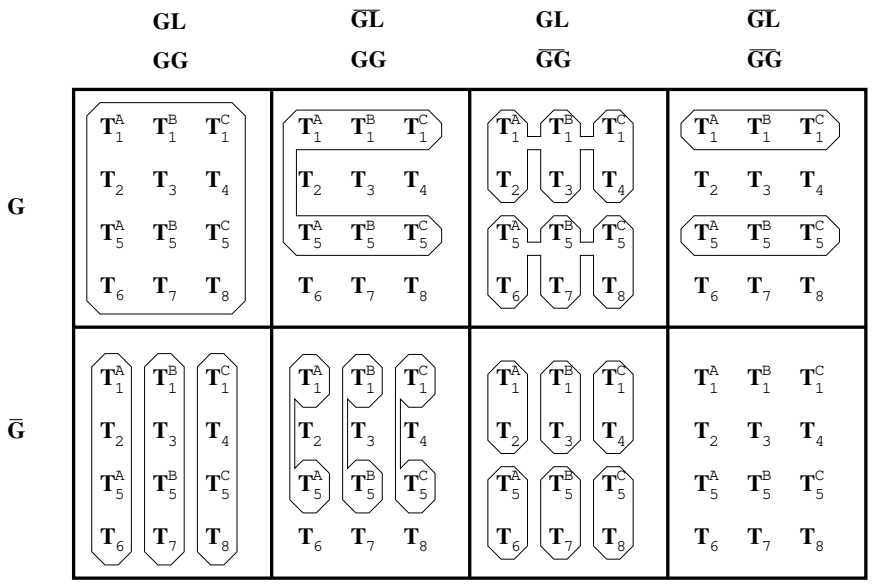

Figure 2: Atomicity Units Made Up of **G**, **GL**, and **GG**

and other related activities in TA must also be undone.

4 Conclusions

Using the flexible atomicity model presented in this paper, a workflow administrator can accurately define the scope of atomicity based upon the desired semantics of the tasks. In this way, DWFMSs can tailor and annotate workflow definitions to produce the desired behavior.

References

[1] Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling.* Kluwer Academic Publishers, Boston, MA, 1995.

[2] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading MA, 1987.

[3] W. Du, K. Elmagarmid, Y. Leu, and S. Ostermann. Effects of Local Autonomy on Global Concurrency Control in Heterogeneous Distributed Database Systems. In *Proc. of Second International Conference on Data and Knowledge Systems for manufacturing and Engineering*, Maryland, October 1989.

[4] Hector Garcia-Molina and Ken Salem. SAGAS. In U. Dayal and I. Traiger, editors, *ACM SIGMOD 1987 Annual Conference*, New York NY, May 1987. ACM Press. *SIGMOD Record*, 16(3):249-259.

[5] George T. Heineman and Gail E. Kaiser. An architecture for integrating concurrency control into environment frameworks. In *17th International Conference on Software Engineering*, pages 305–313, Seattle WA, April 1995. IEEE Computer Society Press.

[6] Nancy A. Lynch. Multilevel atomicity — a new correctness criterion for database concurrency control. *ACM Transactions on Database Systems*, 8(4):484–502, December 1983.

[7] Nandit Soparkar, Henry F. Korth, and Abraham Silberschatz. Failure-resilient transaction management in multidatabases. *Computer*, 24(12):28–36, December 1991.