WPI-CS-TR-96-2 September 1996

A 3-level Atomicity Model for Decentralized Workflow Management Systems

by

Israel Z. Ben-Shaul

George T. Heineman

Computer Science

Technical Report

Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department

100 Institute Road, Worcester, Massachusetts 01609-2280

A 3-level Atomicity Model for Decentralized Workflow Management Systems

Israel Z. Ben-Shaul
Technion-Israel Institute of Technology
Department of Electrical Engineering

George T. Heineman
Worcester Polytechnic Institute
Department of Computer Science

Abstract

A workflow management systems (WFMS) employs a workflow manager (WM) to execute and automate the various activities within a workflow. To protect the consistency of data, the WM encapsulates each activity with a transaction; a transaction manager (TM) then guarantees the atomicity of activities. Since workflows often group several activities together, the TM is responsible for guaranteeing the atomicity of these units. There are scalability issues, however, with centralized WFMSs. Decentralized WFMSs provide an architecture for multiple autonomous WFMSs to interoperate, thus accommodating multiple workflows and geographically-dispersed teams. When atomic units are composed of activities spread across multiple WFMSs, however, there is a conflict between global atomicity and local autonomy of each WFMS. This paper describes a decentralized 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. We describe an architecture that realizes our model and execution paradigm.

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

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

1 Introduction

Workflow management is a broad term for a technology that supports the reengineering of business and information processes [25]. Workflow Management Systems (WFMSs) provide the ability to define, evolve, and execute processes involving multiple human users, activities, and artifacts (e.g., documents). They do so by providing a formalism (e.g., Petri nets, task-graphs) in which processes are defined, and a workflow engine in which the process is executed, where forms of execution include: scheduling and automatically invoking activities according to the control and data flow of a process; reactively triggering activities based on state changes; monitoring the process; and enforcing process consistency constraints. WFMSs have gained popularity in recent years, as evidenced by the large number of research prototypes and products (e.g., InConcert [18],ActionWorkflow [23], ProcessWEAVER [22]).

Many WFMSs have realized the need to complement their process-centric approach with database technology to store diverse information accessed as a workflow progresses, as well as to represent artifacts which are being manipulated by the workflow tasks. In particular, the concept of a transaction as an *atomic* unit of execution has been investigated to preserve the consistency of workflow steps in case of concurrent access, exceptions, and failures. However, conventional transactions are often too restrictive for typical workflow applications, and WFMSs access and manipulate data in ways that cannot abide by such restrictions: (1) Workflow activities (i.e., the basic operations that are carried out as part of a workflow task, such as a financial analysis tool) may be highly interactive, entail much work and require hours or days of operation – rollback of such activities might be undesirable; (2) Activities 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 might need to cooperate with other activities on shared data, but conventional transactions execute in isolation from each other. As a result, various advanced transaction models have been proposed or adapted to workflow management (see for example [1, 39]).

Another dimension in which workflow technology has been rapidly evolving is scalability and decentralization. The ever-increasing globalization in computing and the wide distribution of documents, information, and workers have reinforced the importance of WFMSs to manage (inter-)organizational workflows, but at the same time have presented new challenges to workflow management. Decentralized WFMSs (DWFMSs) must provide mechanisms to enable *cooperation* among individuals, teams, or organizations while preserving the privacy and *autonomy* regarding access to the local workflows and their artifacts and activities. There have been various research efforts in large-scale workflow management (e.g., [2, 3, 11, 25, 34]), and a Workflow Coalition [17] has been established to promote, among other issues, standards for WFMS-interoperability [42].

A natural evolution in workflow technology is then to explore the combination of both directions, i.e, decentralized and transactional WFMSs. This combination (which has also been advocated in [25]) introduces a new challenge: how to reconcile the inherent conflict between the single-site (transactional) autonomy and the multi-site (transactional) atomicity. On one hand, a DWFMS should be able to support the bottom-up definition and atomic execution of cooperative multi-site workflow activities, but on the other hand it may have to respect the autonomy of each WFMS regarding access and manipulation of its private data. In some cases (e.g., when independent organizations collaborate) autonomy is a given constraint, as opposed to being merely a design choice, and therefore cannot be avoided or compromised. Similarly, some activities (e.g., groupware activities [11], or composite activities created on top of pre-existing activities) must access data from multiple sites simultaneously, thus requiring multi-site atomicity. This paper is focused entirely on atomicity of workflow tasks and autonomy of the individual WFMSs within a DWFMS. Issues of concurrency control of the transactions in a DWFMS are outside the scope of this paper; recovery issues, only as they pertain to atomicity, are still discussed.

Motivating Example

Consider the hypothetical DWFMS in Figure 1 composed of three WFMSs: a Banking system, a Plane-Reservation system, and a Car-Reservation system. Each WFMS has its own private, pre-existing workflow definition, users, data, and tools; the sites enter into

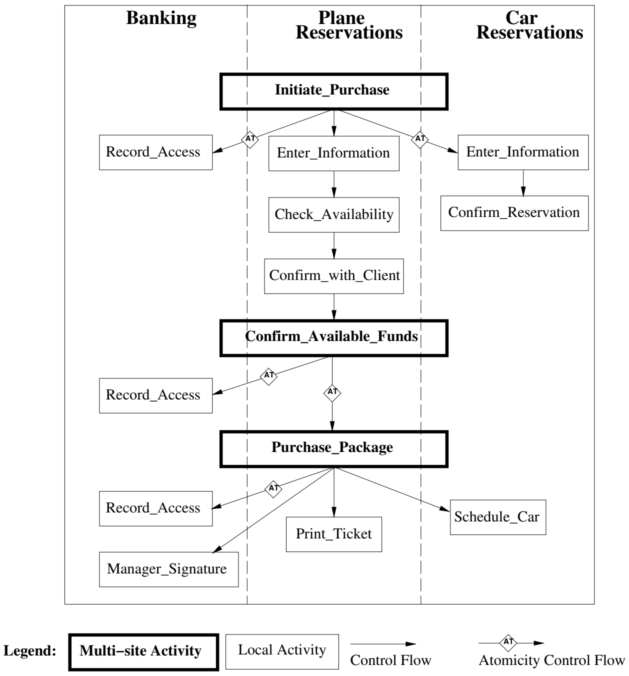

Figure 1: Motivating Example

agreements on a set of multi-site activities in which each site will participate. The sample multi-site workflow task allows travel agents to create travel plans and automatically deduct travel costs directly from a client's bank account. The edges in the figure show the control flow; an edge labeled "AT" between two activities requires that both activities be part of the same atomic unit. This multi-site task contains three multi-site activities:

- *Initiate_Purchase* (*ip*) — a travel agent wishes to book tickets for a round-trip flight from city_1 to city_2, with a rental car reserved in city_2.
- *Check_Available_Funds* (*caf*) — the client's bank account is verified to contain sufficient funds to purchase both the plane ticket and the car reservation.
- *Purchase_Package* (*pp*) — the price of the entire travel package is deducted from the client's bank account, and the package is shipped to the customer.

Each WFMS executes its own local activities as part of its workflow definition (for example, the Banking WFMS has an activity that records each access of a client's bank account), and occasionally the sites engage in a multi-site activity that accesses data from multiple sites. For example, the *caf* activity accesses both local data (the cost of the plane tickets) and remote data (the cost of the car reservation and the client's bank account balance).

Some multi-site activities may have dependencies with other local or multi-site activities. For example, the Banking site may insist that *record_access* succeed or else *caf* should be invalidated, at least from its own perspective. At the same time, it may be desirable to limit the impact that a local activity (possibly unknown in other sites) can have on the global effect of a multi-site task. For example, even if the *record_access* local activity at Banking has failed and *ip* decides to terminate, the information already entered at the Car-Reservation site may be left intact, and even partial effects of *ip* itself (e.g., updates of local data) may be valid, depending on the semantics of the activity and on whether the inter-site consistency overrules intra-site autonomy.

Thus, the main issue to resolve is how the individual WFMSs can define their transactional-interoperability to control the implications from failure of arbitrary activities in a multi-site task. If the DWFMS enforces atomicity of the multi-site task, then the failure of any of these activities will roll back all work at all sites. If no atomicity is enforced, then the failure of an activity may leave the multi-site task in an inconsistent state. This paper describes a flexible and decentralized atomicity model that enables collaboration among geographically-dispersed and autonomous workflows. The model allows each WFMS in the DWFMS to tailor the "scope" and "consistency" of atomicity to fit the semantics of multi-site tasks and to address the autonomous demands of each participating site on a per-task basis. This paper is organized as follows: Section 2 surveys related work and contrasts our approach with similar investigations. Section 3 defines basic terms and describes the execution model and system architecture assumed by the 3-level atomicity model, presented in Section 4. Section 5 discusses issues concerned with the realization of the model in an existing DWFMS from both the language and system perspective. Section 6 summarizes the contributions of this paper and points to future directions.

2 Related Work

In this paper we focus our attention on atomicity and autonomy issues that arise in decentralized WFMSs. DWFMSs are a possible solution to the problem of applying workflow technology on a large scale. Alonso and Schek [3] cite Scalability, Correctness, and Interoperability as three (of five) of the most important limitations of existing workflow technology. We now summarize and compare existing approaches to autonomy and atomicity with our proposed 3-level atomicity model.

There is a clear consensus that existing database technology is not suitable for WFMSs for many reasons [3, 19, 37]; for example, Serializability as a correctness model is considered too restrictive [5]. Many extended transaction models (ETMs) have been developed [24, 32, 33] to define advanced transaction behavior as needed by WFMSs. Most ETMs retain *atomicity* — perhaps the primary objective of transactions — the property that all actions in a transaction occur or none happen. Since transactions have structure, most ETMs extend atomicity to be able to include multiple transactions and subtransactions, such as in nested transactions [36]. This is accomplished typically by commit and abort dependencies [16], the building blocks of atomicity. ACTA [16] and the Transaction Specification and Management Environment (TSME) [26] allow for many more types of dependencies to be created (i.e., *backward-commit-begin* means that a transaction cannot begin before another transaction commits), but we view these as the responsibility of the workflow engine. The transaction managers in a DWFMS should be concerned with managing atomic units, and cooperation is needed when these units spread across multiple sites. ETMs themselves do not necessarily solve the many problems of workflows, and some now view workflow as a superset of transaction models [1].

The architecture we propose for DWFMSs is most similar to a hybrid multidatabase system [40], a solution from the database community for managing multiple, heterogeneous database repositories. The four aspects to autonomy discussed in [40] also apply to DWFMSs. In the same way that local autonomy has implications on global concurrency control [20], autonomy of each WFMS implies that the interoperating WFMSs must be willing to make compromises on local autonomy and global correctness. The Workflow Coalition standard for interoperability [42] does not yet address this issue. An underlying theme in the field of heterogeneous processing is to use data and control abstractions to cope with system heterogeneity and interoperability, by hiding everything that is not pertinent to interoperability, minimize the "exposure" to global control, and determine the desired exposure at each site

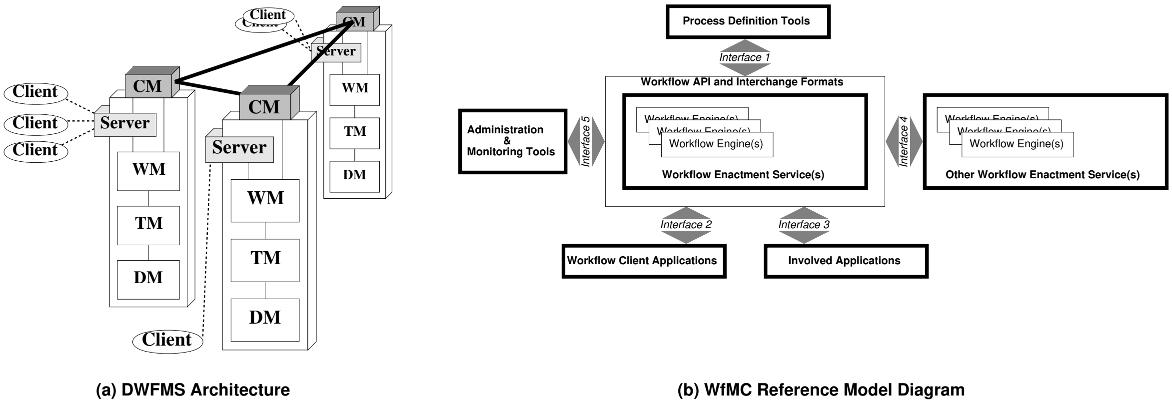

Figure 2: Comparative Architectures

autonomously. The atomicity model outlined in this paper provides these features.

3 A Decentralized Architecture and Execution Model

Our generic DWFMS architecture, depicted in Figure 2a, is compatible with the Workflow Coalition's reference architecture in Figure 2b [17]. A DWFMS consists of a set of local WFMSs that share no resources and communicate via message passing. Each individual WFMS (or workflow engine, in [17]) consists of a local data manager (DM), local transaction manager (TM), and local workflow manager (WM). The WM enacts (e.g., interprets or executes) workflow specifications that are defined using a workflow formalism loaded into each WM (Interface 1 in [17]). Each WFMS supports multiple users and applications through a client/server paradigm (Interface 2 in the reference model). The communication manager (CM) provides the necessary infrastructure support for interconnectivity of the sites (Interface 4), such as the ability for a site to publish information about its DM so that a remote site can access this data (through the TM).

Each WM accesses local data through its local TM and a *local transaction* is created to manage the data. WM can access remote data using services from CM to contact the TM at the remote site. If the access request is granted, the remote TM creates a transaction, and a copy of the requested data item is transferred to, and cached by, the requesting WM. This is in contrast to distributed database systems [13] that employ a global transaction manager that processes any request to access remote data.

Each site maintains its own private workflow and we assume operational autonomy with respect to access to data (more specifically execution and control autonomy as defined in [41]), but we assume throughout the paper design homogeneity, i.e., similar WM, TM, and DM components. Each WFMS is managed by a workflow administrator who defines the particular workflow that the WFMS executes on its data, involving its own users and tools. The administrator of each local WFMS is responsible for creating the desired interconnections at the workflow level with other loosely-coupled DWFMSs.

All operations that are carried out within the WFMS are conducted in the context of workflow *tasks*. A task is a partially-ordered set of workflow *activities*, which are logical groupings of operations. An activity is an important notion that distinguishes workflows from conventional transactional applications. Each activity maps to a single step in the workflow, perhaps involving a user invoking some external tool, such as a spreadsheet application. Workflow formalisms typically guard activities with local constraints, as opposed to the global control flow imposed by the task. For example, FUNSOFT nets [27] (extended Petri nets) allow logical predicates to be attached to transitions (the equivalent of activities in our terminology), task graphs [35] provide predecessor and successor edges, and rules [30] provide pre- and post-conditions. An activity, a_j, *emanates* from another activity, a_i, when the results of executing a_i satisfies the logical guard for and triggers a_j. The notion of emanating activities is also an important characteristic of workflows.

A decentralized workflow is constructed from local workflow tasks and activities spread throughout sites. A *multi-site activity* accesses data from multiple sites and may involve zero, one, or multiple users from the same or from different sites; the activity is necessarily defined in all participating workflow processes. Multi-site activities are the main interoperability building blocks in our multi-site execution model. A *multi-site task* contains at least one multi-site activity, with possibly several local (i.e., involving data only from a single site) activities defined at multiple sites.

A multi-site activity always executes at exactly one coordinating site, that is also the coordinator of the corresponding multi-site task within which the activity operates. The coordinating site first retrieves the data needed for the multi-site activity from the remote sites, then the WM at the coordinating site executes the activity and returns any modified data to the remote sites from which they originated. Before invoking a multi-site activity or upon its completion, local (i.e., single-site) emanating activities may be invoked. Thus, the execution of a multi-site task may be viewed as alternating between local and global modes. Global mode involves synchronous execution of the shared multi-site activities at the coordinating site, involving data from multiple sites and possibly multiple users. Local mode involves execution of local (sub)tasks emanating from the global task at multiple sites. These local tasks execute asynchronously and in parallel, only on local data, solely according to the local workflow definition. Upon completion of a multi-site activity, the task *fans-out* to the local sites to carry out local activities that emanate from the multi-site activity. If another multi-site activity is scheduled for execution, it is enabled when all local activities complete and *fan-in* to the coordinating site. 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 requests each site to carry out its local activities (in local mode). This model demands a high degree of freedom in balancing atomicity and autonomy, because as we show in [9], one may need to limit the impact that (unknown) activities may have on the (local and multi-site) work performed at other sites. Each site can simultaneously be a coordinating site for several multi-site tasks and can be participating in several other multi-site tasks. A peer-based mechanism that actually establishes and maintains agreements over the execution of multi-site tasks among otherwise independent WFMSs is described in detail in [9] and is beyond the scope of this paper.

Figure 3 illustrates execution of multi-site workflow tasks from our motivating example. Site S_P has established an agreement with sites S_B and S_C over activities ip_1, caf_8, and pp_{10}. Note how each site integrates these activities into their own workflow definitions. A multi-site activity, ip_1, is initiated at S_P, involving S_B and S_C. Upon completion, S_P contacts

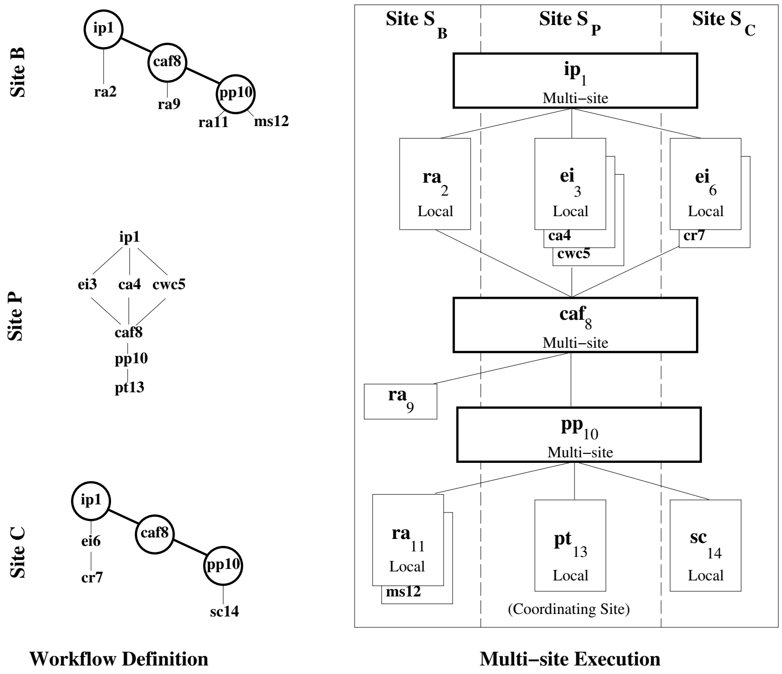

Figure 3: Execution of multi-site workflow tasks

the participating WFMSs, requesting that any emanating local activities be executed, thus activating $ra_2, ei_3, ca_4, cwc_5, ei_6$, and cr_7. These are all executed by the respective WMs; for example, the WM at S_P executes ei_6 first before executing cr_7. Once these all complete, a new round of multi-site activities is initiated at site S_P, causing the execution of caf_8; this execution sequence operates until all activities in the task have been completed.

3.1 Execution Model from TM Perspective

The WM initiates transactions in TM to guarantee the atomicity of activities (whether local or multi-site). If an activity, a_i, only accesses data from its local DM, then a single local transaction, T_i, is initiated by WM to encapsulate the data requests for the activity. In contrast, 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 the TM for each site. Each transaction T_i^j is local to S_j and only accesses local data from that site. This division of a multi-site activity into local transactions retains the autonomy of the individual TMs in the DWFMS.

The TMs are not involved in the details of workflow management and are only concerned with the *dependencies* between transactions (see [15] for a supporting view). To implement the atomicity model presented in this paper, the TMs only need to support two types of dependencies: *commit* and *abort*. If T_i has an abort dependency on T_j, ($T_i \; \mathcal{AD} \; T_j$), then if T_j aborts, T_i must also abort. If T_i has a commit dependency on T_j, ($T_i \; \mathcal{CD} \; T_j$), then T_i cannot commit until T_j finishes (either commits or aborts) [16].

Each TM expects full autonomy over its transactions, therefore the set of transactions for a multi-site activity poses a problem. If a multi-site activity expects to execute as an atomic unit, the commitment of the individual transactions must be synchronized, most likely by using a simple two-phase commit (2PC) protocol [13]. Each TM must then distinguish transactions for local activities from transactions that are part of a multi-site activity. The coordinating site that executes the multi-site activity coordinates the 2PC protocol.

The atomicity of multi-site activities is protected by placing commit and abort dependencies between the coordinating transaction and participating transactions. These cross-site dependencies are allowed if the individual TMs allow them; thus the autonomy of each TM can only be weakened by mutual agreement. The explicit agreements at the WM level are supported by corresponding agreements between the TMs to honor and maintain cross-site dependencies. For each multi-site activity, a_i, the following cross-site dependencies are created between the coordinating site and each participating site, S_j, for $1 < j \leq n$:

$$(T_i^1 \, \mathcal{AD} \, T_i^j) \quad (T_i^j \, \mathcal{AD} \, T_i^1) \quad (T_i^1 \, \mathcal{CD} \, T_i^j)$$

4 Atomicity Model

When decomposing a workflow task into activities, often a sequence of activities must be grouped together atomically to protect sensitive data or to guarantee a consistent transition of the data in DM. We assume that each workflow modeling language has some means of specifying these *atomic units*. For example, one can annotate transitions in Petri nets, edges in Task graphs, and rule chains for rule-based systems. Since workflows can be decentralized across multiple sites, the main question is how to associate proper semantics with these annotations that, like the execution model itself, will be decentralized; there can be no global overseer transaction manager.

One solution is to regard an entire multi-site task as a global atomic transaction. While it may be desired in some cases, it is unacceptable as the sole alternative because it violates autonomy and ignores the problems of typical workflow activities (e.g., long duration). Autonomy is violated because the failure of any activity in the multi-site task will force all activities in the multi-site task at all sites to fail.

Another alternative would be to enclose atomically only single activities (both local and multi-site) by transactions. Such an approach is acceptable from the perspective of autonomy since all sites must agree on shared multi-site activities. However, this alternative does not address the real need to atomically enclose several activities within a workflow task. We can go a step further and attempt to group atomically several multi-site activities along with the intermediate local activities encountered. Once again, while possibly appropriate for some tasks, it may be prohibitively global. In particular, there may be cases where a local site prefers to retain its own consistency even if this violates global consistency.

Thus, a proper general solution should provide means to specify fine-grained atomic units within a multi-site task, and allow these units to be of arbitrary shape and cross sites as needed. Specifically, our approach provides the following:

- 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, the logical units of atomicity.
- If global atomicity is required for correctness, local autonomy is compromised in favor of atomic commitment; local and remote TMs may be affected by the global task.
- 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.

4.1 Three Atomicity Levels and their Combinations

We identify three core levels of atomicity, each of which can be explicitly and separately specified on a per-task basis. The atomicity within a multi-site task is characterized by whether each of the three levels is turned-on or turned-off, thereby enabling eight different atomicity configurations, each with different scope and semantics of atomicity.

1. **G** Atomicity (Global) — This provides atomicity for a single multi-site activity. It requires an atomic-commitment protocol such as two phase commit [14], since each multi-site activity has a subtransaction acting on its behalf at the coordinating site and each participating site. If any of these subtransactions abort, then to preserve atomicity, all subtransactions for the multi-site activity must abort.

2. **L** Atomicity (Local) — This is orthogonal to **G**, i.e., it preserves local autonomy and atomicity, possibly by compromising global atomicity. At site S_j, **L** binds into an atomic unit the local transaction T_i^j, acting on behalf of 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_j from a_i. Since each local activity is encapsulated by exactly one local transaction, **L** only creates dependencies between transactions within the same site.

   Within a given site, S_j, the local transactions, $\mathcal{L}$, 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_i have completed, T_i^j and the local transactions $\mathcal{L}$, can commit.

   If any of these local transactions fails, then the entire set $\mathcal{L}$ fails $\underline{\text{as well as } T_i^j}$, but other local transactions acting on behalf of, or emanating from a_i at other sites are not necessarily affected; it depends on whether or not **G** atomicity was defined. Therefore, the atomicity of a_i might be compromised in favor of retaining atomicity within a given

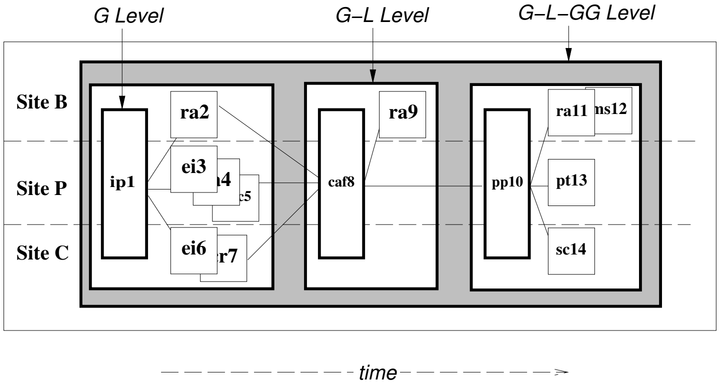

Figure 4: Basic 3 Atomicity Levels

site S_j. Such inconsistencies are easily detected, and may be tolerated, in cases where they make sense semantically, or fixed by a compensating operation (as in Sagas [24]).

3. **GG** Atomicity (Global to Global) — This level provides atomicity across several multi-site activities. When coupled with **G** and **L**, it enforces global atomicity, where any failure in a multi-site task forces it to fail at all sites, and therefore necessarily violates local autonomy. However, since it connects several multi-site activities that 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 4 shows three common and nested atomic units, **G**, **G**-**L**, and **G**-**L**-**GG**, and their different scopes when applied to the execution of the multi-site task shown earlier in Figure 3. The failure of any (sub)transaction forces all transactions within the same atomic unit to fail. The full power of the model is obtained by combining **G**, **L** and **GG** in different ways to create atomic units with different semantics.

Figure 5 illustrates the eight possible atomic units created by the combinations of levels, using a simplified version of the example from Figure 4. **G**-$\overline{\mathbf{L}}$-**GG** binds atomically multi-site activities but has no dependencies with the local emanated subtasks. Thus, unlike **G**-**L**-**GG**, failure at a locally emanating activity does not lead to an abort of the global task and vice-versa. $\overline{\mathbf{G}}$-**L**-$\overline{\mathbf{GG}}$ is the **L**-only local configuration explained earlier, which sacrifices multi-site atomicity in favor of local atomicity. $\overline{\mathbf{G}}$-**L**-**GG** can be viewed as a "long **L**", i.e., site atomicity

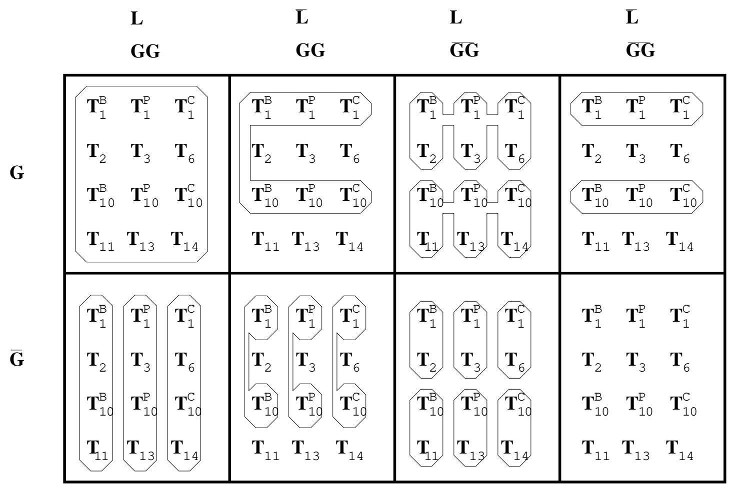

Figure 5: Full Range of Atomic Units

that spans several multi-site activities which by themselves are not cross-site atomic. It is useful, for example, when the multi-site tasks are used to only synchronize or regulate the progress of a set of otherwise independent local tasks with minimal or no inter-site data dependencies. $\overline{\mathbf{G}}$-$\overline{\mathbf{L}}$-**GG** may appear peculiar since it only binds local subtransactions that are part of multi-site (hence "global") activities. However, it can be used in cases where, like in $\overline{\mathbf{G}}$-**L**-**GG**, multi-site activities synchronize mostly independent local processes, but unlike $\overline{\mathbf{G}}$-**L**-**GG**, the consistency-demanding work is done in the multi-site activities, not in the emanating activities. Finally, $\overline{\mathbf{G}}$-$\overline{\mathbf{L}}$-$\overline{\mathbf{GG}}$ is the simplest and degenerate case where each subtransaction is a separate atomic unit.

An interesting aspect to consider in the model is whether all sites participating in the execution of a multi-site task must employ identical atomicity levels. To promote autonomy, the answer should be no. While all sites must agree on the **G**-atomicity since it requires by definition cross-site atomicity, the sites may not necessarily employ same **L**- or **GG**-atomicity. Clearly, there is no reason why **L**, the level that promotes site atomicity, would have to be defined identically in all sites. Indeed, there may be cases where one site must employ **L**-atomicity to preserve some workflow semantics, while another site must *not* employ **L**-atomicity within the same task (as will be seen shortly). It may appear that **GG** must be the identical by all sites since it spans multi-site (hence "global") activities. However, it is important to distinguish the property that a set of multi-site activities are semantically bundled in a task, from the orthogonal (single-site or multi-site) atomicity property over those activities. In particular, since **GG** is concerned with individual site atomicity across the global activities, each site can select to enforce it on a per-task basis. To summarize, the strength of our model is its ability to define fine-grained cross-site atomicity to match the semantics of the interoperating workflows, without consulting a global or brokering entity.

4.2 Motivating Example Revisited

Returning to our motivating example in Figure 1, we now consider various execution scenarios in the face of transaction failure. Within this multi-site task, all sites have **G** and **GG**. For multi-site activity *ip* and *pp*, sites S_P and S_C have $\overline{\mathbf{L}}$. For all three multi-site activities, site S_B has **L**. The first failure can occur when the multi-site activity *ip* is requested. For example, some requested data at site S_P might be locked by a user updating a passenger list and as a result the transaction T_{ip}^{P} is aborted (in this case, abort may be better then blocking since if T_{ip}^{P} were blocked instead, potentially unbounded delays might occur, forcing the travel agent to wait indefinitely). Since both site S_B and S_C have **G** atomicity, T_{ip}^{B} and T_{ip}^{C} are also aborted.

Next, assume that multi-site activity *ip* successfully completes and the individual workflows at each site are executing their local activities which emanate from *ip*. If the transaction for the *Enter_Information* activity for S_C fails and aborts (e.g., because of a semantic or a system error) the multi-site activity *ip* is unaffected since S_C does not have **L** mode for this multi-site task. At a later point, when *caf* is executed, the DWFMS will record that a plane reservation was made without a corresponding car reservation. Contrast this behavior with a failure within the *Record_Access* activity. Since the workflow at S_B specifies that *Record_Access* and *ip* must operate within the same atomic unit, both transactions at S_B must be rolled back. This behavior might be required, for example, if every access request must be logged for future inspection. Since the activity for *ip* is executing in **G**-mode, this means that the corresponding transactions, T_{ip}^{P} and T_{ip}^{C} at S_P and S_C, must also be rolled back. This shows the situation where the failure of a private workflow activity forces the abort of transactions at remote sites.

If *ip* and all of the emanating activities at all sites succeed, site S_P executes the *caf* multi-site activity. If *caf* fails within site S_C, there is no special atomicity requirement between *ip* and *caf*, so only the transactions T^B_{caf} and T^P_{caf} need to be aborted; there is no other effect. Contrast this with a failure in *pp* at site S_B where *caf* and *pp* are directed to complete as an atomic unit. This might be required, for example, to prevent arbitrary updates to occur in the interim between checking the account and debiting the funds from the account. In this case, T^P_{pp} and T^C_{pp} would need to be rolled back, as well as T^B_{caf}, T^P_{caf}, and T^C_{caf}. Since S_B has **L** activated, the second *Record_Access* activity would also need to be rolled back since it requires execution with *caf* in an atomic unit. In the next section we show how the DWFMS can exploit the full potential of this model by specifying the atomic units within each workflow through the use of annotations.

5 Realization of the Model

A workflow modeling language that intends to support some notion of atomicity must be capable of designating parts of a workflow task as atomic. In a single-site workflow with a modeling language that supports explicit control flow, this support can be easily provided by a pair of `begin-atomic` and `end-atomic` directives. In declarative languages with implicit and dynamically-determined control flow, as in rule-based languages, adding such support is less straightforward; specifying atomic units across sites (and workflows) seems even harder. However, declarative modeling paradigms that support implicit communication seem to be well-suited for interoperability (as shown in [38, 21]). We present here the realization of the 3-level atomicity model in the Oz DWFMS [9], focusing more on the language modeling issues than on the underlying implementation of the transaction manager; for a discussion of the latter, see [29, 28].

5.1 Oz **Overview**

Oz is a multi-site DWFMS that supports definition and execution of multiple autonomous workflows. Although targeted originally to support software engineering processes, it has

Report [?care_element:CARE_ELEMENT, ?report:LITERAL, ?vital_signs:LITERAL]:	*Signature*
(and (bind MLM ?mlm suchthat (member [?care_element.post_mlm ?mlm])) (bind HEALTHWORKER ?worker suchthat (linkto [?care_element.care_giver ?worker])) (bind PATIENT ?patient suchthat (?patient.MRN = ?care_element.MRN))):	*Bindings*
(forall ?care_element (?care_element.status <> Reported) (?care_element.status = CheckedOut) **atomicity**)	*Condition*
{ CarePlan_tool report_element ?care_element.report ?report ?mlm.input ?vital_signs return ?report_status ?mlm_output }	*Activity*
(and (?care_element.update_time = CurrentTime) **atomicity** (?mlm.output = ?mlm_output) (link [?patient.track_record ?care_element]) (link [?worker.track_record ?care_element]));	*Effects*

Figure 6: Sample Oz Rule

recently shifted its focus to support general business processes (e.g., see [31] for a healthcare management application). Oz follows the generic architecture shown earlier in Section 3. Each site employs a client-server architecture with multiple clients communicating to a single WM [12]. Across sites, Oz employs a multi-server "share-nothing" architecture, meaning that the workflows, schemas, and instantiated databases are kept separately and disjointly by each site, with no global repository or "shared memory" of any sort [8, 7, 10].

Human interaction with the DWFMS is provided through a *client* that is connected primarily to its local WM. Using the client's connection to its local WM, users can invoke local activities on local data items, under the local process. Oz users can also open connections to remote sites. A remote client can browse through remote databases and get information about remote data (subject to access control permissions). However, a client has no access to remote workflows and manipulation of remote data can only occur within a multi-site activity.

Prior to invoking a multi-site activity, the participating sites follow an agreement-based protocol, called a *Treaty*, which determines various execution and access privileges, establishes a verification scheme to ensure a trustworthy interaction, and effectively turns the activity into a shared multi-site activity by replicating it in the participating sites and integrating it within each local process.

An executing multi-site activity is termed a *Summit* activity. When a Summit activity is attempted, the coordinating WM verifies that it is part of a valid Treaty, retrieves copies of the remote data, and sends the the activity for execution by the client. Upon completion, the coordinating WM directs all participating sites to inspect their local workflows to execute local activities that emanate from the Summit activity. When all sites complete, control returns to the coordinating task, that in turn may initiate a subsequent Summit activity, if any, until there are no more Summit activities, at which time the task is completed.

5.2 Single-site Rules, Atomicity and Annotations

Each single-site activity within an Oz workflow is enclosed in a *rule*. A rule consists mainly[1] of a signature (i.e., a name and typed formal parameters); a (pre)-condition which is a (composite) predicate over the arguments; the actual activity which interfaces to external tools and data; and a set of mutually-exclusive effects. A rule can be invoked either directly by a user, or indirectly, as a result of rule *chaining*, which is the mechanism to invoke emanating activities. When a rule is invoked, its condition is evaluated. If the evaluation fails (i.e., the predicate evaluates to false), WM attempts to automatically satisfy the rule by backward chaining to other rules whose effect may satisfy the failed condition. If the condition is (or has become) satisfied, the activity of the rule is invoked. Upon completion, the activity returns a status code that determines which effect of the rule to assert. The WM then attempts to forward chain to rules whose condition have become satisfied as a result of the assertion. Both backward and forward chaining are recursive. Thus, workflow steps are implicitly interrelated by logical matchings between effects and conditions of rules.

A single-site rule is the smallest atomic unit. This does not mean that the rule actually executes atomically, only that the outcome of its execution is all-or-nothing[2]. A chain of single-site rules can be bundled atomically. Such a chain is called *atomicity* chain and is due to Barghouti [6]. By default, rule chains are not atomic. Non-atomic chains are called

[1] Oz rules actually have various other optional clauses such as data queries and user delegation clauses, but they are irrelevant to this discussion and hence ignored.

[2] In fact, since activities typically execute at the clients the WM usually switches context to serve other requests while the client executes the activity, thus rules seldom really execute atomically.

automation chains simply to distinguish them from the special atomicity chains.

Atomicity chains are denoted by annotations made to predicates of rules. A rule chain exists from rule r_i to rule r_j if a predicate, p_i, asserted by r_i satisfies a predicate, p_j, in the condition of r_j (i.e., r_j emanates from r_i). If both p_i and p_j are marked "atomicity", then an atomicity chain exists which means that if r_j fails the corresponding transactions for both r_j and r_i are rolled back. If p_j is marked "automation" and p_i is marked either "atomicity" or "automation", then an automation chain exists. If automation chaining fails, only the updates of the failed rule are rolled back, without affecting the outcome of rules that were previously executed in that chain. The sample `Report` Oz rule in Figure 6 allows atomicity chaining from its assertion (line 9).

A rule chain may be composed of both atomicity and automation chains. This creates a problem regarding the ordering of the rule chain execution since an abort of an atomicity chain might rollback unnecessarily effects of rules which executed as automation. This might be particularly harmful if these were long-duration activities, in which case their treatment as non-atomic was a deliberate choice to avoid possible rollback. The solution employed in Oz is to execute all atomicity chains first, and any automation rules which are encountered during the atomicity chaining are queued (first-in first-out). Once all immediate atomicity chains complete and commit their work, automation chaining restarts with the queued rules. Since automation chaining can lead to further atomicity chains, this procedure can be recursive. One way to view this form of execution is as a chain of automation rules, with occasional "bursts" of atomicity chaining.

5.3 Multi-site Rules, Atomicity and Annotations

Syntactically, there is no difference between a single-site and a multi-site rule. The difference is that at run-time, a multi-site (Summit) rule is invoked with data from multiple sites, and it must have been agreed upon by a Treaty (hence internally in the WM it is marked properly along with the necessary information such as validation timestamp [9]). The mapping of the three basic atomicity units (as shown in Figure 4) to rules is as follows: the **G** level corresponds to the multi-site activity of a rule; **G**-**L** level corresponds to a multi-site rule, along with its emanating backward and forward *local* chains; finally, the **G**-**L**-**GG** level corresponds to a *global* chain from one multi-site Treaty rule to another, including all emanating local chains. The annotations of the basic levels are as follows.

1. **G** atomicity is the default for multi-site activities, and thus no annotation is required to specify it. To represent $\overline{\mathbf{G}}$, we allow an activity to be marked as non-atomic, meaning that the intra-activity atomicity need not be enforced in case of transaction failure. By agreeing on the semantics of the activity as defined in the Treaty, each site also agrees on the transactional semantics of the activity.
2. **L** atomicity arises when an atomicity effect of a Summit rule matches with the atomicity condition of a local rule, or when two local emanating rules are atomically bundled. No new annotations (beyond the single-site atomicity annotations) are required because the multi-site transactional semantics are implied by the context in which the rules are executed. Similarly, $\overline{\mathbf{L}}$ is represented by automation annotations among the rules. Unlike the **G** case, each site is free to choose the annotations for its local rules. Therefore, the simultaneous execution of local implications of a particular multi-site Treaty rule may be handled as **L** in one site and as $\overline{\mathbf{L}}$ in another site.
3. **GG** atomicity ($\overline{\mathbf{GG}}$) is defined in the same manner as **L** ($\overline{\mathbf{L}}$), except the matching is performed only among multi-site — and hence shared — Treaty rules; all sites employ the same **GG** mode for a given set of multi-site rules.

Thus, the only additional annotation beyond the single-site annotations is for $\overline{\mathbf{G}}$ mode. All other annotations derive the proper semantics from the semantics of the multi-site Summit execution. Once the three basic levels (and their complements) are specifiable, supporting their combination does not incur additional syntax. For example, providing $\overline{\mathbf{G}}$-**L**-$\overline{\mathbf{GG}}$ atomicity in a participating site S_A for a Summit rule R_1 that executes at S_B involves: annotating R_1's activity as automation; annotating R_1's effect(s) and the matching condition of local rules in S_A (if any) as atomicity predicates; and annotating matching conditions of other multi-site rules (if any) in S_B as automation.

One last issue to resolve concerns the order of execution in the multi-site context. In addition to the atomicity vs. automation ordering problem discussed earlier, we are faced with an orthogonal ordering concern — global vs. local execution — and with proper combination of both orthogonal ordering constraints. The solution employed in Oz is to have an atomicity phase followed by an automation phase, where each phase alternates between global and local modes as outlined in Section 3. When a Summit rule completes, all local atomicity rules first execute in the local sites, queuing (locally) any local automation rules encountered. When all sites complete their local atomicity, the next Summit atomicity rule (if any) is executed, followed by all local atomicity, and so forth. When the Summit atomicity phase completes, a global commit occurs. The next step is to execute all automation chains. Again, Summit automation rules fire first, followed by local automation chains, followed by the next Summit automation interval, and so forth. This design reduces the amount of work that would be undone in case of failure.

5.4 Cross-Site Dependencies

Given a TM that supports commit and abort dependencies, our model requires a mechanism for the cross-site dependencies that form as the Summit executes. Our *mediator-based* approach [29] shows how a TM can be extended to augment its behavior (in this case, implement 2PC on a centralized TM), similar to the transaction adapters proposed in [4]. In short, mediator code can be inserted between the WM and TM to insulate WM from the details of 2PC. The **G** level corresponds to the 2PC that synchronizes the transactions for a multi-site activity; **L** corresponds to the nested transaction hierarchy within a single site; **GG** provides the glue between multiple nested transaction hierarchies at each site.

6 Conclusions

Multi-site atomicity is often required to preserve the integrity of collaborative multi-site activities. However, global atomicity and local autonomy are conflicting goals in decentralized and transactional systems, and therefore one has to be compromised often in favor of the other. Workflow management systems require flexible transaction management, thus, we investigated approaches to modeling and managing multi-site workflow transactions that allows a multi-site task to tailor the scope of atomic units of its participant activities. Moreover, such an approach should be decentralized, with no globally enforced policy.

We believe that our 3-level atomicity model and the corresponding realization addresses these issues. In particular:

- The *scope* of the desired multi-site atomicity can be defined in a fine-grained manner. Starting from the three basic levels — a single multi-site activity (**G**), a local intra-site atomicity (**L**), and multiple multi-site activities (**GG**) — and by allowing any intermediate mode formed by combining these levels, workflow administrators have full freedom in selecting the level that best matches the decentralized workflow task.
- *site-autonomy* is maximized, both in specifying and in executing the workflow tasks. The atomicity of multi-site activities and their inter-relationships can be declared only by explicit agreements that must be formed between the involved sites (using Treaties). Moreover, multi-site atomicity specifications are the *only* ones that need to be specified globally; local implications and their relationships to global activities (represented by the **L** level) are private and unknown by remote sites.

The atomicity model presented here assumes a homogeneous workflow formalism, mostly to clearly explain how it was realized in the Oz DWFMS. A Treaty between workflows using different formalisms should still be possible because the logical predicates in conditions and assertions should be common to all workflow formalisms. As described, the Summit execution protocol enforces a particular execution sequence that must be enforced among all WFMS in the DWFMS. To be truly heterogeneous there need to be ways to allow the control flow to be decentralized among sites in the same way that the workflow is. The 3-level atomicity model we have developed should be immediately applicable to any DWFMS struggling to find ways to support failure atomicity.

Acknowledgements

The authors would like to thank Gail Kaiser for her leadership of the Oz system. We also thank Peter Skopp for his numerous contributions, including cache management and deadlock avoidance mechanisms. Finally, we would like thank the anonymous reviewers for their comments and an insightful suggestion that sites may choose different **GG** atomicity within the same task.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, and C. Mohan. Advanced transaction models in workflow contexts. In *12th International Conference on Data Engineering*, pages 574–581, New Orleans, USA, February 1996.

[2] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Gunthor, and Mohan U. Kamath. Exotica/fmqm: A persistent message-based architecture for distributed workflow management. In *IFIP WG 8.1 Workgroup Conference on Information Systems Development for Decentralized Organizations*, Trondheim, Norway, August 1995.

[3] G. Alonso and H. Schek. Research issues in large workflow management systems. In *NSF Workshop on Workflow and Automation in Information Systems*, pages 126–131, May 1996.

[4] Roger Barga and Calton Pu. A practical and modular method to implement extended transaction models. In *21st International Conference on Very Large Data Bases*, Zurich, Switzerland, 1995.

[5] Naser S. Barghouti and Gail E. Kaiser. Concurrency Control in Advanced Database Applications. *ACM Computing Surveys*, 23(3):269–317, September 1991.

[6] Naser S. Barghouti and Gail E. Kaiser. Scaling up rule-based development environments. *International Journal on Software Engineering & Knowledge Engineering*, 2(1):59–78, March 1992.

[7] Israel Z. Ben-Shaul and Gail E. Kaiser. A configuration process for a distributed software development environment. In *2nd International Workshop on Configurable Distributed Systems*, pages 123–134, Pittsburgh PA, March 1994. IEEE Computer Society Press.

[8] Israel Z. Ben-Shaul and Gail E. Kaiser. A paradigm for decentralized process modeling and its realization in the oz environment. In *16th International Conference on Software Engineering*, pages 179–188, Sorrento, Italy, May 1994. IEEE Computer Society Press.

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

[10] Israel Z. Ben-Shaul and Gail E. Kaiser. An architecture for federation of process-centered environments. Technical Report CUCS-015-96, Columbia University Department of Computer Science, May 1996.

[11] Israel Z. Ben-Shaul and Gail E. Kaiser. Integrating groupware activities into workflow management systems. In *7th Israeli Conference on Computer Based Systems and Software Engineering*, pages 140–149, Tel Aviv, Israel, June 1996.

[12] Israel Z. Ben-Shaul, Gail E. Kaiser, and George T. Heineman. An architecture for multi-user software development environments. *Computing Systems, The Journal of the USENIX Association*, 6(2):65–103, Spring 1993.

[13] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading MA, 1987.

[14] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading MA, 1987.

[15] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, and G. Weikum. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. *SIGMOD Record*, 22(3):23–30, September 1993.

[16] Panos K. Chrysanthis and Krithi Ramamritham. Synthesis of Extended Transaction Models using ACTA. *ACM Transactions on Database Systems*, 19(3):450–491, September 1994.

[17] Workflow Management Coalition. http://www.aiai.ed.ac.uk/WfMC.

[18] XSoft Marketing Communications. Inconcert: Workflow software from xsoft, October 1995. http://www.xerox.com/XSoft/DataSheets/InConcert.html.

[19] Umesh Dayal, Hector Garcia-Molina, Mei Hsu, Ben Kao, and Ming-Chien Shan. Third Generation TP Monitors: A Database Challenge. *SIGMOD Record*, 22(2):393–397, June 1993.

[20] 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.

[21] Kenneth J. Goldman et al. The programmer's playground: I/O abstraction for user-configurable distributed applications. *IEEE Transactions on Software Engineering*, 21(9):735–746, September 1995.

[22] Christer Fernström. PROCESS WEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, pages 12–26, Berlin, Germany, February 1993. IEEE Computer Society Press.

[23] Rodrigo F. Flores. The value of a methodology for workflow, 1995. http://www.actiontech.com/market/papers/method5.html.

[24] 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.

[25] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[26] Dimitris Georgakopoulos, Mark Hornick, Piotr Krychniak, and Frank Manola. Specification and management of extended transactions in a programmable transaction environment. In *10th International Conference on Data Engineering*, pages 462–473, Houston TX, February 1994.

[27] Volker Gruhn and Rudiger Jegelka. An evaluation of FUNSOFT nets. In J.C. Derniame, editor, *Software Process Technology Second European Workshop*, number 635 in Lecture Notes in Computer Science, pages 196–214. Springer-Verlag, Trondheim, Norway, September 1992.

[28] George T. Heineman. *A Transaction Manager Component Supporting Extended Transaction Models*. PhD thesis, Columbia University, 1996. Forthcoming.

[29] 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.

[30] George T. Heineman, Gail E. Kaiser, Naser S. Barghouti, and Israel Z. Ben-Shaul. Rule chaining in MARVEL: Dynamic binding of parameters. *IEEE Expert*, 7(6):26–32, December 1992.

[31] Gail E. Kaiser and Wenke Lee. Pay no attention to the man behind the curtain. In *NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*, Athens GA, May 1996. Position paper. In press.

[32] Gail E. Kaiser and Calton Pu. Dynamic restructuring of transactions. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, San Mateo CA, 1992. Morgan Kaufmann. Available as Columbia University Department of Computer Science, CUCS-012-91, August 1991.

[33] Henry F. Korth and Greg Speegle. Formal Aspects of Concurrency Control in Long-Duration Transactions Systems using the NT/PV Model. *ACM Transactions on Database Systems*, 19(3):492–535, September 1994.

[34] Narayanan Krishnakumar and Amit Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2):119–152, 1995.

[35] Peiwei Mi and Walt Scacchi. Process integration in CASE environments. *IEEE Software*, 9(2):45–53, March 1992.

[36] J. Eliot B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. Information Systems. The MIT Press, Cambridge MA, 1985.

[37] Erich Neuhold and Michael Stonebraker (editors). Future Directions in DBMS Research. *SIGMOD Record*, 18(1):17–26, March 1989.

[38] Steven P. Reiss. Connecting tools using message passing in the field environment. *IEEE Software*, 7(4):57–66, July 1990.

[39] Marek Rusinkiewicz and Amit Sheth. Specification and execution of transactional workflows. In Won Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, chapter 29, pages 592–620. ACM Press, New York NY, 1994.

[40] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

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

[42] Workflow Management Coalition Draft Workflow Standard: Interoperability Abstract Specification. ftp://ftp.aiai.ed.ac.uk/pub/projects/WfMC/if4-a.ps.gz.