A 3-level Atomicity Model for Decentralized Process-centered Software Engineering Environments

Israel Z. Ben-Shaul[†] and George T. Heineman[‡]

[†]Technion-Israel Institute of Technology, Department of Electrical Engineering
[‡]Worcester Polytechnic Institute, Department of Computer Science

Abstract. Decentralized process-centered software engineering environments (PSEEs) provide an architecture for interoperability between multiple PSEEs with heterogeneous processes. Atomicity is a standard correctness model for guaranteeing that a set of activities occurs as an atomic unit, or none of them occur at all. Within a single PSEE, atomicity is the concern of its database system. In a decentralized environment, however, the autonomous environments must find ways to cooperate if an atomic unit is split between multiple PSEEs. This paper describes a flexible atomicity model that enables process administrators to reconcile the conflict between local autonomy and global atomicity and cooperatively specify the *scope* of multi-site atomicity based upon the desired semantics of multi-site tasks in the decentralized PSEE.

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

1 Introduction and Motivation

As software systems become more complex and larger in scale, their development and maintenance requires more people with various skills, often organized into groups. In a multi-team software development effort, it is desirable to decentralize the management and allow varying degrees of operational *autonomy*. For example, each team may use their own set of software tools and hardware, their own private files or databases, and their own software processes. Sometimes, such autonomy and privacy considerations are mandatory when the teams belong to different organizations, if out-sourcing occurs, for example. These autonomous teams may share tools and data, agreeing on some common sub-processes to *collaborate* to develop a product.

In previous work [1], Ben-Shaul et al. addressed the interoperability of process-centered software engineering environments (PSEEs) by providing (1) modeling facilities that enable individual sites to mutually agree on common sub-processes (Treaties), and (2) enactment facilities that enable to carry-out the multi-site shared sub-processes while retaining the autonomy and privacy of local sub-processes (Summit). An underlying theme in this work, which was embodied in the Oz system, was the emphasis on decentralization, i.e., avoiding the need to maintain global data, state, or control.

An important aspect of a PSEE is the transactional support it provides. Indeed, the significance and the application of transactions in (centralized) PSEEs has been

addressed extensively in various projects including Marvel, Merlin, Adele, EPOS, and SPADE. In this paper we examine transactional semantics for multi-site Decentralized PSEEs (DPSEEs). More specifically, we focus on multi-site *recovery-atomicity* (henceforth atomicity), a topic that has been hardly addressed in the PSEE community even for centralized PSEEs (which mainly focused on concurrency-control). Atomicity is a grouping of activities such that the outcome of their execution has all-or-nothing semantics; a failure in any of these activities (e.g., system- or user-abort) requires to rollback any of the effects or invoke compensating actions for unrollable effects.

The main difficulty with supporting multi-site atomicity in DPSEEs is reconciling the inherent conflict between the *global* nature of atomicity, which by definition must apply to all activities of an atomic unit, and between the desire to retain maximum *local* site autonomy over the management of (local) data. This conflict is particularly evident when a local activity aborts during the execution of a multi-site task. We would like to reduce the impact that local transaction managers (TM) can have on remote data (managed by other TMs), to a degree permissible by the global task.

2 The Atomicity Model

A process task is a partially-ordered set of activities (each of which may invoke an external tool, e.g., compiler) that defines a logical unit of execution. If an activity a_i only accesses data from its local TM, then a single local transaction T_i is created to encapsulate the data requests for the activity. A multi-site activity a_i involving n sites, however, is associated with n transactions — $T_i^1, T_i^2, \ldots, T_i^n$ — one at each site, where each local transaction defines a unit of atomicity.

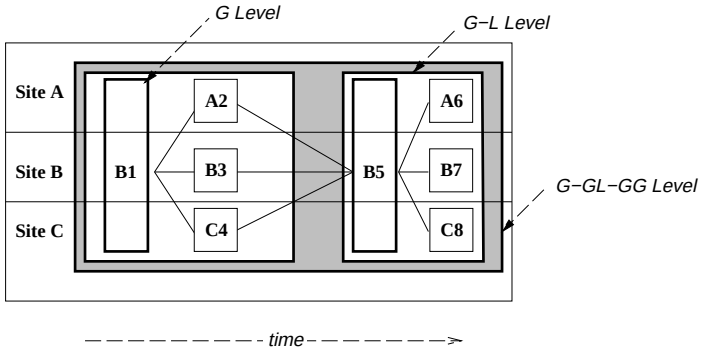

Fig. 1. The 3-level Atomicity Model

A multi-site task consists of interleaved multi-site and/or single-site local activities. When executed, such task alternates between global and local modes. In global mode, a multi-site activity is executed synchronously at one coordinating site, involving multi-site data and possibly multiple users (e.g., for groupware activities). In local mode, each PSEE executes asynchronously any local (sub)tasks on local data. We identify the following three types of atomicity, each of which can be explicitly and separately specified on a per-task basis within a process model:

1. Type **G** (Global) — This type 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 at each participating site. If any of these transactions abort for any reason, all transactions for the multi-site activity must abort, to preserve atomicity. **G** atomicity is the default for a multi-site activity.
2. Type **L** (Local) — This type is orthogonal to **G**, i.e., it preserves the atomicity of multiple transactions running on behalf the same taks in a single site, including (sub)transactions that execute on as part of a multi-site activity. More formally, at a particular site, S_j, **L** binds into an atomic unit the local transaction T_i^j, acting on behalf of the multi-site activity a_i of task t as well as the local transactions $\mathcal{L} = \{T_1^j, T_2^j, \ldots, T_k^j\}$ initiated for the local activities associated with task t at site S_j. If any of these local transactions aborts, then the entire set $\mathcal{L}$ must be rolled back as well as T_i^j, but other local transactions acting on behalf of t in other sites are not necessarily rolled back (it depends on their **G** setting, see below). Therefore the atomicity of activity a_i may be compromised in favor of retaining atomicity within a given site S_j.
3. Type **GG** (Global/Global) — This type atomically binds together several multi-site activities. When coupled with **G** and **L**, 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 that are explicitly specified and known by all sites (by the definition of a global task), the autonomy is *voluntarily* compromised by the local sites.

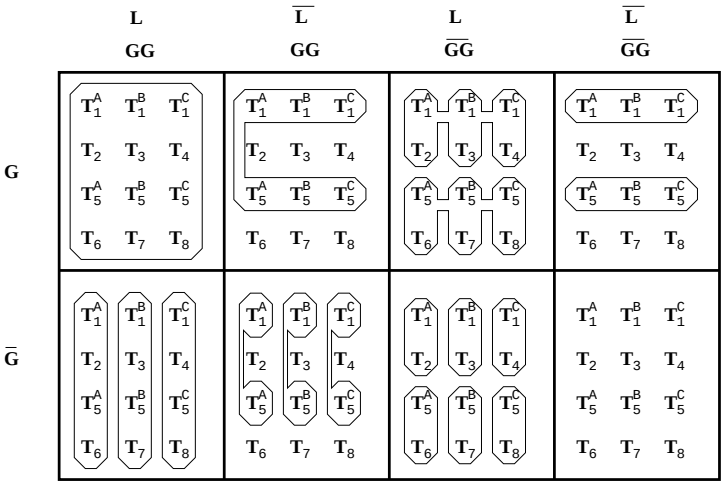

Fig. 2. Atomicity Units Made Up of **G**, **L**, and **GG**

The combinations of the three atomicity types create different units of atomicity that can be applied to fit the desired atomicity semantics of a given task. For example, Figure 1 shows three basic atomicity levels — **G**, **G-L**, and **G-L-GG** — and their different scopes (as represented by rectangles) 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 A and C, requesting that any generated local activities be executed, thus causing A_2, B_3, and C_4. Once these all complete, a new round of multi-site activities is initiated at site B, B_5, causing the execution of local activities A_6, B_7, and C_8. The failure of any transaction forces all transactions within the same atomic unit to abort, but does not affect other parts of the task.

The three combinations mentioned above, which define a hierarchy of nested contexts of atomicity, may be viewed as the "core" levels. However, any other combination may be valid for certain purposes. Figure 2 illustrates the various atomic units created by all eight combinations of types, using the example from Figure 1 (overline denotes lack of a type in a combination). For example, the entry **G**-$\overline{\mathbf{L}}$-**GG** protects the atomicity of the multi-site activities if any of the local emanating tasks fail, whereas $\overline{\mathbf{G}}$-**L**-$\overline{\mathbf{GG}}$ violates multi-site atomicity in favor of preserving local atomicity. This mode may be attractive in cases where the level of collaboration between the sites (and the corresponding level of trust) is low and hence aborting a local transaction cannot affect remote data. The potential inconsistency introduced by the non-atomicity of the multi-site activity may be either ignored, or tolerated by some inconsistency management mechanism, depending on the particular semantics of the activity.

The final issue regarding the flexibility of our model is the choice of atomicity units that each site selects to apply on a given multi-site task. For simplicity, we assumed that all sites in Figure 2 have the same atomicity mode. In general, however, while all sites must agree on the **G** and **GG** types (since they require by definition cross-site atomicity), each site is free to determine whether or not to employ **L** type atomicity, creating additional (less symmetric) "shapes" of atomicity units.

Consider, for example a three-site DPSEE consisting of a coding site (C), a code design site (D), and a quality assurance site (Q), and assume there exists a multi-site activity *integrate* which allows a code developer in C to integrate a newly modified software component into a system being tested at Q. *integrate* first checks that the component has not violated any design constraints, verifies that it passed all of its unit tests, and schedules a QA engineer to inspect any errors during system integration; these actions are performed by local activities at sites D, C and Q, respectively. At a later point, each site may perform further local activities (e.g., site C may invoke a local *report-component-integration-successful* activity), and the sites may join further multi-site activities. A plausible assignment of modes for this task may be **G**-**L** for C and D, and **G** for Q. If site C aborts while executing a local activity emanating from *integrate*, the effects of *integrate* are rolled back at all sites, plus local activities (e.g., unit tests) are rolled back at C and D, but Q's local work is left untouched.

We presented in this paper an abstract model for recovery-atomicity in transactional DPSEEs. Integration of the model with the DPSEE framework, including specification of atomic units, integration with other aspects of transaction management, and concrete realization of the model, are major present and future directions (see [2]).

References

1. Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling*. Kluwer Academic Publishers, Boston, MA, 1995.
2. Israel Z. Ben-Shaul and George T. Heineman. A 3-level atomicity model for decentralized workflow management systems. Technical Report 1013, Technion, Israel Institute of Technology, Department of Electrical Engineering, January 1996.