TAM: A System for Dynamic Transactional Activity Management *

Tong Zhou Ling Liu Calton Pu
Department of Computer Science & Engineering
Oregon Graduate Institute
Portland, OR 97291-1000
{tzhou,lingliu,calton}@cse.ogi.edu

1 Introduction

Electronic commerce (EC) has remarkably reshaped today's business practices. The logic embedded in business transactions are becoming more flexible and sophisticated; the number of components within these business transactions are fastly increasing; and business transactions are getting more distributed and interactive as well. These, on the other hand, make today's business applications more vulnerable to resource availability constraints and consistency problems due to failures. For example, network congestion, server overload, user absence, and other exceptions all might cause temporary or permanent bottlenecks to ongoing business workflows. Developing system support to make workflow activities more reliable, efficient, and adaptive to changes or exceptions therefore poses important yet challenging questions that call for practical solutions.

The TAM project at OGI aims at developing a flexible framework and algorithms that address these challenges. Comparing with other research efforts, our approach is unique in that we combine advanced transaction processing (TP) technologies with adaptive methods to guide our system design, and build the mechanisms by extending the functions of current TP systems.

On the theoretical side, we have developed a Transactional Activity Model (TAM) for the specification and management of activities with transactional properties. TAM is a careful combination of a compositional activity model [5] with well-defined extended transaction models (ETMs) [3] such as split/join transactions [9]. It provides simple and effective specification facilities that allow business process designers to specify the behavioral composition and refinement of complex activities and a wide variety of activity interaction dependencies in a high-level and declarative way. The embedded transactional semantics give business workflows automatic shield from execution anomalies caused by concurrency or failures. TAM also provides a set of dynamic activity restructuring operations with well-defined semantics [7, 11] that allow ongoing transactional activities to bypass execution bottlenecks or deal with failures.

On the practical side, we have built a system as a prototype implementation of TAM, for effectively monitoring and managing transactional applications. An important novel feature of our system is that it is based on the standard OLTP reference architecture [4, 2], and is built on top of a commercial OLTP product Transarc-Encina. Another feature that distinguishes our system is its support for dynamic activity restructuring as an effective facility for activity adaptation. To our knowledge, it is the first implementation of activity restructuring with consideration of transactional properties on OLTP systems. Our implementation continues to follow a special method called "Design for Implementation", which we carefully developed in our previous work on distributed extended transaction management [8]. The method is largely based on component technology, open implementation, program specialization, and plug-in adaptors for practical implementations. Our implementation also utilizes and leverages on the system mechanisms from our previous work, like RTF [1] and OCP [10]. We propose to demonstrate our system prototype at SIGMOD, illustrating its activity visualization, management, and dynamic restructuring features.

2 TAM System Description

As shown in Figure 1, the TAM system has a three-tier architecture: front-end, communication layer, and backend.

The **front-end** (the top tier) consists of Web-enabled graphical user interfaces, implemented using a mixture of Java applets, Javascript, and HTML. It currently has two components: the Activity Specification Pilot and the Activity Instance Monitor.

The **Activity Specification Pilot** allows activity designers to visually specify, modify, and build transactional activities. The graphically specified activities are mapped into activity specifications written in the ActivityFlow Specification Language [6], and handed over (via Specification-based wrappers) to the Activity Code Generator to generate

*The research was supported in part by the National Science Foundation (NSF) under grant IRI-9510112, by the U.S. Department of Defense Advanced Research Projects Agency (DARPA) under contracts F19528-95-C-0193 and MDA972-97-1-0016, by an Intel grant, and by NSERC under grants OGP-0172859.

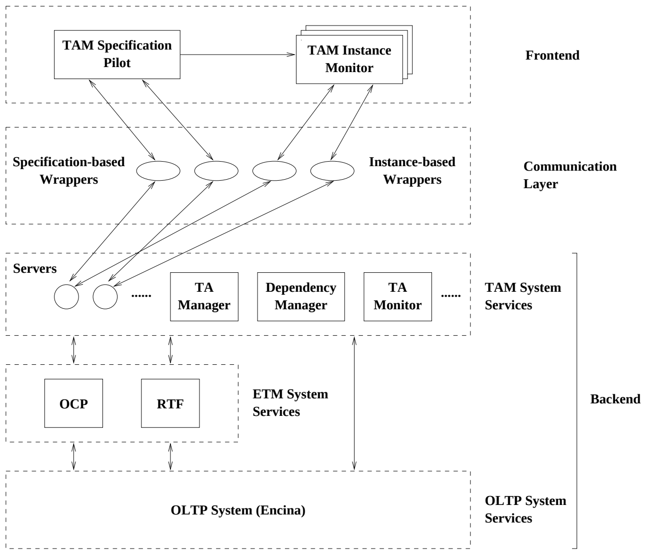


Figure 1: System Architecture

real activity code. This component also provides facilities to start up and shut down the backend TAM System Server. Besides, for a particular activity specification, this component launches different Activity Instance Monitors, one per running activity instance.

The **Activity Instance Monitor** administers and monitors the execution of activity instances, generates alert notifications for potential system problems (e.g., prolonged activity execution delay), and provides a set of dynamic activity restructuring operations for the modification of the execution path of a particular activity instance. The administration tasks include starting and stopping activity instances, as well as querying activity instance execution histories and timing information. The monitoring tasks include animated activity state changes, active execution tracing, and problem alerts. The activity restructuring interface also provides a facility for flexible change adaptation policy specifications on relevant affected activity instances (e.g., a set of instances working for a cooperative task). This component is launched by the Activity Specification Pilot for activity instances on an one-to-one basis.

The **communication layer** (the middle tier) contains a set of communication wrappers. These wrappers are classified into two categories: Specification-based wrappers that serve as the mediators between the Activity Specification Pilot and the backend, and Instance-based wrappers that mediate between the Activity Instance Monitor and the backend. This layer is designed to address some language compatibility and server capability concerns. For example, the current version of the OLTP system on which we build our prototype restricts us to use certain programming languages for applications (e.g., C/C++), making it necessary to provide the hook between these languages and others (like those used in the frontend). As another example, different web servers (e.g., apache, Netscape commerce server, etc.) have different capabilities, also bringing the need to accommodate each in the middle tier.

The **backend** (the bottom tier) has a coarser demarcation in the figure, Conceptually, it actually consists of three sublayers: TAM System Services, ETM System Services, and native OLTP System Services.

OLTP System Services sit at the lowest level in the backend. These are native functions provided by a commercial OLTP system (e.g., Encina), like transaction management, concurrency control, logging and recovery, etc. Our prototype currently uses Transarc's TPM Encina at this sublayer.

The middle sublayer holds what we call **ETM System Services** that support extended transactions [3]. These include the system mechanisms we developed in our previous distributed ETM work [8], like extended transaction manager [1], semantic concurrency control [1], open transaction coordination protocol [10], etc. These services either utilize or extend the functions provided in the bottom OLTP sublayer.

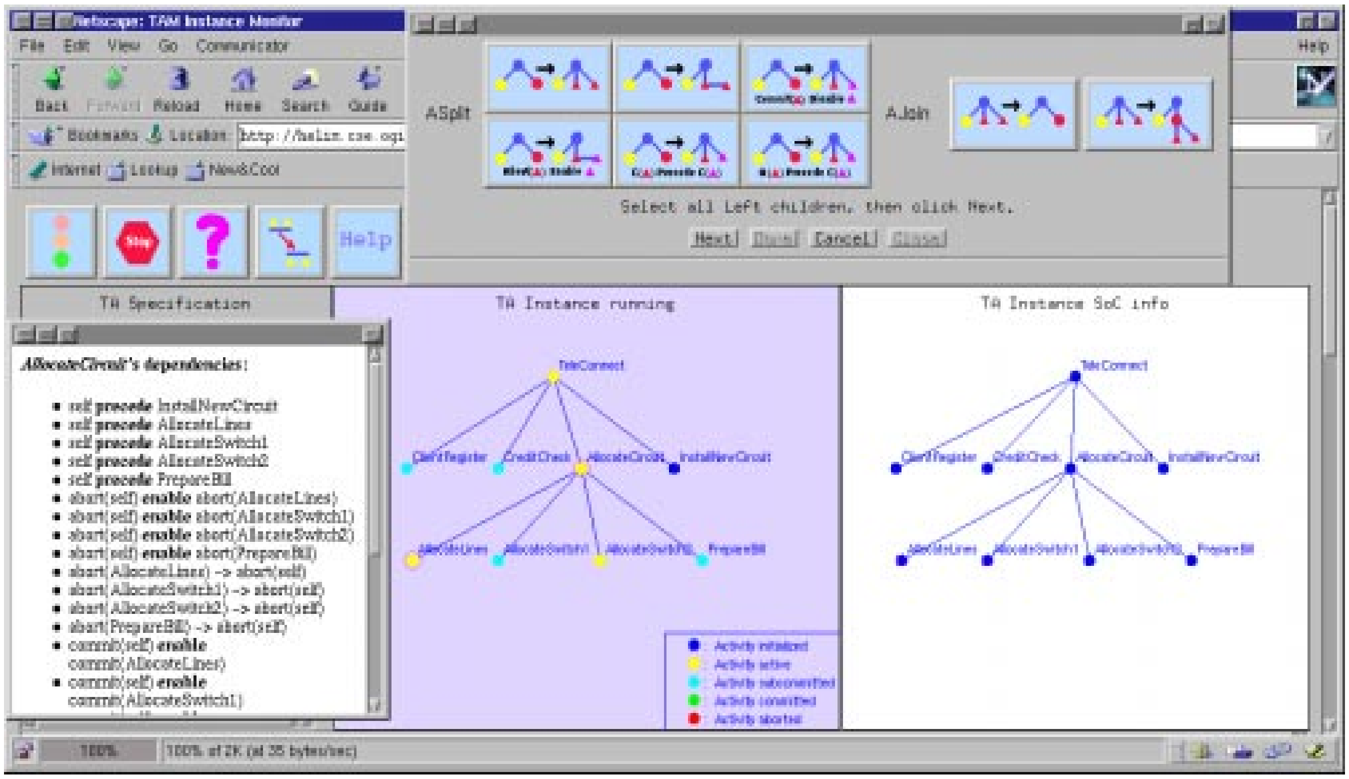
TAM System Services are at the top sublayer of the backend. These include the actual functions that support the frontend capabilities, and either utilize or extend the functions in the two sublayers below. Some of the key components are: the Transactional Activity Manager that actually performs the activity management tasks, like activities' begin, commit, abort, and dynamic restructuring; the Activity Dependency Manager that coordinates with the Transactional Activity Manager to preserve both inter- and intra-activity dependencies; the Activity Monitor component that keeps track of activity instances' execution information, like states, execution traces, timing, etc.; and the Activity Code Generator that generates actual activity code from their specifications.

3 Demo Description

We will demonstrate the following extensions to the underlying OLTP system in our TAM prototype:

- Visual display of specification information about transactional activities, such as activity composition hierarchies and activity dependencies.
- Web-based administration of the TAM system backend, like startup and shutdown of the system servers.
- Execution control and monitoring of ongoing transactional activity instances, like start, stop, animated state changes, execution tracing, and query capabilities. The query interface allows one to conduct simple queries about a single activity, quantified queries over multiple activity instances' execution history, as well as activity execution timing information.

  Figure 2 is a snapshot of the TAM Instance Monitor, along with the Activity Restructuring Interface window on the top and the activity dependency query result window on the left. The canvases show a telecommunication activity instance executing, with colors representing different activity states.
- Problem alerts, and dynamic restructuring of ongoing activity instances based on the alerts (manually done in the demo). This also demonstrates the effects of different change adaptation policies on relevant activity instances when a restructuring happens.



AllocateCircuit's dependencies:

- self **precede** InstallNewCircuit
- self **precede** AllocateLines
- self **precede** AllocateSwitch1
- self **precede** AllocateSwitch2
- self **precede** PrepareBill
- abort(self) **enable** abort(AllocateLines)
- abort(self) **enable** abort(AllocateSwitch1)
- abort(self) **enable** abort(AllocateSwitch2)
- abort(self) **enable** abort(PrepareBill)
- abort(AllocateLines) -> abort(self)
- abort(AllocateSwitch1) -> abort(self)
- abort(AllocateSwitch2) -> abort(self)
- abort(PrepareBill) -> abort(self)
- commit(self) **enable** commit(AllocateLines)
- commit(self) **enable** commit(AllocateSwitch1)

Figure 2: Activity Instance Monitor

The top pop-up window in Figure 2 is a snapshot of the Activity Restructuring Interface. Each image button represents a specialized restructuring primitive with built-in semantics. The interface then guides through the restructuring process via other dialogs, like pinpointing involved activity instances and change adaptation policies.

References

[1] R. S. Barga and C. Pu. A practical and modular method to implement extended transaction models. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[2] P. A. Bernstein. Transaction processing monitors. *Communications of the ACM*, 33(11):75–86, 1990.

[3] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1993.

[4] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[5] L. Liu and R. Meersman. The Basic Building Blocks for Modeling Communication Behavior of Complex Objects: an Activity-driven Approach. *ACM Transactions on Database Systems*, 21(3):157–207, 1996.

[6] L. Liu and C. Pu. Activityflow: Towards incremental specification and flexibile coordination of workflow activities. In *Proceedings of the 16th International Conference on Conceptual Modeling (ER'97)*, November 1997. Los Angeles, California, USA.

[7] L. Liu and C. Pu. Methodical Restructuring of Complex Workflow Activities. In *Proceedings of the 1998 IEEE Conference on Data Engineering*, Orlando, Florida, February 1998.

[8] C. Pu, R. Barga, T. Zhou, and S.-W. Chen. Implementing Extended Transaction Models. In *High Performance Transaction Systems (HPTS) Workshop 1997*, Pacific Grove, California, September 1997.

[9] C. Pu, G. E. Kaiser, and N. Hutchinson. Split-transactions for open-ended activities. In *Proceedings of the 14th International Conference on Very Large Data Bases*, 26-37, August 1988.

[10] T. Zhou, C. Pu, and L. Liu. Adaptable, Efficient, and Modular Coordination of Distributed Extended Transactions. In *Proceedings of the 4th International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[11] T. Zhou, C. Pu, and L. Liu. Dynamic Restructuring of Transactional Workflow Activities: A Practical Implementation Method. In *Proceedings of the 7th International Conference on Information and Knowledge Management*, Washington, D.C., November 1998.