In: Proceedings of the 4^{th} International Conference on Mobile Data Management (MDM'2003), pages 356-361, Melbourne, Australia, January 2003. Lecture Notes in Computer Science (LNCS), Volume 2574, Springer-Verlag.

Transactional Peer-to-Peer Information Processing: The AMOR Approach

Klaus Haller Heiko Schuldt Hans-Jörg Schek
Database Research Group

UMIT Innsbruck Institute of Information Systems
Swiss Federal Institute of Technology
CH-8092 Zürich

Email: {haller,schuldt,schek}@inf.ethz.ch

Abstract

Mobile agent applications are a promising approach to cope with the ever increasing amount of data and services available in large networks (e.g., corporate intranets or the Internet). By using mobile agent technology, a user no longer has to manually browse for certain data and/or services but rather to submit a mobile personal agent that accesses and processes information on her/his behalf (i.e., places orders or bids in some electronic auctions, or fixes dates). These mobile agents operate on top of a peer-to-peer network spanned by the individual providers of data and services. However, support for the correct concurrent and fault-tolerant execution of multiple agents accessing shared resources is vital to agent-based information processing. This paper addresses this problem and shows how agent-based information processing can be enriched by dedicated transactional semantics — despite of the lack of global control which is an inherent characteristic of peer-to-peer environments. We introduce the AMOR project (Agents, MObility, and tRansactions) and we show how conventional concurrency control and recovery protocols have to be extended and generalized such that they can be implemented in a decentralized way and be applied in a truly distributed peer-to-peer environment.

1 Introduction

A consequence of the proliferation of Internet technology is the availability of a vast amount of data. Recent trends in accessing this data propose the use of mobile agent technology [CZ98, KG99]. The most important advantages of this approach are imposed by the aspect of mobility which allows to considerably reduce the amount of data that has to be shipped over the network and to reduce network latency. Mobile agents consist of code and execution state which is both transferred from host to host in order to access data or invoke services there. By using mobile agent technology, a user no longer has to manually browse for certain data and/or services but rather to submit a mobile personal agent that accesses and processes information on her/his behalf (i.e., places orders or bids in some electronic auctions, or fixes dates). To this end, mobile agents operate on top of a peer-to-peer network spanned by the individual providers of data and services.

Most current mobile agent approaches focus on the support of applications which gather and filter data [BGM+99, GKP+01]. However, in order to support another practically relevant class of mobile agent applications, the aspect of *information processing* has also to be considered. The extension of the capabilities of mobile agents systems from information gathering towards information processing in a peer-to-peer-based system environment now goes along with additional requirements that users impose on their mobile agent applications. Essentially, mobile agents have to leave the system in a consistent state, even in the case of failures or when different agents access shared resources and/or services concurrently. This urgently requires support for correct concurrency control and failure handling. Such requirements are well understood

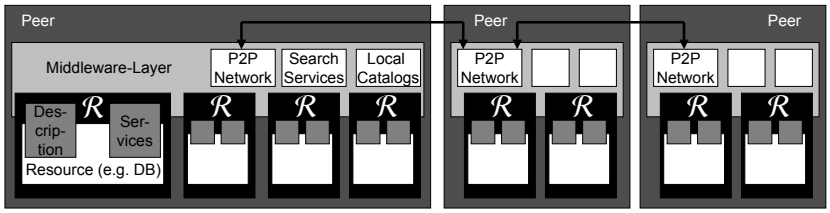

Figure 1: The AMOR System Model

in traditional information systems. Essentially, applications are encompassed into transactions and a centralized transaction manager controls their execution, i.e., it enforces their atomic behavior in case of failures and guarantees synchronization in the case of concurrency. However, in the peer-to-peer environments in which mobile agents execute, such a centralized transaction manager does not exist. Yet, the proper failure handling is delegated to the individual agents and the task of synchronization requires the communication among groups of agents, thus necessities an appropriate protocol to share and distribute information.

The goal of the AMOR project (Agents, MObility, and tRansactions) at ETH Zürich is to bring together transaction processing and peer-to-peer environments and to provide the foundations for correct and reliable peer-to-peer information processing.

The main contribution of this paper is a protocol that allows for the decentralization of concurrency control and recovery. Yet, it has to be noted that concurrency control and recovery are to be solved jointly and in a global manner; however, by applying sophisticated replication techniques to metadata needed for transaction management, we are able to implement support for transactions in a truly distributed way.

The remainder of the paper is structured as follows: The AMOR system architecture is presented in Section 2. Section 3 discusses the AMOR approach to decentralized transaction management. The AMOR prototype is introduced in Section 4. Section 5 discusses related work and Section 6 concludes.

2 The AMOR System

The bottom layer of the AMOR system consists of a set of distributed and autonomous *resources*, e.g., database systems or flat files. These resources are wrapped by so-called *resource agents* ($\mathcal{R}$). Each resource agent is responsible for encapsulating a single resource and for providing a unique interface to the latter. Data objects managed by the underlying resources can only be accessed and/or manipulated by the services provided by the corresponding resource agent. Examples for such services are stored procedures in databases or java code fragments accessing a database via JDBC. Let $\mathcal{R}^*$ denote the universe of all resource agents of the system and $\mathcal{S}^*$ the universe of the services provided by them. We assume the resources wrapped by all $\mathcal{R}_i, \mathcal{R}_k \in \mathcal{R}^*$ to be pairwise disjoint. The independence of resources means that the services of a resource agent only operate on local resources and are not redirected to remote resources.

Resources and therefore also the resource agents are hosted by *peers*. The number of resources is not fixed and may differ from peer to peer. Peers are connected by a middleware layer forming the peer-to-peer (P2P) network. Peers may dynamically decide to join or leave the system such that the P2P network continuously evolves. The overall AMOR system model is depicted in Figure 1.

Applications on top of the AMOR peer-to-peer network are executed by mobile *transactional agents*. Let $\mathcal{T}^*$ be the set of all mobile transactional agents of the system. For the purpose of application execution, each $\mathcal{T}_i$ of $\mathcal{T}^*$ migrates within the network to peers so as to locally invoke services there. Hence, services are used as building blocks for transactional agent applications.

A major concern of the AMOR approach is to enrich distributed applications encapsulated within mobile transactional agents by dedicated transactional semantics. However, due to the inherent distribution and

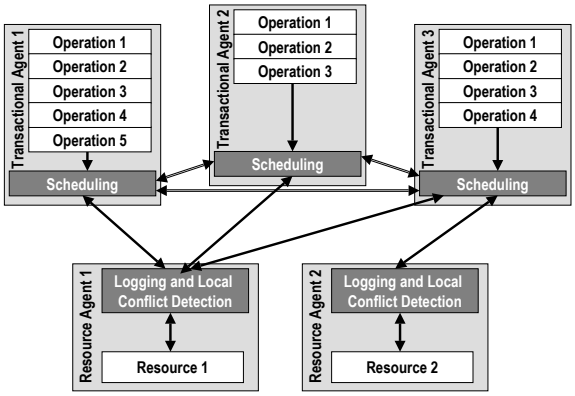

Figure 2: Decentralized AMOR Approach to Transaction Processing

dynamic behavior of the P2P network and due to the autonomy of peers, a centralized instance does not exist. As a consequence, no dedicated transaction manager controlling the execution of the transactions embedded in the $\mathcal{T}_i$ exist. Hence, the tasks of correctly synchronizing the access to shared resources and of properly handling failures is delegated to the $\mathcal{T}_i$. However, this functionality is transparent to the application developer, i.e., to the person implementing a transactional agent.

3 Decentralized Transaction Processing in AMOR

Each $\mathcal{T}_i$ implements an individual, independent, and distributed transaction. Traditionally, transaction management in a distributed environment is provided by a dedicated centralized coordinator, e.g., a TP Monitor [BN97]. The coordinator's task is to orchestrate the execution of distributed transactions and to enforce their atomic commitment by using a 2PC protocol [GR93]. Whereas the centralized approach is highly appropriate for small, well-delimited environments with a fixed number of peers, it cannot be applied to large-scale networks with a large number of distributed, heterogeneous, and autonomous peers In addition to technical issues associated with such large-scale distributed systems, also administrative aspects (i.e., peers might dynamically join or leave the system), a centralized approach would be too restrictive.

Hence, we must strive for a *distributed implementation* of the functionality of a coordinator. Essentially, following the unified theory of concurrency control and recovery [VHBS98], coordination functionality requires both isolation and atomicity to be provided jointly. In terms of recovery, this means that each failure of some $\mathcal{T}_i$ has to be handled properly. To this end, the effects of all services it has invoked so far have to be undone or alternative service invocations leading to a well-defined state have to be enforced. This requires that service invocations are logged which is done by each resource agent mediating transactional agent requests to its resource. In terms of concurrency control, the concurrent invocation of services has to be controlled such that each $\mathcal{T}_i$ is correctly shielded from other transactional agents. Since flow of information is only possible via services manipulating/retrieving data objects, the commutativity behavior of services has to be known (i.e., which service invocations do conflict). Due to the independence of resources, such conflicts may only exist between services wrapped by the same $\mathcal{R}_k$. Hence, conflict detection can also be realized at the level of resource agents. Two service invocations are in *conflict* if their order matters, i.e., if their return values would change when being ordered differently; otherwise, the services *commute*.

Reasoning about global correctness — using information on conflicts — is based on the notion of schedule. A schedule $\mathbf{S}$ is defined as a tuple $\mathbf{S} = (\mathcal{T}_S, <_S)$ reflecting the concurrent execution of a set of transactions $\mathcal{T}_s$ where $<_S$ is the order in which the services of the $\mathcal{T}_i \in \mathcal{T}_S$ are invoked (with $<_i \subseteq <_S$ for all $\mathcal{T}_i$

of $\mathcal{T}_S$). The unified theory of concurrency control and recovery addresses both atomicity and isolation by considering regular services and inverse services (which are used to semantically undo the effects of regular ones) within the same framework, i.e., within a schedule. Serializability with ordered termination (SOT) has been identified as correctness criterion that accounts for both problems [AVA$^+$94]. Moreover, it has been shown that the SOT criterion can be checked by means of the acyclicity of the serialization graph of a schedule [AVA$^+$94]. In short, the serialization graph of a schedule $\mathbf{S}$ is a directed graph where the nodes are the transactions of $\mathbf{S}$ and the directed edges correspond to the conflicts between transactions. Reasoning about the correctness of a schedule based on a serialization graph requires global information to be available at a central coordinator — the transaction manager. However, due to the absence of a centralized transaction manager in AMOR, conflict information on local service invocations have to be communicated from a resource agent to the corresponding $\mathcal{T}_i$. Then, scheduling can be enforced without centralized coordinator but rather by distributing this conflict information between transactional agents, thereby replicating meta information needed for synchronization purposes. In Figure 2, the distributed AMOR approach to transaction management is illustrated.

AMOR follows an optimistic approach to transaction management. This means that each $\mathcal{T}_i$ executes without determining on the spot whether a service invocation of $\mathcal{T}_i$ is allowed or not.

In order to detect incorrect executions as soon as possible, each $\mathcal{T}_i$ has to be equipped with metadata reflecting a multi-agent execution. When a copy of the global serialization graph would be available at each transactional agent, cyclic conflict could be detected and thus resolved immediately. But always maintaining a copy of the complete serialization graph with each $\mathcal{T}_i$ is not practical since it would impose considerable overhead. However, an important observation is that in the type of system we are addressing, more or less closed –albeit not static– communities exist. These communities are sets of closely related resources accessed by the same transactional agents, i.e., agents which aim at addressing the same or similar tasks. Hence, conflicts are only possible between agents invoking services within the same community. We denote the set of transactional agents executing in such a community as *region*. In terms of the data structures maintained for concurrency control purposes, a region corresponds to the nodes of a connected subgraph of the global serialization graph. Consequently, only this connected subgraph (termed *region serialization graph*) has to be replicated among all $\mathcal{T}_i$ of the same region, rather than the complete serialization graph. Obviously, if the serialization graphs of all regions are free of cycles, so is also the global serialization graph. Thus, reasoning about system-wide correctness can be safely shifted to the universe of region serialization graphs. By making use of the partitioning of the system in disjoint regions, AMOR uses replicas of region serialization graphs maintained by each transactional agent to enforce correct multi-agent executions. This requires that i.) conflicts are communicated from resource agents to the corresponding transactional agents and ii.) the replicas of region serialization graphs are kept consistent among all $\mathcal{T}_i$ of the same region.

However, regions are not static. Rather, the composition of regions is affected by the services invoked by the transactional agents of a region. Consider a service invocation of $\mathcal{T}_i$ which imposes a conflict with a service invocation of a $\mathcal{T}_j$ of another region. As a consequence of this conflict, the two originally independent regions have to be merged. Hence, as an additional requirement for consistent metadata management in AMOR, the two region serialization graphs have to be consolidated and finally distributed among all transactional agents of the new region. Similarly, by correctly committing a transactional agent or in case of a rollback, regions may split and so does the corresponding region serialization graph.

4 Prototype

The AMOR prototype implementation is based on the Aglets [LO98] framework which we have extended in several directions. Most importantly, we have added support for agent and place descriptions. This allows to search for either places or agents [HS01]. In short, this extension has led to the implementation of two

new types: P2P-Agents and Agents Management Agents (AMAs). The latter ones are part of the place infrastructure and responsible for managing the information about places. This includes the information on the transactional agents that are currently residing on that specific place. The former type of agents, P2P-Agents, implement mechanisms for sending messages and for migrating to target places by specifying criteria – in other words, predicates which have to be fulfilled for message recipients or destinations of migration.

P2P-Agents are further refined in resource agents and transactional agents as introduced in this paper. In addition to the tasks of a resource agents presented in the previous sections, they provide an XML-description so as to support the search for resources in the peer-to-peer network. The AMOR prototype encompasses implementations of such resource agents, e.g., one for Oracle8 object-relational databases. This resource agent allows to access object-relational tables by wrapping user-defined methods and by providing these services to transactional agents. The other refinement of P2P-Agents is reflected in the concept of transactional agents which implement the protocol presented in this paper.

5 Related Work

In terms of related work in the context of mobile agents, two directions have to be considered: workflow execution and transactions. The mobile agent approach is commonly agreed to be highly suitable for workflow execution, e.g., [CGN96]. Hence, workflows executed by mobile agents are similar to the AMOR mobile transactional agent applications built on top of the services made available by different resource agents. Recent work extends these ideas and deals with the problem of how to schedule mobile agents in case different places are available [XRS01] for a particular service. Other approaches even address the intersection of mobile agent technology and transaction management. First results were achieved by enforcing atomicity and fault tolerance [SP98]. More recent approaches like [SAE01], albeit still concentrating on the atomicity aspect, also provide support for concurrency control. However, this is achieved by combining timestamp-ordering and 2PC, which means with a rather limited degree of concurrency.

6 Summary and Outlook

In this paper, we have presented the ideas of the AMOR project which allows for a novel decentralized implementation of transaction management in a peer-to-peer environment. Applications are embedded within mobile transactional agents. Due to the lack of a central coordinator which is inherent to peer-to-peer systems, the task of enforcing correct concurrency control and failure handling for agent-based applications is shifted to the transactional agents. To this end, sophisticated replication management of metadata needed for synchronization purposes is applied. Each transactional agent receives information about local conflicts when it invokes a service. This information is used to update the local view on the relevant portion of global metadata, i.e., the region serialization graph. The different local replicas of these graphs are kept consistent by means of communication at the agent level.

In our future work, we aim at extending the existing AMOR framework by providing support for shielding unreliable network connections.

References

[AVA+94] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A. Abbadi, H.-J. Schek, and G. Weikum. Unifying Cconcurrency Control and Recovery of Transactions. In *Information Systems*, 1994.

[BGM+99] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus. *Mobile Agents for Distributed Information Retrieval*, pages 355–395. Springer, 1999.

[BN97] P. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann, 1997.

[CGN96] T. Cai, P. Gloor, and S. Nog. Dartflow: A Workflow Management System on the Web using Transportable Agents. Technical Report TR96-283, Dartmouth College, Hanover, NH, 1996.

[CZ98] W. Cockayne and M. Zyda, editors. *Mobile Agents*. Prentice Hall, 1998.

[GKP+01] R. Gray, D. Kotz, R. Peterson, J. Barton, D. Chacón, P. Gerken, M. Hofmann, J. Bradshaw, M. Breedy, R. Jeffers, and N. Suri. Mobile Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task. In *5th International Conference on Mobile Agents (MA), Atlanta, GA*, 2001.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[HS01] K. Haller and H. Schuldt. Using Predicates for Specifying Targets of Migration and Messages in a Peer-to-Peer Mobile Agent Environment. In *5th Int'l Conf. on Mobile Agents (MA), Atlanta, GA*, 2001.

[KG99] D. Kotz and R. Gray. Mobile Agents and the Future of the Internet. *Operating Systems Rev.*, 33(3), 1999.

[LO98] D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley Longman, 1998.

[SAE01] R. Sher, Y. Aridor, and O. Etzion. Mobile Transactional Agents. In *21st International Conference on Distributed Computing Systems (ICDCS), Phoenix, AZ*, 2001.

[SP98] A. Silva and R. Popescu-Zeletin. An Approach for Providing Mobile Agent Fault Tolerance. In *Second Int. Workshop on Mobile Agents (MA), Stuttgart, Germany*, 1998.

[VHBS98] R. Vingralek, H. Hasse-Ye, Y. Breitbart, and H.-J. Schek. Unifying Concurrency Control and Recovery of Transactions with Semantically Rich Operations. *Theoretical Computer Science*, 190(2), 1998.

[XRS01] R. Xie, D. Rus, and C. Stein. Scheduling Multi-Task Agents. In *5th International Conference on Mobile Agents (MA), Atlanta, GA*, 2001.