A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases*

Evaggelia Pitoura and Bharat Bhargava
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398
{pitoura, bb}@cs.purdue.edu

Abstract

Information applications are increasingly required to be distributed among numerous remote sites through both wireless and wired links. Traditional models of distributed computing are inadequate to overcome the communication barrier this generates and to support the development of complex applications. In this paper, we advocate an approach based on agents. Agents are software modules that encapsulate data and code, cooperate to solve complicated tasks, and run at remote sites with minimum interaction with the user. We define an agent-based framework for accessing mobile heterogeneous databases. We then investigate concurrency control and recovery issues and outline possible solutions. Agent-based computing advances database transaction and control flow management concepts and remote programming techniques.

Keywords: agents, mobile computing, workflow, multidatabases, transactions, concurrency control, recovery

1 Introduction

With the rapid development of networking, computing applications increasingly rely on the network to obtain and update information from numerous remote sites. Such applications may often incorporate wireless connections, which are expensive, unreliable, slow and currently not widely available. Given these technical impediments, users of wireless communications will be connected only intermittently with the rest of the distributed system [14, 19].

Today, networking is based on remote procedure calling (RPC), where a network carries messages. To overcome the communication barrier, there is a need for lighter-weight and more flexible architectures. An alternative such approach is remote programming (RP) [26]. In this case, a network carries *agents* which are objects that encapsulate data and procedures that the receiving computer executes. To execute an agent is to perform its procedures in the context of its data. Agents simplify the necessary distribution of computation and overcome the communication barrier by minimizing the number of interchanged messages. Furthermore, agents can easily be customized, thus making the network an open platform for developers. Much recent research has been devoted to agent-based systems [22] and their implementation in new commercial products. Among them, the General Magic Telescript is in use in AT & T PersonalLink servers and in the Sony Magic Link [28, 26].

Most current research on agent-based models focuses on aspects related to intelligence. Although some of the intelligence characteristics attributed to agents are highly unrealistic, there is a short-term product potential in agent technologies intended for database applications, email filters, and group-enabled applications [12]. In this paper, we focus on the consistency and recovery aspects of agent-based computing. We provide a framework for agent-based access to heterogeneous mobile databases, explore the implications of such a model, and identify the aspects in which it differs from traditional database models. We then introduce appropriate workflow constructs and outline solutions for concurrency and recovery. Many of the concepts in this paper have been previously introduced in various contexts; our objective here is to integrate them through the perspective of the agent-based paradigm.

The remainder of this paper is organized as follows. The agent-based model of computing is introduced in Section 2, along with an exploration of its implications for concurrency control and recovery. In Section 3, we present a control flow model for agent-based computing. In Section 4 and 5, respectively, we discuss concurrency control and recovery. In Section 6, we compare our work with related research, and concluding remarks are offered in Section 7.

2 The Agent-Based Model

We shall consider the case of a distributed environment consisting of heterogeneous and autonomous database systems. To provide interoperability, each database system exports a predefined set of operations called *primitive database methods*. Applications interact with these database systems through the submission of *application agents* (or simply *agents*). An application agent can access the databases only by employing primitive database methods. Special agents, called *database agents*, coordinate access to the database items. These database agents are responsible for maintaining the consistency of each database system and for handling recovery in case of failures.

*In *Sigmod Record* 24(3), pp 44–49, September 95

Agents, like distributed processes, model task executions. Multiple agents are executed concurrently and access one or more database systems. Each agent is written in a high-level distributed programming language that allows internal parallelism. The structure of the agent is then translated into a set of dependencies among execution states. This information is used to schedule the components of an agent. Additional dependencies may result from concurrency control protocols used to ensure database consistency. An application agent can communicate with other application agents in the cooperative execution of their assigned tasks. This communication is accomplished by invoking methods to access the local data belonging to another agent. Again, these operations are selected from a pre-specified set of primitive methods called *primitive application methods.* Formally,

Definition 1 (agent) *An agent is an active object ($\mathcal{D}$, $\mathcal{M}$, $\mathcal{SD}, \mathcal{P}$), where $\mathcal{D}$ is a set of local data, $\mathcal{M}$ a set of methods, $\mathcal{D}$ a set of structural dependencies among methods, and $\mathcal{P}$ a set of break and relocation points.*

Definition 2 (method) *A method of an agent is (a) a primitive database or application method that accesses data from a database or another agent, respectively; (b) a local method that accesses the agent's local data; or (c) a combination of local and primitive methods.*

Structural dependencies and break and relocation points are described in Section 3. An agent executes its methods in the context of its data. Methods can modify both local and remote data as well as the specifications of an agent.

2.1 Examples

We will present three agent-based scenarios. The first describes electronic news filtering and is slightly adapted from [16]. The second is located in the context of the electronic market place and is slightly adapted from [27]. The final example pertains to mobile computing and is taken from [14].

Example 1: Electronic News Filtering. Mary creates an application agent to select from the available news items those that match interests which she has previously specified in a profile database. The agent interacts with both the profile and the news database agents, performs some form of text analysis and retrieves the appropriate information. Different application agents are created to express each of Mary's interests; for instance, a "politics agent" searches for politics news while a "literature agent" searches for news about new book releases. The agents accept feedback from Mary and then update her database profile to reflect any changes in her interests.

Example 2: Shopping. John wants to buy a camera. He creates an application agent with specific instructions to find the least expensive camera of a specific brand and purchase it using John's credit card account. To carry this out, John's agent visits the yellow pages database agent. It then communicates with the database agents of each of the shops listed there, negotiates with them regarding the price, returns to the database agent of the shop offering the best value, and purchases the camera. Finally, the agent updates John's personal files by communicating with their supervising database agent.

Example 3: Mobile Computing. Mary, an insurance agent on her way to meet a prospective customer, creates an application agent to check his credit record and other credentials. The application agent continues executing even while Mary turns off her palmtop to save energy. In the process of formulating an individually-tailored policy, the application agent interacts with other application agents including that of Mary's insurance company.

2.2 Characteristics of the agent-based model

We can identify the following characteristics of agent-based computation that have an impact on concurrency control and recovery:

1. Since an agent accesses multiple heterogeneous, autonomous, and distributed database sites, concurrency control is in some respects similar to that in multidatabase systems. There are, however, some significant differences from traditional multidatabase schemas:
   - Agent-based computation is *decentralized.* There is no global transaction manager or central database agent, since the environment is open and evolving. Since agents are submitted from various sites, including mobile stations, it would be unrealistic and inefficient to route all agents through a central point.
   - The decomposition of an agent into local and primitive methods can not be determined at the time of its creation. This decomposition depends on the results of the execution of its previous actions and can be *dynamically* modified at runtime.
   - The *interface* offered by each database system is a collection of primitive methods. In contrast, multidatabase concurrency control assumes that applications interact with the database systems by submitting read and write operations.
2. Agents must coordinate with each other and exchange information, and an agent thus cannot be executed as an isolated transaction. In addition, since agents execute complex activities *advanced control flow* features must be provided.
3. Each agent executes in the context of its own local data.
4. Each agent is a robust, *recoverable object.* In case of failure, not only the data items in the databases should be restored but also the local data and the computation state of the agent.

Figure 1 highlights the architectural differences between traditional multidatabase and agent-based systems.

3 Flow of Control Specification

An application agent consists of a set of methods and a number of control flow specifications. The control flow specifications of an agent include (1) structural dependencies that define the order of method execution; (2) breakpoints that define the points of interaction among agents; and (3) relocation points that define the execution environment.

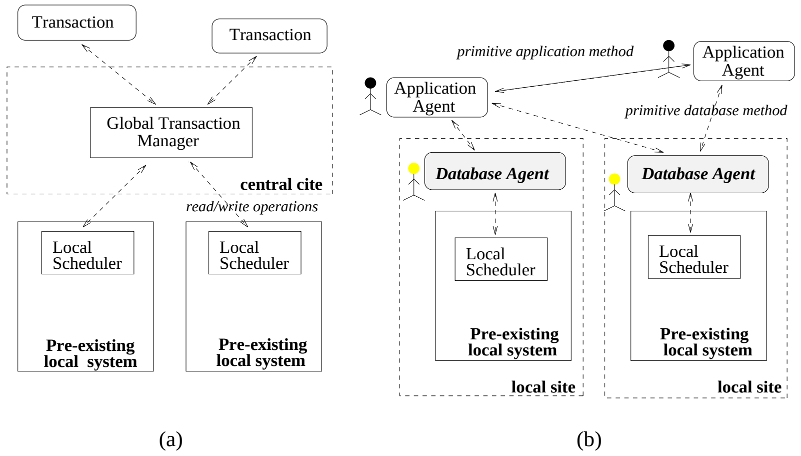

Figure 1: (a) Traditional multidatabase architecture (b) Agent-based architecture

3.1 Structural dependencies

Structural dependencies are dependencies among controllable states of the methods of an agent. In terms of primitive methods, the finest granularity of the lifetime of an agent that can be controlled at the time of its specification or execution is the completion (commit or abort state) and the submission (begin state) of the method. The actual execution time of a primitive method is under the control of the corresponding database or cooperating database agent. In addition, some database agents provide a prepare-to-commit state that indicates that a primitive method has completed execution and its results are about to become permanent. In terms of local methods, an agent has also control over their execution time. We distinguish two types of commitment depending on whether the result of the execution is semantic failure or semantic success.

Definition 3 (structural dependency) *A structural dependency SD is a triple (C, $\mathcal{M}$, $\mathcal{S}$), where C is a specification, $\mathcal{M}$ is a set of methods, and $\mathcal{S}$ is a set of controllable states of the methods $M \in \mathcal{M}$. For a primitive M, a controllable state in $\mathcal{S}$ may be commit (semantic failure or semantic success), abort, prepare-to-commit, or submit, and for a local method it can also be execute.*

We distinguish three types of structural dependency based on the form of C: trigger, ordering, and real-time dependencies.

Definition 4 (trigger) *In a triggering structural dependency (C, $\mathcal{M}$, $\mathcal{S}$), C has the following form: if M_i enters state s_i, then M_j must enter state s_j for M_i, $M_j \in \mathcal{M}$ and s_i, $s_j \in \mathcal{S}$.*

Special cases of triggering structural dependencies include critical, contingency, and compensation methods. *Critical methods* are methods that, when aborted (or fail semantically) cause the entire agent to abort (or fail semantically). *Contingency methods* are methods that are executed as alternatives when a method fails semantically. *Compensation methods* are methods that are executed to semantically undo the effect of a committed method when some other method aborts.

Definition 5 (order) *In an ordering structural dependency (C, $\mathcal{M}$, $\mathcal{S}$), C has one of the following forms: M_i can enter state s_i only after M_j has entered state s_j, or M_i cannot enter state s_i after M_j has entered state s_j for M_i, $M_j \in \mathcal{M}$ and s_i, $s_j \in \mathcal{S}$.*

Ordering structural dependencies can be used to express data flow dependencies, for instance that M_i reads data produced by M_j.

Definition 6 (real-time) *In a real-time structural dependency (C, $\mathcal{M}$, $\mathcal{S}$), C specifies a requirement for the real time submission or completion of the methods in $\mathcal{M}$.*

All methods of an agent are executed in parallel unless otherwise indicated by an ordering dependency or by restrictions imposed by concurrency control. The ordering dictated by concurrency control is necessary for the consistency of the database data or the local data of an agent.

3.2 Cooperation among agents

Since agents are by nature synergistic [11], an agent-based model must support cooperation among application agents. To support agent cooperation, in General Magic, for instance, the invocation of the *meet* command permits two agents to interchange information, if this is authorized by their specifications [28]. The Knowledge Query and Manipulation Language (KQML) under development as part of a large DARPA-sponsored knowledge initiative is a language and a protocol which supports high-level communication among intelligent agents [9]. In KQML, agents communicate by exchanging messages.

Thus, we see that agents, unlike traditional transactions, are not isolated from other concurrently executing agents. This principle may be stated formally with the assistance of concepts found in advanced transaction models. One such appropriate concept is the breakpoint [8]. In this approach, the steps (here, methods) of two transactions (here, agents) can be interleaved only at pre-specified execution points called breakpoints.

Definition 7 (breakpoint) *A breakpoint of an agent A is a triple $(B_s, B_e, \{(A_i, M_j)\})$ where $\{(A_i, M_j)\}$ is a set of pairs of methods and agents, and B_s, B_e are controllable states of methods in A which allow members of $\{(A_i, M_j)\}$ to be executed between states B_s and B_e of A.*

3.3 Relocation

In the remote procedure calling model, programs are immobile and each application is statically installed on a site. Agents, however, are by nature dynamically distributed with their components executed in remote databases or cooperative agents. Agents are also mobile in that their context, i.e., their local data, can change dynamically. To model relocation, we use a notion similar to *delegation* [5, 2].

Definition 8 (relocation point) *A relocation or delegation point of an agent A is a triple $(B_s, B_e, (A', B'_s))$ where B_s, B_e are controllable states of methods in A, A' is another agent, and B'_s is a controllable state of a method in A' and determines that the part of A between the states B_s and B_e is to be executed (delegated) as part of A' after B'_s.*

The ability to relocate the center of computation is of great importance in a computing environment with wireless connections. In such an environment, the location of a user changes dynamically with time. Thus, the distance of a client from an information provider is not a fixed parameter of the cost of the service. Relocating a computation permits the minimization of communication costs and the improvement of response time by reducing the physical distance between information providers and by considering changing network loads and availability. Mobile agents also facilitate load balancing among base stations.

4 Concurrency Control

The execution of an application agent consists of local steps, primitive message steps, and composite message steps. A *local step* is the execution of a local method. A *primitive message* step is the execution of a primitive database or application method. A *composite message step* is the execution of a composite method and consists of a number of primitive and local steps. Thus,

Definition 9 (execution) *An agent execution (history) is a partial order $(T, >)$ where T is a set of local, primitive, and composite steps and break or relocation points and $>$ is an order on the steps and points as determined by flow of control specifications.*

4.1 Maintaining correctness

Multiple agents concurrently access shared resources. Such access involves two aspects of correctness: structural and data correctness. *Structural correctness* ensures the structural properties of agents according to their control flow specifications. We assume that, as in TeleScript [28], there is an engine (an interpreter) within the agent that executes local methods and handles structural dependencies. The specification and enforcement of structural dependencies can be accomplished by various means including petri nets or active rules.

Data correctness maintains the consistency of shared resources and is usually specified through a number of integrity constraints. In the agent-based model, maintaining consistency refers to (a) maintaining the consistency of data in each database system and (b) maintaining the consistency of the local data of each agent. The second part of the definition of correctness is necessary since parts of the code of an agent are executed concurrently and since other cooperative agents access the agent's local data.

The prevailing approach to maintaining data correctness and ensuring the isolation property of an execution is serializability. We assume that each database ensures serializability of histories submitted to its site. We guarantee that each agent A ensures: (a) serializability of the local and primitive methods executed on its data and of the methods delegated to it, and (b) that a nondelegated method M of an agent A' appears among the points B_s and B_e of A only if this is permitted by an appropriate breakpoint. Then,

Theorem 1 *The execution of all agents is serializable (global serializability) if there is a serialization order consistent with the serialization orders assumed by all database sites and agents.*

There are various factors that determine the difficulty of maintaining global serializability, including:

1. The existence of autonomous agents beyond the control of the database agents. These autonomous agents are part of the autonomous pre-existing database systems and are completely hidden from the application and database agents.
2. The existence of interdatabase constraints, that is constraints that span more than one database site. Constraints may also exist among local data of different agents or among local data of an agent and a database. In the absence of such constraints, there is no need for global serializability to maintain data correctness. However, some form of serializability may be necessary to ensure some kind of agent isolation.
3. The type of histories produced by the database agents. For example, for strict histories in which the commitment order of methods is the same as their execution order, the enforcement of global serializability is relatively straightforward.

4.2 A decentralized concurrency control algorithm

We shall now outline a timestamp-based method to ensure global serializability. A commutativity relation is defined for each pair of primitive and local methods. Two methods M_i and M_j commute if they do not conflict; that is, if the result of executing M_i after M_j is the same as executing M_j after M_i. These relations are saved in the form of a compatibility matrix. In a closed-nested transaction model, such as that in [13], conflicts among primitive or local methods result in conflicts among the composite methods with which they are invoked. In open-nested transactions [18], there is no such implication. The algorithm presented is for open-nested transactions but can be adapted to the closed-nested situations by using techniques such as hierarchical timestamps [13].

Upon creation, each application agent receives a *timestamp*. The timestamp is defined to be a combination of the value of the clock and the user's *id*. The timestamp of an agent corresponds to its global serialization order. Each application agent serializes all conflicting methods on its local data based on the timestamp order. An operation on its local data issued by another agent is executed only after ensuring that the two agents are allowed to "meet" at a break or relocation point.

We now describe the submission of a primitive method from an application agent to a database agent. The algorithm is a slight variation of [1] for the case of a database interface of primitive methods. To execute a composite method, each application agent can use techniques such as the semantic-based locks of [18].

Each database agent possesses a variable called an *agent ticket* (AT). In the case of autonomous agents, an additional data item per database site is needed. This data item is physically stored in that site and is called a *physical ticket* (PT). Each database agent keeps a list of the timestamps of all primitive methods that have been submitted to the site. A method that does not commute with a submitted method is not allowed to execute concurrently with it; thus, if such a method arrives with timestamp smaller than AT, it is aborted. Two commutable methods can be executed concurrently. Indirect conflicts among commutable methods may arise through conflicts with the operations of autonomous agents; these can be avoided by forcing direct conflicts among them. This is accomplished by having a database agent execute the following code after a commutable method M of an application agent A is received,

```
get(AT)
if (AT > A's timestamp) abort(M)
else
   submit(M) to database
   then in a critical region
      get(AT)
      if (AT > A's timestamp) abort(M)
      else
         write(PT, A's timestamp)
         send prepare-to-commit(M) to A
         if decision taken to commit M
            set (AT, A's timestamp)
            commit(M)
         else abort(M)
```
□

If there are no autonomous agents, then commutable methods are allowed to execute concurrently without any additional control.

Weak consistency. Distributed systems that include wireless connections are characterized by frequent and predictable disconnections. As a consequence, agent methods executed on mobile hosts usually must rely only on locally available and possibly obsolete data. Inconsistency may be handled by introducing weak primitive methods as part of the interface of a mobile database agent. Weak methods are methods that access only local data and thus are only locally consistent [21]. Their commitment is deferred until connection with other sites becomes feasible.

5 Recovery

Failures in a distributed system include communication and site failures. In the agent-based model, the information that must be recoverable, called the *context*, includes the local data of the agent and the environment of its computation. Following an approach similar to that of Contract [25], we assume that the context is saved in a private database for each agent called *context database*. The context database is stored in stable storage and survives failures.

Instead of overwriting the local data of an agent, several versions are kept in the context database. The value last written to a data item by a committed method is called the last committed value for that data item. A committed context database state with respect to an execution is the state in which each data item contains its last committed value. Methods of an agent that are executed concurrently at different databases or cooperating agents take a copy of the appropriate context database.

We rely on system support provided by database agents to handle database site failures that occur during the execution of a primitive database method of an agent. We assume that such failures are either transparent to the application agent or result in aborting the submitted method. When a site failure occurs after a primitive method was successfully executed but before its results become persistent, the method is compensated. The method is then either retried or considered as aborted. Site failures that occur while an agent's local method or a primitive application method of a cooperative agent is executed are handled by using the local context of the agent. A committed context database state is restored. Communication failures are detected by specifying time-outs. In such cases, when an agent is lost, it is reconstructed using its context.

In order to limit the size of the context database as new versions of local data are stored some old versions must be deleted. An entry is removed if the method by which it was written has been aborted or overwritten by a method that has been committed.

The atomicity property. The atomicity property refers to the requirement that either all or none of the methods of an application agent must be committed. Ensuring atomicity becomes complicated when each database or application agent makes independent decisions on whether to commit or abort a primitive method submitted by an application agent A. In this case, A may need either to compensate (semantically undo) a committed primitive method or to retry an aborted primitive method. However, the atomicity property may not be appropriate for many agents, since aborting the whole computation is usually too costly. Alternative characterizations have been proposed; the interested reader is referred to [3] for an overview of some of these proposed approaches.

6 Related Work

Techniques to support an agent-based model for accessing database systems combine concepts from multidatabase concurrency control, advanced transaction models, and control flow management. [4] offers an excellent survey of the problem of concurrency control in multidatabase systems. We outlined the differences between concurrency control in agent-based systems and multidatabases in Section 2.3. Multidatabase systems that offer method-based interfaces include the DOM project [17] and the VODAK system [15]. The majority of multidatabase transaction management systems adopt a centralized approach; [29, 1] are possible exceptions. Many researchers have identified the need for advanced transaction models (see [7] for examples). ACTA [5] provides a framework based on first-order logic for reasoning about extended transaction models. This model is low-level; a higher-level model based on transaction primitives is described in [2]. These two models can be used to express and to implement respectively some of the control flow characteristics of agents. On the basis of extended transaction models, many researchers have defined control flow specifications [23, 10] along the lines of Section 3. Computations as recoverable objects are discussed in [25]. Finally, transaction management in mobile computing environments is addressed in [6, 20, 30]. The agent-based approach is in compliance with these models.

7 Conclusions

We are currently witnessing the emergence of a new model of distributed computing which is based on the presence of multiple, autonomous, and persistent communicating agents. This model is tailored to the computing requirements of a future that includes wireless connections and numerous distributed sites and for which the onus of coordination must be shifted from users to application programs. In this paper, we have introduced a framework for agent-based access to heterogeneous mobile databases. We have identified the implications of this new computational paradigm on concurrency control and recovery and noted its differences from traditional models. Although much remains to be investigated on this fascinating topic, we have presented an initial outline of its features and pointed to several potential solution to emerging problems.

References

[1] R. K. Batra, M. Rusinkiewics, and D Georgakopoulos. A Decentralized Deadlock-free Concurrency Control Method for Multidatabase Transactions. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, June 1992.

[2] A. Biliris, S. Dar, N. Gehani, H. V. Jagadish, and K. Ramamritham. ASSET: A System for Supporting Extended Transactions. In *Proceedings of the 1994 SIGMOD Conference*, pages 44–54, May 1994.

[3] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Transaction Management in a Failure-Prone Multidatabase System Environment. *VLDB Journal*, 1(1):1–39, 1992.

[4] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–239, 1992.

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

[6] P. K. Chrysanthis. Transaction Processing in Mobile Computing Environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77–83, October 1993.

[7] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[8] A. A. Farrag and M. T. Ozsu. Using Semantic Knowledge of Transactions to Increase Concurrency. *ACM Transactions on Database Systems*, 14(4):503–525, December 1989.

[9] T. Finin, D. McKay, R. Fritzson, and R. McEntire. KQML: An Information and Knowledge Exchange Protocol. In Kazuhiro Fuchi and Toshio Yokoi, editors, *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press, 1994.

[10] D. Georgakopoulos, M. Hornick D., and A. Sheth. Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases, An International Journal*, 3, 1995.

[11] R. Goodwin. Formalizing Properties of Agents. Technical Report CMU-CS-93-159, Carnegie Mellon University, School of Computer Science, 1993.

[12] I. Greif. Desktop Agents in Group-Enabled Products. In [22], pages 100–105.

[13] T. Hadjilacos and V. Hadjilacos. Transaction Synchronization in Object Bases. *Journal of Computer and System Sciences*, 43:2–24, 1991.

[14] T. Imielinksi and B. R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. *Communications of the ACM*, 37(10), October 1994.

[15] W. Klas, P. Fankhauser, P. Muth, T. Rakow, and E. J. Neuhold. Database Integration using the Open Object-Oriented Database System VODAK. In Ahmed Elmagarmid and Omran Bukhres, editors, *Object-Oriented Multidatabases*. Prentice Hall, 1995.

[16] P. Maes. Agents that Reduce Work and Information Overload. In [22] pages 31–40.

[17] F. Manola, S. Heiler, D. Georgakopoulos, M. Hornick, and M. Brodie. Distributed Object Management. *International Journal of Intelligent and Cooperative Information Systems*, 1(1), June 1992.

[18] P. Muth, T. C. Rakow, G. Weikum, P. Brossler, and C. Hasse. Semantic Concurrency Control in Object-Oriented Database Systems. In *Proceedings of the 9th International Conference on Data Engineering*, pages 233–242, 1993.

[19] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In *Proceedings of the Third International Conference on Information and Knowledge Management*, pages 371–378, November 1994.

[20] E. Pitoura and B. Bhargava. Revising Transaction Concepts for Mobile Environments. In *Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.

[21] E. Pitoura and B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, May 1995.

[22] D. Riecken, editor. *Special Issue on Intelligent Agents*. Communications of the ACM, 37(7), July 1994.

[23] M. Rusinkiewicz and A. Sheth. Specification and Execution of Transaction Workflows. In W. Kim, editor, *Modern Database Systems: The Object Model and Beyond*, pages 592–620. Addison Wesley, 1995.

[24] H-J. Scheck, G. Weikum, and W. Schaad. A Multilevel Transaction Approach to Federated DBMS Transaction Management. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 280–287, April 1991.

[25] H. Wachter and A. Reuter. The ConTract Model. In [7], pages 239–264.

[26] J. E. White. Mobile Agents Make a Network an Open Platform for Third-Party Developers. *IEEE Computer, Hot Topics*, pages 89–90, November 1994.

[27] J. E. White. Telescript Technology: Scenes from the Electronic Marketplace. General Magic White Paper, 1994.

[28] J. E. White. Telescript Technology: The Foundation for the Electronic Marketplace. General Magic White Paper, 1994.

[29] A. Wolski and J. Veijalainen. Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase. In *Proceedings of the Parbase90 Conference*, February 1990.

[30] L. H. Yeo and A. Zaslavsky. Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, June 1994.