Transactional Agent Model for Distributed Objects

Tomoaki Kaneda, Youhei Tanaka, Tomoya Enokido, and Makoto Takizawa

Dept. of Computers and Systems Engineering
Tokyo Denki University, Japan
{kaneda, youhei, eno, taki}@takilab.k.dendai.ac.jp

Abstract

A transactional agent is a mobile agent which manipulates objects in one or more than one computer by autonomously ?nding a way to visit the computers so as to satisfy some commitment condition like atomicity in presence of faults of computers. A surrogate can recreate a new incarnation of the agent if the agent itself is faulty. If a destination computer is faulty, the transactional agent ?nds another operational computer to visit. Objects in each computer are locally manipulated by an agent for the computer. After visiting computers, a transactional agent makes a destination on commitment. In addition, objects obtained from a computer have to be delivered to other computers where the transactional agent is performed. We discuss a model of transactional agent and logistics on how to deliver classes for manipulating objects and derived objects to computers where the routing agent to visit. We discuss how to implement a transactional agent on database servers and evaluate the transactional agents.

分散オブジェクトを操作するためのトランザクショナルエージェントモデル

兼田 知明 田中 洋平 榎戸 智也 滝沢 誠
東京電機大学大学院 理工学研究科 情報システム工学専攻

トランザクショナルエージェントは、複数の障害計算機上にあるオブジェクトを計算機障害に対処してから操作する移動エージェントである。計算機上で操作を終了したならば、代理エージェントを生成し、次の計算機に移動する。移動先計算機が障害していたならば、エージェントは他の計算機を探し移動する。また、エージェントが居る計算機がに障害した場合、代理エージェントがエージェントを再生成する。本論文では、トランザクショナルエージェントのフォールトトレランス技術の実装について論じる。

1 Introduction

Various types of objects are distributed in computers. A transaction manipulates multiple objects distributed in computers. A transaction is modeled to be a sequence of methods which satis?es the ACID (atomicity, consistency, isolation, and durability) properties [2]. Huge number and various types of peer computers are interconnected in peer-to-peer (P2P) networks [3]. A mobile agent can autonomously escape from faulty computers and ?nd another operational computers. Mobile agents [5, 14, 22] are programs which move to remote computers locally manipulate objects. We discuss how to realize distributed transactions in mobile agents. A transaction with the ACID properties [2, 8, 9] initiates a subtransaction on each database server, which is realized in mobile agents [9,13,17]. In this paper, a *transactional* agent is a mobile agent which autonomously moves around computers [6].

In addition, we discuss how to reduce communication overheads to transmit classes and objects to a transactional agent in another computer.

After manipulating all or some objects in computers, an agent makes a decision on *commit* or *abort*. In addition, an agent negotiates with another agent which would like to manipulate a same object in a con?icting manner. Through the negotiation, each agent autonomously makes a decision on whether the agent holds or releases the objects [6, 15].

If an agent leaves a computer, objects locked by the agent are automatically released by the after manipulating objects. Hence, an agent creates a *surrogate agent* on leaving a computer so that an agent can abort even after the agent leaves the computer. A surrogate agent still holds locks on objects in a computer on behalf of the agent after the agent leaves the computer.

In this paper, we assume computers may stop by fault. A transactional agent autonomously ?nds another destination computer if a computer where the agent to move is faulty. In addition, an agent and a surrogate are faulty due to the fault of a current computer where the agent and surrogate exist. Some surrogate of the agent which exists on another computer recreates the agent. The new incarnation starts as an agent. Similarly, when a surrogate may be faulty, another surrogate takes a way to recover from the fault.

In section 2, we present a system model. In section 3, we discuss transactional agents. In section 4, we discuss fault-tolerant mechanism of the transactional agent. In section 5, we discuss implementation of transactional agents. In section 6, we evaluate the transactional agent through experiments.

2 System Model

2.1 Objects

A system is composed of *computers* interconnected in reliable networks. Each computer is equipped with a class base (CB) where classes are stored and an *object base* (OB) which is a collection of persistent objects. A *class* is composed of attributes and methods. An object is an instantiation of a class which is an encapsulation of data and methods for manipulating the data. If result obtained by performing a pair of methods op_1 and op_2 on an object depends on the computation order of op_1 and op_2, op_1 and op_2 *con?ict* with one another.

A transaction is modeled to be a sequence of methods, which satis?es the ACID properties [4]. Especially, a transaction can commit only if all the objects are successfully manipulated. A transactional aborts if at least one object can be successfully manipulated. The two-phase commitment protocol [4, 15] is used to realize the atomic commitment. If a method op_1 from a transaction T_1 is performed before a method op_2 from another transaction T_2 which con?icts with op_1, every method op_3 from T_1 has to be performed before every method op_4 from T_2 con?icting with the method op_3. This is the *serializability* property [2, 4]. There are locking protocols [2,4,7] and timestamp ordering protocols [2] to realize the serializability of transactions.

In the locking protocol, a transaction locks an object before manipulating the object. Each computer supports an agent with an *isolation* level [11] which shows when the agent releases objects. In the strict two-phase locking protocol [2], neither dirty read nor cascading abort occur since all the locks are not released before commit or abort.

2.2 Mobile agents

A *mobile agent* is a program which moves around computers and locally manipulates objects in each computer [19, 22]. Mobile agent systems like Aglets [5], Telescript [22], and AgentSpace [14] are so far discussed. A mobile agent is composed of classes. A home computer *home*(c) of a class c is a computer where c is stored. A home computer *home*(A) of a mobile agent A is a home computer of the class of the agent A.

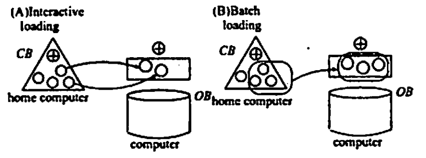

Figure 1. Ways to load classes.

An agent invokes a method *op* in a class. Then, a method of another class is invoked in *op*. The class is required to be loaded to the current computer of the agent. There are two ways to load classes of an agent A from a home computer [Figure 1]. In an *interactive* way, a class c is loaded from a home computer *home*(c) each time a method of c is invoked by the agent A. Another way is a *batch* one where a collection of multiple classes are loaded. If an agent invokes more number of methods, the interaction time between the current computer and the home computer can be more reduced than the interactive way.

3 Transactional Agents

3.1 Model of transactional agent

We discuss how to realize a transaction which manipulates more than one object on computers with some commitment condition in a mobile agent. A *transactional agent* is a mobile agent which satis?es the following properties:

1. autonomously decides on which computer to visit.
2. manipulates objects on one or more than one computer.
3. commits only if some commitment condition intrinsic to the agent is satis?ed, otherwise aborts.

For simplicity, a term *agent* means a transactional agent in this paper. An agent A is composed of three subagents: *routing* agent *RC(A)*, *commitment* agent $CC(A)$, and *manipulation* agent $MC(A,\ D_1)$, ..., $MC(A,\ D_n)$, where D_i stands for a target computer of the agent A. That is, $A = \langle RC(A), CC(A), MC(A)\rangle$ where $MC(A) = \{MC(A, D_1), ..., MC(A, D_n)\}$. Here, let *Dom*(A) be a set of *target* computers D_1, ..., D_n of an agent A. First, an agent A on a current computer has to move to a computer in the target domain *Dom(A)*. A computer D_j to which an agent A on a current computer D_i moves is referred to as *destination* computer of A on D_i. An agent A has to autonomously make a decision on which computer in the target domain *Dom*(A) to visit. In $RC(A)$, a destination computer is selected. Then, the agent A moves to the destination computer. Here, an agent ?rst ?nds a candidate set of possible destination computers which have objects to be manipulated after manipulating objects in the current computer. Then, the agent selects one target computer in the candidate computers and moves to the computer.

Secondly, an agent A manipulates objects in a current computer D. The agent A loads a manipulation agent $MC(A,\ D)$ for manipulating objects from the home computer *home*($MC(A, D)$).

Lastly, an agent makes a decision on whether the agent can commit or abort after visiting target computers.

3.2 Routing agent

The agent A visits a computer D_j. Here, objects in D_j are manipulated through the manipulation agent $MC(A, D_j)$ by using objects which are obtained in other computers. Thus, the manipulation classes in an agent are related with input-output relation. Objects which are inputs and outputs are referred to as *intermediate* objects. Here, $D_i \overset{x}{\Rightarrow} D_j$ shows that the manipulation agent $MC(A, D_i)$ outputs an intermediate object x and $MC(A, D_j)$ in D_j uses x as an input. If $D_i \overset{x}{\Rightarrow} D_j$ for an agent A, the agent A has to visit D_i before D_j. The input-output relation is shown in an input-output graph as shown in Figure 2.

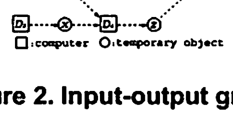

Figure 2. Input-output graph

There are two types of nodes, *computer* and *object*. A directed edge from a computer D_i to an object x shows that the manipulation agent $MC(A, D_i)$ outputs x. A directed edge from x to a computer D_i indicates that $MC(A, D_j)$ outputs x.

From the input-output graph, an agent A decides in which order the agent visits. A directed acyclic graph (DAG) $Map(A)$ named a *map* is created from the input-output graph [Figure 3]. In a map, a node D shows a computer D with a manipulation agent $MC(A, D_i)$. A directed edge $D_1 \rightarrow D_2$ shows that an object base (OB_2) in a computer D_2 is required to be manipulated by $MC(A, D_2)$ after $MC(A, D_1)$ in a computer D_1. $D_1 \rightarrow^* D_2$ if and only if (*iff*) $D_1 \rightarrow D_2$ or $D_1 \rightarrow D_3 \rightarrow^* D_2$ for some computer D_3. D_1 and D_2 are independent ($D_1 \parallel D_2$) if neither $D_1 \rightarrow^* D_2$ nor $D_2 \rightarrow^* D_1$. Here, an agent A can in parallel visit D_1 and D_2. Each node D_1 is assigned $MC(A, D_1)$ through which objects in D_1 are manipulated. Figure 3 shows an example of a map $Map(A)$.

Intermediate objects in $Out(A, D_i)$ obtained by manipulating target objects in the computer D_i are used to manipulate objects in another computer D_j. There are following ways to bring an intermediate object x obtained in D_i to D_j:

1. An agent A carries x to D_j.
2. x is transfered from D_i to the computer before the agent A arrives at D_i.
3. x is transfered from D_i to D_j on request of the agent A on D_j.

$$\begin{tikzcd}
D_1 \arrow[r] \arrow[dr] & D_3 \arrow[r] & D_5 \\
D_2 \arrow[r] & D_4 \arrow[ur] &
\end{tikzcd}$$

Figure 3. Map.

A routing agent $RC(A)$ with a map $Map(A)$ is moving around computers [Figure 3]. A collection I of computers which do not have any in-coming edge are found in $Map(A)$. One node D_i is selected in the set I so as to satisfy some condition. The agent A moves to the computer D_i. Here, a manipulation agent $MC(A, D_i)$ is loaded to D_i from the home computer. After manipulating objects in D_i, D_i is removed from $Map(A)$. Another destination node D_j is selected as presented here. Then, the agent A moves to D_j with $Map(A)$.

Then the agent A is started on the computer. The computer is referred to as *base computer* of the agent A. An agent A leaves the base computer for a computer D_i to manipulate objects. Here, D_i is a *current* computer of the agent A, denoted *current*(A). If the agent A invokes a method t of a class c, c is searched in the network as follows:

1. The cache of the current computer is ?rst searched for c. If c is found, t is invoked.
2. If not, the class base (CB_i) of D_i is locally searched. If c is found in CB_i, c is taken to invoke t.
3. Otherwise, c is transferred from the home computer *home*(c) into D_i.

A history $H(A)$ of an agent A shows a sequence of computers which the agent A has visited. On leaving a computer, the computer is recorded in the history $H(A)$.

3.3 Manipulation agent

A manipulation agent is composed of application-speci?c classes and library classes.

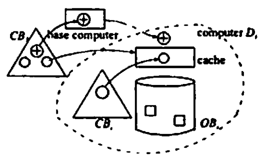

Figure 4. Home computer.

If an agent A invokes a method t of a class c in a computer, the class c is loaded from the home computer *home*(c) to the cache in the computer [Figure 4]. Then, the method t of c is performed in the computer. If another agent B invokes t of c in the computer, c in the cache is used to invoke t without loading c. Thus, if classes are cashed in a computer, methods in the classes are locally invoked in the computer without any communication.

3.4 Commitment agent

A scope $Scp(A)$ of an agent A means a set of computers where the agent A possibly manipulates objects. If an agent A ?nishes manipulating objects in each computer, the following *commitment* condition of the agent A is checked by the commitment agent $CC(A)$:

1. *Atomic commitment*: an agent is successfully performed on all the computers in $Scp(A)$.
2. *Majority commitment*: an agent is successfully performed on more than half of the computers in $Scp(A)$.
3. *At-least-one commitment*: an agent is successfully performed on at least one computer in $Scp(A)$.
4. $\binom{n}{r}$ *commitment*: an agent is successfully performed on more than r out of n computers ($r \leq n$) in $Scp(A)$.
5. *Application speci?c commitment*: condition speci?ed by application is satis?ed.

A commitment condition is speci?ed for each agent A by an application. The commitment condition is checked by a commitment agent $CC(A)$ of the agent A. There are still discussions on when the commitment condition of an agent A can be checked while the agent A is moving around computers. Let $H(A)$ be a *history* of an agent A.

3.5 Resolution of con?iction

Suppose an agent A moves to a computer D_j from another computer D_i. The agnet A cannot be performed on D_j if there is an agent or surrogate B con?icting with A. Here, the agent A can take one of the following ways:

1. *Wait*: The agent A in D_i *waits* until the agent A can land at D_j.
2. *Escape*: The agent A *finds another* computer D_k which has objects to be possibly manipulated before D_j.

3. *Negotiate*: The agent A *negotiates* with the agent B in D_j. After the negotiation, the agent A takes over B.
4. *Abort*: The agent A *aborts*.

If the agent B waits for release of an object held by the agent A, a pair of the agent A and B are deadlocked. If the timer expires, the agent A takes a following way:

1. The agent A retreats to a computer D_j in the history $H(A)$. All the surrogates of A which have been performed after performed on D_j are aborted.
2. Then, the surrogate agent A_j on D_j recreates a new incarnation of the agent A. The agent A ?nds another destination computer D_h [Figure 5].

The surrogate A_j to which the agent A retreats plays a role of checkpoint [12] of A. Differently from traditional checkpoints [12], the agent A retreating to some surrogate A_j autonomously ?nds an operational computer which may be different from one which the agent A has visited.

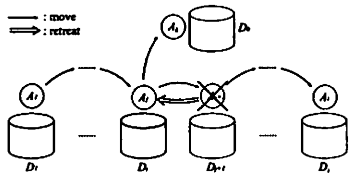

Figure 5. Retreatment.

Suppose a surrogate agent B holds an object in a computer D_j. An agent A would like to manipulate the object but con?icts with the surrogate agent B in D_j. The surrogate agent B makes a following decision depending on the commitment conditions of B:

1. *Atomic commitment*: The agent A waits until the surrogate B ?nishes.
2. *At-least-one commitment*: If the surrogate B knows at least one sibling surrogate of B is committable, B releases the object and aborts. B informs the other sibling surrogates of this abort.
3. *Majority commitment*: If the surrogate B knows more than half of the sibling surrogates are committable, B releases the object and aborts. B informs the other surrogates of this abort.
4. $\binom{n}{r}$ *commitment*: If the surrogate B knows more than or equal to r sibling surrogate agents are committable, B releases the object and aborts.

4 Fault-Tolerance

4.1 Forwarding and backwarding

Computers may be faulty in networks. We assume computers may stop by fault. An agent is faulty only if a current computer of the agent is faulty. Suppose a transactional agent A ?nishes manipulating objects on a computer D_i. The agent A selects one computer D_j from the map $Map(A)$. The agent A detects by timeout mechanism that the computer D_j is faulty if the agent A does not receive any response from D_j. If the computer D_j is operational, the agent A leaves D_i for D_j. Here, suppose D_j is faulty. The agent A tries to ?nd another destination computer D_k in the map $Map(A)$. If found, the agent A moves to the computer D_k if D_k is operational, as presented here.

If the transactional agent A cannot ?nd another destination computer in the map $Map(A)$, the agent A backs to the preceding D_k, i.e. the agent A has come to the current computer D_i from D_k. The map $Map(A)$ is restored to one when the agent A had left the computer D_k. The node D_i is removed from the map $Map(A)$. Then, the agent in D_k tries to ?nd another destination computer in the map $Map(A)$.

4.2 Fault of agent and surrogate

A transactional agent A leaves its surrogate agent A_i on a computer D_i. The surrogate agent A_i holds objects after the agent A leaves the computer D_i. The surrogate agent A_i releases objects on before the agent A terminates depending on the solution condition of the agent A.

A transactional agent A and surrogate agent A_i are faulty if a current computer when A and A_i exist is faulty. First, let us consider case an agent A is faulty, on a computer D_i. Suppose that the agent A comes from another computer D_j named predecessor of D_i to the computer D_i. The surrogate A_j on the computer D_j detects that the agent A is faulty on the computer D_i. Here, the surrogate agent A_i recreates a new incarnation of the agent A. The agent A takes another destination computer D_k in the map $Map(A)$. If found, the agent A one of the following strategies:

1. waits until the computer D_i is recovered.
2. backs to the precedent computer from D_j.

A surrogate A_i on a computer D_i may be faulty as well. The precedent surrogate A_j on computer D_j detects the fault of the surrogate agent A_i.

5 Implementation of Transactional Agent

5.1 Surrogate agents

When an agent A leaves a computer D_i, a surrogate A_i still holds objects in D_i which are manipulated by the agent A. Surrogate agents commit or abort according to the decision of the agent. Surrogates of an agent A are referred to as *sibling* surrogates of A. The agent A creates A_j and moves to another computer D_k. Here, A_i and A_k are most preceding and most succeeding agents of A_j. Thus, when the agent A ?nishes visiting all the computers, some surrogate agent may not exist due to the fault and abortion in negotiation with other agents. The agent A starts the negotiation procedure with its surrogates $A_1, \ldots, A_m$. If a commitment condition on $A_1, \ldots, A_m$ is satis?ed by the commitment agent $CC(A)$, the agent A commits. On the other hand, the commitment condition is not satis?ed, the agent A aborts.

Suppose an agent A moves to a computer D_j from another computer D_i. The agent A cannot be performed on D_j if there is another agent or surrogate agent B con?icting with the agent A. The authors [6, 15] discuss how to resolve the con?iction though negotiations among agents.

5.2 Implementation

We discuss how to realize agents. An agent is implemented in Aglets and composed of a *routing*, *manipulation*, and *commitment agents*.

A routing agent $RC(A)$ is transfered from a computer to another computer. When routing agent $RC(A)$ arrives at a computer D_i, a manipulation agent $MC(A, D_i)$is created by $RC(A)$.

An object base (OB) is realized in a relational database system. An agent manipulates *table* objects by issuing SQL commands in a current computer D_i. The computation of each agent A on D_i is realized as a local *transaction* on a database system in D_i. If the agent A leaves D_i, the transaction for the agent A commits or aborts. Even if the agent A leaves D_i, objects manipulated are required to be still held by the agent A because the agent A may abort after leaving D_i. Therefore, a *surrogate agent* is newly introduced as discussed in the preceding sub section. The surrogate agent is composed of $MC(A, D_i)$ and an *object agent* OBA. Each object agent (OBA) behaves as follows:

1. On arrival at a computer D_i, the routing agent $RC(A)$ initiates a manipulation agent $MC(A,D_i)$ and an object agent OBA_i on D_i. OBA_i initiates a transaction on an object base OB_i.
2. If $MC(A,D_i)$ issues a method for manipulating objects.
3. If the agent A ?nishes, the agent A leaves D_i.
4. OBA_i commits or aborts if the agent A sends *commit* and *abort* requests to A_i.

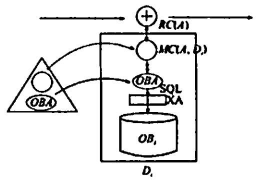

Figure 6. Object agent (OBA).

An object agent OBA is independent of types of database systems like Oracle and Sybase. OBA class can be loaded to a computer with any type of database system. Each time an agent arrives at a computer, an instance OBA_i of OBA class is loaded from the home computer of the agent A into a computer D_i. If an agent comes to D_i from another home computer, OBA class is loaded to D_i from the home computer. Thus, OBA instances are accumulated in the cache. In order to resolve this problem, OBA class is loaded as follows:

1. There is one home computer *home*(OBA).
2. If the OBA class is not cached in the current computer, the OBA class is loaded from *home*(OBA).
3. If the OBA class could not be loaded from *home*(OBA).

The routing agent $RC(A)$ leaves a computer D_i if the manipulation agent $MC(A, D_i)$?nishes manipulating objects in D_i. $MC(A, D_i)$ recreates a new incarnation of $RC(A)$ if the agent A stops due to the computer fault.

An agent A can commit if all or some of the surrogates commit depending on the commitment condition. Communication among an agent and its surrogate agents is realized by using the XA interface [23] which supports the two-phase commitment protocol [16] [Figure 6]. Each surrogate agent issues a *prepare* request to a computer on receipt of a *prepare* message from the agent A. If *prepare* is successfully performed, the surrogate agent sends a *prepared* message to the agent A. Here, the surrogate agent is *committable*. The agent A receives responses from the surrogate agents after sending *prepare* to the surrogates. On receipt of the responses from surrogate agents, the agent A makes a decision on *commit* or *abort* based on the commitment condition.

Next, we discuss how to support robustness against faults of computers. Suppose a surrogate agent A_i of an agent A stops after sending *prepared*. Here, the surrogate agent is committable. On recovery of the committable surrogate, the surrogate agent unilaterly commits if the surrogate agent is committable in the at-least-one commitment condition. In the atomic condition, A_i asks the other surrogates if they had committed. Suppose A_i is abortable, i.e. faulty before receiving *prepared*. On recovery, A_i unilaterly aborts.

6 Evaluation

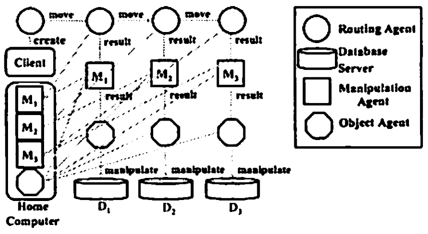

Figure 7. Evaluation model

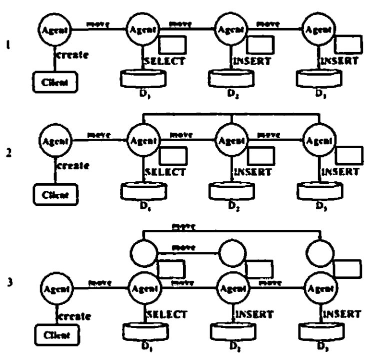

Figure 8. Evaluation model

We evaluate the agent which is implemented in Aglets. In the evaluation, There are three server computers D_1, D_2, and D_3. An agent is created in a computer C. There is another computer H, which is a home computer of the manipulation agents and object agent. D_1, D_2, and D_3 are realized in personal computers (Pentium 3) with Oracle database systems. The computers are interconnected in the 1Gbps Ethernet.

First, an agent A is initiated in C. The agent A ?nds in which order D_1, D_2, and D_3 to be visited as discussed in this paper. Here, the agent A visits D_1, D_2, and D_3 in this order as shown in Figure 7. On arrival of the agent A on D_i, the manipulation agent M_i and object agent OBA_i are loaded to D_i [Figure 7].

In this evaluation, there are following types of agents A and B:

A. The manipulation agents M_1 on D_1 derives intermediate object I from the object base. The object base in D_2 and D_3 are updated by using I.

B. M_1 and M_2 derive objects from the object bases in D_1 and D_2 to intermediate objects I_1 and I_2. Then, the object base in D_3 is manipulated by using I_1 and I_2.

There are three ways to deliver intermediate objects derived in a computer to another computer where the objects are used as discussed in section 3 [Figure 8].

1. The agent A carries intermediate objects.
2. After the agent A arrives at D_j, the agent A requests the computer D_i to send the intermediate objects.
3. The agent A transfers the intermediate object I obtained to D_j before leaving D_i.

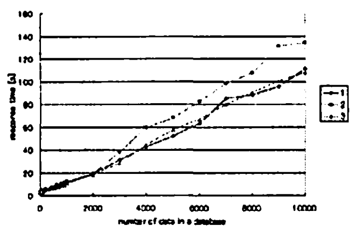

Figure 9. Response A

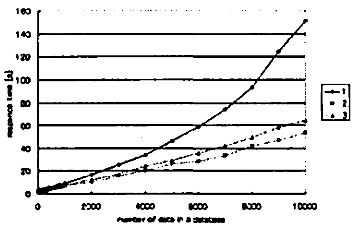

Figure 10. Response B

The total response time of an agent is measured for number of intermediate objects. Figures 9 and 10 show the response time for the types of agents A and B. The second and third ways to deliver intermediate objects to destination computers imply shorter responce time than the ?rst way.

7 Concluding Remarks

The authors discussed a transactional agent model to manipulate objects in multiple computers with types of commitment constraints in presence of computer faults. A transactional agent autonomausly ?nd a distination computer to visit, moves to a computer, and then locally manipulates objects. We discussed how to implement transactional agents in Aglets and Oracle. We evaluated the transactional agent in terms of response time.

References

[1] American National Standards Institute. *The Database Language SQL*, 1986.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[3] L. Gong. *JXTA: A Network Programming Environment*, pages 88–95. IEEE Internet Computing,, 2001.

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

[5] IBM Corporation. *Aglets Software Development Kit Home*. http://www.trl.ibm.com/aglets/.

[6] T. Komiya, T. Enokido, and M. Takizawa. Mobile agent model for transaction processing on distributed objects. *Information Sciences*, 154:23–38, 2003.

[7] F. H. Korth. Locking Primitives in a Database System. *Journal of ACM*, 30(1):55–79, 1989.

[8] N. A. Lynch, M. Merritt, A. F. W. Weihl, and R. R. Yager. *Atomic Transactions*. Morgan Kaufmann, 1994.

[9] K. Nagi. *Transactional Agents : Towards a Robust Multi-Agent System*. Springer-Verlag, 2001.

[10] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf. *Coordination of Internet Agents*. Springer-Verlag, 2001.

[11] Oracle Corporation. *Oracle8i Concepts Vol. 1 Release 8.1.5*, 1999.

[12] R. S. Pamula and P. K. Srimani. Checkpointing Strategies for Database Systems. *Proc. of the 15th Annual Conf. on Computer Science, IEEE Computer Society*, pages 88–97, 1987.

[13] S. Pleisch. *State of the Art of Mobile Agent Computing - Security, Fault Tolerance, and Transaction Support*. IBM Corporation, 1999.

[14] I. Satoh. A Mobile Agent-based Framework for Active Networks,. *Proc. of IEEE Systems, Man, and Cybernetics Conference (SMC'99)*, pages 71–76, 1999.

[15] M. Shiraishi, T. Enokido, and M. Takizawa. Fault-Tolerant Mobile Agent in Distributed Objects Systems. *Proc. of the 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, pages 145–151, 2003.

[16] D. Skeen. Nonblocking Commitment Protocols. *Proc. of ACM SIGMOD*, pages 133–147, 1982.

[17] A. D. Stefano, L. L. Bello, and C. Santoro. A Distributed Heterogeneous Database System Based on Mobile Agents. *Proc. of the 7th Workshop on Enabling Technologies (WETICE'98), IEEE Computer Society*, pages 223–229, 1998.

[18] Sun Microsystems Inc. *JDBC Data Access API*. http://java.sun.com/products/jdbc/.

[19] Sun Microsystems Inc. *The Source for Java (TM) Technology*. http://java.sun.com/.

[20] Sun Microsystems Inc. *Trail: JAR ?les*.

[21] Sybase Inc. *SYBASE SQL Server*. http://www.sybase.com/.

[22] J. E. White. *Telescript Technology : The Foundation for the Electronic Marketplace*. General Magic Inc., 1994.

[23] X/Open Company Ltd. *X/Open CAE Speci?cation Distributed Transaction Processing: The XA Speci?cation*, 1991.

[24] S. Young and D. Aitel. *The Hacker's Handbook: The Strategy Behind Breaking into and Defending Networks*. Auerbach Publications, 2003.