Transactional Agent Model for Fault-Tolerant Object Systems

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 multiple computers by autonomously finding a way to visit the computers. The transactional agent commits only if its commitment condition like atomicity is satisfied in presence of faults of computers. On leaving a computer, an agent creates a surrogate agent which holds objects manipulated. 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 finds another operational computer to visit. After visiting computers, a transactional agent makes a destination on commitment according to its commitment condition. We discuss design and implementation of the transactional agent which is tolerant of computer faults.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Transaction processing

General Terms

Algorithms, Reliability

Keywords

Mobile agent, Transaction, Fault-Tolerant

1. INTRODUCTION

A transaction manipulates multiple objects distributed in computers through methods. Objects are encapsulations of data and methods for manipulating the data. A transaction is modeled to be a sequence of methods which satisfies the ACID (atomicity, consistency, isolation, and durability) properties [8, 9]. Huge number and various types of peer computers are interconnected in peer-to-peer (P2P) networks [3]. Personal computers easily get faulty not only by crash but also by hackers and intrusions. A mobile agent can autonomously escape from faulty computers by moving to another operational computer. Mobile agents [5, 19] are programs which move to remote computers and then locally manipulate objects on the computers.

An ACID transaction initiates a subtransaction on each database server, which is realized in mobile agents [16, 9, 13]. In this paper, a *transactional agent* is a mobile agent which autonomously decides in which order the agent visits computers in presence of computer faults, and locally manipulates objects in a current computer with not only atomicity but also other types of commitment conditions like at-least-one condition [6]. After manipulating all or some objects in computers, an agent makes a decision on *commit* or *abort*. For example, an agent atomically commits only if all objects in the computers are successfully manipulated [4]. An agent commits if objects in at least one of the computers are successfully manipulated. In addition, an agent negotiates with another agent which would like to manipulate a same object in a conflicting manner. Through the negotiation, each agent autonomously makes a decision on whether the agent holds or releases the objects [6, 14].

If an agent leaves a computer, objects locked by the agent are automatically released. Hence, once leaving a computer, an agent cannot abort. An agent creates a *surrogate agent* on leaving a computer. A surrogate agent still holds locks on objects in a computer on behalf of the agent after the agent leaves.

A transactional agent autonomously finds another destination computer if a destination computer is faulty. An agent and surrogate are faulty if the current computer is faulty. Some surrogate of the agent which exists on another computer recreates a new incarnation of the agent. Similarly, if a surrogate may be faulty, another surrogate detects the fault and takes a way to recover from the fault. For example, if an agent takes an at least one commitment condition, a fault of the surrogate can be neglected as long as at-least-one surrogate is operational.

In section 2, we present a system model. In section 3, we discuss transactional agents. In section 4, we discuss fault-tolerant mechanism. In sections 5 and 6, we discuss implementation and evaluation of transactional agents.

2. SYSTEM MODEL

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. If result obtained by performing a pair of methods op_1 and op_2 on an object depends on the computation order, op_1 and op_2 *conflict* with one another. For example, a pair of methods *increment* and *reset* conflict on a *counter* object. On the other hand, *increment* and *decrement* do not conflict, i.e. are *compatible*.

A transaction is modeled to be a sequence of methods, which satisfies the ACID properties [4]. Especially, a transaction can commit only if all the objects are successfully manipulated. If a method op_1 from a transaction T_1 is performed before a method op_2 from another transaction T_2 which conflicts with op_1, every method op_3 from T_1 has to be performed before every method op_4 from T_2 conflicting with the method op_3. This is the *serializability* property [2, 4]. Locking protocols [2, 4, 7] are used to realize the serializability of transactions. Here, a transaction locks an object before manipulating the object.

A *mobile agent* is a program which moves around computers and locally manipulates objects in each computer [5, 18, 19]. A mobile agent is composed of classes. A home computer $home(c)$ of a class c is a computer where the class c is stored. For example, each class c is identified by a pair of IP address of a home computer $home(c)$ and a local path to the directory where the class c is stored. A home computer $home(A)$ of a mobile agent A is a home computer of the class of the agent A.

3. TRANSACTIONAL AGENTS

3.1 Model of transactional agent

A *transactional agent* is a mobile agent which satisfies the following properties:

1. autonomously decides on which computer to visit.
2. manipulates objects on multiple computer.
3. commits only if some commitment condition of the agent is satisfied, otherwise aborts.

For simplicity, a term *agent* means a transactional agent in this paper. *Target* objects are objects to be manipulated by an agent. $Target$ computers have the target objects. An agent A is composed of *routing* $RC(A)$, *commitment* $CC(A)$, and *manipulation* agents $MC(A, D_1)$, ..., $MC(A, D_n)$, where D_i stands for a target computer of the agent A. 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 $Dom(A)$. A computer D_j to which an agent A on D_i moves is a *destination* computer. An agent A has to autonomously make a decision on which computer to visit. In the routing agent $RC(A)$, a destination computer is selected. Then, the agent A moves to the destination computer. Here, an agent first finds a candidate set of possible destination computers. Then, the agent selects one target computer in the candidate computers and moves to the computer.

Secondly, a transactional agent A manipulates objects in a current computer D. The agent A initiates a manipulation agent $MC(A, D)$ for manipulating objects in the current computer D from the home computer. If an object base is realized in a relational database system [11], objects are manipulated by issuing SQL commands in $MC(A, D)$.

Lastly, a transactional agent makes a decision on whether the agent can commit or abort after visiting target computers. A traditional transaction [2] *atomically* commits only if objects in all the target computers are successfully manipulated. In this paper, we consider other types of commitment conditions [6]. For example, in the at-least-one commitment, a transaction can commit only if objects in at least one target computer are successfully manipulated.

3.2 Routing agent

A transactional agent A locally manipulates objects in a computer D_i through the manipulation agent $MC(A, D_i)$ and then outputs intermediate objects $OUT(A, D_i)$. In the meanwhile, the agent A visits another computer D_j. Here, objects in D_j are manipulated through the manipulation agent $MC(A, D_j)$ by using the intermediate objects $In(A, D_j)$ ($=OUT(A,\ D_i)$). Thus, the manipulation classes are related with input-output relation. Here, $D_i \stackrel{x}{\Rightarrow} D_j$ shows that the manipulation agent $MC(A, D_i)$ outputs an intermediate object x which is used by $MC(A, D_j)$. If $D_i \stackrel{x}{\Rightarrow} D_j$, the agent A has to visit D_i before D_j and the intermediate object x has to be delivered to D_j. The input-output relation is shown in an input-output graph as shown in Figure 1.

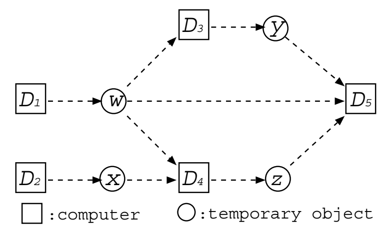

Figure 1: Input-output graph

There are *computer* and *object* nodes. Directed edges $D_i \dashrightarrow x$ and $x \dashrightarrow D_i$ show that the manipulation agent $MC(A, D_i)$ outputs and inputs an object x, respectively. In Figure 1, the agent A outputs an intermediate object w in D_i. The agent A uses x in D_3, D_4, and D_5. This means the agent A is required to visit D_3, D_4, and D_5 after D_1.

From the input-output graph, a transactional 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 2]. Here, a node D shows a computer D with a manipulation agent $MC(A,\ D)$. A directed edge $D_1 \rightarrow D_2$ a computer D_2 is required to be manipulated after 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, a transactional agent A can visit the computers D_1 and D_2 in any order and can in parallel visit the computers D_1 and D_2. Figure 2 shows an example of a map $Map(A)$ obtained from the input-output graph of Figure 1. Here, an agent A is required to visit a computer D_3 after D_1, D_4 after D_2 and D_3, and D_5 after D_4. On the other hand, an agent A can visit D_1 and D_2 in any order, even in parallel.

In Figure 1, the intermediate object w has to be delivered to D_3, D_4, and D_5. There are following ways to bring an intermediate object x obtained in D_i to D_j:

1. A transactional agent A carries the intermediate object x to D_j.
2. x is transfered from D_i to D_j before A arrives at D_j.
3. x is transfered from D_i to D_j after A arrives at D_j.

A routing agent $RC(A)$ of a transactional agent A with a map $Map(A)$ is moving around computers [Figure 3]. First,

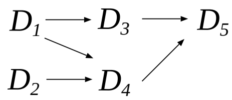

Figure 2: Map.

a collection I of computers which do not have any in-coming edge are found in $Map(A)$. For example, $I = \{D_1, D_2\}$ in Figure 2. One computer D_i is selected in I so as to satisfy some condition, e.g. D_i nearest to the current computer is selected. For example, an agent takes a computer D_1 in Figure 2. The agent A moves to 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 D_j is selected and A moves to D_j.

Initially, a routing agent $RC(A)$ of the agent A is loaded and started on a computer. The computer is a *base* computer $base(A)$ of the agent A. An agent A leaves the base computer for a computer D_i. Here, D_i is a *current* computer $current(A)$ of A. If the agent A invokes a method t of a class c on D_i, the class c is searched:

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

A history $H(A)$ shows a sequence of computers which an agent A has visited.

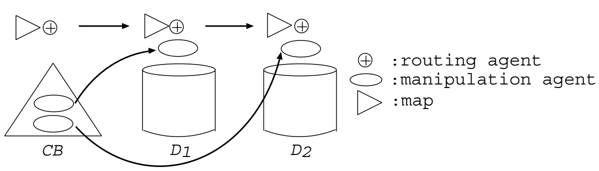

Figure 3: Mobile agent.

3.3 Manipulation agent

A manipulation agent is composed of not only application-specific classes but also library classes like JDBC [17] and JAVA classes [18]. Each computer is assumed to support a platform to perform a mobile agent on an object base (OB). A platform includes *cache* and *class base* (CB). The routing, manipulation, and commitment agents of a transactional agent A are stored in the class base (CB) of the home computer $home(A)$. If an agent A invokes a method t of a class c in a computer D_i, the class c is loaded from the home computer $home(c)$ to the cache in D_i. Then, the method t of the class c is performed in D_i. If a method u of another class d is invoked in the method t, the class d is loaded from the home computer $home(d)$ as well as the class c. Meanwhile, if another agent B invokes a method t of the class c in D_i, the class c in the cache is used to invoke the method t without loading the class c. Thus, if classes are cashed in a computer D_i, methods in the classes are locally invoked in D_i without any communication. Otherwise, it takes a longer time to invoke methods since classes with the methods are transferred from the home computers in networks. Here, the class c is loaded i.e. *cached* to D_i. The method t of the class c is performed on D_i. If another agent B comes to D_i after A has left D_i, B can take usage of the class c in the cache.

3.4 Commitment agent

If a transactional agent A finishes manipulating objects in each computer, the following *commitment* condition is checked by the commitment agent $CC(A)$:

1. *Atomic commitment*: an agent is successfully performed on all the computers in the domain $Dom(A)$, i.e. all-or-nothing principle used in the traditional two-phase commitment protocol [4, 15].
2. *Majority commitment*: an agent is successfully performed on more than half of the computers in $Dom(A)$.
3. *At-least-one commitment*: an agent is successfully performed on at least one computer in $Dom(A)$.
4. $\binom{n}{r}$ *commitment*: an agent is successfully performed on more than r out of n computers ($r \leq n$) in $Dom(A)$.
5. *Application specific commitment*: condition specified by application is satisfied.

3.5 Resolution of confliction

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 conflicting with A. Here, the agent A can take one of the following ways:

1. $Wait$: The agent A in the computer D_i *waits* until the agent A can land at a computer D_j.
2. *Escape*: The agent A $finds$ another computer D_k which has objects to be manipulated before D_j.
3. $Negotiate$: The agent A *negotiates* with the agent B in D_j. After the negotiation, B releases the objects or aborts.
4. *Abort*: The agent A *aborts*.

Deadlock among agents may occur. 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 surrogates preceding 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 finds another destination computer D_h.

The surrogate A_j to which the agent A retreats plays a role of checkpoint [12].

Suppose a surrogate agent B holds an object in a computer D_j. An agent A would like to manipulate the object but conflicts with B in D_j. The surrogate B makes a following decision:

1. *Atomic commitment*: The agent A waits until the surrogate B finishes.
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 after informing 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 after informing the other surrogates.
4. $\binom{n}{r}$ *commitment*: If the surrogate B knows more than

or equal to r sibling surrogate agents are committable, the surrogate B releases the object and aborts.

4. FAULT-TOLERANT AGENT

We assume computers may stop by fault and networks are reliable. A transactional agent is faulty only if a current computer of the agent is faulty. Suppose an agent A finishes manipulating objects on a computer D_i. The agent A selects one computer D_j from the map $Map(A,\ D_i)$. The agent A detects by timeout mechanism that D_j is faulty. The agent A tries to find another destination computer D_k [Figure 4]. If found, A moves to D_k as presented here. If A cannot find another destination computer in $Map(A,\ D_i)$, the agent A backs to the preceding computer D_k [Figure 5]. D_i is removed from $Map(A,\ D_k)$. Then, the agent in D_k tries to find another destination computer in $Map(A,\ D_k)$.

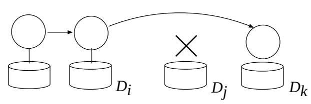

Figure 4: Forwarding recovery.

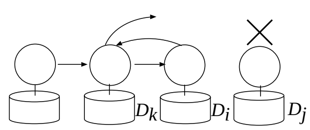

Figure 5: Backwarding recovery.

An agent A leaves its surrogate agent A_i on a computer D_i. The surrogate A_i holds objects even after the agent A leaves D_i. An agent A and surrogate agent A_i stop if the current computers are faulty. First, suppose an agent A stops on the current computer D_j. Suppose that the agent A comes from D_i to D_j. The surrogate A_i on D_i detects that the agent A stops on D_j. Here, A_i takes one of the following actions:

1. Find a succeeding surrogate A_k of A_i and skips A_j.
2. Recreate a new incarnation of the agent A.

If the commitment condition is not atomic, the surrogate A_j takes the first one, i.e. skips the fault of A_j. For the atomic condition, A_i recreates a new incarnation of the agent A. The agent A takes another destination computer D_k in $Map(A,\ D_i)$. If found, the agent A moves to D_k. Otherwise, A waits until the computer D_i is recovered or backs to the precedent computer from D_j.

A surrogate A_i on a computer D_i may be faulty as well. A preceding surrogate A_j on D_j detects the fault of A_i. Suppose a surrogate agent A_i of A exists on D_i. A_{i+1} and A_{i-1} show the succeeding and precedeing surrogate agents of A_i, respectively [Figure 6]. A_i periodically sends an enquiry message AYL (are you alive) to A_{i+1} and A_{i-1} to check if A_{i+1} and A_i are alive. On receipt of the AYL message, a surrogate sends back a response message IAL (I am alive). Thus, a faulty surrogate is detected by the succeeding and preceding a surrogate with timeout mechanism.

If A_i detects the stop of A_{i+1}, A_i does the follwings:

1. A new incarnation of the agent A is recreated on D_i.
2. From the map $Map(A,\ D_i)$, a new destination D different from D_{i-1} is detected.

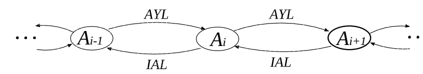

Figure 6: Fault detection.

3. If detected, the agent A moves to D. Otherwise, A_i informs A_{i-1} of *abort* and then aborts. A_{i-1} does the procedure from step 1.

If the surrogate A_i detects the stop of the preceding surrogate A_{i-1} or receives an *abort* message for A_{i-1}, A_i informs the succeeding surrogate A_{i+1} of abort. On receipt of the abort message from A_i, A_{i+1} forwards the *abort* message to A_{i+2} and then aborts. Thus, *abort* messages are eventually forwarded up to the agent A. In Figure 7, suppose A_2 stops. A pair of surrogates A_1 and A_3 detect the stop of A_2. A_1 creates a new incarnation A' of the agent A. The obsolete incarnation A still is moving to D_6. The succeeding surrogate A_3 of A_2 sends an *abort* message to A_4. If the abort message catches up the agent A, A can be aborted. Otherwise, the obsolete incarnation A cannot stop. Thus, there might exist multiple incarnations of an agent.

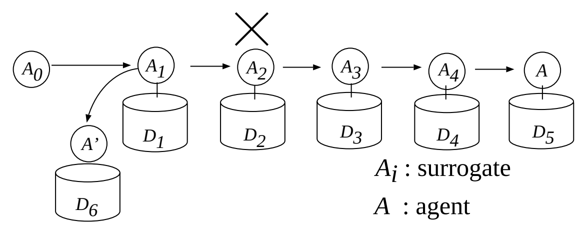

Figure 7: Incarnations of an agent.

On receipt of an AYL message from the preceding surrogate A_{i-1}, A_i sends an IAL message with the address information which A_i knows of surrogates are backwarding to preceding surrogates. If the surrogate A_i finds A_{i-1} to be faulty, A_i sends an *abort* message to not only A_{i+1} but also a surrogate whose address A_i knows and which is nearest to the current computer of A. By this method, an *abort* message can more easily catch up with the agent mapped the agent can be aborted.

5. IMPLEMENTATION

We discuss how to implement transactional agents in Aglets. A transactional agent A is composed of *routing*, *manipulation*, and *commitment* subagents. An routing agent $RC(A)$ with a map $Map(A)$ is transfered from one computer to another. When an agent A, i.e. routing agent $RC(A)$ arrives at a computer D_i, a *manipulation agent* $MC(A, D_i)$ is created by loading the manipulation class.

An object base (OB) is realized in a relational database system, Oracle [11]. A transactional agent manipulates *table* objects by issuing SQL commands, i.e. **select**, **insert**, **delete**, and **update** in a current computer D_i. The computation of each agent A on a computer D_i is realized as a local *transaction* on a database system. If the agent A leaves D_i, the transaction for A commits or aborts. That is, objects manipulated by A are released. Even if the agent A leaves D_i, the objects manipulated by A are required to be still held because A may abort after leaving D_i. If the objects are released, the agent is *unrecoverable*. Therefore, a *surrogate agent* is created on D_i. The surrogate agent is composed of a manipulation agent $MC(A,\ D_i)$ and an *object agent* OBA_i. OBA_i behaves as follows:

1. On arrival at a computer D_i, the routing agent $RC(A)$ of an agent A initiates a manipulation agent $MC(A,\ D_i)$ and an object agent OBA_i on D_i, i.e. $MC(A,\ D_i)$ and OBA classes are loaded. OBA_i initiates a transaction on an object base OB_i.
2. If $MC(A,D_i)$ issues a method for manipulating objects, OBA_i issues SQL commands to the database system in D_i.
3. If the agent A finishes, A leaves D_i. However, OBA_i is still operational and holding the objects in D_i.
4. OBA_i commits and aborts if the agent A sends *commit* and *abort* requests to the surrogate A_i, respectively.

An object agent OBA_i stays on a computer D_i while holding objects even if the agent A leaves D_i. OBA_i is a local transaction on an object base OB_i. On completion of the agent A, OBA_i and $MC(A, D_i)$ are terminated.

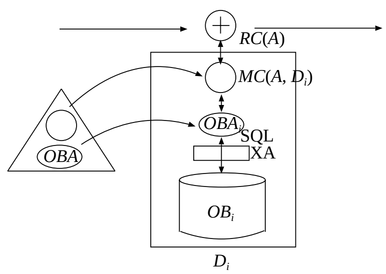

Figure 8: Object agent (OBA).

An OBA class can be loaded to a computer with any type of database system. If a transactional agent comes to D_i from another home computer, an 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, an OBA class is loaded as follows:

1. If the OBA class is not cached in the current computer, the OBA class is loaded from $home(OBA)$.
2. If the OBA class could not be loaded from $home(OBA)$, an OBA class in the home computer of the agent is loaded to a computer.

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

A transactional agent A can commit if all or some of the surrogates commit depending on the commitment condition. For example, a transactional agent commits if all the surrogate agents successfully exist. Communication among an agent and its surrogate agents is realized by using the XA interface [20] which supports the two-phase commitment protocol [15] [Figure 8]. Each surrogate agent issues a *prepare* request to a computer on receipt of a *prepare* message from A. If *prepare* is successfully performed, the surrogate agent sends a *prepared* message to A. Here, the surrogate agent is *committable*. Otherwise, the surrogate agent aborts after sending *aborted* to A. 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. For example, if the atomic condition holds, A sends *commit* only if *prepared* is received from every surrogate. The agent A sends *abort* to all committable agents if an *aborted* message is received from at least one surrogate. On receipt of *abort*, a committable surrogate aborts. In the at-least-one commitment condition, A sends *commit* to all committable surrogates only if *prepared* is received from at least one surrogate.

Next, we discuss how to support robustness against faults of computers. Suppose a surrogate agent A_i of a transactional agent A stops after sending *prepared*. Here, A_i is committable. On recovery of the committable surrogate A_i, A_i 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

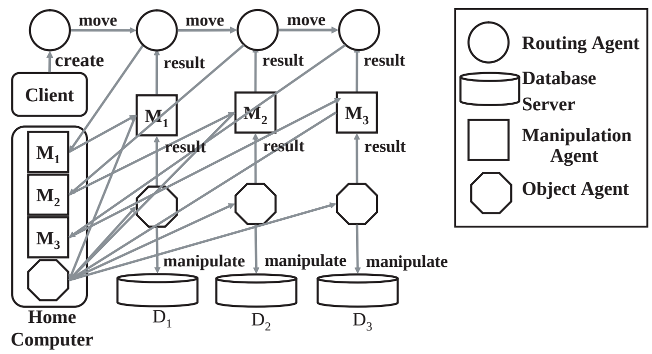

Figure 9: Evaluation model

We evaluate the transactional agent which is implemented in Aglets. In the evaluation, There are three server computers D_1, D_2, and D_3. A transactonal agent is created in a computer C by loading classes from the home computer h. The servers D_1, D_2, and D_3 are realized in personal computers (Pentium 3) with Oracle database systems, which are interconnected in the 1Gbps Ethernet.

First, a transactional agent A is initiated in a base computer C. The agent A finds in which order D_1, D_2, and D_3 to be visited. Here, the agent A visits D_1, D_2, and D_3 in this order as shown in Figure 9. On arrival of the agent A on D_i, the manipulation agent $MC(A,\ D_i)$ and object agent OBA_i are loaded to D_i [Figure 9].

We consider that following types of transactional agents:

A. The manipulation agents $MC(A,\ D_1)$ derives intermediate object I from the object base. The object bases in D_2 and D_3 are updated by using the object I, i.e. objects in I are added to the object base.
B. $MC(A,\ D_1)$ and $MC(A,\ D_2)$ derive objects to intermediate objects I_1 and I_2, respectively. 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 to another computer:

1. The transactional agent A carries intermediate objects to a destination computer D_j from D_i.

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

The total response time of a transactional agent is measured for number of intermediate objects, i.e. number of tuples deriverd in computeres. Figures 10 and 11 show the response time for the types of transactional agents A and B, respectively. The second and third ways to deliver intermediate objects to destination computers imply shorter responce time than the first way.

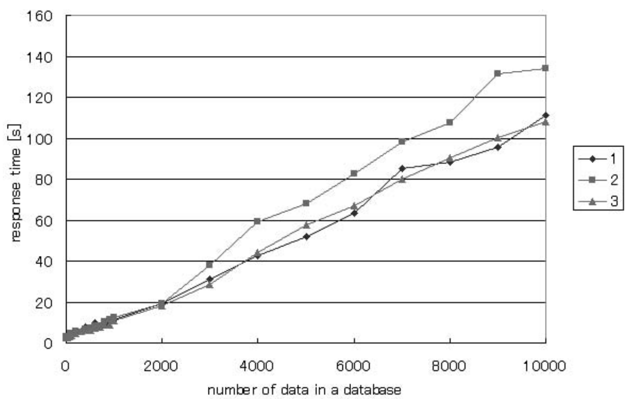

Figure 10: Response A

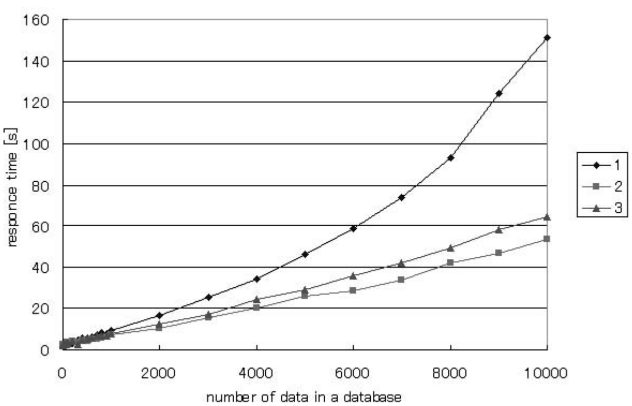

Figure 11: Response B

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 finds a distination computer, 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.

8. 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] 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.

[15] D. Skeen. Nonblocking commitment protocols. *Proc. of ACM SIGMOD*, pages 133–147, 1982.

[16] 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.

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

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

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

[20] X/Open Company Ltd. *X/Open CAE Specification Distributed Transaction Processing: The XA Specification*, 1991.