Negotiation Strategy for Multiple Autonomous Agents *

Chiaki Yahata and Makoto Takizawa †

Dept. of Computers and Systems Engineering
Tokyo Denki University ‡
e-mail {chii,taki}@takilab.k.dendai.ac.jp

A *cooperating database system* is composed of multiple agents interconnected by communication networks where some agents provide database systems. It purposes to provide easy access to various kinds of multiple autonomous database systems under a situation that the system configuration is changed dynamically. An agent is an autonomous system which assists users with accessing multiple database systems. Each agent takes the request from the user. The agent may ask other agents to answer the request so as to meet the user's requirement by doing the negotiation with them on whether and how they could answer the request. In this paper, we present a model of the cooperating database system and discuss a negotiation protocol among multiple agents.

1 Introduction

Various kinds of autonomous database systems including existing enterprise database systems and personal database systems are interconnected by communication networks. Distributed database systems [2, 7, 8, 10, 11, 13, 14, 15] are systems including multiple, possibly heterogeneous database systems interconnected by communication networks, where users can access multiple database systems without being conscious of their heterogeneity, autonomy, and distribution. There are two kinds of distributed database systems. One is an *integrated* distributed database system (a tightly coupled system[13], where one *global schema* on all the database systems is defined for the users by a global administrator [8, 14, 15]. Through the global schema, users can access all the database systems as if they were one database system which provides the global schema. The other one is a *multi-database system* [9]. Instead of providing one global schema on all the database systems, users can define dynamically their views on a subset of the database systems in the distributed database system. It is named a *dynamic integration* of multiple database systems.

The groupware applications [5] include various kinds of database systems like enterprise database systems and personal database systems. In addition to deriving information from the existing well-organized enterprise database systems, it is important to access less-defined information kept by each individual. Furthermore, new database systems may be added and some database systems may stop the service. In the presence of various kinds of database systems, it is difficult for users to find what kinds of database systems are included, where they exist, and how they are manipulated. In order to provide easy access to multiple database systems, our system is composed of *agents*. Each database system is a kind of an agent. The agent assists users with their accessing multiple database systems. Each user issues a request to access some

---

[*] 自律的エージェント間の交渉方式
[†] 矢羽田 千哲　滝沢 誠
[‡] 東京電機大学 理工学部 経営工学科

data without being conscious of where it is and how it is accessed it. The agent takes the request to access multiple database systems, and may ask another agent to obtain the answer of the user's request if it cannot answer the request. The agent has to do the negotiation with other agents on what and how they can do. Thus, each agent not only provides a database system but also helps users with manipulating other database systems. Through the negotiation with another agent, each agent can obtain information on what kind of database the agent has. A *cooperating database system* is a system which is composed of multiple agents interconnected by communication networks. In this paper, we would like to present the architecture of the cooperating database system and a protocol for doing the negotiation among multiple agents.

In section 2, a system model of the *cooperating database system* is presented. In section 3, we discuss the *acquaintance* relation among the agents. In section 4, a protocol for doing the negotiation among multiple agents is discussed. In section 5, a learning method for each agents to obtain information on the change of the system state is presented.

2 System Model

A *cooperating database system* is composed of multiple agents interconnected by communication networks [Figure 1]. An *agent* is a system which provides a database system and may access another agent to answer user's requests. The agent considers the database as a collection of data objects. Objects are tuples in the relational database system[4]. Record occurrences are objects in the network-type database system [3]. This means that the agent provides users with heterogeneity-independent access to the database systems. Each user U issues an agent A a request R which describes what data objects U would like to access. The agent A takes the request R from U, and finds what agents have objects required by U. A accesses its own database if the database includes the objects, and asks another agent to answer the request. Even if A has the objects, A may ask another agent to answer R if A thinks of it to be more suitable to answer the requests, e.g. from the performance point of view.

Each agent is autonomous. That is, each agent can decide what object it has and how it would behave for the request. For example, an agent usually answers the request but sometimes may not. Even if some strategy for accessing multiple agents is pre-decided based on the statistical information like [16], the agents may not behave as expected in the strategy, for example, because the states or policies of the agents may be changed. Hence, negotiation among agents is required to make clear what and how each agent can do. The agents does the negotiation with other agents to find what agents have the objects and how they could obtain them. Thus, the users can manipulate multiple database systems through agents without being conscious of the heterogeneity, distribution, and autonomy of the database systems.

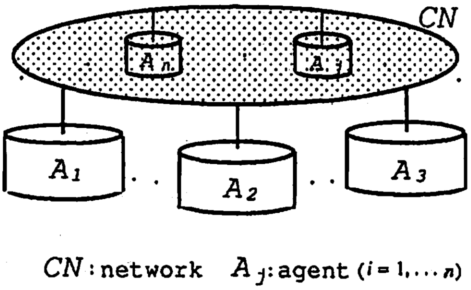


Figure 1: Cooperating database system

3 Agents

An agent is an autonomous system which accesses multiple database systems through negotiation with other agents. We present the types, behaviour, and structure of the agents in this section.

3.1 Passive and active agents

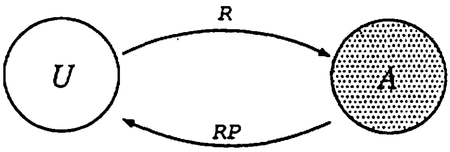
Since the cooperating database system includes huge number of agents interconnected by communication networks, it is difficult, maybe impossible for each user to obtain the following information :

1. what kinds of agents are included,
2. what kind of database system each agent has, and
3. how each agent can answer requests, e.g. on the response time and processing time.

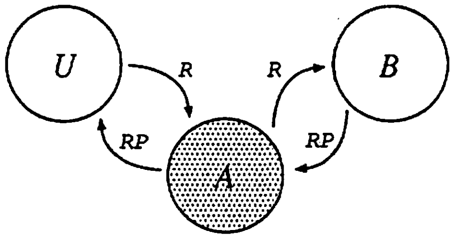
Hence, we need a mechanism named an *agent* which assists users with obtaining information on and manipulating multiple database systems.

There are two kinds of agents, i.e. *passive* and *active* ones [Figure 2]. The passive agent A takes a request R from a requester U, i.e. a user or another agent, and then answers R if A can answer R. A sends the answer of R back to U. If A cannot answer R, A informs U of the failure. For example, suppose that U sends a request R to A to obtain objects on *Tokyo*. If A has objects on *Tokyo*, A sends the objects to U. Otherwise, A informs U of the failure. Thus, the passive agent does not issue requests to another agent. Conventional database systems and server systems like print servers are examples of the passive agents.

On the other hand, the active agent A can issue a request to another agent. For example, if A cannot answer a request R, A can send R to another agent which A thinks can answer R. Even if A can answer R, A can send R to another agent B if A thinks that B better, e.g. faster than A [Figure 2(b)]. Furthermore, A can decompose R to subrequests $R_1, \ldots, R_n$ ($n \geq 1$) and send each R_i to an agent A_i which A thinks can answer R_i ($i = 1, \ldots, n$). Thus, the active agent is a system which can issue requests to another agent by itself.



(a)passive agent A



(b)active agent A

Figure 2: Active and passive agents

There is one kind of the passive agent named a *kind* agent. If a kind agent A knows what agent, say B can answer the request R from U, A informs U of B when A cannot answer R. On receipt of the reply from A, U may send R to B. The conventional distributed database systems [13] are composed of passive agents, i.e. database systems. On the other hand, the cooperating database system includes not only passive but also active agents.

3.2 Behaviour of agent

On receipt of a request R from a user or agent U, A behaves as follows.

[Behaviour of agent]

1. A decomposes R into subrequests R_1, ..., R_n ($n \geq 1$) .
2. A decides what agent A_i can answer each subrequest R_i ($i = 1, \ldots, n$).
3. A asks each A_i if A_i can answer R_i, A_i

negotiates with A_i on how A_i can answer R_i if A_i can answer R_i, e.g. how long it takes to answer R_i. Otherwise, another agent is tried to be found for R_i at step 2.

4. A asks A_i to answer R_i according to the way negotiated in step 3. A_i answers R_i and sends back the reply RP_i to A.

5. A collects the results RP_1, ..., RP_n from A_1, ..., A_n, respectively, and generates the result RP of R from RP_1, ..., RP_n. A sends RP to U. □

The step 1 is a *decomposition* of the request. The step 2 is an *allocation* of subrequests to agents. The step 3 is a *negotiation* among the agents. The step 4 is an *execution* of the request. The step 5 is a *composition* of replies from the agents.

3.3 Structure of agent

The cooperating database system is composed of agents interconnected by a communication network CN as shown in Figure 3. Each agent A_i is composed of two parts, i.e. *head* H_i and *body* B_i. B_i includes a database system DBS_i. DBS_i is composed of a database DB_i which is a collection of *objects*. B_i manipulates objects in DB_i. For each object o, $Term_o$ is a collection of *terms* $\{t_1, \ldots, t_m\}$. Each term corresponds to a *keyword* in the information-retrieval systems [12]. The meaning of o is defined as $Term_o$. For example, suppose that each object in a database on cities represents a city. An object denoting *Tokyo* can be given to a set of terms $\{$*Capital*, *Japan*, *Tokyo*, ...$\}$. Here, let O be a set of objects and T be a set of terms in the system. Each agent A has a subset O_A of O and a subset T_A of T. For two agents A and B, O_A and O_B may not be disjoint, and T_A and T_B may not either. That is, each object and term can exist redundantly in multiple agents. For each term t in T, $Obj(t)$ denotes a set of objects on t in O. $Obj_A(t)$ denotes objects on t in A if t is in $T_A(t)$, i.e. *A knows about t*. Here, $Obj_A(t) \subseteq Obj(t)$.

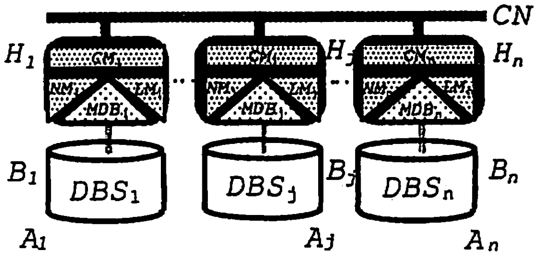


Figure 3: Agents

The head H_i is composed of a *metadatabase* MDB_i, *communication* module CM_i, *learning* module LM_i, and *negotiation* module NM_i [Figure 3]. MDB_i is a collection T_i of terms structured by *is_a* and *part_of* relations. Figure 4 shows the metadatabase of two agents A and B. T_A = $\{$*Capital, London, Paris, Tokyo*$\}$ is structured, like "*London*, *Paris*, and *Tokyo* are *Capitals*". A has a term *Tokyo* and an object on *Tokyo*. A may know that B knows *Tokyo* as shown in Figure 4. Thus, each term in an agent shows objects in its own database or terms in another agent.

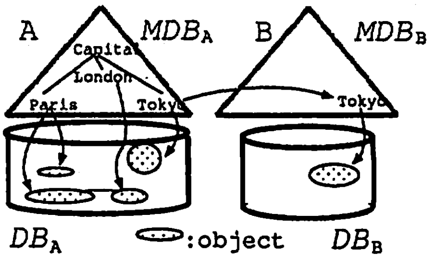


Figure 4: Metadatabases

Each agent A_i can obtain information on terms through communicating with another agent. LM_i changes MDB_i by adding new terms and relations among terms obtained from another agent. Since MDB_i is finite, MDB_i is eventually fully engaged by terms. LM_i removes terms which E_i thinks are unuseful.

NM_i executes the protocol for the negotiation among the agents.

4 Acquaintances

Let MDB_A and DB_A be a metadatabase and a database of an agent A, respectively. Each term t in MDB_A denotes not only objects on t which A has but also another agent which A knows has t. MDB_A is T_A. If MDB_A includes t, A is referred to as *knows* about t. A *directly know* about t iff DB_A includes some object on t, i.e. $DB_A \cap Obj(t) \neq \phi$. A is referred to as *indirectly know* about t if A knows about t but does not directly know about t. Here, although A has no object about t, A has t in MDB_A. Hence, A cannot obtain objects on t from DB_A but can ask another agent denoted by t in MDB_A which directly or indirectly knows about t. If A directly knows about t, A can obtain objects on t from DB_A.

[Definition] A is an *acquaintance* of B on t ($A \xrightarrow{t} B$) iff A knows that B knows about t. For some term t, $A \rightarrow B$ if $A \xrightarrow{t} B$. □

If A cannot answer a request R on t, A can send R to an acquaintance B of A on t.

[Definition] For some term t, if $A \xrightarrow{t} B$, $B \xrightarrow{t} C$, and not $B \xrightarrow{t} A$, A *transitively knows* C about t (or C is an *indirect acquaintance* of A) ($A \overset{t}{\hookrightarrow} C$). A *directly knows* B about t (written $A \overset{t}{\Rightarrow} B$) (or B is a *direct acquaintance* of A) iff $A \xrightarrow{t} B$ and not $A \overset{t}{\hookrightarrow} C$. □

It is clear that $A \xrightarrow{t} B$ if $A \overset{t}{\hookrightarrow} B$. We assume that A directly knows B if A can access B, i.e. A has the access right on B. If A transitively knows B about t, A cannot access B because A may have no access right on B. Hence, A can access only the direct acquaintances. There are two ways to access B. One way is that A finds the direct acquaintance C such that $C \rightarrow B$ and asks C to access B. In the other way, A has to obtain the access right on B and then A accesses B directly. In order to obtain the access right on B, A has to do the negotiation with B. There are two ways to obtain the access right on B. In the first way, A asks B directly to grant the access right to A. In the second way, A first asks the direct acquaintance C to allow A to access B. C asks B if A could access B.

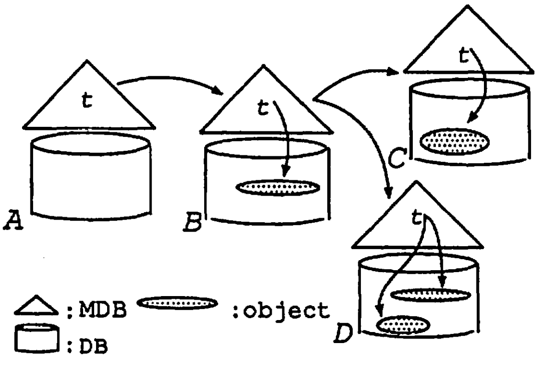


Figure 5: Acquaintances

Figure 5 shows an acquaintance relation on a term t among four agents A, B, C, and D. Each directed arc $A \rightarrow B$ shows that B is a direct acquaintance of A. C and D are indirect acquaintances of A. A indirectly knows about t since A has no object on t and A knows that B knows about t. On the other hand, B, C, and D directly know about t because they have objects on t. Here, A can directly ask B but can neither directly ask C nor D. On receipt of a request from A, B may obtain objects on t in DB_B and may ask C or D to get objects on t. It depends on the autonomy of B.

Since each agent is autonomous, B might not be an acquaintance of A at present even if A has thought that B is the acquaintance of A. For example, although A thinks that B still knows about t, t may be removed from B. B is referred to as *close* acquaintance of A if A always knows what B knows. If B informs A of the change each time MDB_B

is changed, B can be a close acquaintance of A.

[Example] Let us consider a *shopping* system C [Figure 6] as an example of the *cooperating database system*. C includes agents, i.e. *department stores* *Dept* and *Store*. *Dept* has acquaintances, i.e. *Shoes*, *Clothes*, *Accessory*, *Daily_necessity*, and so on. *Clothes* has acquaintances, *Suit*, *Shirts*, and *Sports*. A user U can access *Dept* if U would like to buy something, without being conscious of where they could buy it. If U would like to buy a collection of a suit, shirt, and ties, U asks *Dept* to obtain what U would like to buy. *Dept* decomposes R into subrequests R_1 to *Clothes* and R_2 to *Accessory*. On receipt of R_1 from U, *Clothes* decomposes R_1 into R_{11} for *Suit* and R_{12} for *Shirt*. *Suit* selects a suit and sends the information to *Clothes*. Then, *Clothes* asks *Shirt* to select a shirt which goes well with the suit. *Clothes* sends the reply RP_1, i.e. a suit and a shirt back to U. *Accessory* also sends the reply RP_2, i.e. a list of ties. *Dept* checks whether the ties go well with to the shirt based on the colour. If it is OK, *Dept* sends back the collection of the suit, shirt, and ties to U. If not, *Dept* asks *Accessory* to show another list of ties. □

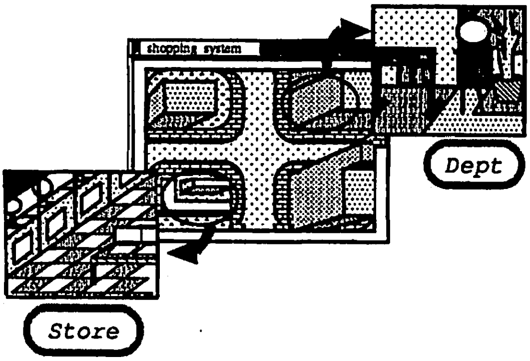


Figure 6: Shopping system C

5 Negotiation

In this paper, we would like to think about only retrieval operations on multiple database systems because it is difficult to consider update operations on multiple database systems and most users would rather retrieve data objects.

5.1 Requests

First, an agent A takes a request R from a requester U. R is composed of a qualification Q and a *preference* P, i.e. $R = \langle Q, P \rangle$. Q is written as follows. Let t and *qual* denote a term and qualification, respectively. *qual* is defined as $qual_1 \mid qual_2$, $qual_1$ & $qual_2$, $qual_1 - qual_2$, or t. *Result*(*qual*) is the meaning of *qual* which is defined as follows. *Result*(t) is a set of objects on t, i.e. $\{o \mid o \in Obj(t)\}$. $Result(qual_1 \mid qual_2) = Result(qual_1) \cup Result(qual_2)$. $Result(qual_1$ & $qual_2) = Result(qual_1) \cap Result(qual_2)$. $Result(qual_1 - qual_2) = Result(qual_1) - Result(qual_2)$.

There may be multiple ways to obtain *Result*(Q). The preference P is used to select one way among them. The preference P is in a form of $\langle P_1, \ldots, P_m \rangle$, $\{P_1, \ldots, P_m\}$, or $[P_1, \ldots, P_m]$ ($m \geq 0$) where P_i is a preference or a preference predicate. A preference predicate is given in a form of *pitem* θ *value*, where θ is a comparison operator, *pitem* is one of *communication_time*, *response_time*, *processing_time*, *agent*,and *completeness*. $\langle P_1, \ldots, P_m \rangle$ means that P_i is preferred to P_k if $i < k$. Let W_0 be a set of ways which can obtain *Result*(Q). First, a subset W_1 of W_o which satisfies P_1 is obtained. Thus, $W_i \subseteq W_{i-1}$ which satisfies P_i is obtained from W_{i-1} until W_i gets a singleton. If $W_i = \phi$, one way in W_{i-1} is selected. In $\{P_1, \ldots, P_m\}$, one way which satisfies all $P_1, \ldots, P_m$ is selected. For $[P_1, \ldots, P_m]$, one way with satisfies at least one of $P_1, \ldots, P_m$ is selected.

Suppose that there are two agents A and

B, which are acquaintances of an agent U, which U thinks know about *Tokyo*. Suppose that A is faster but farther from U than B. Suppose that the preference is $\langle$*communication_cost, processing_time* ≤ 50 $\rangle$. This means that U prefers less communication time. If there are still multiple ways whose communication costs are the smallest, one way whose *processing time* ≤ 50 is selected. U selects B because B is nearer to U, and then asks B to execute the request. If the preference is $\{$ *communication_cost, processing_time* ≤ 50 $\}$, U selects one way not only which has the *minimum communication cost* but also whose *processing time* ≤ 50. The preference $\langle$*agent* $= A\rangle$ means that A is preferred to be used to obtain the result. $\langle$*processing_time* $\leq 50\rangle$ means that U would like to obtain the result in the total processing time ≤ 50. Let *Answer*(R) be a set of objects obtained by the cooperating database system. If $\langle$*completeness* $=$ *Partial*$\rangle$, *Answer*(R) may be a subset of *Result*(Q). If $\langle$*completeness* $=$ *Full* $\rangle$, *Answer*(R) has to be *Result*(Q). □

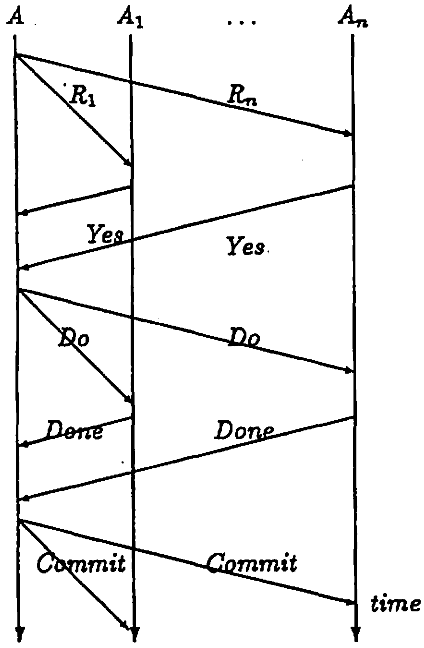


Figure 7: Negotiation protocol

5.2 Negotiation protocol

A negotiation protocol is shown as follows.

[Negotiation procedure][Figure 7]

1. A takes a request $R = \langle Q, P\rangle$ from a requester U. If A can answer R, A executes R. Otherwise, A decomposes R into $R_1, \ldots, R_n$ $(n \geq 1)$, where $R_i = \langle Q_i, P_i\rangle$ $(i = 1, \ldots, n)$. If R cannot be decomposed, A sends the *Failure* back to U.

2. A finds for each R_i an acquaintance agent A_i which A thinks can answer R_i $(i = 1, \ldots, n)$. If no agent can be found for R_i, R_i is further decomposed into smaller subrequests $R_{i1}, \ldots, R_{im_i}$ $(m_i \geq 2)$. Then, this step is repeated until some agent is allocated to each subrequest. If R_i cannot be further decomposed, all the executions of the subrequests are aborted, i.e. A sends *Abort* messages to $A_1, \ldots, A_n$. Then, R is tried to be differently decomposed by the step 1.

3. A asks each A_i whether A_i can answer R_i and how A_i can answer R_i. If A_i cannot answer R_i, A tries to find another acquaintance at step 2. If A cannot find any agent for R_i, R_i is tried to be further decomposed by returning to the step 2.

4. A asks A_i to execute R_i by sending a *Do* message to A_i. On receipt of the *Do*, A_i executes R_i. If A_i can not obtain the answer of R_i, A_i sends the *Failure* to A. On receipt of the *Failure* from some A_i, A returns to step 3 and tries to find another candidate of R_i. If A_i can obtain the answer RP_i of R_i, A_i sends the *Done* message with RP_i to A.

5. A integrates all answers $RP_1, \ldots, RP_n$ into an answer RP for R. A sends the RP back to U. □

If R_i changes the state of A_i, i.e. R_i is an update operation on DB_i, the update data obtained by R_i is saved into the secure storage, i.e. a log L_i of A_i at step 4. Then, A_i sends the *Done* to A. On receipt of all the *Done* messages, A sends *Commit* messages to $A_1, \ldots, A_n$. On receipt of the *Commit*, A_i changes the state by using the update data in L_i. This process is similar to the two-phase commitment [6].

Suppose that A_i takes $R_i = \langle Q_i, P_i \rangle$ from A at step 3. If A_i knows all the terms included in Q_i, A_i can answer R_i. Otherwise, A_i sends back the *Failure* of R_i to A. Next, A_i considers how A_i can obtain $Answer(R_i)$. A_i has to obtain $Answer(R_i)$ so as to satisfy the preference P_i. If A_i cannot satisfy P_i, A_i sends A a message describing how A_i cannot satisfy P_i. Suppose that P_i is a set $\{P_{i1}, \ldots, P_{im_i}\}$ of preferences. If A_i can obtain the result of R_i so as to satisfy a subset $AP_i = \{AP_{i1}, \ldots, AP_{ih_i}\} \subseteq P_i$, A_i sends AP_i to A. On the other hand, suppose that P_i is a list $\langle P_{i1}, \ldots, P_{im_i} \rangle$. Suppose that A_i can satisfy $AP_i = \langle AP_{i1}, \ldots, AP_{ik} \rangle$ where $\{AP_{i1}, \ldots, AP_{ik_i}\} \subseteq \{P_{i1}, \ldots, P_{im_i}\}$ and the order of preferences in P_i may not be preserved in AP_i. If A can accept AP_i, A asks A_i to execute $\langle R_i, AP_i \rangle$. Otherwise, A cannot ask A_i.

Suppose that there are multiple candidates A_{i1}, ..., A_{im_i} ($m_i \geq 2$) for a request R_i. One way is to select one agent, say A_{ij}, and ask A_{ij} to execute R_i as presented above. Another way is to send R_i to all A_{i1}, ..., A_{im_i} and then ask all of them to execute R_i in parallel. If at least one A_{ij} of $A_{i1}, \ldots, A_{im_i}$ could return the answer RP_{ij} of R_i, A can consider RP_{ij} as the answer RP_i of R_i.

Suppose that R is decomposed into $R_1, \ldots, R_n$. There may be some precedence relation → among them. $R_i \rightarrow R_j$ means that R_j has to be executed after R_j completes. If there is no precedence relation among R_i and R_j, R_i and R_j can be executed in parallel. At step 1, a partially ordered set $\{R_1, \ldots, R_n\}$ on → is obtained from R.

[Example] Let us consider the shopping system C as shown in Figure 8. Suppose that U would like to buy a collection of a suit, tie, and shirt. Here, U prefers that the wool suit colours grey, the cotton shirt colours white. The shirt costs about 10,000 yen. U would like to have a tie going well with the suit in colour. U would like to buy the suit and shirt at a store A or B. U prefers A to B.

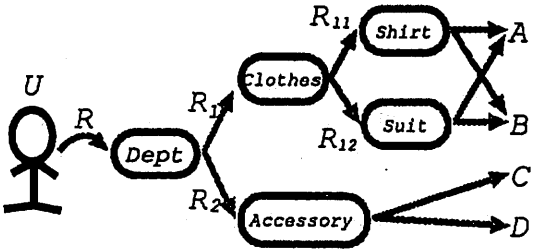


Figure 8: Decomposition

1. U sends *Dept* a request R = ⟨ { { *suit.colour* = "*Grey*" & *suit.material* = "*Wool*" } & { *shirt.colour* = "*White*" & *shirt.material* = "*Cotton*" & *shirt.cost like 10,000 yen* } & { *matches* (*tie, suit*) } }, { ⟨⟨*suit, shirt* : *A, B*⟩, ⟨*tie* : *C, D*⟩⟩ } ⟩.
2. *Dept* takes R from U. *Dept* decomposes R into two subrequests R_1 = ⟨ {{ *suit.colour* = "*Grey*" & *suit.material* ="*Wool*" } & { *shirt.colour* = "*White*" & *shirt.material* = "*Cotton*" & *cost like 10,000yen* }}, ⟨*suit,shirt* : *A, B*⟩⟩ and R_2 = ⟨*matches* (*tie, Suit*), ⟨*C, D*⟩⟩. R_1 is sent to *Clothes* and R_2 to *Accessory*.
3. *Clothes* takes R_1, and decomposes R_1 into R_{11} = ⟨{*colour* = "*Grey*" & *material* = "*Wool*"}, ⟨*A*, *B*⟩⟩ and R_{12} = ⟨{*colour* = "*White*" & *material* = "*Cotton*"}, ⟨*A*, *B*⟩⟩. R_{11} and R_{12} are sent to *Suit* and *Shirt*, respectively.
4. *Suit* sends R_{11} to A according to the preference. If A could obtain the suit

satisfying the qualification of R_1, A sends the reply back to *Suit*. Otherwise, *Suit* sends R_{11} to B. *Shirt* negotiates with A and B with respect to R_{12} similar to *Suit*. *Suit* and *Shirt* send back the replies to *Clothes*.

5. After receiving the replies, *Clothes* sends $R_2 = \langle\{clour = \text{"}Grey\text{"}\}, \langle C, D\rangle\rangle$ to *Accessory*.
6. *Accessory* sends R_2 to C according to the preference. If C could not answer R_2, *Accessory* sends R_2 to D. If *Accessory* gets the tie going well with *Suit*, *Accessory* returns the reply to *Dept*. *Dept* receives the replies from *Clothes* and *Accessory*.
7. Finally, *Dept* sends a list of the suit, shirt, and tie to U. □

6 Learning

An agent A can obtain newly terms and relations among terms from another agent through the negotiation. The terms and relations among the terms are stored in MDB_A. The process is named a *learning*.

An example of the learning in an agent A is shown in Figure 9. A knows that *London* and *Paris* are *capitals*, but does not know of *Tokyo*. On the other hand, agents B and C know that *Tokyo* is a capital. B and C are acquaintances of A. Suppose that A takes a request to obtain a set of capitals. A asks B and C to obtain the capitals. B and C return the sets of capitals derived from DB_B and DB_C, i.e. $RP_B = \{Tokyo, London\}$ and $RP_C = \{Tokyo, Paris\}$, respectively. On receipt of the replies RP_B and RP_C from B and C, A newly knows that *Tokyo* is a capital. A adds a new term *Tokyo* and a new *is_a* relation "*Tokyo* is a *Capital*" in MDB_A.

The metadatabases are finite. If MDB_A is too full to store new terms and relations, some terms and relations are removed from MDB_A. It is named an *oblivion* process.

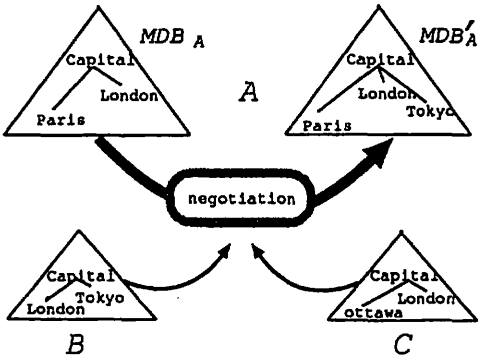


Figure 9: Learning

Problem is what terms and relations to be removed from MDB_A. The following rules to remove terms in MDB_A are adopted.

1. The terms which A directly knows are not removed.
2. If the terms are frequently used, they are not removed.
3. The terms of the higher levels are not removed.

Suppose that terms t_1 and t_2 are tried to be stored in MDB_A but MDB_A is full. If A directly knows about t_1 but not about t_2, t_2 can be removed because some agent different from A directly knows about t_2. If t_1 and t_2 could be removed and t_1 has been used more frequently than t_2, t_2 can be removed. If t_1 is not at a higher level than t_2 in MDB_A, t_2 can be removed. The terms of a higher level mean that they represent more abstract information than the terms of lower levels. Even if more-detailed information is forgotten, we can restore the information if we still remember the abstract information, e.g who knows about it. Thus, the terms of lower levels can be removed.

7 Concluding Remarks

In this paper, we have discussed the architecture of the cooperating database system which is composed of multiple agents interconnected by the communication network. The cooperating database system includes not only passive but also active agents although the conventional distributed database systems include only passive agents, i.e. database systems. We have shown a negotiation protocol among agents. By this procedure, agents can obtain the reply by taking advantage of another agent. We have also shown how to maintain the metadatabases, i.e. learning module.

References

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

[2] Ceri, S., and Pelagatti, G., "Distributed Databases - Principles and Systems," *McGraw-Hill*, 1984.

[3] CODASYL, "Data Description Language Journal of Development," *Canadian Government Publishing Center*, 1973.

[4] Codd, E. F., "A Relational Model for Large Data Banks," *CACM,* Vol.13, No.6, 1970, pp. 377-387.

[5] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. of ACM*, No.1, 1991, pp.38-58.

[6] Eswaren, K. P., Gray, J., Lorie, R. A., and Traiger, I. L., "The Notion of Consistency and Predicate Locks in Database Systems," *CACM*, Vol.19, No.11, 1976, pp.624-637.

[7] "Proc. of the IEEE, Special Issue on Distributed Database System," 1987.

[8] Landers, T., and Rosenberg, R., "An Overview of Multibase," *Distributed Databases, Schneider, H.J., Ed., North-Holland,* 1982, pp.153-184.

[9] Litwin, W. and Abdellatif, A., "An Overview of Multidatabase Manipulation Language MDSL," *IEEE Proc.* Vol.75, No.5, 1987, pp.621-632.

[10] "Data Processing - Open Systems Interconnection - Basic Reference Model," DP7498, 1980.

[11] Ozsu, M. T. and Valduriez, P., "Principle of Distributed Database Systems," *Prentice-Hall*, 1990.

[12] Salton, G., Mcgill, M. J., "Introduction to Modern Information Retrieval," *McGraw-Hill International Book Company*, 1983.

[13] Sheth, A. P. and Larson, J. A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, Vol.22, No.3, 1990, pp.183-236.

[14] Smith, J., et al. "Multibase: Integrating heterogeneous distributed database systems," *Proceedings of the National Computer Conference,* 1981.

[15] Takizawa, M., "Distributed Database System JDDBS," *JARECT, Vol.7, Computer Science and Technologies (Kitagawa, T., ed.), Ohmsha and North-Holland,* 1983.

[16] Yu, C., and Chang, C,. "Distributed query processing," *ACM Comput. Surv.*, Vol. 16, No. 4, 1984, pp.399-433.