

Bir Örnek Dağıtık Veri Tabanının Dizge Tanımı

77357

Bora İ. Kumova

77357

Aralık, 1998

İZMİR

System Specification for an Example Distributed Database

**A Thesis Submitted to the Graduate School of Natural and Applied
Sciences of
Dokuz Eylül University**

**In Partial Fulfilment of the Requirements for the Degree of Master of
Science in Computer Engineering, Software Engineering Programme**

**By
Bora İ. Kumova**

**December, 1998
İZMİR**

Bir Örnek Dağıtık Veri Tabanının Dizge Tanımı

Dokuz Eylül Üniversitesi

Fen Bilimleri Enstitüsü

Yüksek Lisans Tezi

Bilgisayar Mühendisliği Bölümü, Yazılım Anabilim Dalı

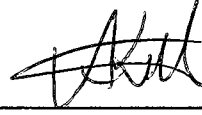
Bora İ. Kumova

Aralık, 1998

İzmir

M.Sc. Thesis Examination Result Form

We certificate that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as thesis for the degree of Master of Science.



Assoc. Prof. Dr. Alp Kut
(Advisor)

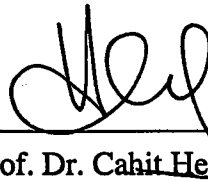


Assoc. Prof. Dr. Malcolm Heywood
(Comity Member)



Asst. Prof. Dr. Reyat Yilmaz
(Comity Member)

Approved for the
Graduate School of Natural and Applied Sciences



Prof. Dr. Cahit Helvacı
Director

Graduate School of Natural and Applied Sciences

Acknowledgements

Thanks are due to my advisor, for giving me latitude for interdisciplinary research, inside the defined scope.



Bora I. Kumova

Abstract

Currently, a rapidly growing amount of structured data is becoming available over the network. Most of them reside on workstations within relatively small databases. Users are interested in sharing their structured data and in having own views of that distributed data. An attempt to combine this data in a bottom-up design and try to guarantee overall data consistency would fail with current distributed database management systems. The reasons are the existence of various types of heterogeneity and the dynamic behaviour of the overall data structure. Both are caused by the fact that each owner usually insists on the autonomy of his database.

Our approach is to provide for distributed database management on each client separately. Each user can deal with the dynamic nature of the distributed data structures by managing his views by himself. Furthermore, a user environment with access to shared data can be utilised for co-operative teamwork of users. Thus, we propose view-based distributed database management with guaranteed data integrity, until data structures change some where in the distributed database. Further, we propose an agent-based solution for database connectivity. Agents can be designed to handle with heterogeneity and as object-oriented, intelligent entities are inherently suitable implementing distributed database management in a distributed fashion, which fits the best the network characteristics.

Özet

Halen, büyük bir hız ile çoğalarak biçimlenmiş veri, ağ üzerinden kullanılabilir hale geliyor. Bunların çoğu iş istasyonlarında oldukça küçük veri tabanlarında bulunuyor. Kullanıcılar, kendi biçimlenmiş verilerini paylaşmak ve bu dağıtık veri tabanına kendi açılarından bakmak istiyorlar. Bu verileri, bir alttan-yukarı tasarımı ile birleştirme ve kapsamlı veri tutarlılığı sağlama giriřimi, var olan dağıtık veri tabanı dizgeleri ile başırlamaz. Nedenleri ise, deęiřik tür ayrıcalıkların var olmasında ve kapsamlı veri yapıların devingen davranışlı olmasındadır. İkisi de, kullanıcıların genelde kendi veri tabanlarının bağımsız olmasında direndiklerinden kaynaklanmaktadır.

Bizim yaklaşımımız ise, dağıtık veri tabanı yönetimini her istemcide ayrı ayrı sağlamaktır. Her kullanıcı, kendi görüş açılarını yöneterek dağıtık veri yapıların devingen davranışı ile kendi ilgilenebilir. Bundan da öte, paylaşılan verilere erişimi olan bir kullanıcı ortamı, işbirlikli takım çalışması için yararlandırılabilir. Buna göre biz, görüş açısına dayanan, veri yapıları dağıtık veri tabanında her hangi bir yerde deęişinceye deęin veri tutarlılığı güvenceli olan, dağıtık veri tabanı yönetimi öneriyoruz. Ayrıca, aracıya dayanan bir veri tabanı bağlantısı öneriyoruz. Aracılar tür ayrıcalıkları ile başa çıkabilecek nitelikte tasarlanabilirler ve nesneye dayalı, akıllı varlıklar olarak, ağ niteliklerine en uygun gelen, dağıtık bir biçimde dağıtık veri tabanı yönetimini gerçekleřtirmek için doęal olarak uygundur.

Contents

Contents	X
List of Tables	XIII
List of Figures	XIV
Introduction	1

Chapter One

Distributed Database Systems

1.1 Distributed Database Management	4
1.2 Distributed Database System Architecture	5
1.3 Distributed Database Design	8
1.4 Data Distribution	9
1.5 Transaction Management.....	12
1.6 Distributed Concurrency Control.....	13
1.7 Data Security.....	15

Chapter Two

Software Agents

2.1 Conception.....	17
2.2 Properties	19
2.3 Architectures	21
2.4 Knowledge Base	24
2.5 Communication Languages	26

2.6 Agent Classifications	28
2.7 Co-operation	30

Chapter Three

Existing Agent-based Information Systems

3.1 WEBCON	32
3.2 MIND	33
3.3 RETSINA	36
3.4 InfoSleuth	38
3.5 Infomaster	40

Chapter Four

AgentTeam Framework

4.1 Distributed Database Model	42
4.2 Agent Model	46
4.3 Agent Types	53
4.4 AgentCom	55
4.5 User Co-operation Model	56
4.6 Security Concepts	57

Chapter Five

CourseMan Prototype

5.1 Distributed Database Architecture	59
5.2 Agent Architecture	61
5.3 Co-operative User Work	63
5.4 Security Concepts	64
5.5 System Configuration	64

Conclusions.....	67
Abbreviations.....	72
References.....	73

Appendix

A. BNF of AgentCom.....	82
B. Entity Relationship Diagram of the Distributed Database.....	84
C. Data Dictionary of Distributed Database.....	86
D. Network Configuration.....	94
E. Utilised Tools.....	96
F. Software Design of the Prototype.....	98
G. Data Structures of the Knowledge Base.....	101

List of Tables

Table 1	Some Characteristics of the Communication Models	69
---------	--	----



List of Figures

Figure 1.1	Layers of Transparency.....	6
Figure 1.2	DBMS Implementation Alternatives.....	7
Figure 2.1	A Complete Utility-based Agent	22
Figure 2.2	Structure of a Mobile Agent	23
Figure 2.3	A Taxonomy of Agent Technologies.....	27
Figure 3.1	WEBCON Architecture	33
Figure 3.2	MIND Architecture.....	35
Figure 3.3	The RETSINA Distributed Agent Organisation	36
Figure 3.4	InfoSleuth Dynamic and Broker-based Agent Architecture	38
Figure 3.5	InfoSleuth Agent Layers	39
Figure 3.6	Infomaster Agent Architecture	41
Figure 4.1	Abstraction Levels over Heterogeneous DBMSs	43
Figure 4.2	Domain of the Framework AgentTeam	44
Figure 4.3	Database Connectivity Model of AgentTeam.....	46
Figure 4.4	Architecture and Environment of an AgentTeam Agent	48
Figure 4.5	Life-cycle Model of an AgentTeam Agent	48
Figure 4.6	Knowledge Base Structure of an AgentTeam Agent.....	50
Figure 4.7	Hierarchical Structure of the DBMS Template	50
Figure 4.8	Communication Model of an AgentTeam Agent	52

Figure 4.9	Communication Structure of the Agent Types	54
Figure 4.10	Three Levels Co-operation Model of AgentTeam.....	57
Figure 5.1:	Wrapping between the Data Representation Forms	62
Figure 5.2:	User Interface of CourseMan.....	63
Figure 5.3	Current Network Configuration of CourseMan.....	65
Figure B.1	Conceptual Scheme as E/R Diagram of the University Course Enterprise	84
Figure D.1	Current Machine Configuration, Data Distribution, and Access Rights in CourseMan.....	95
Figure E.1	Basic Tools and their Connectivity in CourseMan.....	96
Figure F.1	Class Diagram.....	99
Figure G.1:	Node Structure of a Semantic Node	101
Figure G.2:	AgentCom Syntax Represented in a Semantic Net Template	102

Introduction

Traditionally, a distributed database is designed in a top-down or a bottom-up fashion and the management system is designed to guaranty global data integrity. A tight communication between the components of such a management system is necessary, in order to hold the distributed data consistent [Özsu 91]. However, for growing number of involved sites, global data integrity becomes unreasonable, because of the growing communication overhead [Stonebraker et al. 96].

Different techniques have been suggested to relax the guarantee for global data integrity temporarily, to provide for more system flexibility [Desai 90], [Stonebraker et al. 96]. In the extreme case of distributing a database management system, the local databases are autonomous and global database management is performed in form of co-operation among involved database systems. An example is the current situation in the Internet, where, caused by heterogeneity, the database management systems are not even capable to communicate with each other. A standard protocol that could enable such a communication should first build a homogeneous layer [Özsu 91], [Genesereth et al. 97], and then it should enable for distributed database management.

Since, the agent concept defines an autonomous, communicative, and problem solving entity, that exists in a distributed environment, it promises to be a suitable approach to implement distributed database management with agents. The agent concept is classified as a subsidiary of artificial intelligence and discussed in the literature under distributed artificial intelligence, in form of multiple agent systems and their properties [Green et al. 97], [Norvig 95]. The main objective of multi-agent systems is distributed problem solving, whereas that of distributed database management systems is the management of distributed data. However, for both there are some similarities, such as the heterogeneous

environment, in which the system entities operate, autonomy, which the system entities possess, and co-operation, which is required for each involved entity to solve local problems.

The idea to combine agent technology with databases have already been discussed in the literature [Kandzia et al. 97]. Agents as components of a database management system, such as query optimiser or storage manager, can introduce more flexibility into the system as well as for the system designer [Akker et al. 97]. Concepts for such systems are for example, a multi database management system with components designed as agents and where in a centralised approach global integrity is guaranteed [Doğaç et al. 98]. A further example is a multi reactive database system, where the rules of each database are considered to be implemented by an agent and where the system goal is, like in the former example, to guarantee global data integrity centralised [Babin et al. 97].

This work is an attempt to combine the computer science and engineering disciplines distributed database management and agents in a framework that is suitable for heterogeneous and dynamic networks.

For collecting information material, following research disciplines have been of interest in this work:

- Distributed databases management: Distributed, heterogeneous, multi databases
- Artificial intelligence: Knowledge representation, distributed problem solving (intelligent agents)
- Software engineering: Object-oriented software design and implementation
- Distributed systems: Co-operative work, co-ordination, information retrieval, network-centric systems

The two major goals of this work are the:

- Development of a framework for connecting multiple databases and to provide for distributed database management
- Implementation of a prototype that can serve as a test-bed for experiments and for further extensions in terms of distributed database management and agent capabilities

Some requirements for the framework are:

- Provide a homogeneous working environment for co-operative work of a user team based on shared distributed databases
- Enable users to create their own views on distributed databases
- Enable users to handle with dynamic data structures of the distributed database

Some requirements for the test-bed are:

- The test-bed itself should be a distributed system
- The scalability of the test-bed should be unlimited, with respect to involved hosts and data sources

We assume that a heterogeneous network is characterised by differences in:

- Native data representation
- Native programming languages
- Operating systems
- Network transport protocols

In this work, frequently in database terminology used term “user”, we interpret as any kind of user of data or system functionality, such as an application, an administrator, or an end-user.

The structure of this work is organised as follows. In chapter one, current theory of distributed database management is reviewed, in chapter two, current theory about the agent concept is discussed. In chapter three, a review on some existing systems is given, which attempt to combine distributed database management with agent technology. In chapter four, the framework AgentTeam is introduced, which is our solution model for the thesis. In the last chapter the prototype implementation of AgentTeam, namely CourseMan, is discussed. Finally, the work concludes with an evaluation of the work and some suggestions for further work. Implementation-oriented details are listed in the appendix.

Chapter One

Distributed Database Systems

Theory, architectures, and algorithms that underlay distributed database management systems are summarised in this chapter. The aim at reflecting here discussions from the related literature is to identify the design issues for our framework. However, we evaluate these issues from the viewpoint of distributed multi database systems, since this is the chosen implementation model for the framework.

1.1 Distributed Database Management

Though implementation models are discussed later in this chapter, we use the acronym DDBMS hereafter, in place of distributed, heterogeneous, multi database or database management system, even if the currently discussed system only partially satisfies these properties.

The major functionality of distributed database management is concerned with distributed query processing, concurrency control, distributed commitment and recovery, deadlock detection, query optimisation, security, and heterogeneity.

Advantages

A detailed discussion of advantages and disadvantages of distributed systems and DDBMS can be found in [Tanenbaum 92] and [Desai 90] respectively.

- *Sharing*: Users at a given site are able to access data stored at other sites and at the same time retain control over the data at their own site.
- *Availability and reliability*: Even when a site is down, the system remains available. With replicated data, the failure of one site still allows access to the replicated copy

of the data from another site. The remaining sites continue to function. The greater accessibility enhances the reliability of the system.

- *Incremental growth*: As further information is needed, new sites can be added independently from already existing sites.
- *Parallel evaluation*: A query involving data from several sites can be subdivided into sub-queries and the parts evaluated in parallel.

Disadvantages

Disadvantages of DDBMSs over centralised ones are:

- *Co-operation mechanisms needed*: Each site must provide connectivity mechanisms in order to enable mutual communication. A common communication language for involved sites is required. Dedicated control mechanisms for distributed systems are necessary, to allow for co-ordinated work.
- *Transaction time overhead*: Due to time consuming message transmission, transaction times over the network usually result in multiples of the time that would be needed for a local transaction.
- *Operation overhead*: Sum of all operations is a magnitude greater than the sum of operations in a centralised environment where no co-operation and co-ordination procedures must be performed.
- *Synchronisation mechanisms needed*: Concurrent update operations must be synchronised, in order to guarantee data consistency.

1.2 Distributed Database System Architecture

Various architectural models have been proposed and discussed for DDBMSs [Desai 90], [Özkarahan 97]. Two of them are generalise frameworks that represent the major architectural structures common to the most DDBMSs. One emphasises user's perspective by building several functional abstraction levels upon data. The other classifies DDBMSs according implementation alternatives. Both are discussed in the next sub-chapters. A further one is the standardisation effort ANSI/SPARC that, from the perspective of data organisation, builds the functional abstraction levels internal view, conceptual view, and

external view. The first one deals with the physical storage and location of data. The second level deals with relationships between the data. The highest level is devoted to modelling the interface between system and end-user. Further discussions on architectures can be found in [Özkarahan 97].

Layered System Transparencies

System transparency hides lower-level details of an abstraction. Usually, an abstraction is made according to a concept. Depending on the system domain, concepts may vary, resulting in different abstractions over the same details. Also, a hierarchy of abstractions is possible, by abstracting an abstraction further according another concept. For DDBMS following layers of transparencies are being suggested [Özsu et al. 91] (Figure 1.1).

Data independence: Refers to the immunity of users from data structures. This means that changes in the data structure or in its storage structures will not effect the user.

Network Transparency: This type refers to the transparency of data from locations in the network, and that for data and other database objects unique names have to be provided.

Replication Transparency: Here, the user should not be involved into the existence of replicated data, the number of replicas, their locations, and how updates on replicas are handled to avoid inconsistent DDBMS states.

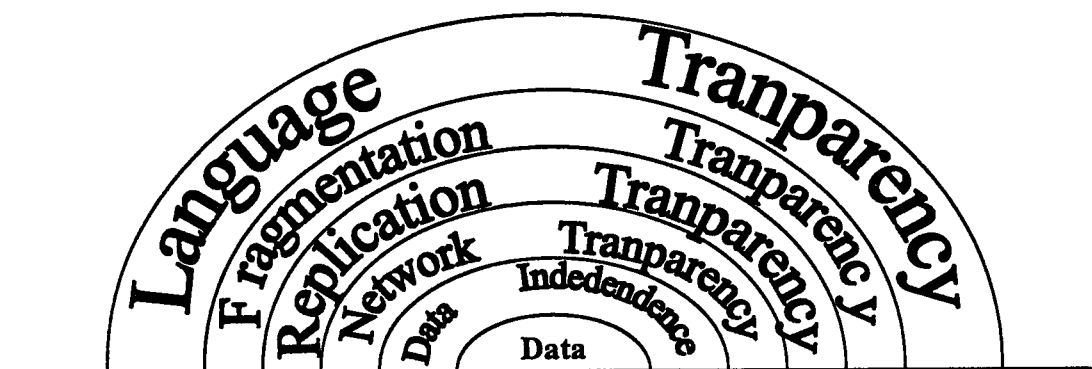


Figure 1.1: Layers of Transparency [Özsu et al. 91]

Fragmentation transparency: Hides the facts that data objects may be fragmented in the DDBMS, their locations, and how the system handles with queries that were specified on the entire data object.

Language transparency: In this abstraction, users need not to know traditional data access languages, such as SQL or QUEL. Instead, they can access data over more flexible and user-friendly languages, such as fourth-generation languages and natural languages.

Implementation Alternatives

A further architectural issue is the implementation of a DDBMS [Özsu et al. 91]. A classification of DDBMSs with respect to the autonomy of local systems, their distribution, and their heterogeneity is depicted in (Figure 1.2). Any alternative along the three coordinates and their combinations is possible.

Autonomy: Refers to the degree of control granted to participating DBMSs inside a DDBMS. It indicates the degree to which individual DBMSs can operate independently. The scale may vary from total isolation to tight integration of a DBMS. In the former case,

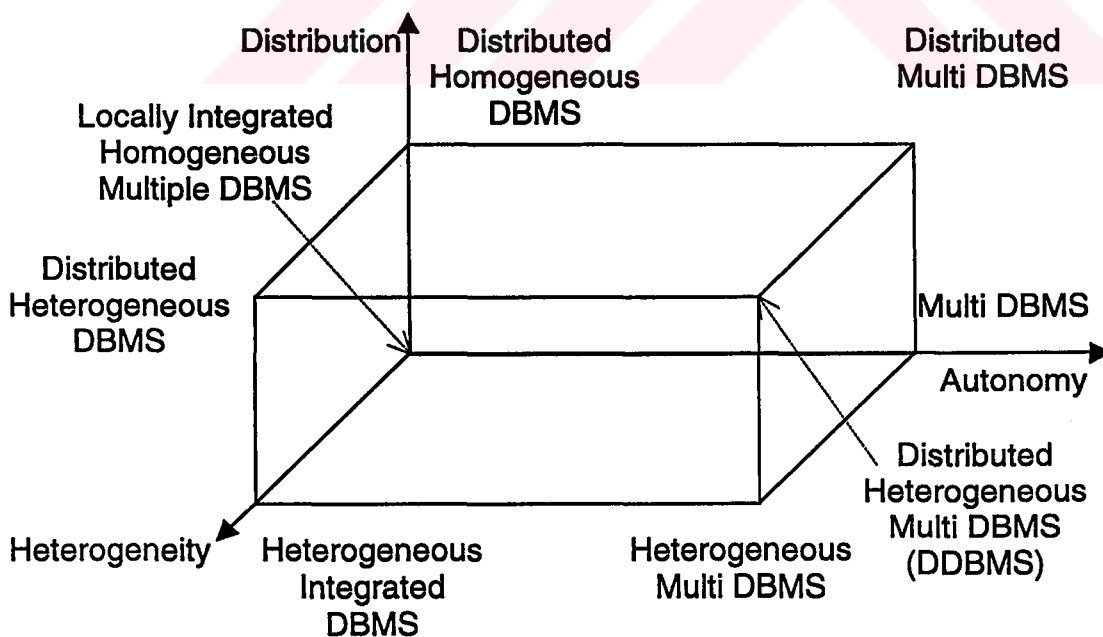


Figure 1.2: DBMS Implementation Alternatives [Özsu et al. 91]

the DBMS is not aware of other DBMSs and it does not export any of its control mechanisms to be used from outside. In the latter case, a single image of the entire database is available to any user who wants to share the information, which may reside in multiple databases.

Distribution: Deals with the distribution of data. Only two cases are considered, data may be stored central on one machine or it may be distributed over more than one machine.

Heterogeneity: Heterogeneity may occur in various forms, ranging from hardware heterogeneity, different operating systems, different network protocols to variations in data management, such as data representation forms, query languages, transaction management protocols.

1.3 Distributed Database Design

Three properties influence the design of DDBMSs: sharing, access pattern, and level of knowledge about the access pattern. Sharing considers whether data and programme will reside private on a site, or whether they will be shared in the DDBMS. Access pattern considers whether user requests may be static or dynamic. The latter one considers the degree of information about user access patterns in the design phase.

Two alternative strategies have been identified for designing distributed databases [Özsu et al. 91]: top-down approach and bottom-up approach.

Top-down design: In the first design phase a new database is designed, in the second phase its components are distributed. The result of the first phase is a global conceptual schema. The results of the second phase are the local conceptual schemas.

Bottom-up design: This design strategy involves integrating already existing databases into one database. Starting at the bottom, the individual local conceptual schemas are integrated into the global conceptual schema. This type of design is suitable for integrating heterogeneous databases.

1.4 Data Distribution

Criteria for data distribution are performance, reliability, and availability considerations, which can be summarised and expressed in form of access costs. Access to local data is usually faster than access over the network and provides for highest availability. On the other hand, network connections are more failure-prone. Depending on which portion of data is needed on a site and how it will be used there, two methods are differentiated that provide practical solutions for data distribution: fragmentation and replication.

Fragmentation

A fragment is a portion or region of data that is frequently accessed by a user. Fragmentation algorithms deal with finding such data fragments and with managing their distribution and with integrating them again. Some common fragmentation strategies are discussed below. Further details on fragmentation strategies and algorithms can be found in [Özsu et al. 91].

Horizontal fragmentation: Partitions a relation along its tuples. Primary horizontal fragmentation is performed using predicates that are defined on that relation. Whereas, derived horizontal fragmentation is the partitioning of a relation that results from predicates being defined on other relations.

Vertical fragmentation: Partitions a table along its attributes. Vertical fragments can be determined, based on data usage values. For example in a statistical approach, by using a clustering algorithm [Özsu et al. 91].

Hybrid fragmentation: In some cases, according to user requirements, it may be necessary to fragment first horizontally, then vertically, or vice versa.

Disjoined fragmentation: In any combination of the above fragmentation types applied onto a table, fragments may be non-disjoined or disjoined; in other words, different data regions may overlap each other or not.

Replication

The decision on whether to replicate data or not, which data to replicate, and whether to allow read-only or write permissions, depends on the specific configuration and the characteristics of each system, and is usually made according to expertise and rules of thumb. In the following, based on some generalised assumptions, replication is discussed in mathematical terms.

For below calculations we assume ideal system conditions, characterised by following properties:

- Serial accesses to a disk result in constant response times.
- Concurrent accesses to a disk result in exponentially growing response times.
- Remote access results in longer response times than local access.
- Remote response times are independent from network load.
- Number of concurrent clients is great.
- In case without replicas, the number of clients is great enough, so that number of concurrent accesses to the server will result in exponentially increasing response times.
- All data is equally worthy to all users, i.e. all data has same information content for all users.

Read-only accesses on replicas: In case of read-only access, allowed on all replicas of the same data, the total access costs will be smaller than the total costs for remote access without replicas. The cost difference will increase with increasing number of accesses.

Thus, under normal conditions following holds: $C_l(i) < C_r(i)$

$C_l(i)$: For the case of replicas, total cost function C for i times local read accesses on all clients in parallel. For increasing i , this function will increase linear.

$C_r(i)$: For the case without replicas, total cost function C for i times remote read accesses from clients concurrently to the server. For increasing i , this function will increase exponential.

For sufficient great i : $nC_d + C_l(i) < C_r(i)$

nC_d : For the case of replicas, total costs C for transferring a replica of data d to all n clients. nC_d is independent form i , and therefore grows only with increasing n .

For very great i : $nC_d + C_l(i) \approx C_l(i)$

And the cost difference increases dramatically: $C_l(i) \ll C_r(i)$

Write accesses on replicas: In case of one write access, made on one replica and read-only accesses on all the other replicas, additional costs for updating all replicas niC_u , including the original data on the server, must be calculated: $nC_d + (i+1)C_l + niC_u$

For sufficient great i : $nC_d + (i+1)C_l + niC_u < C_r(i+1)$

For very great i : $iC_l + iC_u \ll C_r(i)$

For increasing i , niC_u will increase linear.

For the duration of an update on a replica, the distributed database can enter an inconsistent state: I_u . Where, I_u can be determined by the longest delayed update of all parallel updates.

In case of updates made on different clients' replica j , additional costs for synchronising the concurrent updates $C_s(j)$ must be considered:

$$nC_d + (i+j)C_l + n(i+j)C_u + C_s(j)$$

Where, $C_s(j)$, and depending on it I_u , will grow exponentially with increasing j .

For sufficient great i : $nC_d + (i+j)C_l + n(i+j)C_u + C_s(j)$ increases exponentially, as well as $C_r(i+j)$

For very great i : $C_l(i+j) + C_u(i+j) + C_s(j)$ and $C_r(i+j)$

In both latter cases both functions result in unacceptable high values.

In this case neither distributed synchronisation of replicas nor centralised access without replication becomes acceptable. However, in case of a centralised synchronisation algorithm for replicas, all update requests are resolved by a global synchronisation mechanism, which considerably can reduce network traffic. In this case and with replication, $C_s(j)$, and depending on it I_u , will grow only linear with increasing j .

The final conclusion from above calculations is that, if $(i \gg j) \wedge (j \ll n)$ then replication can be profitable. In any other case and with a distributed synchronisation algorithm, replication would result in exponential increasing additional costs. However, with a centralised synchronisation algorithm, additional costs could be reduced to a linear increasing one.

The problem of performing an update exactly at the same time on all replicas can only approximately be solved, since even in case of a global clock mechanism, the final synchronisation messages sent to the replicas could be delayed in the network. A practical conclusion from above calculations is that, update permissions should be granted only to very few replicas.

An advantage of fragmentation over replication is that parallel updates on different fragments of a table are possible without any synchronisation. Whereas, a disadvantage is that, the query mechanism must be able to logically reconstruct a table from its fragments, for example for further users, how may request several fragments of the same table.

1.5 Transaction Management

A transaction is a consistent and reliable computation on database data that can change the state of the database. Since, transactions can occur concurrently on the same data, a co-ordination mechanism must be provided to manage transactions. This management will be responsible for secure execution of transactions and will move the database from one consistent state to another. To achieve this, a transaction manager has to deal with following situations.

Conditions of transactions: If a transaction was successful, then it commits and all of its updates become permanent. If it was not successful, then it aborts and all of its updates are undone, which is known as rollback.

Characteristics of transactions: Transaction management depends further on the characteristics of transactions. A transaction can include actions to read write, insert, or delete data.

Properties of transactions: The consistency and reliability aspects of transactions are due to the property atomicity, consistency, isolation, durability, and serialisability.

Atomicity refers to the fact that a transaction is treated as a unit of operation. Therefore, either all the transaction's actions are completed, or none of them are. Consistency of a transaction is its correctness. A correct transaction maps one consistent database state to another. Isolation requires each transaction to see a consistent database at all times. In other words, an executing transaction cannot reveal its results to other concurrent transactions before its commitment. Durability refers to that property of transactions, which ensures that once a transaction commits, its results are permanent and cannot be erased from the database. Serialisability refers to the property that if several transactions are executed concurrently, the result must be the same as if they were executed serially in some order.

Architectural components: A distributed execution monitor consists of a transaction manager, a scheduler, and the recovery managers. The first one is responsible for coordinating the execution of the database operations on behalf of an application at client site. The scheduler is responsible for the implementation of a specific concurrency control algorithm for synchronising access to the database. This component usually resides at server site. Recovery managers' functionality is to implement the local procedures by which the local database can be recovered to a failure.

Discussions on the formalisation of the transaction concept can be found in [Özsu et al. 91].

1.6 Distributed Concurrency Control

Concurrency control deals with the isolation and consistency properties of transactions. To manage concurrent access to data a special control mechanism must be provided in a DDBMS. It has to take in account the existence of fragmentation and replication of data that deadlocks may occur in multi-user environment of a DDBMS.

Primitive Mechanisms

Concurrency control algorithms can consist of one or a combination of the following primitive mechanisms: pessimistic or optimistic, locking-based or timestamp-based.

Pessimistic: Pessimistic algorithms synchronise the concurrent execution of transactions early in their execution life cycle. Transaction follows the sequences validate, read, compute, and write.

Optimistic: Optimistic algorithms delay the synchronisation of transactions until their termination. A transaction follow the sequences read, compute, validate, and write.

Locking-based: The main idea of locking-based concurrency control is to ensure that the data shared by conflicting operations is accessed by one operation at a time. This is accomplished by locking the related data. In locking-based systems, the scheduler is a lock manager. Further cases are distinguished according the degree of distribution of the lock manager activities: centralised locking, primary copy locking, and decentralised locking. In centralised locking, one of the sites in the network is designated as the primary site where the lock tables for the entire database are stored. In primary copy locking, one of the copies of the lock tables is designated as the master copy. In decentralised locking, lock tables of related local data are stored locally. Each scheduler is responsible only for the lock tables of its site. One locking-based concurrency algorithm that easily enables resolving concurrency, is strict two-phase locking. First, in the growing phase, locks are obtained and data accessed. Second, in the shrinking phase, all locks are released together.

Timestamp-based: Involves organising the execution order of transactions so that they maintain mutually consistent. This ordering is maintained by assigning timestamps to both the transactions and the data items of the database.

Economy-based: Mariposa is a wide-area DDBMS [Stonebraker et al. 96] that does not employ global synchronisation for distributed fragments and replicas. Instead, its distributed DBMS architecture supports a microeconomic paradigm for query and storage optimisation that seeks a local-optimum scheduling cost. This mechanism causes temporarily inconsistent replicas, but allows for scalability to a large number of co-operating sites.

Deadlock Management

Any lock-based concurrency control algorithm may result in deadlocks, since there is mutual exclusion of access to shared data and transactions may mutually wait on locks. A

deadlock is a phenomenon that can not be resolved by a control mechanism alone. Three methods are known for handling deadlocks: prevention, avoidance, and detection and resolution.

Deadlock prevention: The transaction manager checks a transaction when it is first initiated and does not permit it to proceed, if it may cause a deadlock. To perform this check, it is required that all of the data that will be accessed by a transaction be pre-declared. Since, access to certain data may depend on conditions that may not be resolved until runtime, such a method is not practicable.

Deadlock avoidance: a simple algorithm in avoiding deadlocks is to order the resources according to some access patterns and insist that each process request access to these resources in that order. Another alternative is to make use of transaction timestamps to prioritise transactions and resolve deadlocks by aborting transactions with higher/lower priorities.

Deadlock detection and resolution: A tool in analysing deadlocks is a wait-for graph. It is a directed graph that represents the wait-for relationship among transactions. The nodes of this graph represent the concurrent transactions in the system. An arc $T1 \rightarrow T2$ exists in the graph, if transaction $T1$ is waiting for $T2$ to release a lock. In distributed systems besides the local wait-for graphs a global one is necessary, since two transactions that participate in a deadlock condition may be running at different sites. Deadlock detection is done, by studying the global wait-for graph for the formation of cycles. Resolution is done by the selection of one or more arbitrary transactions that will be pre-empted and aborted, in order to break the cycle.

1.7 Data Security

The topics of semantic data control are view management, security control, and semantic integrity control [Özsu et al. 91]. We will discuss only some aspects of data security. Data security is an important function of a database that protects data against unauthorised access. Data security includes two aspects: data protection and authorisation control.

Data protection: Is required to prevent unauthorised users from understanding the physical content of data. The main data protection approach is data encryption, which is useful both, for information stored on disk and for information exchanged on a network.

Authentication control: Three main actors are involved in authorisation control users, operations on database objects, and database objects. Authorisation control consists of checking whether a given triple (user, operation, object) can be allowed to proceed. The introduction of a user in the system is typically done by a pair (user name, password). In DDBMSs authorisation information of the system may be replicated to all involved sites or each site may maintain only its local authentication information.



Chapter Two

Software Agents

Software agents are a new technology that encompasses various branches of the research fields artificial intelligence, co-operation and co-ordination, intelligent user interfaces, information processing, object-oriented software engineering, and others. Since, agent technology is a rapidly evolving area, there is still no consensus on a model for software agents, like there is one for example for DDBMSs. In this chapter we summarise the architectural properties of various agent models and present them unified in form of a general framework for software agents.

2.1 Conception

For the designer, an agent is an abstraction and solution idea over a small problem inside a considered domain. Similar to the fact that object-oriented programming is a programming paradigm, agent-based software design is a design paradigm

Definitions

Various definitions for software agents can be found in the literature. Most of them have the following two common patterns:

- Each definition is made in context of a specific domain, and in this view it emphasises some agent properties more and some less.
- Most definitions focus on the fact that an agent is an independent object-oriented software entity with uniform functionality.

Some of the definitions of a software agent are given below:

- *Artificial intelligence* [Green et al. 97]: A computational entity, which acts on behalf of other entities in an autonomous fashion, performs its actions with some level of pro-activity and/or reactivity, and exhibits some level of the key attributes of learning, co-operation, and mobility.
- *Artificial intelligence* [Russell et al. 95]: An intelligent agent has multiple goals, limited resources, and a dynamic real-time task.
- *Robotics* [Russell et al. 95]: Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.
- *Information retrieval* [Sycara et al. 96]: A programme that acts on behalf of its human user in order to perform laborious information gathering tasks.
- *Interface agents* [Çağlayan et al. 97]: A personal assistant that performs tasks on behalf of a user.
- *Business applications* [Çağlayan et al. 97]: A computing entity that performs user delegated tasks autonomously.
- *WebMate* [Chen et al. 97]: A stand-alone proxy that can monitor user's actions to provide information for learning and search refinement, and can interact with a user.

Advantages

The technical advantages will become apparent with the discussion of agent properties in the next section. Here, some of its user-oriented qualities are mentioned.

- *Efficiency*: It can compute any amount of dynamic and unstructured information and transform it to the desired information.
- *Object-orientation*: An agent is an object-oriented entity. It hides the complexity of its tasks and data structures and presents a uniform functionality.
- *Acceptance*: Because of its user friendly appearance and behaviour, agents usually enjoy better user acceptance.
- *Dynamic adaptation*: Agents have the ability to autonomously react to changes in their environment. For example changing user behaviour or changing information structures.

Disadvantages

An agent is a new software concept in which many beneficial software technologies are being tried to combine. Until agents and developers have reached a sufficing high maturity, following disadvantages of agent technology can be observed:

- *Expensive*: An agent is a synthesis of high-quality software properties. Currently, it is more costly to develop an agent than a conventional software programme.
- *Insufficient behaviour*: Most existing agents possess actually only a few of the suggested software attributes.
- *Process time overhead*: When multiple agents are involved in a task, additional time for co-operation and co-ordination is required.

2.2 Properties

Most of the characteristics of agents are already mentioned in their definitions. In our discussion, we separate agent properties for which a mathematical model was available from those, which were described informally.

Formal Definitions

A formal description of the following properties in Z notation is given in [Goodwin 94].

General agent properties:

- *Capable*: An agent is capable, if it possesses the effectors needed to accomplish the task.
- *Perceptive*: An agent is perceptive, if it can distinguish salient characteristics of the world that would allow it to use its effectors to achieve the task.
- *Successful*: An agent is successful to the extent that it accomplishes the specified task in the given environment.
- *Reactive*: An agent is reactive, if it is able to respond sufficiently quickly to events in the world, to allow it to be successful.
- *Reflexive*: An agent is reflexive, if it behaves in a stimulus-response fashion.

The first three are related to each other, so that successful implies perceptive, and perceptive implies capable.

Deliberative agent properties:

- *Predictive*: An agent is predictive, if its model of the world works is sufficiently accurate to allow it to correctly predict how it can achieve the task.
- *Imperative*: An agent is interpretative, if it can correctly interpret its sensor readings.
- *Rational*: An agent is rational, if it chooses to perform commands that it predicts will achieve its goals.
- *Sound*: An agent is sound, if it is predictive, interpretative, and rational.

Informal Definitions

The following is a brief list of most of the properties discussed in the literature [Sycara et al. 96], [Çağlayan et al. 97], [Green et al. 97], [Russell et al. 95].

- *Delegated/taskable*: The agent can perform a task on behalf of a user or another agent.
- *Autonomous*: The agent can operate without direct intervention to the extent of the user's specified delegation. Autonomy can range from simple knowledge-based decision making to negotiation.
- *Monitor/perceptive*: The agent is able to monitor/perceive its environment.
- *Actuation*: The agent is able to affect its environment.
- *Communication*: The agent is capable to communicate with users or other agents.
- *Active*: The agent can initiate problem-solving activities.
- *Adaptive*: The agent can learn and self-organise its knowledge structures.
- *Co-operative/collaborative*: The agent can co-operate and/or collaborate with humans and other agents, to exchange knowledge and to resolve conflicts and inconsistencies in knowledge.
- *Persistent*: The agent is capable of long periods of unattended operation.
- *Mobile*: The agent can move itself, including its state and data, between different environments and between different sites.

- *Trustworthy*: An agent should serve user's needs in a reliable way, so that user will develop trust in its performance.
- *Secure*: An agent should protect its task and its knowledge against unauthorised access.

2.3 Architectures

In a functional view three functional system attributes can be identified that influence the architecture and behaviour of agents:

- *Environment*, in which the agent works
- *Task*, assigned to the agent
- *Success*, the agent achieves.

Introducing any of the above properties that were discussed in the previous section into this system, or increasing/decreasing one of them will modify its architecture and behaviour. Usually, this is the way to design a new agent for a specific domain or to adapt an already existing one in a new domain.

We will discuss two architectural models for agents. One model has its roots in artificial intelligence and robotics, and emphasis the intelligent behaviour of an agent [Russell et al. 95]. The other model has its roots in mobile agents, which is a specific discipline inside the agent research community [Green et al. 97].

Robotics

In this view, an agent primarily as a working environment, has sensors for monitoring the environment, has effectors for effecting the environment, and has goals (*Figure 2.1*). While it tries to reach its goals, it makes actions. Its actions are caused by condition-action rules. To model the world and to choose between actions, it maintains its internal states. Further, the agent has utility functions that allow it to distinguish the quality of a decision. These components are also considered as agent classification criteria, which is discussed in the last section of this chapter.

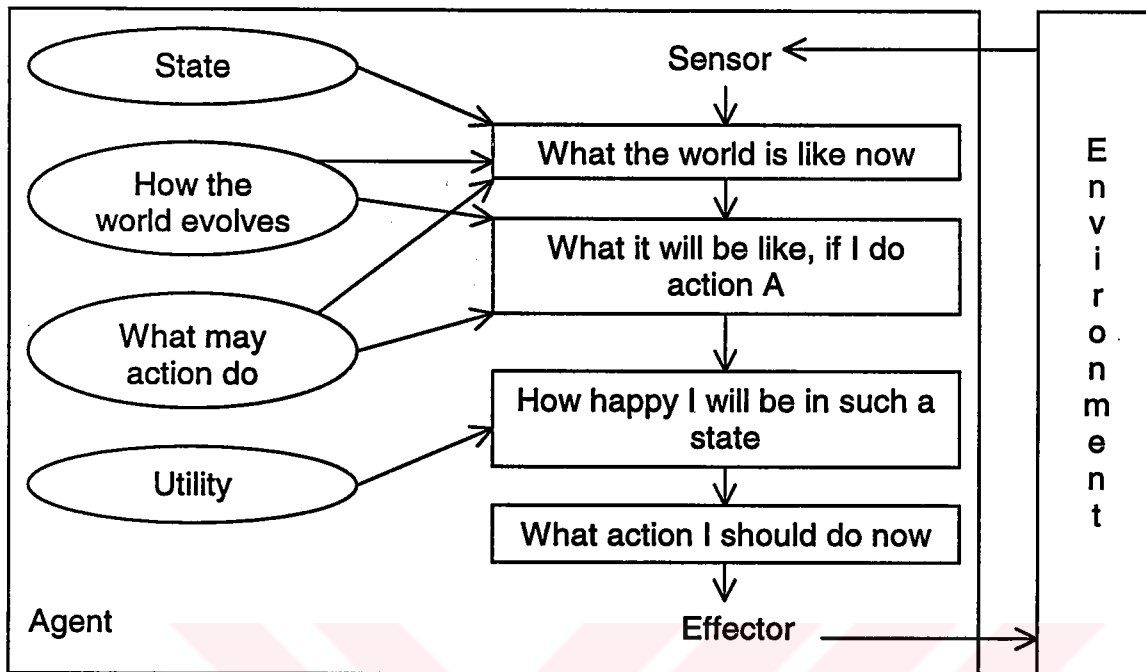


Figure 2.1: A Complete Utility-base Agent [Russell et al. 95]

Mobile Agents

A mobile agent is a software entity, which exists in a software environment. It inherits some of the characteristics of an agent. A mobile agent must contain all of the following models: an agent model, life-cycle model, a computational model, a security model, a communication model, and a navigation model (*Figure 2.2*) [Green et al. 97].

Agent model: This model defines the internal structure of the intelligent agent that is part of a mobile agent. It defines the autonomous, learning, and co-operative characteristics of an agent. Additionally, it specifies the reactive and proactive nature of agents.

Life-cycle model: This model defines the different execution states of a mobile agent and the events that cause the movement from one state to another. Thus, it is closely related to the computational model, which describes how the execution occurs.

Computational model: The computational model defines how a mobile agent executes, when it is in a running state. As part of this model a set of primitive instructions

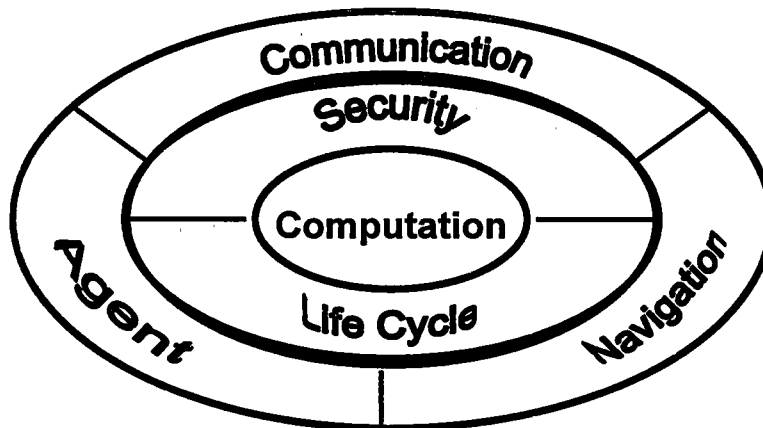


Figure 2.2: Structure of a Mobile Agent [Green et al. 97]

must be specified. This defines the computational abilities of an agent. These include data manipulation instructions and thread control instructions.

Security model: Mobile agent security can be split into two broad areas. The first involves the protection of host nodes from destructive mobile agents, while the second involves the protection of mobile agents from destructive hosts. The former falls into four main categories: Leakage, which is acquisition of data by an unauthorised party. Tampering, which is alteration of data by an unauthorised party. Resource stealing, which is use of facilities by an unauthorised party. Vandalism, which is malicious inference with host's data or facilities with no clear profile to the perpetrator. Standard protection techniques are cryptography, authentication, digital signatures, and trust hierarchies.

Communication model: This model defines the relationships of an agent with other entities in the computing environment. These entities include users, other agents, the host mobile agent environment, and other systems, such as CORBA based distributed systems. Communication is used, when accessing services outside of the mobile agent, during co-operation and co-ordination between mobile agents and other entities, and finally to facilitate competitive behaviour between self-interested agents. Communication protocols can range from simple e-mail and remote procedure call to more complex ones, such as KQML. An agent can implement multiple protocols, which is known as multilingual behaviour. In this case the need for transaction between the protocols comes.

Navigation model: This model concerns itself with all aspects of agent mobility from the discovery and resolution of destination hosts to the manner in which a mobile agent is transported. It should include naming conventions for all entities in the system, such as agents, hosts, services, and sources. Further, access to information regarding a mobile agent environment. Also, the ability to move a mobile agent into a suspended life cycle state ready for transporting to a remote host.

2.4 Knowledge Base

In order for an agent to reason logically, it must be provided with a knowledge base and the related logical operations. A knowledge base is a set of representations of fact about the world. Each individual representation is called a sentence. The sentences are expressed in a knowledge representation language [Russell et al. 95]. This section is a brief overview on knowledge bases and the logical operations that can be performed on it.

Representation Forms

Several representation forms are known in artificial intelligence, such as predicate calculus, semantic nets, frames, conceptual dependencies, and scripts.

Conceptual dependencies: Are structures, which allow representing the dependencies between the components of an action. It is a theory for the representation of the semantics of a sentence in natural language [Rich et al. 91].

Scripts: A script is a generic event description. It is used to represent frequently occurring event sequences.

Predicate Logic

The most popular knowledge representation form that was chosen for logical agents is first order predicate logic. The logic assumes that the world consists of objects with individual identities, properties that distinguish them from other objects, and relations among the objects. A detailed discussion of predicate logic in the context of agents can be

found in [Russell et al. 95]. It will suffice to mention here the operations that are allowed on a knowledge base in predicate logic and to discuss ontology briefly.

Logical operations: Primary operations on data are read and write. To read data from a knowledge base, first some variable values must be provided to the inference mechanism. Then the inference deduces matching properties of the knowledge base. A rule stored in the knowledge base can be preconditioned with these properties. If the precondition of a rule becomes true, then the rule is evaluated. The evaluation can contain any kind of actions. Writing data into a knowledge base can be done by modifying, inserting, or deleting a variable, a property, or a rule. Before writing data into a knowledge base, a consistency check must be performed. Any qualitative update of the knowledge base is denoted learning.

Ontology: ontology is a specification of a conceptualisation. The term is borrowed from philosophy, where ontology is a systematic account of existence. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects and the describable relationships among them represents ontology [Gruber 94].

Semantic Nets

A semantic net is a labelled digraph used to describe relations, including properties of objects, concepts, or actions. This is a more structured approach, which states collecting together facts about particular objects and event types, and arranging the types into a large taxonomic hierarchy analogous to a biological taxonomy. In frame systems objects are represented as nodes in a graph and links between the nodes represent binary relations. In frame systems links are thought of as slots in one frame, whereas in semantic nets they are thought of as arrows between nodes. The meaning and implementation of the two types of systems can be identical. An arrow between two nodes of a semantic net can also be represented as a predicate. It is widely accepted that any semantic net can principally be transferred in a representation in first order predicate logic [Schalkoff 90].

2.5 Communication Languages

An agent communication language is used by intelligent software agents in order to exchange knowledge and/or to co-ordinate co-operative work. It is designed specifically to describe and facilitate communication among two or more agents. An alternative approach is the blackboard as a flexible common communication channel [Engelmore et al. 88], [Cohen et al. 94], [Demirörs 95]. Several agents can post information to the blackboard or read from there concurrently. Main drawbacks of such systems are that concurrent access to the blackboard can easily become a bottleneck. Further, a blackboard system is inherently centralised and therefore not very suitable for distributed systems. A taxonomy for agent technology classifying agent communication languages is depicted in (Figure 2.3). Discussions on agent programming languages can be found in [Dam 97].

Requirements

Following requirements have been identified for an agent communication language [Mayfield et al. 95].

Form: An agent communication language should be declarative, syntactically simple, and readable by people. It should be concise, yet easy to parse and to generate. It should be linear or easy to translate into a linear character stream. It should be extensible.

Context: A communication language should be layered in a way that fits well with other systems. In particular, a distinction should be made between the communication language, which expresses communicative acts, and the content language, which expresses facts about the domain.

Semantics: Language semantics should be unambiguous. It should exhibit canonical form. Because a communication language is intended for interaction that extends over time among spatially dispersed applications, location and time should be addressed by the semantics.

Implementation: The implementation should be efficient, both in term of speed and bandwidth utilisation. The interface should be easy to use. It should be easy to integrate or build application programme interface for a wide variety of programming languages.

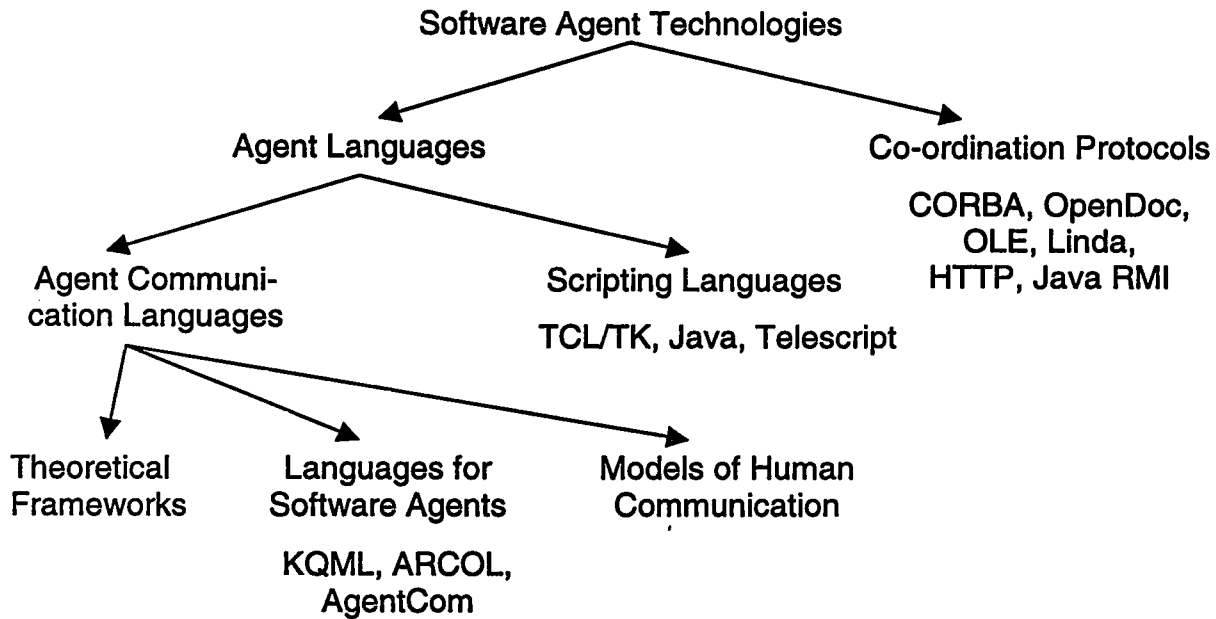


Figure 2.3: A Taxonomy of Agent Technologies [Mayfield et al. 95]

Networking: The language should support all of the basic connection types, point-to-point, multi cast, and broadcast. Both synchronous and asynchronous connections should be supported.

Environment: To provide a communication channel to the outside world, which will be distributed, heterogeneous, and dynamic. A communication language must provide for coping with them. It must support interoperability with other languages and protocols.

Reliability: A communication language must support reliable and secure communication among agents. Provisions for secure and private exchanges between agents should be supported. There should be a way to guarantee authentication of agents.

KQML

The knowledge Query and Manipulation Language (KQML) is a language and protocol for exchanging information and knowledge [Mayfield et al. 95]. It is a language that is designed to support interaction among intelligent agents. Communication takes place on three layers. Interaction protocol for co-ordination purposes, communication

language to exchange knowledge, and the transport protocol that is the actual transport mechanism, such as TCP, SMTP, HTTP. The message primitives are called performatives and are based on human speech acts. The syntax of KQML is based on a balanced-parentheses list. Each performative can have several arguments in form of keyword/value pairs. KQML has currently more than forty performatives [Finin et al. 93]. It has been implemented in several multi-agent systems, some of which will be discussed in the next chapter. Each implementation of KQML resulted in different language dialects [Odubiyi et al. 97]. In existing systems intra-system communication is reported to be successful. But, because of different implemented language semantics, inter-system communication seems currently to be the major problem [Nodine et al. 98b]. But, effort is made to standardise semantics for KQML [Labrou et al. 97].

ARCOL

Based on the experiences with KQML, ARtimis COmmunication Language (ARCOL) [Sadek et al. 97] has been developed that complies with the Foundation for Intelligent Physical Agents (FIPA) standards [FIPA 97]. FIPA has developed a requirements list containing formal descriptions for language properties. The main objective of the FIPA engagement is to set standards for the syntax as well as for the semantics of an agent communication language. ARCOL has many similarities with KQML. It is based on first order predicate logic and its message primitives are performatives. Unlike KQML, ARCOL utilises modal logic to realise standard language semantics.

2.6 Agent Classifications

Agents can be grouped according some common architectural and behavioural properties. One common classification criterion is based on the properties autonomous, adaptive, and co-operative.

Single-agent systems

Agents that are not capable to co-operate with other agents are stand-alone systems. They can be further classified according their degree of intelligence and their task properties.

According intelligent behaviour [Russell et al. 95]: The following classes are ordered according cumulative functionality.

- *Simple reflex agent*: It works by finding a rule, whose condition matches the current situation, and then doing the action associated with that rule.
- *Agent that keeps track of the world*: Is a reflex agent that maintains its state, in order to have choices for action.
- *Goal-based agent*: Are agents that keep track of the world and that can search, plan, and maintain a goal.
- *Utility-based agent*: Is a goal-based agent that maintains utilities, in order to find the best solutions.

According task properties [Green et al. 97]: All following agents are classified as user or interface agents, but they differ from each other in their assigned tasks.

- *Information filtering agent*: Can filter information sources, which is a single directed communication with the information sources.
- *Information retrieval agent*: Can communicate bi-directional with information sources.
- *Personal digital assistant agent*: Can support user's routine tasks.

Multi-Agent Systems

A multi-agent system is a loosely coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities [Green et al. 97]. An essential property of multi-agent systems is that one common communication language exists and that principally any agent can communicate with any other agent of the system. According agents' willingness to co-operate with each other, two multi-agent systems are differentiated, co-operative and antagonistic ones.

Co-operative multi-agent systems: A system for multiple agents is designed in a top-down fashion. All agents are designed to behave co-operative, in order to reach their group goals. A further essential property of such systems is the existence of a common co-ordination protocol, in order to redirect each other through the overall search space.

Antagonistic multi-agent systems: A system in which multiple already existing agents are involved, is designed in a bottom-up fashion. Since, each agent was designed to independently reach its individual goals, any communication with other agents will be self-interested. The agents do not co-operate they compete. In such systems the agents are not aware of a team goal, if at all there exist a team goal.

Further discussions on co-ordination protocols, negotiation techniques, and on mobile agents can be found in [Green et al. 97].

Commercial Applications

Interesting is also the following classification of commercial available software agents [Çağlayan et al. 97].

Desktop agents: Operating system agents, application agents, application suit agents.

Internet agents: Web-search agents, information-filtering agents, off-line delivery agents, notification agents, and other service agents.

Intranet agents: Intranet-search agents, information-filtering agents, process automation agents, database agents, mobile agents.

2.7 Co-operation

Distributed systems rarely exist in the form of isolated entities nor are they themselves isolated against other systems. Multi-agent systems are inherently communicative systems. The overall system knowledge is distributed over the agents. In order for an agent to proceed its tasks successfully, the agents usually share the overall knowledge. If for some agents in a multi-agent system there exist a team goal, then involved agents will co-operate with each other, to contribute their knowledge.

Collaboration: In the classification of agents, we have already mentioned the co-operative-antagonistic spectrum. We define collaboration as a special case characterised by both co-operation and competition, where the agent has some temporary sub-goals. Any individual goal of a collaborative agent must be a sub-goal that is required to reach either a team goal or an antagonistic goal. In general, an agent will always temporarily behave collaborative with other agents.

Competition: An agent may be communicative, but only for the benefit of its antagonistic goals. In some cases it may co-operate with other agents to contribute its knowledge and to facilitate competitive behaviour, but it does not know about any team goals. A competitive agent knows only its antagonistic goals.

Co-ordination: Co-ordination is the regulation of different activities to find a harmonisation according a goal. In multi-agent systems there is a need for co-ordination, for without it, any benefits of interaction vanish and the agent team quickly degenerates into a collection of individuals with a chaotic behaviour. Essentially, co-ordination is a process in which agents engage in order to ensure a community of individual agents acting in a coherent and harmonious manner. For the implementation global control mechanisms are designed, to co-ordinate the co-operative work of multiple agents. Subject to co-ordination are active and passive system entities that can be accessed concurrently. An active entity may be a service or a functionality of an agent. A passive entity may be data or knowledge of the knowledge base. In general, co-ordination is thought to be orthogonal to computation [Ciancarini et al. 98]. Therefore, various co-ordination languages were developed [Papadopoulos et al. 98b]. According their co-ordination mechanisms they are classified as control-based, event-driven, state- defined, and/or data-driven.

Chapter Three

Existing Agent-based Information Systems

This chapter is a review on five existing agent-based information systems. According to their initial design goals they can be separated into two groups. The focus in the design of the first two systems was to build an information retrieval system and a DDBMS respectively, and to provide the system with simple reflex agents. The focus in the design of the last three was to build multi-agent systems that behave intelligently in information retrieval tasks.

3.1 WEBCON

WEBCON is a commercial tool for connecting an SQL database to the WWW [Zoller et al. 98]. The tool is an application generator that takes information from a database and a user to output customised HTML pages for retrieving data from that database.

Architecture

The connection of the database to a HTML page is realised on server side by using the Common Gateway Interface (CGI) of the Web server (*Figure 3.1*). Customised database views are created with HTML pages by mapping database types to HTML types.

The gateway uses the data dictionary of the DBMS to generate enriched HTML templates. To generate customised standard HTML pages from enriched HTML templates. And finally, to retrieve data from the database. Enriched HTML templates include special commands for communicating with the DBMS. Retrieved data is temporarily stored in

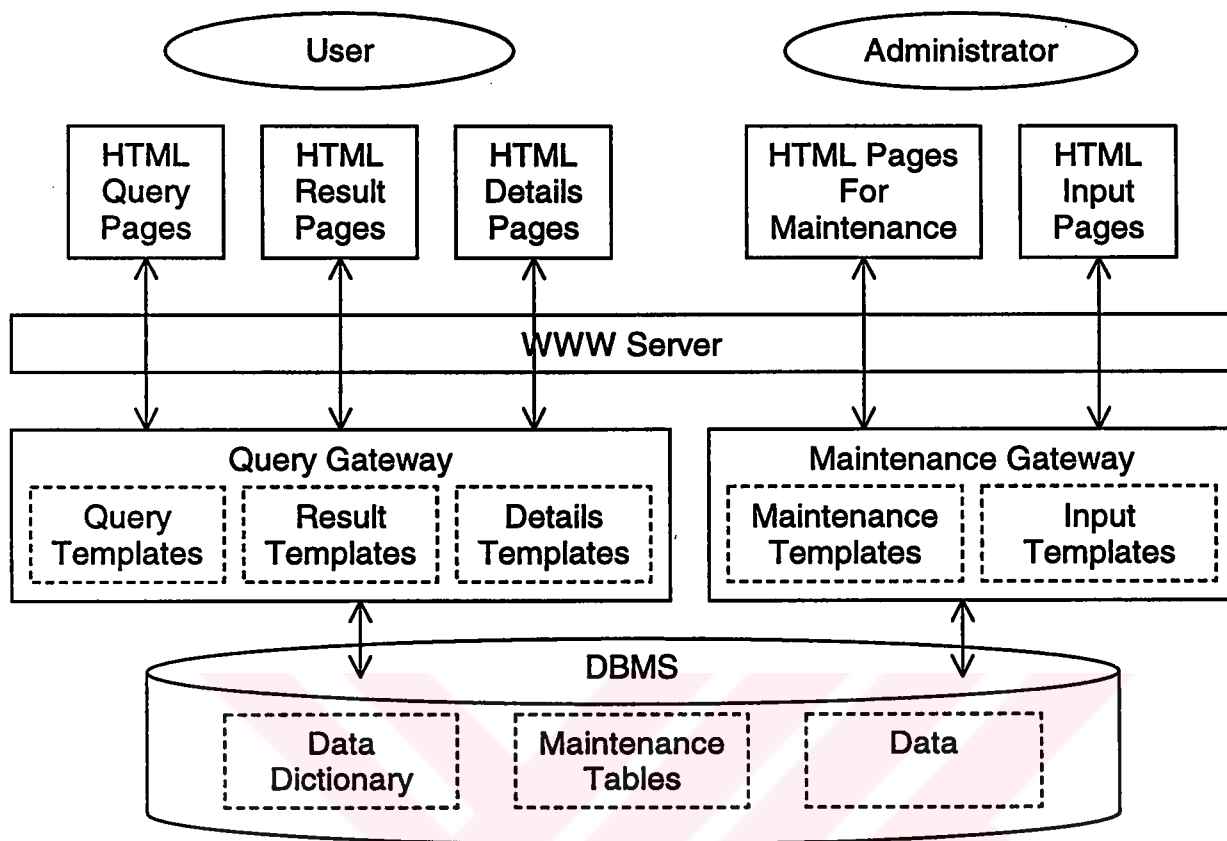


Figure 3.1: WEBCON Architecture [Zoller et al. 98]

HTML result pages. Based on the result pages, sub-queries can be performed in HTML detailed pages.

Two different gateways are used, one for maintenance and one for customised user queries. The maintenance gateway utilises the data dictionary of the DBMS to generate maintenance tables. From maintenance tables and query, result, and details templates according customised HTML pages are created.

Utilisation of the data dictionary makes the tool insensitive to changes within the database schema. Further, this technique provides for installation flexibility against data dictionaries of different SQL DBMSs.

Evaluation

WEBCON does not provide access to multiple DBMSs in one environment. Its query pages are linked only to one DBMS at a time. But, multiple users independently can access the same DBMS over the Internet when each user downloads the HTML pages to its client. In this sense, each customised HTML page can be considered as a database view.

3.2 MIND

METU Interoperable DBMS (MIND) [Doğaç et al. 95] is a multi-database system that is based on a distributed object-computing platform with CORBA (Common Object Request Broker Architecture [Orfali et al. 96]) as the communication mechanism [Doğaç et al. 98]. Global schema integration and transaction management is employed to provide for global data integrity.

Architecture

Each local DBMS is connected to the system over a local database agent (*Figure 3.2*). The local database agents are responsible for maintaining export schemas provided by the local DBMSs represented in the canonical data model, translating the queries received in the global query language to the local query language, and providing an interface to the local DBMSs. This layer provides a virtually homogeneous set of database objects. The global layer of MIND, which contains a global transaction manager, a global query processor and a schema integrator, is developed on top of this layer.

Global transaction management component is responsible for the management of global, distributed transactions. It keeps track of sub-transactions, handles global commit or global abort using two-phase commit protocols over local database agents and detects global deadlocks.

Global query management component is responsible for parsing and decomposing the queries according to the information obtained from schema integration service and for optimisation of the global queries. After a global query is decomposed, the global queries are sent to the involved local database agents.

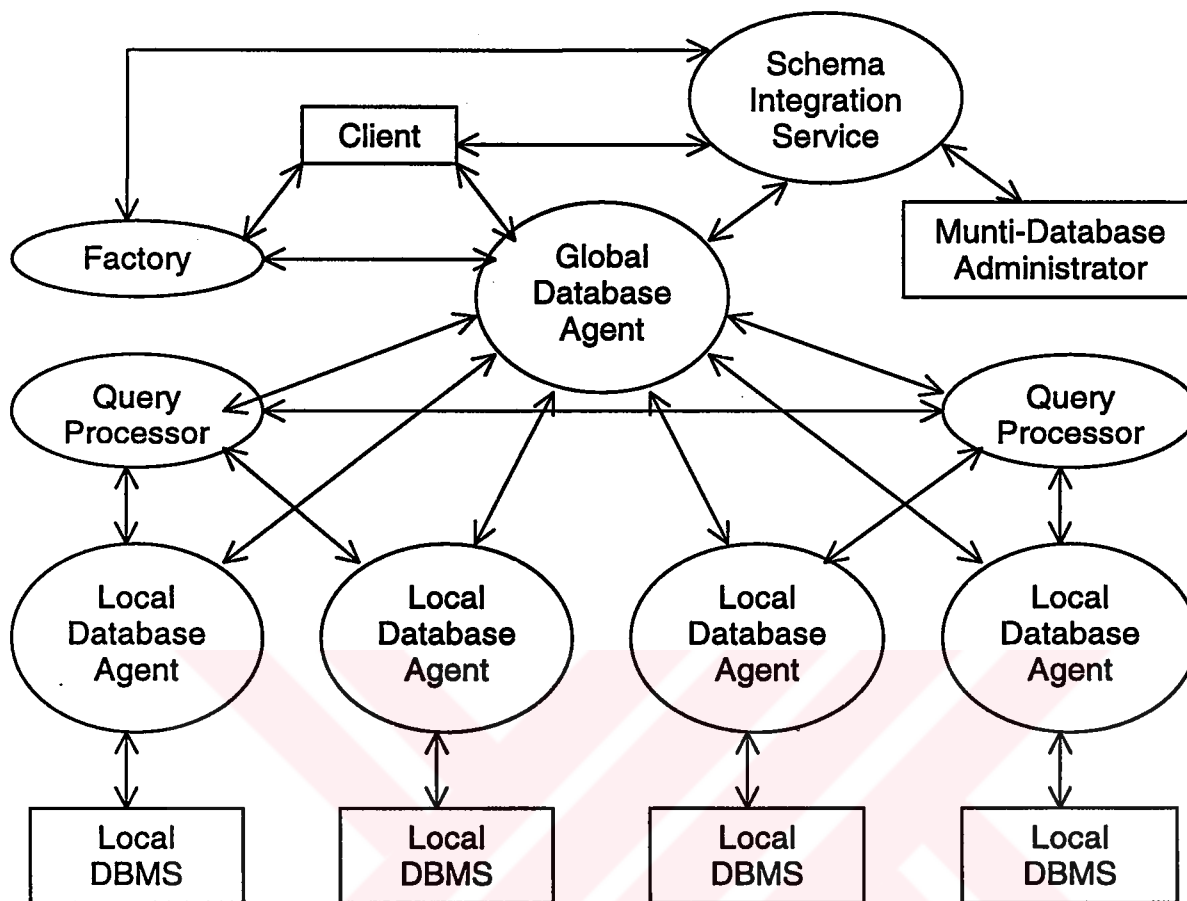


Figure 3.2: MIND Architecture [Doğaç et al. 98]

MIND query optimisation addresses the optimisation of post-processing queries that combine results returned by the local database agents. Query optimisation is performed at run-time by query processor objects. Global query manager may use as many of query processors running in parallel as necessary.

Schema integration service holds the global schema information. The multi-database administrator builds the integrated schema as a view over export schemas. MIND provides the user a common data model and a single global query language based on SQL.

Evaluation

MIND was designed according the traditional approach, where multiple DBMSs are integrated in a bottom-up design [Özsu et al. 98]. However, scalability of such

implementations are restricted, since their centralised global management would result in unacceptable response times for a large number of users and/or a large number of connected DBMSs. The agents in MIND are rather simple reflex agents.

3.3 RETSINA

Reusable Task Structure-based Intelligent Network Agents (RETSINA) is a framework for distributed intelligent agents. It is a distributed adaptive collection of agents that co-ordinate to retrieve, filter, and fuse information relevant to the user, task, and situation as well as anticipating a user's information needs [Sycara et al. 96].

Architecture

RETSINA has three types of agents (*Figure 3.3*). Interface agents interact with the user, receiving user specifications and delivering results. The main functions of an interface

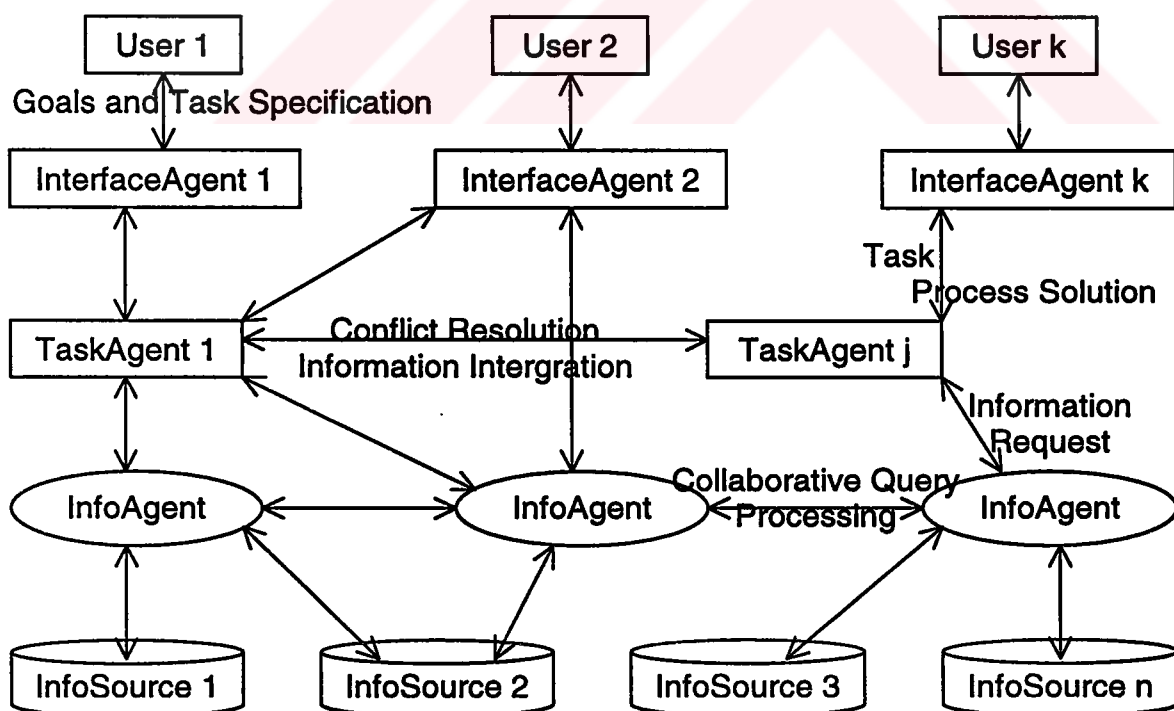


Figure 3.3: The RETSINA Distributed Agent Organisation [Sycara et al. 96]

agent include: Collecting relevant information from the user to initiate a task, presenting relevant information including results and explanations, asking the user for additional information during problem solving, and asking for user confirmation, when necessary.

Task agents help user agents perform tasks by formulating problem solving plans and carrying out these plans through querying and exchanging information with other agents. Task agents have knowledge of the task domain, and which other task assistants or information assistants are relevant to performing various parts of the task. Task assistants have strategies for resolving conflicts and fusing information retrieval by information agents. A task agent performs most of the autonomous problem solving. It exhibits a higher level of sophistication and complexity than either an interface or an information agent. A task agent receives user delegated task specifications from an interface agent. It interprets the specifications and extracts problem solving goals, forms plans to satisfy these goals, identifies information seeking sub-goals that are presented in its plans, decomposes the plans and co-ordinates with appropriate task agents or information agents for plan execution, monitoring, and results composition.

Information agents provide access to a heterogeneous collection of information sources. They have models of the associated information resources, and strategies for source selection, information access, conflict resolution, and information fusion. An information agent's activities are initiated either top-down, by a user or task agent through queries, or bottom-up through monitoring information sources for the occurrence of particular information patterns. Once the monitored-for condition has been observed, the information agent sends notification messages to agents that have registered interest in the occurrence of particular information patterns.

The framework includes KQML for inter-agent communication. According RETSINA for example WebMate [Chen et al. 97] was implemented, which is a personal software agent that accompanies a user when he browses and searches on the WWW.

Evaluation

RETSINA has served as a framework for several multi-agent systems implementing information retrieval tasks. The information agents can communicate with information

sources, but they do not deal with database management. Functionality of DBMSs and DDBMSs is not considered in RETSINA.

3.4 InfoSleuth

InfoSleuth is an architecture and toolkit for deploying agent systems. The InfoSleuth environment focuses on information gathering and analysis over diverse and dynamic networks of multimedia information sources. The emphasis behind InfoSleuth is to establish a stable infrastructure and interaction machinery such that disparate groups and organisations can independently develop agents that meet and work together in the context of an InfoSleuth application [Nodine et al. 98a].

Architecture

The InfoSleuth model defines a framework for loosely collected agents, based on semantic advertisements and then dynamically composing agents based on application needs (*Figure 3.4*).

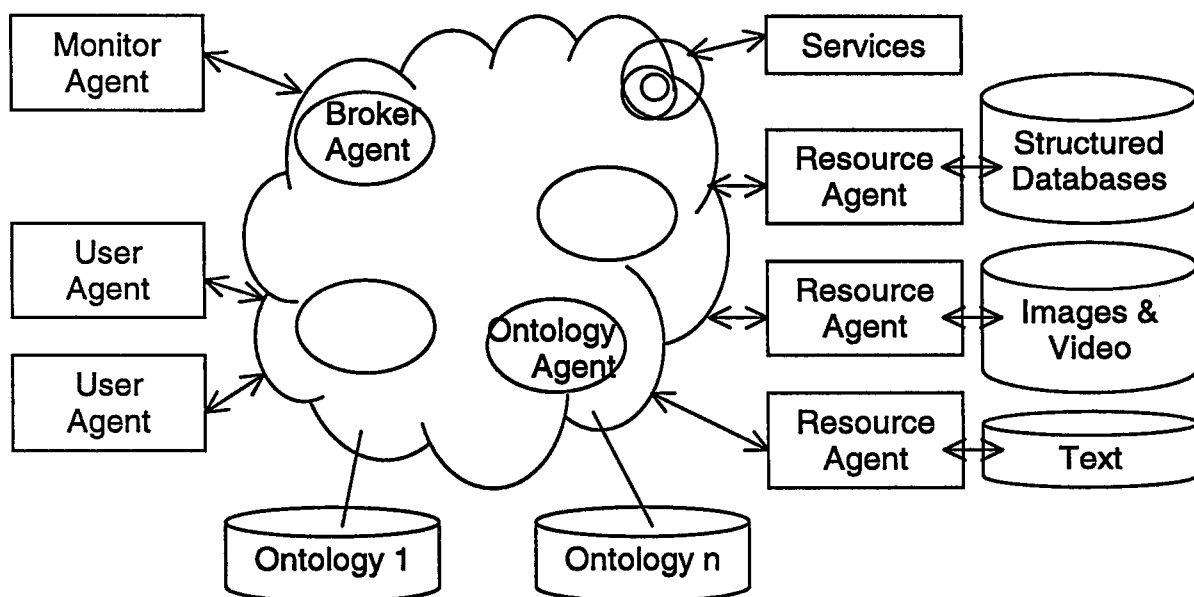


Figure 3.4: InfoSleuth Dynamic and Broker-based Agent Architecture [Nodine et al. 98]

The system consists of a layered agent shell used for rapid and consistent creation and monitoring of agents in an InfoSleuth environment (*Figure 3.5*). The agent message layer maps logical KQML requests made by the conversation layer into and out of physical network exchanges with other agents. The conversation layer defines and enforces conversation policies for a group of co-operating agents. The generic agent layer embodies the services crucial to the operation of all agents that participate in an InfoSleuth community. Within InfoSleuth, there are two such services, the first is the semantic matchmaking service, which enables an agent that is requesting a service, and to locate another agent that can provide that service. The second service provides information on the knowledge accessible through the ontology defined within the community. The agent application layer implements the functionality specific to the agent itself.

It is the intention of the layered agent shell architecture, to allow the agent developer to focus on his effort primarily on the agent application layer, and to inherit the agent shell capabilities and the optional services that it supports.

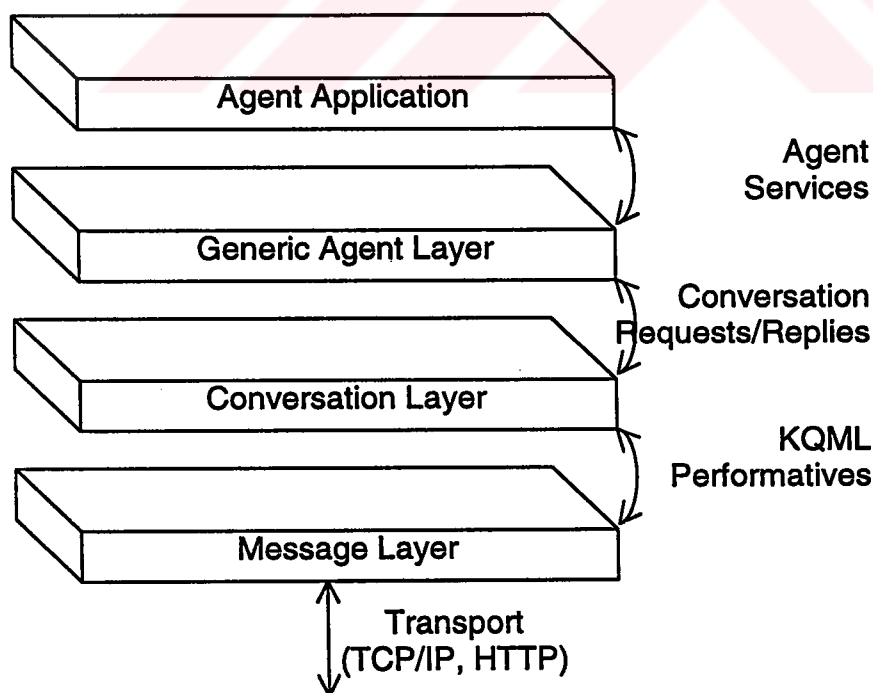


Figure 3.5: InfoSleuth Agent Layers [Nodine et al. 98a]

Evaluation

InfoSleuth is a toolkit for deploying agent applications that focus on information gathering and analysis. The tool does not support the integration of database management functionality into agents to be developed. Remarkable is the tool-based approach of scaling an existing multi-agent system. This provides for an abstraction over the agent communication language and guarantees that the agents are always capable for communicating with each other.

3.5 Infomaster

Infomaster is an information integration system that provides access to multiple distributed heterogeneous information sources on the Internet, thus giving the illusion of a centralised, homogeneous information system. Infomaster creates a so-called virtual data warehouse [Genesereth et al. 97].

Architecture

Infomaster handles both, structural and content translation to resolve differences between multiple applications for the collected data. Infomaster can connect to various databases using wrappers, such as for Z39.50, SQL databases through ODBC, EDI transactions and other WWW sources (*Figure 3.6*).

The core of Infomaster is a facilitator that determines which sources contain the information necessary to answer the query efficiently, designs a strategy for answering the query, and performs transactions to convert source information to a common form or the form requested by the user.

Infomaster uses rules and constraints to describe information sources and translations among these sources. These rules and constraints are stored in a knowledge base. Infomaster has a programmatic interface, which supports KQML, KIF, and vocabularies of terms.

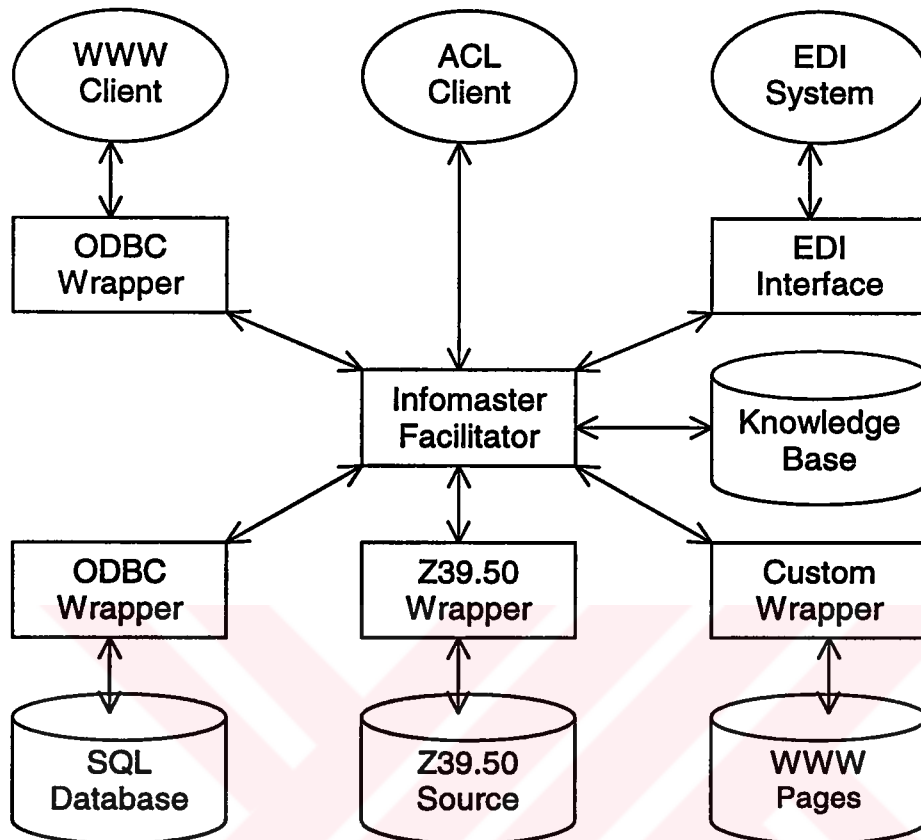


Figure 3.6: Infomaster Agent Architecture [Genesereth et al. 97]

Evaluation

Infomaster is an information integration tool. The system bridges differences in schemata and terminology between existing databases. This makes the system possible to provide a uniform user interface to a collection of heterogeneous information sources. Remarkable is the rule-base reference schema that translates heterogeneous information sources into the knowledge interchange format (KIF). The tool provides rules for semantic evaluation of information, in order to resolve possible inconsistencies in information. Infomaster too, is not concerned with database management.

Chapter Four

AgentTeam Framework

AgentTeam is a framework for a multi-agent system that implements a distributed, heterogeneous, multi DBMS. It is thought for interactive system supporting teamwork of users. The framework is designed in an object-oriented view, where the objects are identified according internally required data structures and major system functionality. In this chapter we introduce the framework consisting of several models. First, the DDBMS model and the agent model are discussed. Thereafter, some important aspects of the agent model are refined within the following sections. These are the agent types, the agent communication language AgentCom, the user co-operation model, and the security concepts of AgentTeam.

4.1 Distributed Database Model

The distributed database consists of shared databases of heterogeneous DBMSs possibly located on different sites. The shared databases are used to store any required data, to store data replicas, and to store data fragments. The functionality of the local DBMSs is utilised by schedulers for local database management on each server. Global database management is performed by a transaction manager and by one or more schedulers in a distributed fashion (*Figure 4.1*). On each client one transaction manager exists. But transaction managers usually do not communicate with each other.

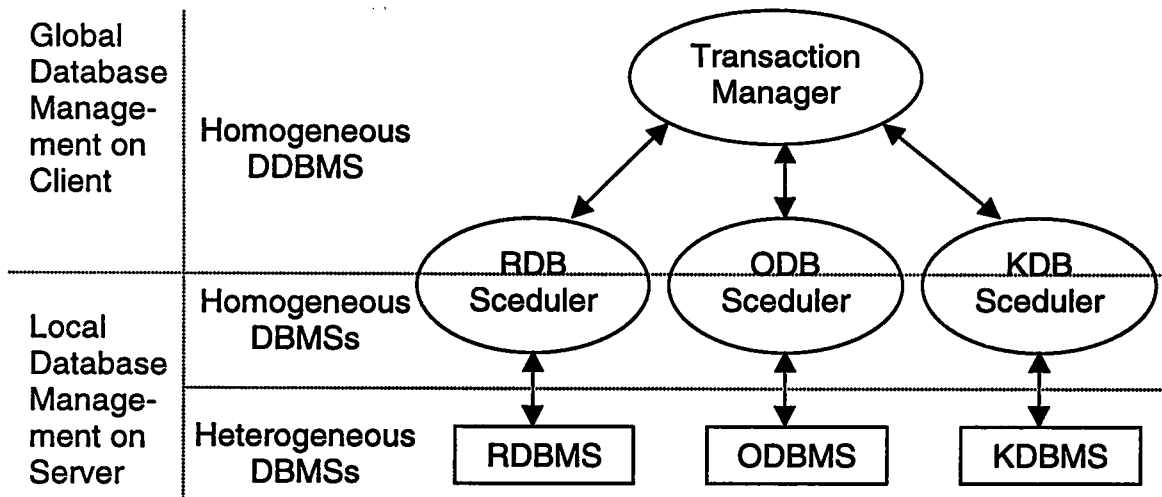


Figure 4.1: Abstraction Levels over Heterogeneous DBMSs

Transaction Management

No global transaction manager exists in the framework. Copies of an identical transaction manager on each involved client and copies of an identical scheduler on each involved database server perform distributed transaction management (*Figure 4.2*).

To handle with failures on sites, distributed commitment is employed in form of two-phase commit. It consists of a voting phase, where sub-transactions are requested to vote on their readiness to commit or abort, and a decision phase, where the decision as to whether all sub-transactions should commit or abort is made and carried out.

Concurrent access to data is resolved with Two-phase locking. First, all lock requests of a transaction are made, than all locks are released.

Each scheduler on server site resolves serialisability of concurrent transactions. Schedulers can communicate with each other to manage replication and fragmentation.

The transaction manager at client site handles the execution of transactions. Based on the data required for a specific transaction, it identifies sub-transactions, delegates them to the related schedulers, and co-ordinates their parallel incoming results.

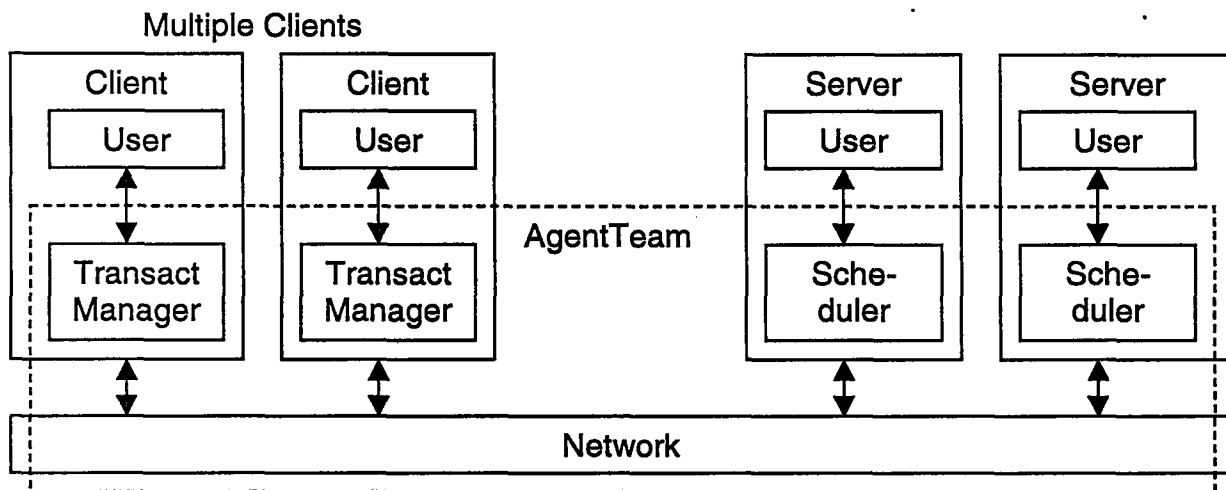


Figure 4.2: Domain of the Framework AgentTeam

Deadlock Handling

There is no mechanism designed to explicitly detect global deadlocks. Instead, a delay-driven mechanism is used to rollback and restart a transaction. This mechanism captures three possible delays:

- Delay caused by network overloading.
- Delay caused by server overloading.
- Delay caused by a global deadlock.

The decision to such a simplified mechanism has practical reasons, which is three-fold. Firstly, network response times are usually a magnitude higher than response times of DBMSs. Though, network protocols themselves can be assumed to be reliable, in some situations a server can cancel a transaction. Under these circumstances it is not worth to implement a global deadlock detection algorithm, which in most cases would save only a tiny amount of time compared with network delays. Thirdly, since this is an interactive system, a user can define the maximum delay time himself, which the user will usually calculate according the current network response times.

Data Distribution

Data managed by the system can be distributed over all involved DBMSs. Both, replication and fragmentation are carried out hidden from the user automated in the background. The objective of both is to achieve locality of reference. Consistency of local data is guaranteed by utilising the related mechanisms of the local DBMSs. Consistency of global data is guaranteed by additional functionality implemented in the schedulers. Schedulers communicate with each other to distribute data and to hold the distributed database consistent. The data distribution policy is determined principally based on data access statistics. A user view may require gathering data from different sites. In this case the transaction manager of this site will establish sub-transactions and will send them to the scheduler of each involved site.

Optimisation Issues

Since, remote data transfer usually is slower than local data transfer, a practical rule is first to reduce the data amount locally then to transfer it over the network. In this sense, following rules of thumb will increase remote data management:

Local filtering: Data should be filtered first, before transferring to another site.

Local joining: If tables to be joined are available locally, then they should be joined first locally using semi-join of the form $E \bowtie R$, where E is an entity and R is the relation.

Attribute transfer: If E and R reside on different sites, then transfer only required attributes, preferably only keys, to the remote site.

Moving less data amount: If E and R reside on different sites, then E should be transferred to the site where R resides. In case of horizontal fragmentation, the fragment with smaller cardinality should be transferred to the site with the greater cardinality.

Database Connectivity

In AgentTeam connectivity of system components conceptually consists of three abstraction layers data, knowledge, and information (*Figure 4.3*). Data is exchanged via the agent communication language AgentCom. Data is represented in AgentCom in form of

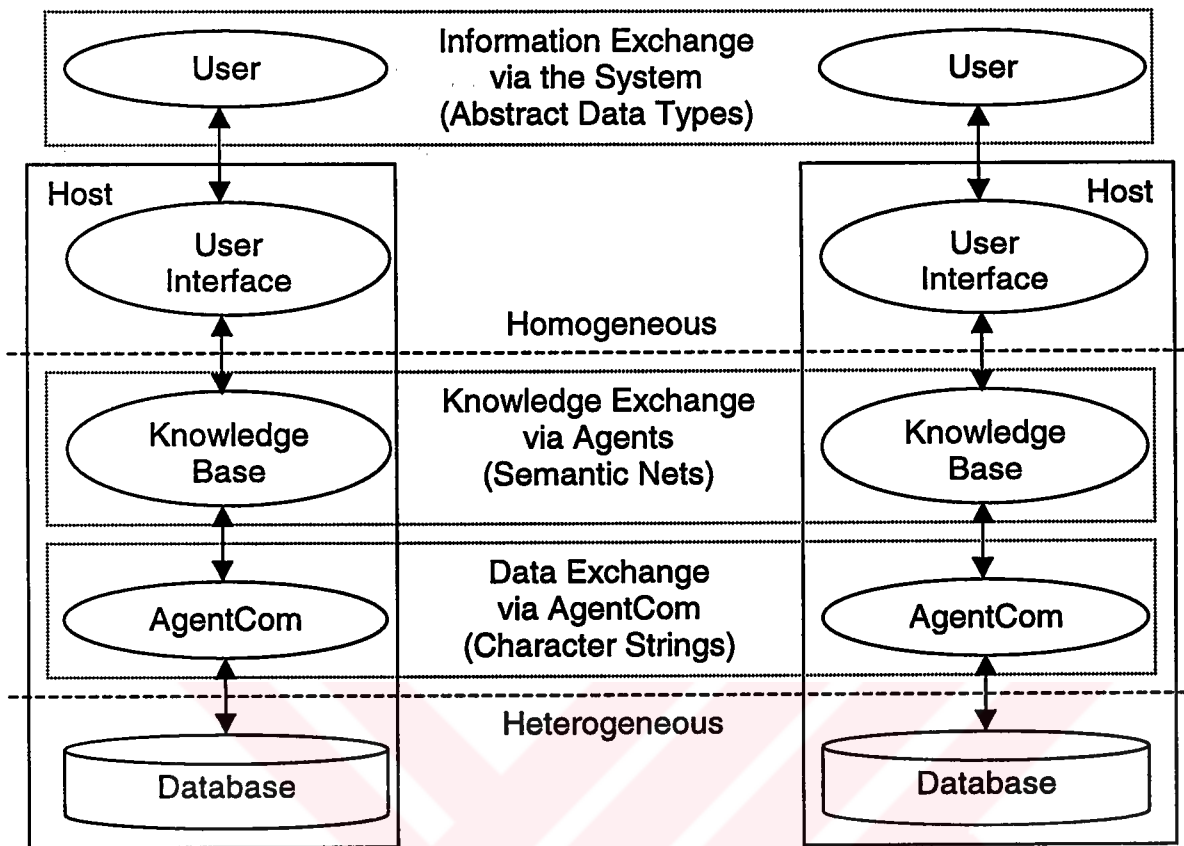


Figure 4.3: Database Connectivity Model of AgentTeam

character strings. Knowledge is exchanged between agents in form of semantic nets. Information is exchanged between users via the system in form of abstract data types.

4.2 Agent Model

This is a model for a single software agent with some properties for successfully proceeding delegated tasks, for reasoning logically, and the capability to communicate in an agent communication language with other agents.

Properties

An AgentTeam agent should possess following properties in order to perform database management functionality successful and sound.

Capable: Can receive tasks over the communication language AgentCom.

Perceptive: Always ready to become active, since it is event-driven.

Successful: Makes the accomplishment of a transaction or sub-transaction always to its goal.

Reactive: Can return results sufficiently quickly, since the workload of a transaction or sub-transaction is distributed over several agents.

Reflexive: Always returns the result of a transaction or sub-transaction.

Predictive: Distributed database management and transaction processing is distributed to different agent types each responsible for specific functionality of the DDBMS. Results

Imperative: Agents can distinguish agent types and know each other's capabilities, so that they will delegate each other correct and interpretable transactions or sub-transactions.

Rational: All agents can reason logically and can optimise the execution of a transaction or sub-transaction.

Architecture

The architecture of an AgentTeam agent consists of several abstract components, including the environment (*Figure 4.4*). The environment of an agent is database management by considering the existence of users, other agents, and various sources. The agent interacts to the environment via communication languages, which is the user interface, the agent communication language AgentCom, various native host languages, and different protocols to access heterogeneous sources. Its internal structure consists of control mechanisms for learning from sample operations, planning the optimal execution of a transaction or sub-transaction, co-ordinating the execution of transactions and sub-transactions, and of the knowledge base. The knowledge base contains particular database management functionality in form of meta-knowledge and all current transactions represented as goals.

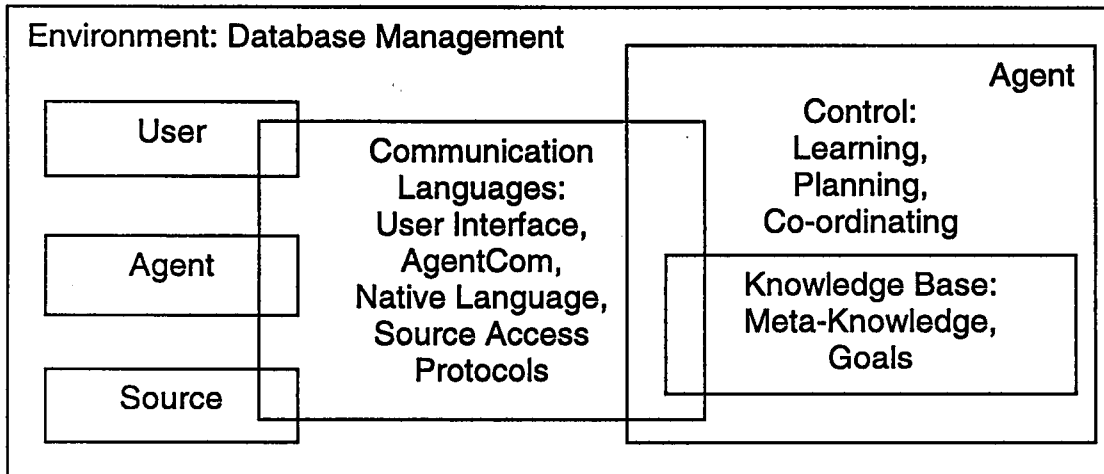


Figure 4.4. Architecture and Environment of an AgentTeam Agent

Life-cycle Model

An agent has four major states sleep, wait, communicate, and evaluate (*Figure 4.5*). The possible states and state transitions are explained briefly below.

Sleeping: An agent is sleeping, when it is currently persistent or is moving. For example when it is currently inactive, because its state and data is stored in a database or is moving from one machine to another machine. From sleeping an agent may transit only to

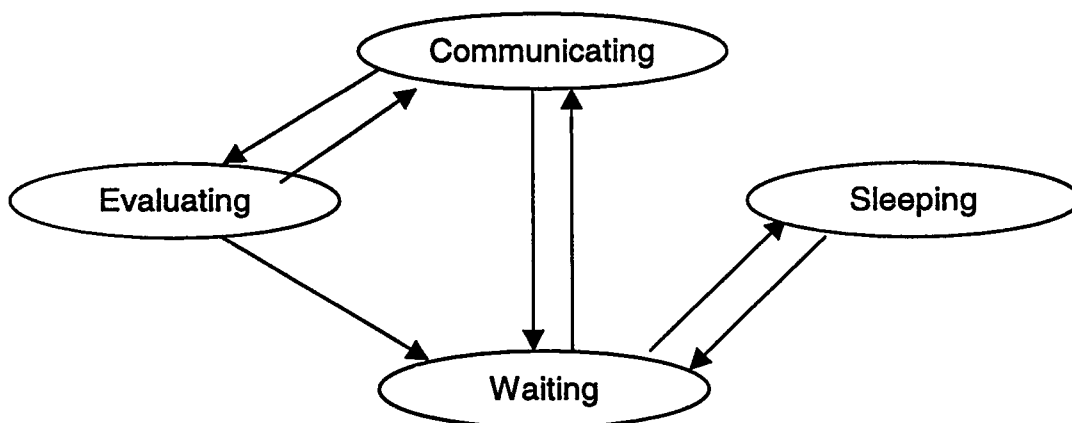


Figure 4.5: Life-cycle Model of an AgentTeam Agent

the waiting state, for example caused by an event.

Waiting: An agent is waiting, if currently no event has activated any of its functionality. From waiting it may transit to sleeping, if no events have occurred for a while. It may transit to communicating, if a communication event, if it receives a communication event.

Communicating: An agent is communicating in all types of interaction with other agents, which can be co-operate, collaborate, or compete. AgentTeam agents are usually always collaborative. It may change to waiting, when for example network congestion occurs. It may change to evaluating, when a source is requested or should be released, or gets any other task delegated.

Evaluating: An agent is evaluating, when transforming a protocol, learning, reasoning, filtering database data, or retrieving database data. It may transit to waiting, when it is expecting further knowledge from other agents, in order to complete its tasks, or if all tasks have been completed.

Knowledge Base

In order to find a general-purpose data structure that is capable for representing a broad range of objects and that can easily be visualised, we have chosen semantic nets. Since, a semantic net can also easily be transferred into first order predicate logic, it can be utilised for logical reasoning. Therefore, knowledge is represented in the knowledge base of an agent in form of semantic nets.

Semantic net: A semantic net is a connected, directed, and labelled graph, where a node represents an object or an attribute value and an arc between two nodes represents a relation, a property, or an operation. A semantic net is always stored as a connected graph.

Knowledge base concepts: Depending on the agent type, the knowledge base of an agent may contain several concepts represented in form of semantic nets. These are the data dictionaries of local databases with references to distributed data, transaction management or transaction scheduling, the communication history of an agent for statistical evaluations,

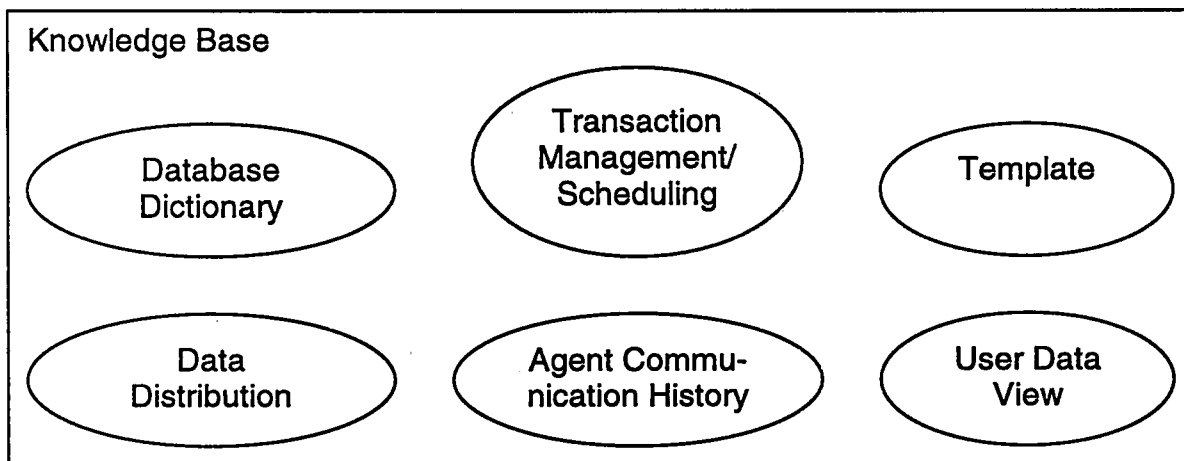


Figure 4.6: Knowledge Base Concepts of an AgentTeam Agent

the user defined data views, and templates for each of these concepts (*Figure 4.6*). Several user defined data views may be stored at the same time in the knowledge base.

Template: A template represents the common structure of a class of concepts and is used to traverse a sample concept of that class currently stored in the same knowledge base. For each class of concepts there must exist exactly one template. For example, the template of the concept RDBMS consists of the node RDBMS on top and one or more nodes for databases, tables, attributes, and values in a hierarchy (*Figure 4.7*). Since, all agent shares

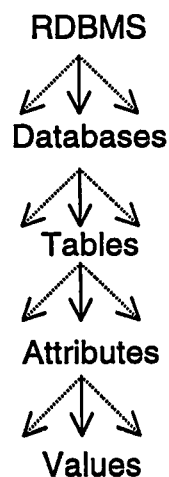


Figure 4.7: Hierarchical Structure of the DBMS Template

the same knowledge base structure and inference mechanism, exchanging a template and one or more related concepts is equivalent with exchanging knowledge between agents. In other words, if two agents store equal templates in their knowledge base, then they will be able to interpret each other's concepts that belong to the class represented by that template

Inference Mechanism

The logical evaluation of the semantic nets is performed by a special mechanism that we call inference mechanism. Nodes are interpreted as facts. A relationship between nodes, which is graphically drawn as an arc, is interpreted as a predicate. The inference mechanism proceeds in three phases: searching, evaluating and executing.

- *Searching*: According the related template, traversing a given semantic net, while seeking for a given attribute name. The attribute name may be the name of a node.
- *Evaluating*: Is performed by evaluating the attributes of the found node according the given attribute values.
- *Executing*: Is performed by executing the operations of the found node.

Communication Model

An AgentTeam agent may communicate principally in the three modes co-operate, collaborate, or compete (*Figure 4.8*).

Co-operating: In this mode an agent knows only team goals, it has no antagonistic goals. It may have individual goals, but these will be sub-goals required to reach the team goals independently. All involved agents tightly work together to successfully terminate a transaction. In other words, a transaction is proceeded by the agents always in co-operate.

Collaborating: In this mode an agent has two or more goals, which may be co-operative or antagonistic in nature. These goals always will temporarily be required as sub-goals. In this case two or more transactions must be proceeded concurrently by the involved agents. For example when a user has started several transactions immediately in succession or two users have started their transactions in parallel and some involved agents must proceed some of the sub-transactions concurrently. To achieve its team goals, an

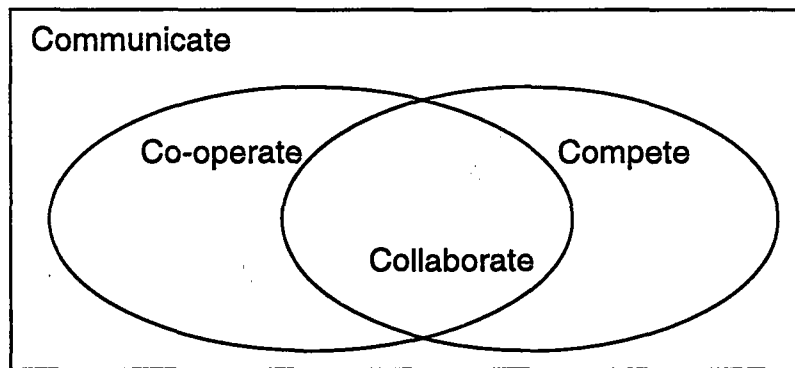


Figure 4.8: Communication Model of an AgentTeam Agent

agent may have some sub-goals, which may be collaborative inside the agent team, but competing against agents outside that team. A further example for collaboration is, if an agent involved in a team communicates with an agent outside that group.

Competition: In this mode of communication an agent has only antagonistic goals, it has no team goals. A competing agent may temporarily take over a sub-goal of a team goal of other agents, if that sub-goal will contribute to its antagonistic goals. In this case it will co-operate with other agents temporarily. AgentTeam agents will never behave competitive to each other. But, it may be advantageous, with respect to their own goals, to enter this mode, when communicating with agents of other domains or applications.

Co-ordination Model

Different entities, in form of functionality and data, need to be co-ordinated in AgentTeam as agent internal and agent-external control. Since, the system is transaction-oriented, initiated by a user transaction, control begins centrally by the user agent, flows over its task agents in parallel, then over the resource agents in parallel, and ends with the last related activities of the involved database agents in parallel. In particular following entities are co-ordinated.

Collaborative work: The co-ordination of the collaborative work of AgentTeam agents is implicit in transaction management and scheduling. For example, for each parallel transaction of a user the user agent invokes a task agent, sub-transactions of a transaction

are co-ordinated by one task agent, concurrent transactions are serialised by the related database agent.

Access rights: Access to database data is co-ordinated by the access rights of each involved DBMS and over the above discussed functionality for distributed transaction management.

Consistency: Data consistency of the distributed database is co-ordinated according the discussed functionality for distributed transaction management.

With respect to these co-ordination mechanisms, the agents operate semi-autonomous. Further global or central synchronisation is not required.

4.3 Agent Types

To reduce complexity of agents and their responsibilities, distributed database management were divided into independent system tasks. Accordingly, we have synthesised four types of agents, by emphasising related properties and functionality of the general agent model. The agent types, their responsibilities, and their communication structure are discussed below. According their location and assigned tasks, they can be grouped in client-site and server-site agents (*Figure 4.9*).

User Agent: A user agent is mainly responsible for managing user transactions. Is created from a template, which is located on each server, stays persistent and will be invoked by user. One exists for each user, with the purpose of managing user's data and information links. It provides the user with an interface for adding, modifying, deleting, and retrieving data shared by the user team members. It is responsible for subdividing a user transaction into tasks and distributing them to task agents. Knowledge of the location of data sources and the kind of protocol in which that data were retrieved in previous communications is retained.

Task Agent: A task agent implements the transaction management functionality of the DDBMS. It is specifically responsible for executing transactions, identifying sub-transactions, and for co-ordinated execution of transactions and sub-transactions. It is created by the user agent and exists for the duration of a transaction. A transaction may

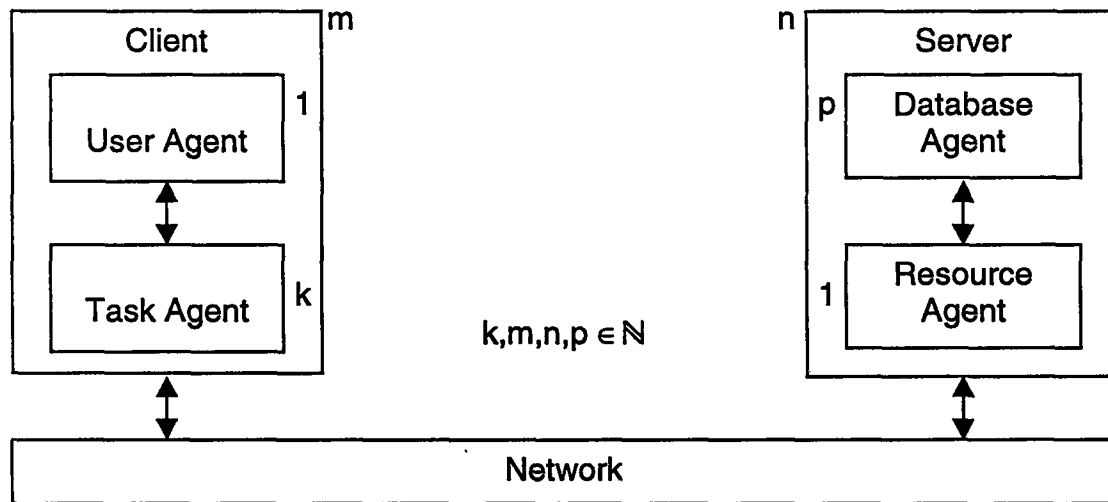


Figure 4.9: Communication Structure of the Agent Types

result in one or more AgentCom sessions. A task agent acts as the client site connectivity that gains access to a remote resource agent.

Resource Agent: A resource agent is mainly responsible for brokering of AgentCom messages between task agents and database agents. Only one exists on a server, where it provides the server site connectivity. It organises the persistency of database agents and is responsible for the brokering of messages between task agents and the different database agents that reside on the same host. Finally, it seeks for a database agent that is suitable for sender's domain.

Database Agent: A database agent implements the scheduling functionality of the DDBMS. It is specifically responsible for scheduling concurrent transactions, managing data distribution, and providing the actual access to the database. It is created, or if already existing, is invoked by the resource agent. It provides an interface to data resources and stays persistent. One exists for each resource and is responsible for translation between AgentCom and site's native language.

In order to implement persistency of agents, in general, a new agent is created for each required domain and type, if it does not exist yet. Otherwise, the currently existing agent is invoked.

4.4 AgentCom

AgentCom is the common communication language of the AgentTeam agents. Following requirements influenced the design of the language:

- Network capable, serial
- Exchange data of any form
- Exchange knowledge between agents
- Co-ordinate co-operative work of agents
- Enable agents to support higher order functionality, such as to store user defined views, or to support the co-operative work of user teams

Language Primitives

AgentCom has several message types that can be communicated syntactically independently from each other. The syntax is based on list structures. Sent messages are pure character strings. For a brief overview, all language primitives are grouped into messages and keywords and listed below.

```
message = {sessionBegin, sessionEnd, give, take, delete,  
          respond}
```

```
keyword = {error, address, name, domain, topic, sessionId,  
          messageId, conference, communicationMode, language,  
          object}
```

A formal description of the language in extended BNF is given in (*Appendix A*).

The basic semantic of the language is to deliver the communication partner some knowledge or to receive some knowledge. It is noteworthy, that the language itself does not provides for logical operation, in the sense of predicate calculus. However, related combinations of the language primitives can express any more complicated communicative intention, such as the intention to let the partner perform some functionality, or to reason about some facts.

Semantics

A session can consist of several messages. Each message can contain a variable number of keywords. Depending on the current communication status, an agent may decide to include a keyword or not. Furthermore, each keyword can contain a variable number of values.

At the language level data is exchanged, whereas, knowledge can be exchanged at the semantic level, for example by exchanging a knowledge base concept or template. Data to be exchanged between communication partners is represented in form of attribute/value pairs, which are named object/object-value in the language. Since, each message can contain a variable number of attribute-value pairs, a complete semantic net can be sent within a single AgentCom message or parts of it.

4.5 User Co-operation Model

From the perspective of a user the system consists of one AgentTeam client, which is the local machine, and one or more AgentTeam database servers. All AgentTeam hosts build a community consisting of several domains. Users of a domain are thought to be a user team working temporarily together, until some team goals are reached. Principally, an AgentTeam client can communicate with any AgentTeam server, independent from a specific domain.

Working Environment

The working environment of a user provides abilities to inspect the data dictionary of a remote database on an AgentTeam server, to view the tuples of a table, to create views by joining tables and defining filters on selected data. A user can incrementally introduce new information into the personal working environment by creating new views, modifying, or deleting them. Further properties of the user environment are:

- User is aware of heterogeneity
- User is aware of distributed data, but not involved into the distribution management
- User must provide for valid data access rights, in order access remote data

Users can principally co-operate with each other directly, by exchanging database views, or indirectly by updating some remote data over a view.

There are actually three types of goals in the AgentTeam framework. Goals of a user team, individual user goals, and agent team goals inside the system (*Figure 4.10*).

4.6 Security Concepts

Since, the framework is designed for distributed systems, any component of the framework must consider security and reliability for functionality as well as for data against any possible external attack. Following system components should comply with security standard, in order to guarantee acceptable overall system security and reliability.

Secure hosts: All involved sites, clients and servers, should be secure platforms, for example by using proxies.

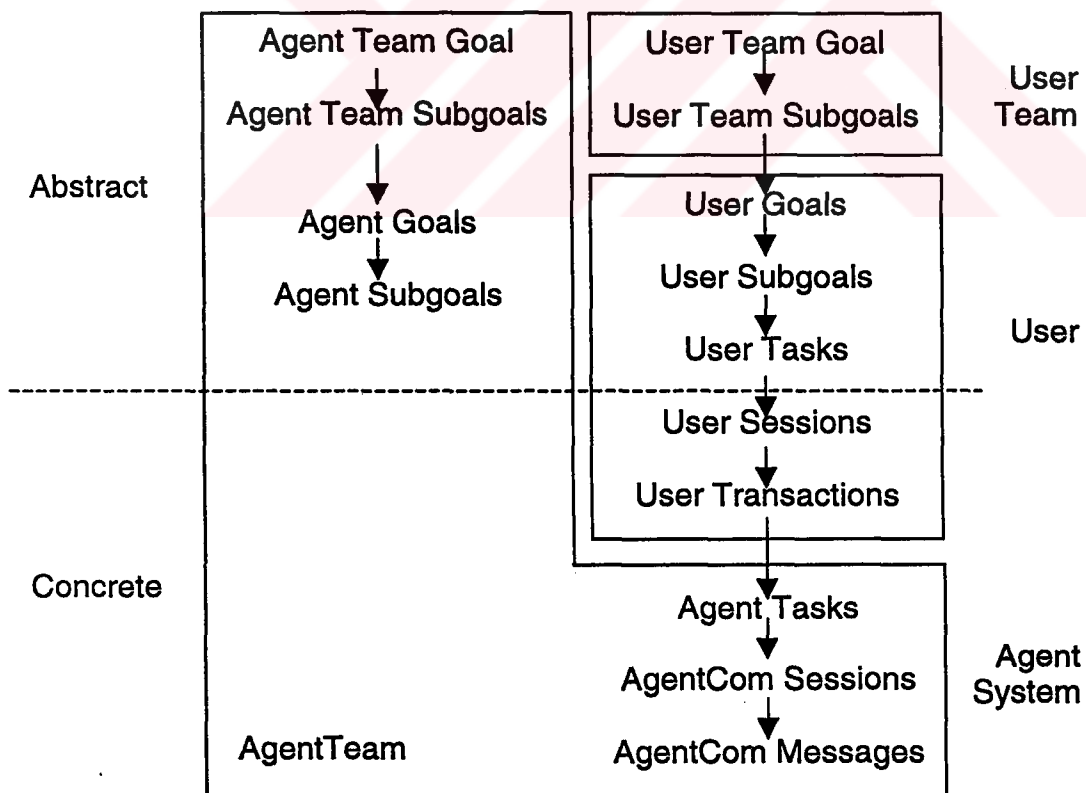


Figure 4.10: Three Levels Co-operation Model of AgentTeam

Secure agents: The agents should provide for security hierarchies, including signatures, user authentication, and syntax and semantic check for received AgentCom messages.

Secure transactions: Transactions should be communicated based on secure message primitives.

Secure messages: Since, an AgentCom message is a character string, it should be encrypted, before sending over the network.



Chapter Five

CourseMan Prototype

This chapter covers the feasibility study for the framework AgentTeam. Because of limited was available for the implementation and limited resources for the realisation of some of the concepts, only some important features of the framework could be considered in the design of the prototype. In this chapter we discuss the implementation-oriented design concepts of CourseMan. Further implementation-oriented details are listed separately in the appendices.

5.1 Distributed Database Architecture

The distributed database of CourseMan is designed in a top-down fashion according the requirements of the enterprise, which is the management of university courses.

Enterprise

Detailed data concerning the courses and involved persons are required. To store temporarily required data for the duration of a term, some further data structures are necessary. Data related to assignments, midterms and the final of a course have to be stored temporarily, until the overall grade of each student and each course is calculated.

Identified entities are department, lecturer, assistant, student, course, assignment, midterm, and final. The relationships are assist, offer, teach, enrol, assign, examine midterm, and examine final. A full description of the enterprise with the conceptual scheme in form of an E/R diagram is given in (*Appendix B, C*).

Data Dictionary

Data of the enterprise can be grouped in the classes master data and temporary data (*Appendix C*). Master data represents the basic objects of the domain. Temporary data is created for each course separately at the beginning of each term. It is required only for the duration of that term. At the end of a term the results are updated against related master data and all temporary data is deleted.

Tables Assignment, assign, Midterm, examiM, Final, and eaxmiF are created at run time by the user agent, to hold temporarily required CourseMan data. When user creates a new course to be taught, then required data for the related assignments, midterms, and the final is stored in these tables. After a course is over, the user agent deletes all related temporary data.

Data Distribution

Lecturers and assistants usually are responsible for the same students within a class. Therefore, the same person frequently updates data of related students. Thus, fragmenting frequently updated tables horizontally and storing them on the local machines will increase system response time. The objective of horizontal fragmentation is to partition a relation into tuple sets and to distribute the sets to different users. The distribution criteria are unstructured, since users with update permission randomly select students. Always only one user requires tuples of temporary tables. Principally, CourseMan users do not compete for updates on temporary data. Therefore, an update on a tuple always fragments the related table horizontally. This is in conformance with the principal of data locality.

Database Connectivity

The connectivity of CourseMan to databases is realised over the ODBC protocol. A database agent communicates directly with the database source, which is a single file in the operating system environment. ODBC configurations are made local on server site, where the database resides. Clients are zero-configured, which has the advantage that any machine can join the CourseMan system without having to configure anything on client site. The data access language all involved databases is SQL.

5.2 Agent Architecture

Common agent properties were implemented in form of a super class. All agent types inherit these capabilities and add additional functionality, such as user interface functionality, transaction management, and scheduling functionality.

Knowledge Base

Currently maintained concepts of the knowledge base are the RDBMS template and data dictionary of the local database including attributes for fragmented tables, a template for user data views and one user data view stored as a topic.

To exchange a semantic net, the sending agent wraps the semantic net into attribute-value pairs to be sent in AgentCom. The receiving agent wraps the attribute-value pairs back to its semantic net representation. The data wrapping according the communication structure of the agent types is depicted in (*Figure 5.1*).

Sample AgentCom Session

The sample communication below illustrates how a connection between a task agent and a database agent can be established. The messages are fragments of one session.

```
(sessionBegin (address '193.x.x.x' '193.x.x.y')
  (name 'D1' 'D2')      (domain 'Course') (sessionID '300')
  (messageID '1')      (communicationMode 'RO')
  (language ('SQL' 'Java')))
```

D1 sends this message to D2. The first address is that of the sender, the second is that of the receiver, respectively for name.

```
(respond (name 'D2' 'D1')      (domain 'Course')
  (sessionID '300') (messageID '1')
  (object ('Student' 'enrol')
    ('SID' 'Name' 'Address')
    ('SID' 'CID' 'AssignAverage' 'MidtemAverage'
      'FinalGrade' 'Grade')))
```

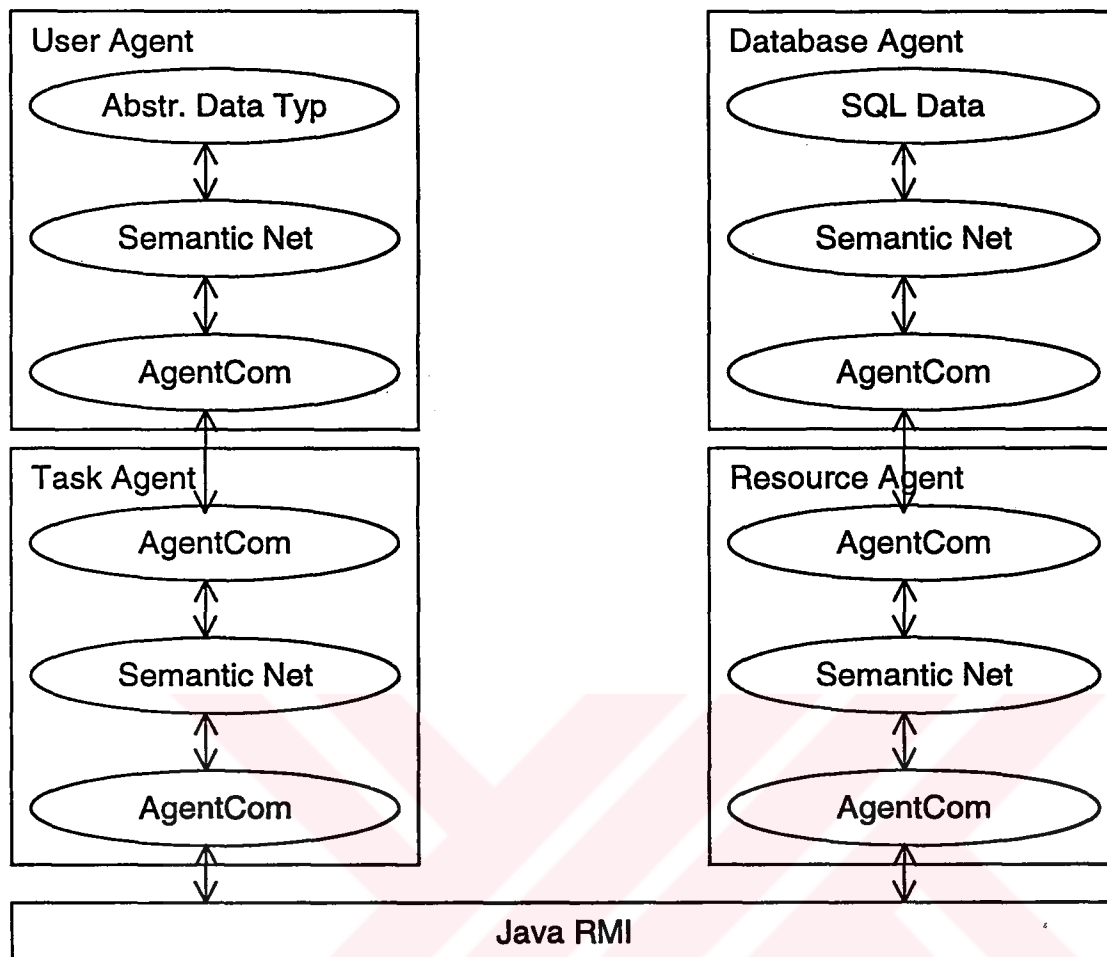


Figure 5.1: Wrapping between the Data Representation Forms

D2 is responding with the same session and message identifications. D2 is indicating that it is involved in the domain mentioned. It sends the data dictionary of the local database.

```
(take      (sessionID '300')  (messageID '2')
  (communicationMode 'XX') (object ('Student')
    ('SID' 'Name' 'Address')      ('SID' > '1998000')))
```

D1 is requesting D2 to resolve the indicated object-value pair.

```
(respond (sessionID '300')  (messageID '2')
  (object ('SID' 'Name' 'Address')
    ('1998076', 'Baris Bacaksiz', 'Bostanli-Izmir')
    ('1998106', 'Aycin Bilir', 'Bornova-Izmir')
    ('1998028', 'Burcu Batmis', 'Karsiyaka-Izmir'));
```

D2 has resolved the object-value pair as a table-attribute pair. Since, the attribute names are not final, their values are retrieved and sent back to D1 as an object-value pair.

5.3 Co-operative User Work

Basically, each user can randomly view required data in the user interface. Co-operative user work is based on shared data, which can be updated by users with update permission and viewed by all users.

User Interface

The user agent presents the user an interface with following capabilities (*Figure 5.2*):

- Selection of remote servers
- Selection of remote databases
- Selection, update, or creation of tables of a database

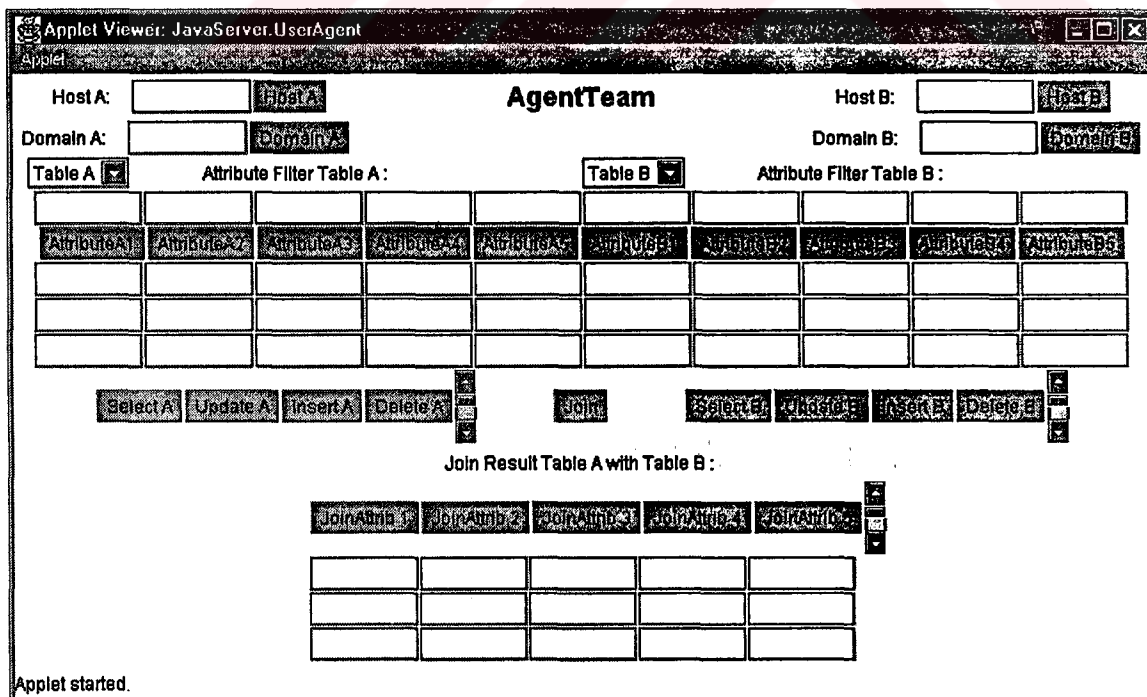


Figure 5.2: User Interface of CourseMan

- Selection, update, or insertion of tuples of a table

Sample Scenario for the Management of a Course

A sample scenario for the management of a course could occur as follows: A lecturer informs its user agent about a new course. The user agent creates a task agent to get required master data. The task agent establishes a connection to related database agents and retrieves the master data. During the current term, the lecturer gives assignments, makes midterms, and the final. For each of these tasks the user agent creates one or more new task agents, who organise required data. A course usually exists for the duration of a term. During that term, the lecturer and its assistants may update, or students may query CourseMan data. Each authorised user will have its own user agent. At the end of a term, the lecturer evaluates all data, calculates final course grades, and delivers the results. The results are stored in the master database. The user agent assists nearby and finally deletes all temporarily used data in CourseMan. A user agent will always assist the same user for the duration of one course, as well as for all other courses of that user.

5.4 Security Concepts

In the current implementation of CourseMan the data access rights of the involved DBMSs are utilised. On all involved database an identical user account exists, which is used by the database agent to gain access to the local database.

5.5 System Configuration

All servers involved in CourseMan are located in the department. The current host configuration consists of following machines (*Figure 5.3*):

- *Web server alpha*: The HTML application with all required executables, which are downloaded to a client at run time.
- *Master data server menekse*: Some of the master data is stored in the tables Course, Lecturer, teach, Assistant, assist, Department, and offer.
- *Master data server bilgin*: Some of the master data is stored in the tables Student and enrol.

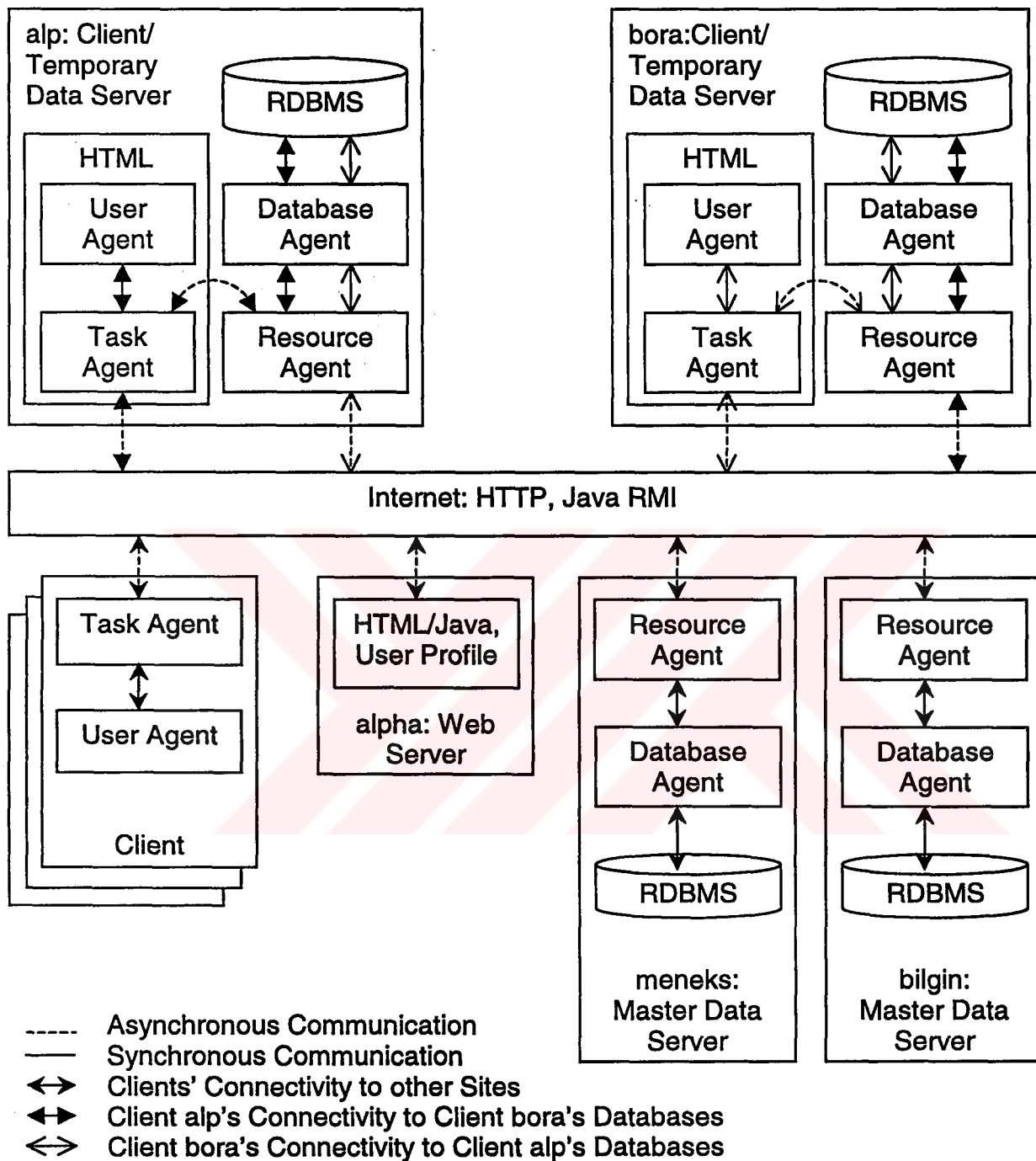


Figure 5.3: Current Network Configuration of CourseMan

- *Client/temporary data server alp:* Some of the temporary data is stored in the tables Assignment, assign, Midterm, examiM, Final, and examiF.

- *Client/temporary data server bora*: Some of the temporary data is stored in the tables Assignment, assign, Midterm, examiM, Final, and examiF.

Current machine configuration with the data distribution, and access rights are given in (*Appendix D*).

Scalability

Currently only one database is connected to CourseMan on a host. Before a database can be utilised by CourseMan, there must exist an SQL DBMS with the related ODBC configuration. A further domain or database source can be introducing to a local resource agent dynamically. The first reference to that database will cause the resource agent to create a further database agent on the local machine. This functionality of the resource agent results in unlimited database scalability in CourseMan.

A further CourseMan server can be introduced to the system, by invoking a resource agent locally on the chosen machine. Thereafter, a configuration for at least one database should be made, as described above. Another alternative is given for users with update permission. In this case the first attempt to fragment a table will cause the system to dynamically create a resource agent locally, if no one already exists. Then, this host will become a CourseMan client and a CourseMan server for temporary data.

A further CourseMan client can be introduced to the system, by downloading a user agent to the local machine, which can be any machine, since zero-configuration for clients is assumed by CourseMan.

Implementation

Details of the utilised tools and the software design of CourseMan can be found in (*Appendix D, E, F, G*). The whole prototype software is implemented in the programming language Java [JDK 97], [Gosling et al. 96].

Conclusions

In this work we have reviewed current research and technology on DDBMSs, agents, and existing systems, which attempt to combine both of them. However, all of the analysed systems were designed with an emphasis on either multi database management or on information retrieval. None of them were concerned with distributed database management of shared databases in a dynamic environment.

We have addressed this problem by designing a system that can successfully work in a dynamic environment, since itself is inherently a dynamic, distributed system. For such a system we have presented a framework, namely AgentTeam, and tested its feasibility in the prototype implementation CourseMan.

Comparison with Related Work

Meriposa: The idea of a DDBMS with temporarily allowed global data inconsistencies was already realised with the wide-area DDBMS Mariposa [Stonebraket et al. 96]. Goals of this project stated are scalability to a large number of co-operating sites, data mobility, no global synchronisation, total local autonomy, and easy configurable policies. A user must charge for each access on data of the DDBMS, which is entered into a personal bank account. The main differences between Mariposa and AgentTeam are that Mariposa resolves concurrency over an economy principle and that it does not utilise the agent concept.

MIND: MIND is a multi DBMS [Doğaç et al. 95] but does not deal with data distribution. Database management in MIND is particularly centralised by global transaction management, whereas in AgentTeam there is one transaction manager on each

client locally, which provides for more user-oriented distributed database management capabilities.

RETSINA: The agent types and their communication structures of RETSINA [Sycara et al. 96] were partially adopted in AgentTeam. However, the architecture of RETSINA does not incorporate co-operative work of user teams, and does not consider the user requirement for information exchange at user level. The main difference between and AgentTeam is the domain for which they were designed. RETSINA is a framework for information retrieval, whereas AgentTeam is a framework for information exchange among users based on shared databases.

InfoSleuth: InfoSleuth [Nodine et al. 98b] is more a software-engineering tool for developing intelligent agents. Developed agents can be deployed into a system environment with uniform communication capabilities. It would be of interest, if, based on AgentTeam and KQML, a system for distributed database management could be developed within the InfoSleuth development environment.

Infomaster: The information integration tool Infomaster [Genesereth et al. 97] is a further example for information retrieval. Its rule-based schema integration facilitator is an approach to homogenise heterogeneous data sources as well as structured and unstructured data. This could be a solution concept, when extending AgentTeam to encompass unstructured data, too.

Some characteristics of AgentCom are summarised and compared to those of KQML and distributed objects (*Table 1*). The main differences between the languages are the capability of representing native data in the language, interaction with native languages, and controlling the flow of execution. Firstly, any native data can be represented in form of attribute-value pairs in AgentCom, whereas native data must be available in logical form to be represented in KQML. Secondly, since AgentCom not necessarily is logical, it can be implemented in or can interact with any native language, whereas implementation of and interaction with KQML requires logical languages. A further difference is that the flow of execution between communicating partners is data-driven with AgentCom, but logical with KQML.

Table 1. Some Characteristics of the Communication Models

Protocol	Agent Languages		Distributed Objects
Characteristic	AgentCom	KQML	
Idea	Knowledge Exchange	Knowledge Exchange	Remote Object-link
OS Platform	Any	Any	Any
Implementation Language	Any	Any	Object-oriented
Native data Representation	Any	Logical	Object-oriented
Native Language	Any	Logical	Object-oriented
Control Flow	Data-driven	Logical	Functional
Language Syntax	List String	List String	Object/Method Invocation
Language Semantic	In Language, in Agent's Knowledge Base	In Language, in Agent's Knowledge Base	Non
Co-ordination Protocol	Any (CourseMan: HTTP, RMI)	Any	CORBA, DCOM OpenDoc, RMI, etc.
Communication Mode	Synchronous/ Asynchronous	Synchronous/ Asynchronous	Synchronous
Security	Not in Language	Not in Language	In Protocol

Suggested Further Work

Based on some expectations in the progress of the field, following further work is suggested.

Standardisation: Any further extension or improvement of AgentTeam should closely be related the standardisation efforts in the field. For example, standardisation efforts of the telecommunications companies for a common agent communication language are expected to set a milestone for automated information exchange [FIPA 97].

Extending AgentTeam: Furthermore, AgentTeam can serve as an underlying model upon which other systems can be constructed, for example workflow management or virtual data-warehouse, or multimedia information exchange.

Data dictionary parser: Dynamic evaluation of the data dictionary of a database implemented in WEBCON [Zoller et al. 98] provides for further system flexibility. This functionality could be implemented in the database agent, for dynamically recognising the data structures of a shared database.


Extending AgentCom: Evaluation of AgentCom in the domain of CourseMan or in other domains, for example to exchange multi media data or unstructured data and according extension of the language, and refinement of the syntax.

Extending CourseMan: Since, CourseMan will serve as a test bed and for stepwise extensions, following further extensions are suggested:

- Transaction management and scheduling functionality to be represented as knowledge base concepts, and to be stored in form of semantic nets in knowledge base
- Replication of relations, to increase system availability and reliability
- Allowing joins of relations located at different sites
- Dynamic evaluation of data dictionaries, which will make connectivity insensitive to changes within the database scheme
- Introducing further AgentTeam servers into the current configuration inside the current intranet solution and/or building an internet solution, by introducing further servers outside the intranet
- Extending the system to the UNIX platform, to test platform heterogeneity
- Introducing further DBMSs on a local server
- Introducing more intelligent behaviour for user agents, for example by increasing the learning capabilities, and/or by designing intelligent user interfaces

- Refinement of the mathematical model for data fragmentation and data replication that was presented in chapter one. We propose the design of a fuzzyfied [Kumova 92] decision support system for automated data distribution. This functionality could be implemented in the database agents
- *System interoperability*: A wrapper for AgentCom and KQML is of further interest, to enable communication with each other

Software development: For rapid prototyping, it is of further interest to utilise a software environment that supports agent development. For example at tool-level, InfoSleuth provides a layered shell, in which standard agent architecture is enforced [Nodine et al. 98a]. At programming language-level, standard routine libraries can facilitate the implementation of the designed agent properties [Bigus et al. 98].



Abbreviations

ANSI	American National Standards Institute
ARCOL	ARtimis COmmunication Language
BNF	Backus-Naur Form
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
DDBMS	Distributed DataBase Management System
E/R	Entity/Relationship
FIPA	Foundation for Intelligent Physical Agents
HTML	HyperText Mark-up Language
HTTP	HyperText Transport Protocol
KDBMS	Knowledge-based DBMS
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
ODBC	Open DataBase Connectivity
ODBMS	Object-oriented DBMS
QUEL	QUEry Language
RDBMS	Relational DBMS
RMI	Remote Method Invocation
SQL	Structured Query Language
SPARC	Standards Planning and Requirements Committee
TCP	Transmission Control Protocol
IP	Internet Protocol
URL	Universal Resource Locator
WWW	World Wide Web

References

- [Akker et al. 97] Akker, Johan van den; Siebes, Arno; 1997, "Enriching Active Databases with Agent Technology", in Kandzia, Peter; Klusch, Matthias; eds., *Cooperative Information Agents*, Springer-Verlag, Berlin, 1997
- [Ambite et al. 97] Ambite, José Luis; Knoblock, Craig A., 1997, "Agents for Information Gathering", *IEEE Expert*, Vol.12 No. 5 pp.2-4
- [Çağlayan et al. 97] Çağlayan, Alper; Harrison, Colin G., 1997, "Agent Sourcebook: A Complete Guide to Desktop, Internet, and Intranet Agents", John Wiley Sons Inc., New York
- [Babin et al. 97] Babin, Gilbert; Maamar, Zakaria; Chaib-draa, Brahim; 1997, "Metadatabase Meets Distributed AI", in Kandzia, Peter; Klusch, Matthias; eds., *Cooperative Information Agents*, Springer-Verlag, Berlin, 1997
- [Bigus et al. 98] Bigus, Joseph P.; Bigus, Jennifer, 1998, "Constructing Intelligent Agents with Java - A Programmer's Guide to Smart Applications", John Wiley Sons Inc., New York
- [Chavez et al. 97] Chavez, Anthony; Moukas, Alexandros; Maes, Pattie, 1997, "Challenger: A Multi-agent System for Distributed Resource Allocation", *Autonomous Agents '97*, Marina Del Ray
- [Chen et al. 97] Chen, Liren; Sycara, Katia; 1997, "WebMate: A Personal Agent for Browsing and Searching", TR, Carnegie Mellon University, Pittsburgh

- [Ciancarini et al. 98] Ciancarini, P.; Rossi, D.; 1998, "Coordinating Distributed Applets with Shada/Java", *SAC'98*
- [Cohen et al. 94] Philip R. Cohen, Philip R. Cheyer, Michelle Wang, Soon Cheol Baeg, 1994, "An Open Agent Architecture", *AAAI Spring Symposium*, pp. 1-8, March
- [Dam 97] Dam, Mads; eds., 1997, "Analysis and Verification of Multiple-Agent Languages", LOMAPS'96, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin
- [Demirors 95] Demirörs, Elif; 1995, "A Blackboard Framework for Supporting Team in Software Development", PhD thesis, Southern Methodist University
- [Desai 90] Desai, Bipin C.; 1990, "An Introduction to Database Systems", West Publishing Company, St. Paul
- [Doğaç et al. 95] Doğaç, Asuman; Dengi, Cevdet; Kılıç, E.; Ozhan, G.; Özcan, F.; Nural, S.; Evrendilek, C.; Halıcı, U.; Arpınar, B.; Köksal, P.; Kesim, N.; Mancuhan, S.; 1995, "METU Interoperable Database System", *ACM SIGMOD*, Vol. 24, No. 3
- [Doğaç et al. 98] Doğaç, Asuman; Dengi, Cevdet; Özsu, M.Tamer; 1998, "Building Interoperable Databases on Distributed Object Management Platforms", *Communications of the ACM*
- [Duschka et al. 97] Duschka, Oliver M.; Genesereth, Michael R.; 1997, "Infomaster-An Information Integration Tool", International Workshop on Intelligent Information Integration, Freiburg
- [Engelmore et al. 88] Engelmore, Robert; Morgan, Tony; 1998, "Blackboard Systems", Edison-Wesley Publishing Company, Wokingham
- [Evans et al. 97] Evans, Eric; Rogers, Daniel; 1997, "Using Java Applets and CORBA for Multi-User Distributed Applications", *IEEE Internet Computing*, Vol.1 No.3 pp.43-55

- [Finin et al. 92] Finin, Tim; Fritzson, RichMcKay, Donald, 1992, "A Language and Protocol to Support Intelligent Agent Interoperability",
- [Finin et al. 93] Finin, Tim; Wiederhold, Gio; Weber, Jay; Genesereth, Michael; Fritzson, Richard; McKay, Donald; McGuire, James; Pelavin, Richard; Shapiro, Stuart; Beck, Chris; 1993, "Specification of the KQML Agent-Communication Language", draft,
- [Finin et al. 94] Finin, Tim; Labrou, Yannis; Mayfield, James; 1994, "KQML as an Agent Communication Language", Draft, <http://www.cs.umbc.edu/kqml>, University of Maryland Baltimore County, Baltimore
- [FIPA 97] "Foundation for Intelligent Physical Agents"; 1997, *FIPA 97 Specification*, part 2, Agent Communication Language, <http://drogo.cselt.stet.it/fipa/spec/fipa97/fipa97.htm>
- <http://www.cs.cmu.edu/~softagents>
- [Genesereth et al. 97] Genesereth, Michael R.; Keller, Arthur M.; Duschka, Oliver M.; 1997, "Infomaster: An Information Integration System", *ACM SIGMOD*
- [Gosling et al. 96] Gosling, James; Joy, Bill; Steel, Guy; 1996, "The Java Language Specification", Sun Microsystems Inc., Palo Alto
- [Goodwin 94] Goodwin, Richerd; 1994, "Formalizing Properties of Agents", *Journal of Logic and Computation*, Vol. 5, Issue 6
- [Green et al.97] Green, Shaw; Hurst, Leon; Nangle, Brenda; Cunningham, Pádraig; Somers, Fergal; Evans, Richard; 1997, "Software Agents: A review", http://www.cs.tcd.ie/research_groups/aig/iag/pubreview.ps.gz
- [Gruber 94] Gruber, Thomas R.; 1994, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", in Guarino, Nicola; Poli, Roberto; *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers

- [Hayes-Roth 92] Hayes-Roth, Barbara, 1992, "Opportunistic Control of Action in Intelligent Agents", TR, Stanford University, Palo Alto
- [Huhns 97] Huhns, Michael N.; 1997, "Ontologies for Agents", *IEEE Internet Computing*, Vol.1 No.6 pp.81-83
- [Huhns et al. 97] Huhns, Michael N.; Singh, Munindar P.; 1997, "Conversational Agents", *IEEE Internet Computing*, Vol.1 No.2 pp.73-75
- [JDK 97] 1997, "JDK 1.1.5 Documentation", Sun Microsystems Inc., Palo Alto
- [Joseph et al. 91] Joseph, W. Sullivan; Sharman, W. Tyler; 1991, "Intelligent User Interfaces", *ACM Press*, New York
- [Kandzia et al. 97] Kandzia, Peter; Klusch, Matthias; eds., 1997, "Cooperative Information Agents", CIA'97, *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin
- [Krone et al. 98] Krone, Oliver; Chantemargue, Fabrice; Dagaëff, Thierry; Schumacher, Michael; Hirsbrunner, Béat; 1998, "Coordinating Autonomous Entities", *SAC'98*
- [Kumova 92] Kumova, Bora İ.; 1992, "Untersuchung der Einsatzmöglichkeiten von Fuzzy-Logiken in einem regelbasierten Planungssystem", BSc thesis, Fachhochschule Furtwangen
- [Kumova et al. 98a] Kumova, Bora İ.; Kut, Alp; 1998, "A Communication Model for Distributed Agents", *ISCIS'98*, Antalya, <http://www.cs.deu.edu.tr/~kumova>
- [Kumova et al. 98b] Kumova, Bora İ.; Kut, Alp; 1998, "Agent-based System Co-operation", *PDP'99*, Madeira, <http://www.cs.deu.edu.tr/~kumova>
- [Kumova et al. 98c] Kumova, Bora İ.; Kut, Alp; 1998, "Agent-based Connectivity for Distributed DBMSs", technical report, <http://www.cs.deu.edu.tr/~kumova>

- [Labrou et al. 98] Labrou, Yannis; Finin, Tim; 1998, "Semantics for an Agent Communication Language", in Sing, Munindar P.; Rao, Anand; Wooldridge, Michael J.; eds., *Intelligent Agents IV - Agent Theories, Architectures, and Languages*, Springer-Verlag, Berlin
- [Lander 96] Lander, Susan E.; 1997, "Issues in Multiagent Design Systems", *IEEE Expert*, Vol.12 No. 2 pp.18-26
- [Mayfield et al. 95] Mayfield, James; Labrou, Yannis; Finin, Tim; 1995, "Evaluation of KQML as an Agent Communication Language", *Intelligent Agents Volume II - Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages*, in Wooldridge, M.; Muller, J.P.; Tambe, M.; eds., *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, 1996
- [Niemeyer et al. 96] Niemeyer, Patric; Peck, Joshua; 1996, "Exploring Java", O'Reilly, Cambridge
- [Norvig 95] Norvig, Russell, 1995, "Artificial Intelligence - A Modern Approach", Prentice-Hall, New Jersey
- [Nodine et al. 98a] Nodine, Marian; Perry, Brad; Unruh, Amy; 1998, "Experience with InfoSleuth Agent Architecture", *AAAI-98 Workshop on Software Tools for Developing Agents*
- [Nodine et al. 98b] Nodine, Marian; Chandrasekara, Damith; 1998, "Agent Communities", technical report, MCC
- [Odubiyi et al. 97] Odubiyi, Judé B.; Kocur, David J.; Weinstein, Stuart M.; Wakin, Nagi; Srivastava, Sadanand; Gokey, Chris; Graham, JoAnna; 1997, "SAIRE – A Scalable Agent-based Information Retrieval Engine", *Autonomous Agents '97*, Marina Del Ray
- [Orfali 96] Orfali, Robert; Harkey, Dan; Edwards, Jari; 1996, "The Essential Distributed Objects Survival Guide", John Wiley Sons, New York

- [Özsu 91] Özsu, M. Tamer; Valduriez, Patrick; 1991, "Principles of Distributed Database Systems", Prentice Hall, Englewood Cliffs
- [Özkarahan 97] Özkarahan, Esen; 1997, "Database Management – Concepts, Design, and Practice", Saray Medical Publication, İzmir
- [Papadopoulos et al. 98a] Papadopoulos, Georg A.; Arbab, Farhad; 1998, "Modeling Activities in Information Systems using the Coordination Language MANIFOLD", SAC'98
- [Papadopoulos et al. 98b] Papadopoulos, Georg A.; Arbab, Farhad; 1998, "Coordination Models and Languages", *Advances in Computers*, No 46, Academic Press
- [Rich et al. 91] Rich, Elaine; Knight, Kevin; 1991, "Artificial Intelligence", McGraw-Hill, New York
- [Russell et al. 95] Russell, Stuart J.; Norvig, Peter; 1995, "Artificial Intelligence – A Modern Approach", Prentice Hall, Englewood Cliffs
- [Sabin 98] Sabin, Roberta Evans; Yap, Tieng K.; 1998, "Integrating Information Retrieval Techniques with Traditional DB Methods in a Web-Based Database Browser",
- [Sadek et al. 97] Sadek, D.; Bretier, P.; Panaget, F.; 1997, "ARCOL agent communication language and MCP, CAP and SAP agent's cooperativeness protocols", proposal, France Télécom
- [Schalkoff 90] Schalkoff, J. Robert; 1990, "Artificial Intelligence: An Engineering Approach", McGraw-Hill, New York
- [Şeker et al. 98] Şeker, Güzin; Kut, Alp; Kumova, Bora İ.; 1998, "Security Concepts for Accessing Distributed Databases via Java and JDBC", BAS98, İzmir, <http://www.cs.deu.edu.tr/~kumova>

- [Sing et al. 98] Sing, Munindar P.; Rao, Anand; Wooldridge, Michael J.; eds., 1998, "Intelligent Agents IV - Agent Theories, Architectures, and Languages", ATAL'97, Springer-Verlag, Berlin
- [Singh 98] Singh, Narinder; 1998, "Unifying Heterogeneous Information Models", *Communications of the ACM*, Vol.41 No.5
- [Stonebraker et al. 96] Stonebraker, Michael; Aoki, Paul M.; Litwin, Witold; Pfeffer, Avi; Sah, Adam; Sidell, Jeff; Staelin, Carl; Yu, Andrew; 1996, "Mariposa: a wide-area distributed database system", *The VLDB Journal*, Vol. 5, pp. 48-63, Springer-Verlag, Berlin
- [Sycara et al. 97] Sycara, Katia; Decker, Keith; Pannu, Ananddeep; Williamson, Mike; Zeng, Dajun; 1997, "Distributed Intelligent Agents", *IEEE Expert*, Dec 96, <http://www.cs.cmu.edu/~softagents>
- [Tanenbaum 89] Tanenbaum, Andrew S.; 1989, "Computer Networks", Prentice Hall, Englewood Cliffs
- [Tanenbaum 92] Tanenbaum, Andrew S.; 1992, "Modern Operating Systems", Prentice Hall, Englewood Cliffs
- [Tanenbaum 95] Tanenbaum, Andrew S., 1995, "Distributed Operating Systems", Prentice-Hall, New Jersey
- [Thompson et al. 97] Thompson, Dean; Watkins, Damien; 1997, "Comparison between CORBA and DCOM: Architectures for Distributed Computing", *Tools Asia'97*, Beijing
- [Vitek et al. 97] Vitek, Jan; Tschudin, Christian; eds., 1997, "Mobile Object Systems – Towards the Programmable Internet", Springer-Verlag, Berlin
- [Vraneš et al. 95] Vraneš, Sanja; Stanojević, Mladen; 1995, "Integrating Multiple Paradigms within the Blackboard Framework", *IEEE Transactions on Software Engineering*, Vol. 21, No 3, Mar 1995, IEEE Computer Society Press

[Zoller et al. 98] Zoller, Peter; Sommer, Ulrike; 1998, "WEBCON: A Toolkit for an Automatic, Data Dictionary Based Connection of Databases to the WWW", SAC'98, Atlanta



Appendix

Implementation-oriented details of the prototype are listed in this appendix.



Appendix A

BNF of AgentCom

To shorten the kind of expressing iterative rules and thereby increase readability, exponential non-terminals extend the BNF presented here. Thus, an exponential non-terminal represents an iterative rule.

```

<session> ::= <message> <respond>
<message> ::= <sessionBegin> (<give > | < take > | <delete>)*
               <sessionEnd>
<sessionBegin> ::= (sessionBegin <keyword>+)
<sessionEnd> ::= (sessionEnd <keyword>+)
<give> ::= (give <keyword>+)
<take> ::= (take <keyword>+)
<delete> ::= (delete <keyword>+)
<respond> ::= (respond <keyword>+)
<keyword> ::= (error (eValue)) |
               (address (aSource aDestination)) |
               (name (nSource nDestination)) |
               (domain (dValue+)) |
               (sessionId (sIdValuei)) |
               (messageId (mIdValue)) |
               (conference (agentNameValue+)) |
               (communicationMode ({RW, RO, XX}) |
               (language (lValue+)) |
               (<object>)*
<object > ::= object ((objectNamenj) ((objectValuenj)+)
               <condition>) ; nij > 0

```

$\langle \text{condition} \rangle ::= ((A \oplus B) \oplus \langle \text{condition} \rangle) \mid \varepsilon$

$A, B \in \{\text{objectName}^m\} \quad ; \quad \sum_{i=1}^{\text{sessions}} \sum_{j=1}^{\text{messages}} n_{ij} \leq m \leq m_s \leq m_{KB}$

$\oplus ::= != \mid \&\& \mid || \mid < \mid > \mid = \mid + \mid - \mid * \mid / \mid \varepsilon$

i : session counter

j : message counter; restarted for each new session

n_{ij} : number of unique objects totally exchanged between two agents in current session

m : number of unique objects totally exchanged between two agents in all sessions

m_s : number of unique objects of a semantic net

m_{KB} : number of unique objects of all semantic nets in the knowledge base of an agent

In case of synchronous communication, a response message is returned asynchronously. In case of asynchronous communication, a response message is sent back from server to the client explicitly.

Appendix B

Entity Relationship Diagram of the Distributed Database

To represent the domain of the management of university courses from a database-oriented view, entity relationship diagrams were employed. Accordingly, the conceptual schema of the university course enterprise is depicted in (Figure B.1). The diagram consists of two sub-graphs, marked by different line types. The sub-graph in solid lines represents

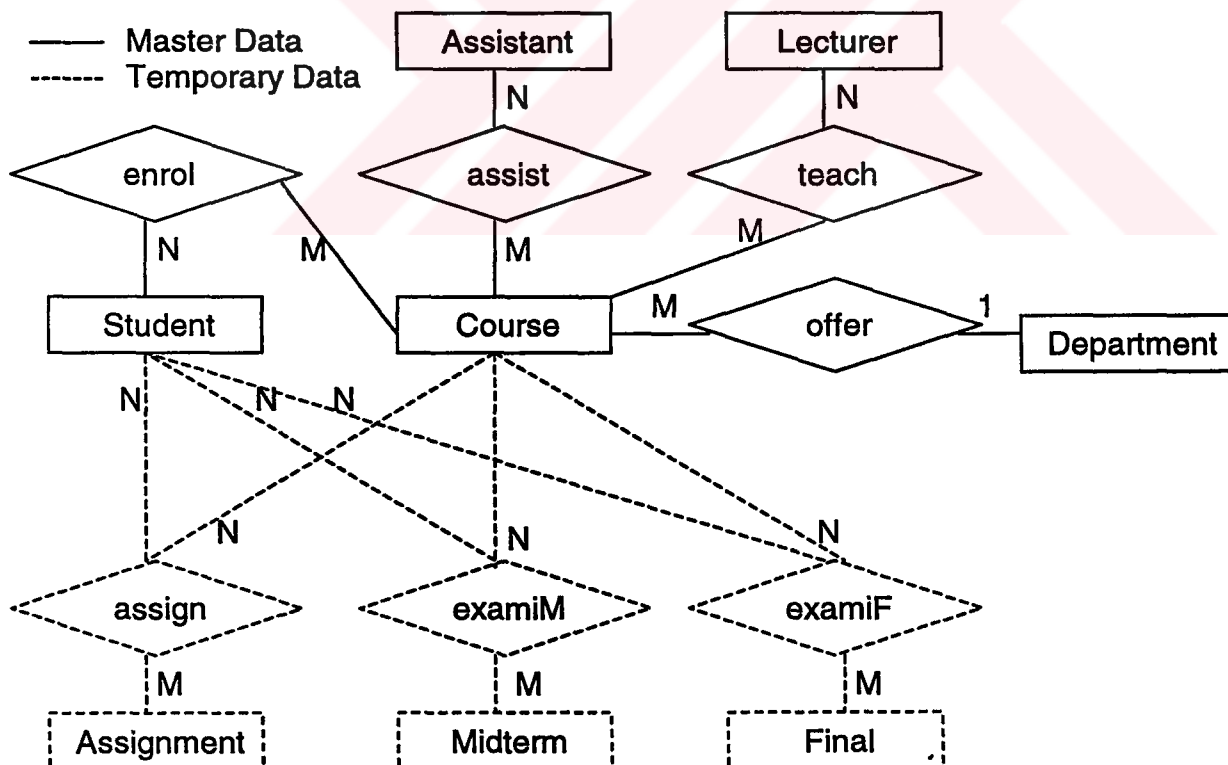


Figure B.1: Conceptual Scheme as E/R Diagram of the University Course Enterprise

the master data. The sub-graph in dashed lines represents temporary CourseMan data required for the duration of a term.



Appendix C

Data Dictionary of the Distributed Database

The data dictionary of the distributed database consists of a collection of SQL statements. They are intended for creating the required data structures and to fill them with some initial data. Data is classified into master data and temporary data. Both are created in two different DBMSs each.

Master Data

The database on server menekse was created inside the MS-SQL Server interactively. Thereafter, the required tables were created with following SQL statements.

```
/* Default Database: CourseMan */
```

```
/* Entity: All courses */
```

```
CREATE TABLE Course (  
    CID          VARCHAR(10) NOT NULL UNIQUE,  
    Title        VARCHAR(50),  
    Description   VARCHAR(100),  
    PRIMARY KEY (CID)  
)
```

```
/* Relation: All offered courses of all departments */
```

```
CREATE TABLE offer (  
    CID          VARCHAR(10) NOT NULL,  
    DeptID       VARCHAR(10) NOT NULL,  
    PRIMARY KEY (CID, DeptID)  
)
```

/* Entity: All departments */

```
CREATE TABLE Department (  
    DeptID          VARCHAR(5) NOT NULL UNIQUE,  
    Name            VARCHAR(20),  
    Description      VARCHAR(100),  
    PRIMARY KEY (DeptID)  
)
```

/* Relation: All courses of all lecturers */

```
CREATE TABLE teach (  
    CID             VARCHAR(10) NOT NULL,  
    LID             VARCHAR(10) NOT NULL,  
    PRIMARY KEY (CID, LID)  
)
```

/* Entity: All lecturers */

```
CREATE TABLE Lecturer (  
    LID             VARCHAR(10) NOT NULL UNIQUE,  
    Name            VARCHAR(30),  
    Address          VARCHAR(50),  
    PRIMARY KEY (LID)  
)
```

/* Relation: All courses of all assistants */

```
CREATE TABLE assist (  
    CID             VARCHAR(10) NOT NULL,  
    AID             VARCHAR(10) NOT NULL,  
    PRIMARY KEY (CID, AID)  
)
```

/* Entity: All assistants */

```
CREATE TABLE Assistant (  
    AID             VARCHAR(10) NOT NULL UNIQUE,  
    Name            VARCHAR(30),
```

```
Address    VARCHAR(50),  
PRIMARY KEY (AID)  
)
```

Initial required master data is inserted into the tables on menekse with following SQL statements.

```
INSERT INTO Course (CID, Title, Description)  
VALUES ('CSE101', 'Algorithms and Programming I',  
        'Introduction to Data Structures and Algorithms')  
INSERT INTO Course (CID, Title, Description)  
VALUES ('CSE450', 'Distributed Databases', 'Introduction to  
        Distributed Databases')  
INSERT INTO Course (CID, Title, Description)  
VALUES ('CSE509', 'Distributed Databases', 'Introduction to  
        Distributed Databases')  
  
INSERT INTO offer (CID, DeptID)  
VALUES ('CSE101', 'CSE')  
INSERT INTO offer (CID, DeptID)  
VALUES ('CSE450', 'CSE')  
INSERT INTO offer (CID, DeptID)  
VALUES ('CSE509', 'CSE')  
  
INSERT INTO Department (DeptID, Name, Description)  
VALUES ('CSE', 'Computer Science and Engineering', 'Academic  
        Programme: BSc, MSc, Phd')  
INSERT INTO Department (DeptID, Name, Description)  
VALUES ('MATH', 'Mathematics', 'Academic Programme: BSc, MSc,  
        Phd')  
INSERT INTO Department (DeptID, Name, Description)  
VALUES ('ENG', 'English', 'Academic Programme: BSc, MSc,  
        Phd')  
  
INSERT INTO teach (CID, LID)
```

```
VALUES ('CSE101', 'YD303')
INSERT INTO teach (CID, LID)
VALUES ('CSE450', 'YD303')
INSERT INTO teach (CID, LID)
VALUES ('CSE509', 'YD303')

INSERT INTO Lecturer (LID, Name, Address)
VALUES ('P180', 'Esen Ozkarahan', 'Bostanli-Izmir')
INSERT INTO Lecturer (LID, Name, Address)
VALUES ('YD203', 'Alp Kut', 'Hatay-Izmir')
INSERT INTO Lecturer (LID, Name, Address)
VALUES ('OG138', 'Adil Alpkocak', 'Karsiyaka-Izmir')

INSERT INTO assist (CID, AID)
VALUES ('CSE101', 'A3175')
INSERT INTO assist (CID, AID)
VALUES ('CSE450', 'A3175')
INSERT INTO assist (CID, AID)
VALUES ('CSE509', 'A3175')

INSERT INTO Assistant (AID, Name, Address)
VALUES ('A3175', 'Bora I. Kumova', 'Karsiyaka-Izmir')
INSERT INTO Assistant (AID, Name, Address)
VALUES ('A2936', 'Erdinc Tarniverdi', 'Bornova-Izmir')
```

The database and its tables were created on server bilgin inside Oracle DBMS interactively with following SQL statements.

```
CREATE DATABASE "C:\CourseMan\CourseMan.gdb";
```

```
CONNECT "C:\CourseMan\CourseMan.gdb";
USER "CourseMan" PASSWORD "masterkey";
```

```
/* Entity: All students */
```

```
CREATE TABLE Student (  
    SID          VARCHAR(10) NOT NULL UNIQUE,  
    Name         VARCHAR(30),  
    Address      VARCHAR(50),  
    PRIMARY KEY (SID)  
);
```

/* Relation: All courses of all students */

```
CREATE TABLE enrol (  
    SID          VARCHAR(10) NOT NULL,  
    CID          VARCHAR(10) NOT NULL,  
    AssignAverage  FLOAT,  
    MidtermAverage FLOAT,  
    FinalGrade     FLOAT,  
    Grade          FLOAT,  
    PRIMARY KEY (SID, CID)  
);
```

```
COMMIT;
```

```
EXIT;
```

Initial required master data is inserted into the tables on bilgin with following SQL statements.

```
INSERT INTO Student (SID, Name, Address)  
VALUES ('1996021', 'Yilmaz Gul', 'Hatay-Izmir');  
INSERT INTO Student (SID, Name, Address)  
VALUES ('1997002', 'Ciler Ayabakan', 'Hatay-Izmir');  
INSERT INTO Student (SID, Name, Address)  
VALUES ('1997059', 'Sencan Yerebakan', 'Karsiyaka-Izmir');  
INSERT INTO Student (SID, Name, Address)  
VALUES ('1998076', 'Baris Bacaksiz', 'Bostanli-Izmir');  
INSERT INTO Student (SID, Name, Address)  
VALUES ('1998106', 'Aycin Bilir', 'Bornova-Izmir');
```

```
INSERT INTO Student (SID, Name, Address)
VALUES ('1998028', 'Burcu Batmis', 'Karsiyaka-Izmir');

INSERT INTO enrol (SID, CID, AssignAverage, MidtermAverage,
    FinalGrade, Grade)
VALUES ('1996021', 'CSE101', NULL, NULL, NULL, NULL);
INSERT INTO enrol (SID, CID, AssignAverage, MidtermAverage,
    FinalGrade, Grade)
VALUES ('1997002', 'CSE101', NULL, NULL, NULL, NULL);
INSERT INTO enrol (SID, CID, AssignAverage, MidtermAverage,
    FinalGrade, Grade)
VALUES ('1997059', 'CSE101', NULL, NULL, NULL, NULL);
INSERT INTO enrol (SID, CID, AssignAverage, MidtermAverage,
    FinalGrade, Grade)
VALUES ('1998076', 'CSE101', NULL, NULL, NULL, NULL);
INSERT INTO enrol (SID, CID, AssignAverage, MidtermAverage,
    FinalGrade, Grade)
VALUES ('1998106', 'CSE101', NULL, NULL, NULL, NULL);
INSERT INTO enrol (SID, CID, AssignAverage, MidtermAverage,
    FinalGrade, Grade)
VALUES ('1998028', 'CSE101', NULL, NULL, NULL, NULL);
```

Temporary Data

If it does not exist yet, a temporarily required table is created by the application on client alp and/or bora by using following SQL statements. In case of horizontal fragmentation, the related SQL statement is used anew to create a table for each further fragment.

```
/* Entity: All assignments of all courses */
CREATE TABLE Assigment (
    CID          VARCHAR(10) NOT NULL UNIQUE,
    ANo          INTEGER NOT NULL UNIQUE,
```



```
Assigned      DATE,
Delivery      DATE,
Text          VARCHAR(100),
PRIMARY KEY (CID, ANo)
);

/* Relation: All assignments of all courses of all students*/
CREATE TABLE assign (
  CID          VARCHAR(10) NOT NULL,
  SID          VARCHAR(10) NOT NULL,
  ANo          INTEGER NOT NULL,
  Delivered    DATE,
  Grade        VARCHAR(2),
  PRIMARY KEY (CID, SID, ANo)
);

/* Entity: All midterms of all courses */
CREATE TABLE Midterm (
  CID          VARCHAR(10) NOT NULL UNIQUE,
  MNo          INTEGER NOT NULL UNIQUE,
  Date         DATE,
  Text         VARCHAR(100),
  PRIMARY KEY (CID, MNo)
);

/* Relation: All midterms of all courses of all students*/
CREATE TABLE examiM (
  CID          VARCHAR(10) NOT NULL,
  SID          VARCHAR(10) NOT NULL,
  MNo          INTEGER NOT NULL,
  Grade        VARCHAR(2),
  PRIMARY KEY (CID, SID, MNo)
);

/* Entity: All finals of all courses */
```

```
CREATE TABLE Final (  
    CID          VARCHAR(10) NOT NULL UNIQUE,  
    Date         DATE,  
    Text         VARCHAR(100),  
    PRIMARY KEY (CID)  
);
```

/* Relation: All finals of all courses of all students*/

```
CREATE TABLE examiF (  
    CID          VARCHAR(10) NOT NULL,  
    SID          VARCHAR(10) NOT NULL,  
    Grade        VARCHAR(2),  
    PRIMARY KEY (CID, SID)  
);
```

Appendix D

Network Configuration

The distributed components of CourseMan, application and data, are located on several machines of the department. All machines are connected to the Internet. User access rights restrict the manipulation of CourseMan data.

Five machines are involved into the management of a course in CourseMan. The HTML pages and Java executable binaries are located on the Web server alpha. Master data is distributed over the two machines menekse and bilgin (*Figure D.1*) and stored there in the DBMSs MS-SQL Server and Oracle, respectively. Master data cannot be moved, it is fixed to the location where it was created. Temporary data can be distributed over the machines alp and bora and stored there in the DBMSs MS-SQL Server and InterBase, respectively. The location of temporary data can be changed dynamically through fragmentation. Table fragments can be exchanged between the clients alp and bora. User access rights on database data are organised on database level. The user on alp can update any master data and any temporary data in any form. The user on bora can update any temporary data in any form, but can only read master data (*Figure D.1*).

Machine Configuration: The operating system on all involved sites is Microsoft NT.

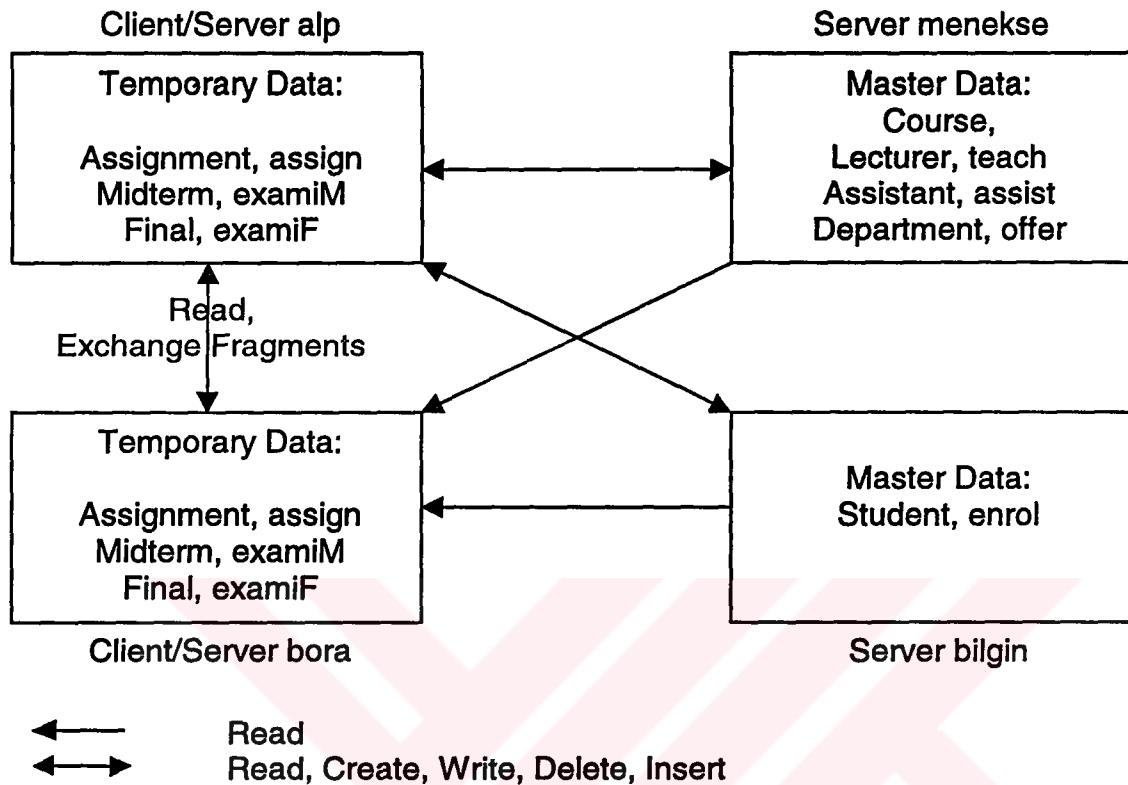


Figure D.1: Current Machine Configuration, Data Distribution, and Access Rights in CourseMan

All involved DBMSs are autonomous and are not aware of each other.

Appendix E

Utilised Tools

The whole CourseMan system consists of four modules represented by the four agent types. All agents are written entirely in Java scripts.

On client side, the user agent is embedded in an HTML page as an applet. At run-time, the user agent dynamically creates one or more task agents, which establish the network connection to resource agents at server sites. To receive messages from database servers asynchronously, the user agent creates a server side skeleton on the client that listens on the standard port of the RMI system. All programmes running at client site are bound to one Java script (*Figure E.1*).

On server side, the resource agent is introduced to the operating system as a service

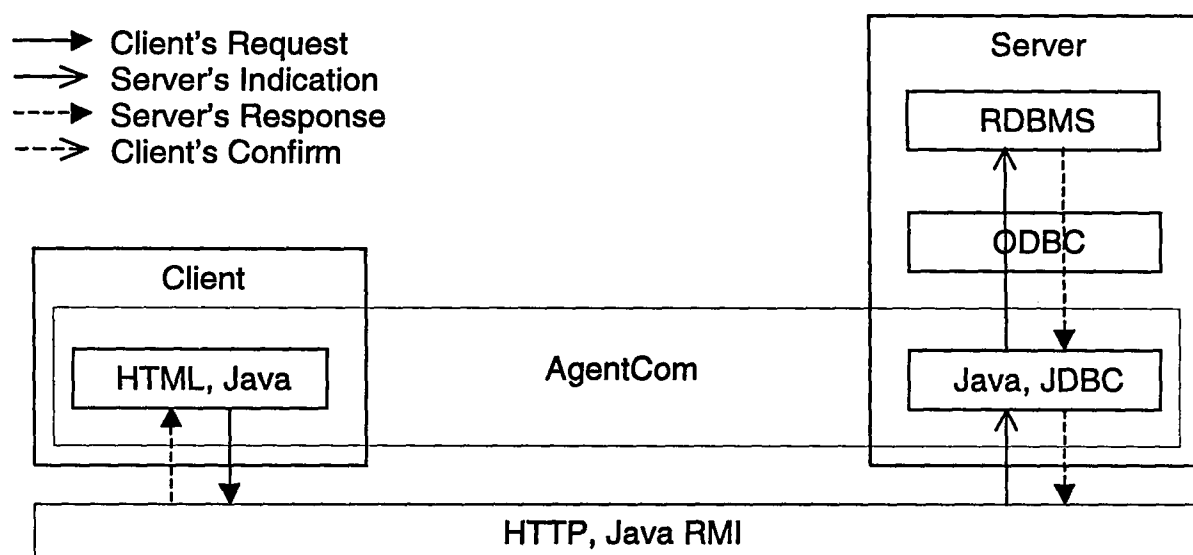


Figure E.1: Basic Tools and their Connectivity in CourseMan

programme to be started at boot time. The resource agent is written as Java application. With the first client side request to a database, it creates a database agent for that database. A database agent can access a database over a related ODBC configuration, by using Java's JDBC-ODBC Bridge. All programmes running at server site are bound to one Java script (*Figure E.1*).

All agents implement the AgentCom language interface to communicate with each other. Communication between agents that are compiled to one code is made synchronously. Communication between agents over the network is done in asynchronous mode.



Appendix F

Software Design of the Prototype

Class Diagram

CourseMan consists of various object-oriented classes. Their interrelationships are depicted in (*Figure F.1*). Besides the utilisation of standard classes of the tool's library following classes were developed.

SemanticNode: Provides basic functionality to construct and modify a semantic node, and to find its attributes.

KnowledgeBase: It creates and uses *SemanticNode* and provides the functionality to construct, store, and modify a semantic net, and to find nodes. It maintains a list of all stored templates, where each template represents a group of concepts, and a list of all stored concepts.

AgentCom: Implements the agent communication language *AgentCom*. Uses the *KnowledgeBase* to wrap between data represented in *AgentCom* and its semantic net representation.

InferenceMechanism: Uses the *KnowledgeBase* and implements mechanisms to evaluate the state of a topic, which is the semantic net with the current focus. It determines firing conditions and executes them.

Agent: This is a super class that creates one instance of *KnowledgeBase* and one instance of *InferenceMechanism*. For each *AgentCom* message an *AgentCom* instance is created. It is further an abstract class that forces its sub-classes to implement some methods.

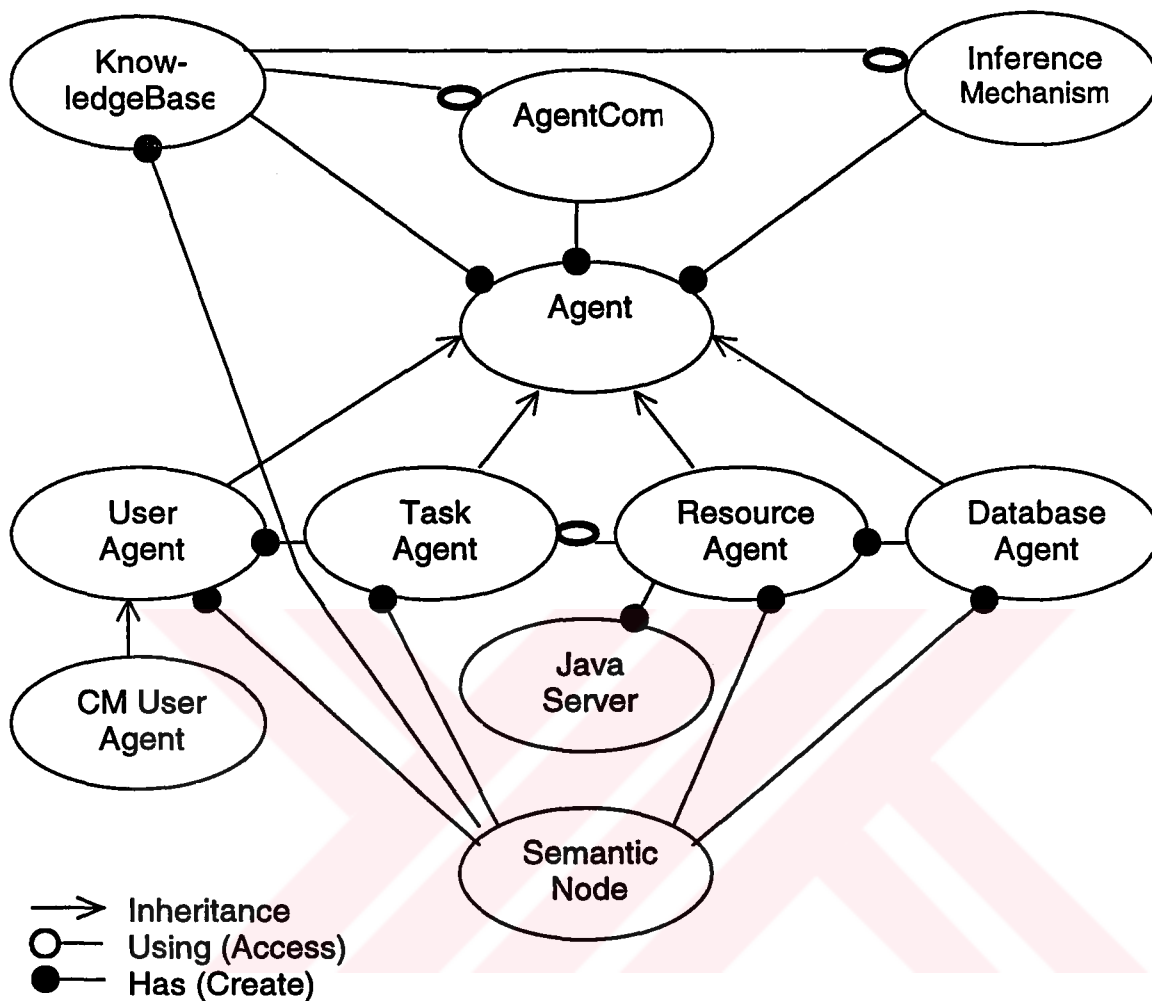


Figure F.1: Class Diagram of the CourseMan Software

UserAgent: It is a sub-class of Agent. It uses SemanticNode to retrieve the knowledge base and to wrap between semantic net data and data that can be displayed in the user interface. It creates one or more TaskAgents to execute user transactions and other tasks.

CMUserAgent: This is the CourseMan User Agent. It implements a default user interface for the CourseMan domain. It is a sub-class of UserAgent.

TaskAgent: It is a sub-class of Agent. It uses SemanticNode to retrieve the knowledge base. It uses remote created ResourceAgents. Divides a user transaction into one or more sub-transactions and establishes for each one an AgentCom session.

JavaServer: This class implement the client site stub and the server site skeleton. It is a demon process that listens on a TCP/IP port for incoming AgentCom messages. When started on server site, it creates one ResourceAgent.

ResourceAgent: It is a sub-class of Agent. It uses SemanticNode to retrieve the knowledge base. Creates one or more DBAgents and brokers AgentCom messages between them and TaskAgents.

DatabaseAgent: It is a sub-class of Agent. It uses SemanticNode to retrieve the knowledge base and to wrap between semantic net data and SQL data. It implements database access according the related data dictionary.



Appendix G

Data Structures of the Knowledge Base

The knowledge base of a CourseMan agent contains several concepts stored in form of semantic nets. A semantic net is a directed graph of connected nodes with attributes.

Semantic Node

The nodes of a semantic net have a common structure (*Figure G.1*). Each node has a name represented as a character string. It has an attribute list and a value list, which represent attribute-value pairs. The relationships between the two lists can be 1:1 or 1:m, where m is the number of values of an attribute. For example the number of values of an attribute of a table will be equal to the cardinality of that table. A node has further a link list and a type list for representing relationships to other semantic nodes. Their list elements are related to each other only 1:1.

SemanticNode	
Name:	String
Attribute:	List of String
Value:	List of Object
Link:	List of SemanticNode
Type:	List of String

Figure G.1: Node Structure of a Semantic Node

Semantic Net

A semantic net of the knowledge base consists of at least one semantic node. A semantic net with unconnected graphs would cause a syntax error. The link semantics of a sample net are not proved. For example whether it has a hierarchical structure, cycles, or bi-directional connections between nodes. As an example, the semantic net representation of the AgentCom language is depicted in (Figure G.2). The nodes in italic are placeholders for representing equal rank of nodes in the same hierarchy level. In other words, the syntactic correct occurrence of the language primitives is represented in hierarchical levels.

Concept

The knowledge base of an agent contains two main groups of semantic nets: templates and topics. A template defines the basic structure for a group of similar topics. It is used by the inference mechanism to traverse related topics. All templates of a knowledge base are stored in a list (Figure G.2). Also, all topics of a knowledge base are stored in a further list.

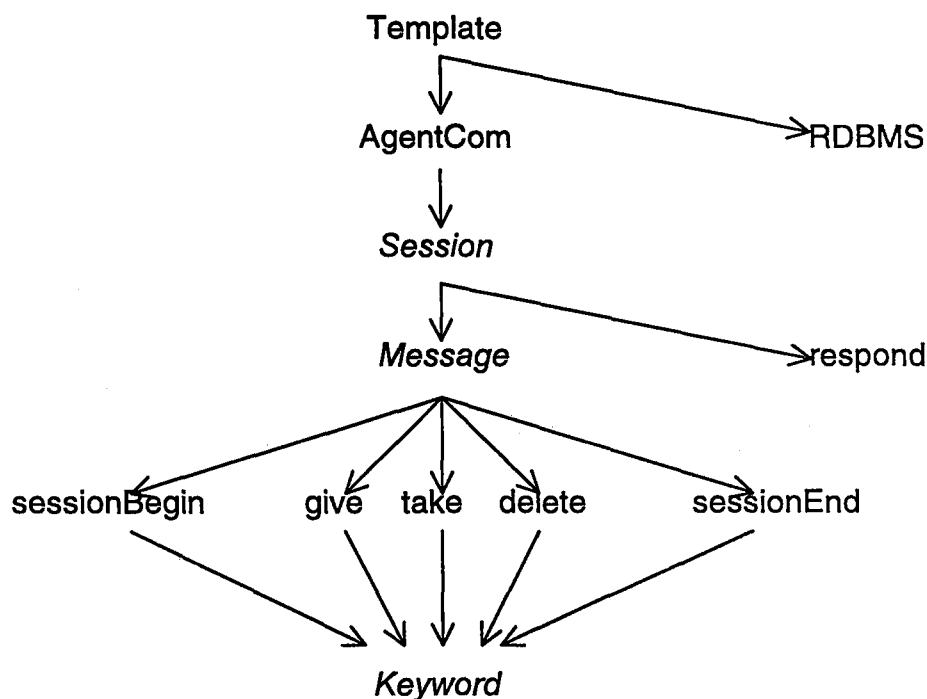


Figure G.2: AgentCom Syntax Represented in a Semantic Net Template