Mobile agents using the Tracy environment

Thomas Rietzler

Submitted in partial fulfilment of the requirements of Napier University for the degree of Bachelor of Science with Honours in Computing

Supervised by Dr. Bill Buchanan

School of Computing
May 2002

Authorship declaration

I, Thomas Rietzler, confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed.
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work.
3. I have acknowledged all main sources of help.
4. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself.
5. I have read and understand the penalties associated with plagiarism.

Abstract

Mobile agents are often presented as the future of distributed computing. They introduce a new approach to the traditional client/server architecture, which hides the network complexity to the end-user and makes data transfers asynchronous. This is more and more appreciable in a world where the overall network structure is dynamic, due to the mobility of computer component themselves like mobile phones, and the fact that servers providing new services appear everyday on the Internet while some other disappear.

Mobile agents have tremendous potential, and are subject to a lot of research at the moment, even if applications of it are not yet wide spread. Agent technology is up to now only use by academia and a few industrials. But they are expected to become more popular in the years to come. Their application includes user tracking, improved client-server communications, and for auditing purposes like network monitoring.

This report presents an application of mobile agents under the mobile agent system Tracy. Its main components are databases, mobile agents, and stationary agents to interface services with the mobile agent system.

Gathering data over a distributed system, if not organized, requires a client to connect and query every server on the distributed system. The application developed here collects data without having to know which server to query: agent technology is in charge of the distribution and collection of data. This data is taken from databases, and extraction is filtered by SQL queries.

This document reports the rigorous testing of the Tracy environment, with details on how the application developed with it was designed and implemented, what it really does, and how well it performs.

Table of content

List of illustrations

Acknowledgements

I would like to thanks my supervisor, Bill Buchanan, for his help during this project, and for providing the subject of this project which was a pleasure to study.

1 Introduction

1.1 Background

Mobile agent technology is nowadays one of the most active research topics in computer science. It is now proven that many areas could benefit from mobile agents. Many papers like Mobile Objects and Mobile Agents: The Future of Distributed Computing? [7] investigated on mobile agents' concept and theories. Most of the current research on mobile agents has two general goals: reduction of network traffic and asynchronous interaction. This research is driven by the fact that mobile agents will be soon developed at larger scale, for commercial application. Kotz and Gray [1] predict that, *within a few years, nearly all major Internet sites will be capable of hosting and willing to host some form of mobile code or mobile agents*. But before such a deployment, some issues need to be investigated and some problems need to be fixed. Security of mobile agent systems and interoperability are surely the main concerns for commercial development of this technology. Rothermel, Hohl, Radouniklis [5] investigates on mobile agent security. Standardization efforts are mostly done by the Foundation for Intelligent Physical Agent (FIPA) and the Object Management Group (OMG) with Mobile Agent System Interoperability Facility (MASIF).

Academic research is more concerned about how to improve mobility and performance. Following is the call for paper for the ECBS 2002 [8] Conference and Workshops which reflect the actual research areas in mobile agents:

We seek research contributions and experience reports that address the performance, interoperability, and applications of mobile agent systems. We particularly solicit submissions covering but not limited to:

- *Mobile agent system architecture design issues.*
- *"Components-off-the-shelf" for mobile agent systems.*
- *Quantitative and qualitative analysis and comparison of mobile agent systems.*
- *Benchmarks and performance analysis of mobile agent systems.*
- *Design Pattern to improve the performance of mobile agent systems.*
- *Performance of mobile agents vs. other approaches.*
- *Scalability of mobile agent systems.*
- *Resource control.*
- *Mobile agent applications.*
- *Design patterns and abstractions that promote interoperability of mobile agent systems.*
- *Work related to the interoperability and standardization of mobile agent systems.*
- *Experiences with mobile agent testbeds and publicly accessible servers.*
- *Agency and service discovery for mobile agents.*
- *Critical reviews of existing approaches.*

1.2 Aims of the project

The University of Jena, in Germany, have developed a mobile agent environment called Tracy which can be used to develop mobile agents using Java. The aim of this project is to review this environment by testing it with the development of an application designed to distribute, collect, and filter data over a distributed system, and test the performance of mobile agents created. The application is required to run over a distributed system, and be capable of transferring, collecting and filtering data over it.

The report splits into three main parts:

- **Part 1 [Investigation]**. The first part of this report relates to an investigation on mobile agents and the Tracy environment, under which the application was developed.
- **Part 2 [Design]**. The second part relates to the design of the application itself, what kind of application was chosen, how it works and explaining different strategies of mobile agents within this application.
- **Part 3 [Implementation and Evaluation]**. The third part exposes some details concerning the implementation, as well as the evaluation of the application.

2 Mobile agents theory and investigation on Tracy

This part relates to theory and concepts needed to understand the work done. It is the result of the study of the Tracy environment, needed before developing an application using mobile agents.

2.1 Introduction to mobile agents

2.1.1 Software agents

The definition of a software agent is always subject to discussion. Many definitions are already available, and even if the term can sometimes be misused, no definition is right or wrong, and no definition is universally recognize. Franklin and Graesser [2] try to define at best different kind of agents and try to classify them as precisely as possible.

Basically, agents differ from standard computer program by the fact they don't constitute an application by themselves. They are characterized by their small size. They aim at reducing human workload by only interacting with users at the beginning or the end of their process.

Agents have some characteristics:

- **Authority**: they have delegation, represent someone and act on behalf of someone.
- **Reactivity**: they react to stimuli and learn from their environment.
- **Proactivity**: they can take initiatives, they are goal driven.
- **Autonomy**: they have control over their own actions.
- **Social ability**: they are able to communicate with other entities, and may cooperate and collaborate with them to reach their goals.

2.1.2 Mobile agents

A mobile agent is an agent that has one more characteristic: its code is mobile. While a stationary agent executes its code on the same host all its lifetime, a mobile agent have the ability to transport and execute itself over a network, in a heterogeneous environment.

By code mobility, it is meant that the mobile agent not only transfers its code, but also it's being: code, data and state. It is possible for it to begin an operation on one host and continue it on another, while updating its data after the visit of each host.

2.1.3 Agent server

An agent server is also known as the agent execution environment. An agent servers controls agents: it creates then, executes them, transfers and terminates them. It provides some services such as inter-agent communication.

2.1.4 Mobility

A feature of mobile agent systems is their ability to move mobile agent from one place to another. There are two ways to support mobility of a mobile agent: weak and strong migration.

In strong migration, the agent is transferred with its code, data and its complete state: it allows the mobile agent to be executed exactly from where it was left before migrating: it resumes itself.

Weak migration is every other kind of migration which is not strong, therefore which only transfer code and data when migrating the agent.

2.1.5 Agent architecture vs. client/server architecture

In the traditional client server architecture, all connections to server are initiated from the client. The advantage is that these connections can be managed in parallel (Figure 1). With an agent architecture (Figure 2 the network is dynamic, and the client does not have to know the structure of the network. The fact that agents communicate with a high-level communication language and work over a logical network makes them more reliable as the logical network which is dynamic will adapt itself to the current conditions and be less affected by network failure.

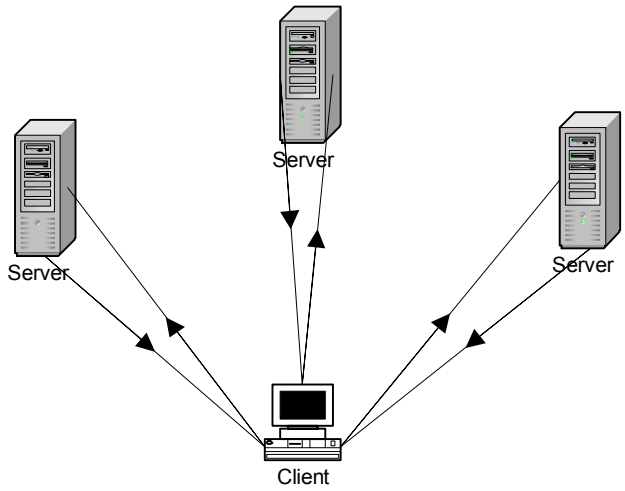


Figure 1: Traditional client/server communication

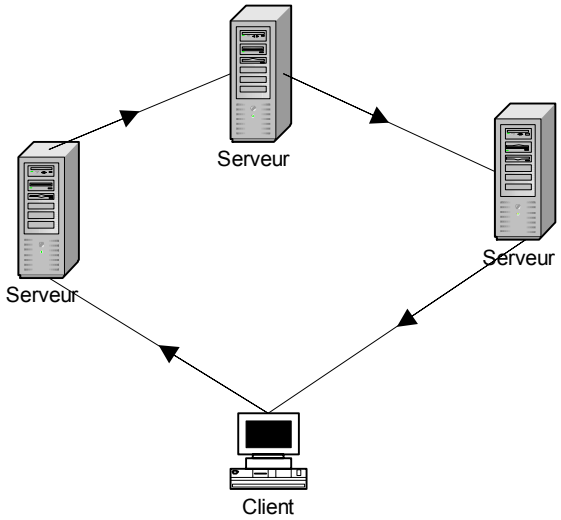


Figure 2: Communication with the use of mobile agents

Mobile agents differ from traditional client/server application by moving themselves where the data are, instead of moving the data to where the application resides. This improves performance of the data collection by requiring less bandwidth.

Another advantage is the fact that mobile agents can perform their task asynchronously, or offline: the host of origin can initiate agents, tell them to migrate, go offline, and when back online, waiting for agents to come back.

2.1.6 Mobile agents environments

Several mobile agent environments have been developed, as outlined by Naylor, *et al* [12]. Each of them has its own functionality: interoperability, migration option, language used, ease of implementation, and many other functionalities. Someone has to choose it depending on the requirement and aim of its project.

Some people will only use one for research on how to improve mobile agent environments and test the architectures to compare them, as well as improve the current design. Some other will need one to develop an application, to test if an environment could be easily deployed or is suitable for commercial purposes.

Well-known environments are Telescript, Aglets [13], Concordia and JATLite [14] just to name a few. Naylor [12] evaluated three mobile agent development toolkits are evaluated, Voyager from Object Space, JATLite and the Aglets Software Development Kit (ASDK) from IBM, and chose Aglets as being the most suitable for the prototype. Several developers, though, have had problems in implementing mobile agents with the Aglet development system. Tracy seems to overcome many of these development problems.

2.2 What is Tracy?

Tracy is a mobile agent system, being developed at Friedrich Schiller University Jena, in Germany [10]. It is programmed in Java and is therefore not platform dependant, and was designed with two goals:

- For research purposes on mobile agents, Tracy was developed with an open agent migration model.
- To provide a mobile agent system suitable for application development relying on mobile agents.

Mobile agent technology makes distributed application easier to manage and program as the agent environment does things a developer would have to program with a traditional architecture. While in TCP/IP, the developer has to deal with 4 OSI layers, from the transport layer when programming sockets to the application layer, Tracy deals with transport, session and presentation layers, and so the developer only has to build his application on top of it.

<table>
<tr><th>Protocol
layer</th><th>OSI</th><th colspan="3">Tracy</th><th>TCP/IP</th></tr>
<tr><td>7</td><td>Application</td><td colspan="3">Application</td><td rowspan="4">Application</td></tr>
<tr><td rowspan="2">6</td><td rowspan="2">Presentation</td><td colspan="3">Agent system layer: agent management</td></tr>
<tr><td colspan="3">Package manager layer: migration process management</td></tr>
<tr><td>5</td><td>Session</td><td rowspan="4">Net transmission layer: data exchange between agent servers</td><td colspan="2">Net Manager</td></tr>
<tr><td rowspan="3">4</td><td rowspan="3">Transport</td><td colspan="2">SATP (simple agent transfer protocol)</td><td rowspan="3">Transport</td></tr>
<tr><td>RMI</td><td>SSL</td></tr>
<tr><td>TCP</td><td>UDP</td></tr>
<tr><td>3</td><td>Network</td><td colspan="3" rowspan="3">Java Virtual Machine dependant</td><td>Internet</td></tr>
<tr><td>2</td><td>Data link</td><td rowspan="2">Network access</td></tr>
<tr><td>1</td><td>Physical</td></tr>
</table>

Figure 3: The Tracy architecture: Tracy, OSI and TCP/IP

Figure 3 shows how Tracy fits within the OSI model (the Tracy environment is in green) and shows the details of the Tracy layers themselves in comparison to OSI and TCP/IP. But it doesn't mean that the developer cannot interact with Tracy: the developer can make some choice, such the transmission strategy (net transmission layer), or the migration strategy to use (package process management layer).

2.3 How does Tracy works?

2.3.1 Management of the logical network and agent servers

As seen in Section 2.1.5, a mobile agent system's particularity is to be independent from the physical network. The mobile agent system has to maintain a logical network in some way. In Tracy, an agent server is referred as a node. A node is made of a hostname, and a server name. These are the value specified in the property file.

In version 0.54 of Tracy, which is the one used for this project, an agent server can either be a domain manager, or a single node. When an agent server is started, it first looks its neighbourhood for a manager. If it finds one, it registered to the manager, and acts after as a single node. If it does not find a manager, which means the agent server is the first one on the network, it starts itself as the manager. The neighbourhood in Tracy 0.54 is the subnetwork on which the agent server is started, which the agent server polls by UDP broadcast. This means that the size of the Tracy logical network with version 0.54 is restricted to the size of the subnetwork. The domains are dynamic: if the domain manager goes offline, another node will relay it. The domain manager keeps a record of all the nodes registered to it, pinging them regularly to keep the network up to date. Some system agents are in charge of this job. A Tracy domain manager behaves like a DNS server in an Internet network.

This difference of concept between physical and logical network is represented in Figure 4. In red, the Tracy logical network created by agent server interconnection. In black, the network and process to system link. Tracy 0.6, which is not publicly released at this time, introduces the notion of "master node", which acts as global domain manager for local subnetwork managers. It removes any domain limit for the Tracy network.

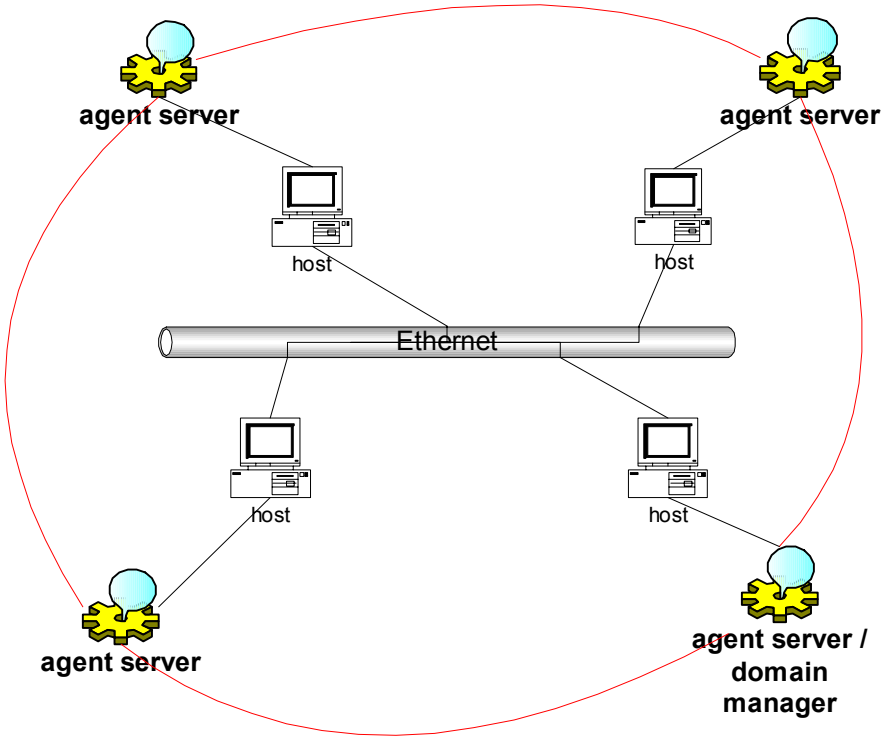


Figure 4: Physical network and Tracy network

2.3.2 Types of agents

There are two categories, and three types of agents in Tracy:

Stationary agents: For security purposes, mobile agents cannot have direct access to the host system. Mobile agents are restricted to communicate with the agent server and other agents. Stationary agents don't have the ability to migrate, and they are therefore considered as secure and granted access to the host. They are used as a dynamic interface between mobile agents with which they can communicate, and the host system. Tracy defines two types of stationary agents:

- **Gateway agents**
  By convention, gateway agents are used to interface with host's applications.
- **System agents**
  By convention, system agents are used to interface with host's operating system. In reality, in Tracy 0.54, they are gateway agents which have two more methods to read and write system entity to the blackboard[1]. A gateway agent has also this ability to do so with some other methods, so the difference between both agents is, up to now and according to what I was able to access[2], purely conventional.

[1] cf. agent communication later on
[2] the source code of Tracy is not available to the public

Mobile agents. Mobile agents are agents travelling between agent servers. They dock to agent servers, and communicate only via the agent server. They cannot execute themselves out of this environment. To reach their goals, they have to communicate with other agents. They are transferred by agent server, on their own request (they have control ever their own actions) or the request of the agent server. The migration strategy, which the agent server has to take into consideration to transfer a mobile agent, is specified in the mobile agent code. They are referred by a name, and a home platform, and they can be killed by themselves, or by the agent server that host them.

2.3.3 Agent communication

There are two communication channels in Tracy within agents:

- **Message passing**. Message passing is a synchronous communication channel between agents. The sender can be either the agent server or an agent, the recipient an agent. The recipient agent has to be present on the agent server to get the mail. If an agent is trying to mail another agent which is not there, an exception is thrown. An agent in a waiting state still listens for incoming messages. The reception of a message sets its state back to running. The implementation of messages allows only the transfer of character string.
- **Blackboard**. The blackboard is an asynchronous communication channel between agents. The blackboard is maintained by the agent server, and acts as a mailbox for agents. Agents have to check the blackboard regularly, the delivery is not automated. The implementation of the blackboard allows the storage of any kind of objects on the blackboard. Some types are predefined, like strings, images, XML content, but the programmer can easily defines its own types.

2.3.4 Migration

Migration strategy and transmission strategy are both open in Tracy: it is possible to make some choices, depending on what could be best for the application. Different push and pull strategies have been implemented. Push strategies consist in connecting to another agent server and transfer the mobile agent's code (upload from source to destination), while the pull strategy is initiated by the recipient, who loads the mobile agent (download from the source to the destination).

The mobility offered by Tracy is currently a weak form of mobility. Migration of Mobile Agents in Java: Problems, Classification and Solutions [6] tells why strong mobility is not easy to implement in Java and gives some solutions.

2.4 Start Tracy

2.4.1 Software requirements

Tracy is not given with everything it needs to run in its package. Tracy first required a Java Virtual Machine (JVM), version 1.3 or later. Either a Java Runtime Environment (JRE) or a Java Development Kit (JDK) would do it. I used the JDK version 1.3.0_02. Then, she requires some JARs (Java Archives) from Sun. These are the archives for encryption services: JCE (Java Cryptography Extension) version 1.0.2 or later, and JSSE (Java Secure Socket Extensions) version 1.2.1 or later. These JARs as well as Tracy's JARs must be added to the classpath.

2.4.2 Access control

Tracy has an access control mechanism, which authenticate users before starting an agent server. The User Manager, which allows the root of Tracy to set user rights, must then be configured before the first agent server is started.

When starting a new agent server, a user is asked to provide his login and password. These are compared with values in the tracy.user file, in the Tracy root directory. As you can see on the screen shot of the User Manager in Figure 5, it is possible to set different permissions for each user. The last permission, editing user database, is usually only allowed by the root.

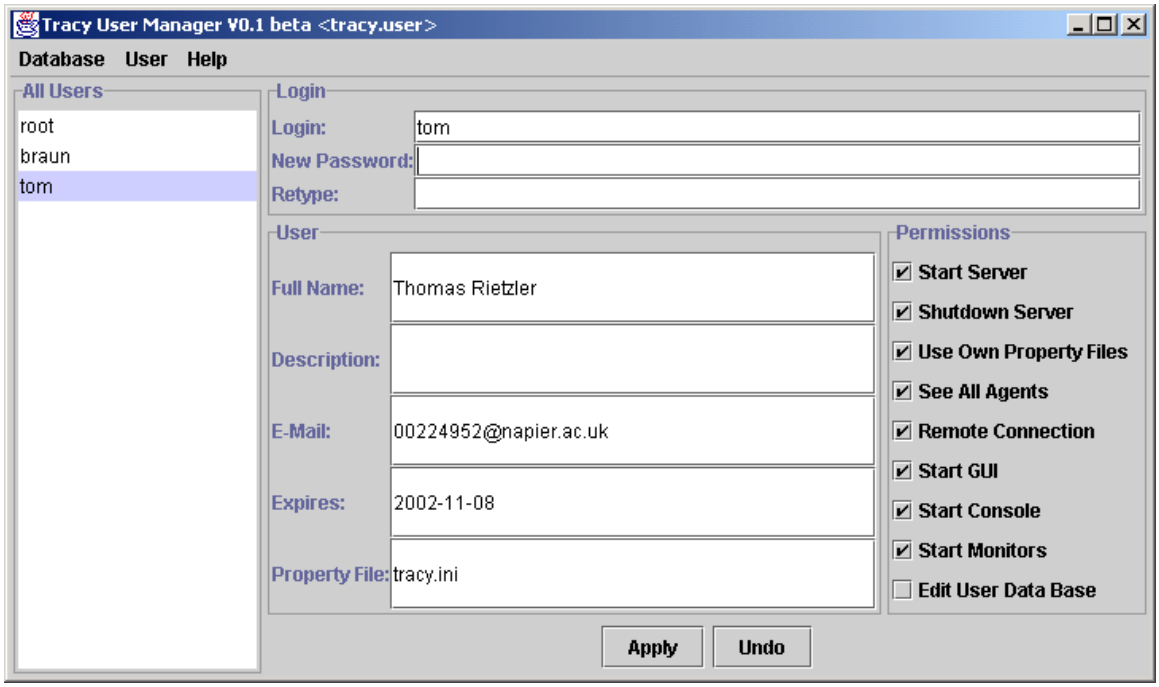


Figure 5: Tracy User Manager main screen

2.4.3 Property file

The property file (by default tracy.ini) is the configuration file for the agent server. It contains information such as its name, its hostname, and the properties for the net layer (type of connection) with port numbers to use. Transmission techniques are configured here, while migration options have to be specified in the code of agents themselves.

2.4.4 Starting and monitoring agents

The Tracy GUI is a useful tool while developing mobile agents with Tracy. It allows the developer to start and interact with agents, with a user-friendly tool. It is possible to start and stop agents, send messages to agents, read and post data on the blackboard, see which agents are currently on the hosts. All mobile agents coming in and leaving the host are logged. It was really indispensable during the development.

But while the GUI is very useful during the development phase, it is not very user friendly, and when the development of the application is finished, there is no more need to have all the debugging option the GUI provides. The last step in the development of the application is to instantiate a Tracy server within a Java class,

and start agents from this class. The user interfaces, if any, have to be coded within the main class or the gateway agents.

The reader should by now have a good understanding of what mobile agents are, what their characteristics are, and by what Tracy characterize herself. Technical details of Tracy are presented and explained in papers [3, 4] published by the developer's team.

3 Requirements, analysis, design

This part reports what kind of application it was decided to implement. It tells why the chosen application was chosen, what improvement this application could bring compared to actual technology, and how this application works.

3.1 Requirements for the application

In theory, every kind of application based on the standard client/server architecture can be ported to mobile agent system architecture. But mobile agent systems have their strength and weaknesses compared to standard client/server architecture. Several kind of application can benefit from mobile agent. Here are just a few of them:

- **Parallel processing**: the mobile agent is on charge of distributing work to online hosts, and to retrieve the results to dispatch the global result on the home host.
- Information dissemination: the mobile agent can be used to broadcast information, perform remote updates.
- **Network management**, monitoring: the small size of agents is used as an advantage to keep network resources free for application. We find as well intelligent agents which are agents coupled with AI algorithm for intrusion detection.
- **Distributed information retrieval**, electronic commerce: the mobile agent is used for data collection and filtering to improve information searching.

The requirement of the technical part for the project was very vague at the beginning; it mainly depended on the results of the investigation on the environment.

My supervisor, Bill Buchanan, was open to any idea. His general ideas as specified in the contract were to transfer data around a distributed system, or collected data around a network.

3.2 Analysis and design

My ideas for an application were either a network management tool, either an ecommerce-oriented tool. The choice was free. Research on the Internet on what has already been done was mainly focused on network management, so it was decided to orient the design to the ecommerce tool to do something innovative.

In order not to limit the application to a specific area (a specific business), it was decided that agents would collect data from a database, so any business could be interested. A business selling goods or services online has therefore just to put the available products in the database, like businesses today put products list on their web pages. Usually all Ebusinesses have a database behind their web site. Here, the web site which bridges the user to the database is replaced by a stationary agent. The user request is carried by a mobile agent, like an HTTP request is carried to a web site. The difference is that the mobile agent already contains a request when it arrives on the host, whereas with HTTP, a client has to connect to the web site, make his choice and request the goods.

This kind of system is for demonstration only, its main disadvantage is that all concurrent businesses targeted by the mobile agent need to have the same database structure, as the request passed by mobile agent is unique, and is meant to be distributed to all databases.

The overall application behaves as if a user was querying a huge database consisting in several databases. The distribution of queries, and the system of data collection is invisible to the user.

3.2.1 Design of the client and server sides of the application

Figure 6 shows the overall application design as described before, and defines what is meant by client side and server side of the application. Specific design of each software component had to be constantly reviewed during the implementation phase, as of course not everything worked as planned during the implementation phase.

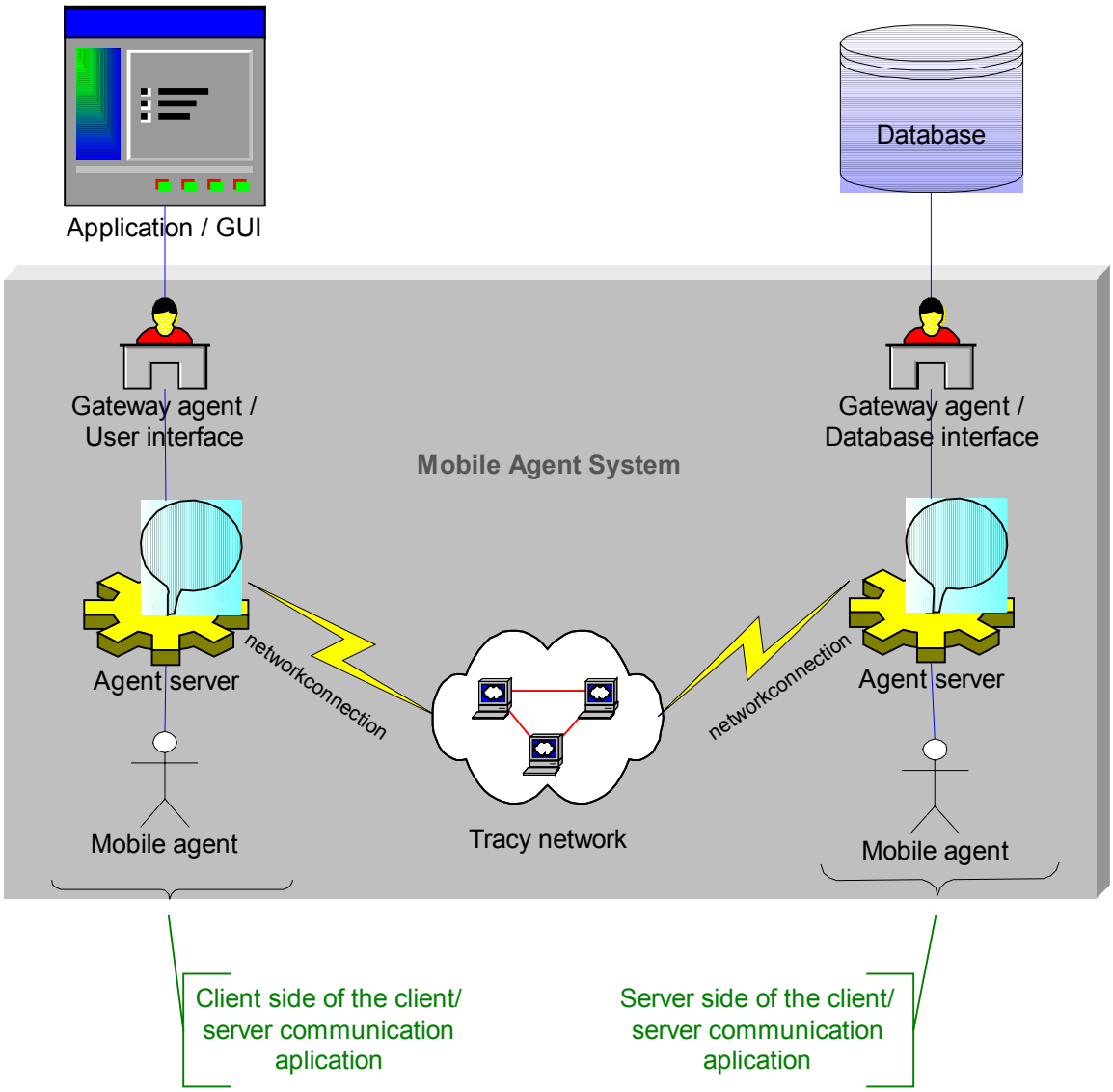


Figure 6: Overall application design

3.2.2 Design of gateway agents

Server side gateway agent
The gateway agent on the server side is a bridge between the database and the mobile agent. It listens for messages from mobile agents, and when it gets one, extracts the query of it. This query consists in:

- A sequence number
- The name of the database
- The SQL query

The gateway agent then forwards the SQL query to the specified database (step 2 of figure 7), and post the results of the query on the blackboard (step 4 of figure 7), with the sequence number as part of the label on the blackboard so that the mobile agent, that provided this same sequence number, can find its data.
The aim of this sequence number is for many mobile agents to do requests to a gateway agent without confusion.

The aim of the name of the database is that it is then possible for a server to host several databases and have only one gateway agent to bridge them to the mobile agent system.

SQL queries allow the design to be compatible with any kind of database. Even if the developed version only connects to ODBC databases, a new driver, specific to a database, is easily implemented in this gateway agent. It is therefore possible to access a wide variety of database, over a wide range of operating system (as Tracy is platform independent) with the same mobile agent: the mobile agent can operate over a heterogeneous network and with heterogeneous databases.

The last action of the gateway agent for the query processing is to send a message to the mobile agent: in case this one is waiting, it will wake it up (step 5 of Figure 7). Data has to be passed through the blackboard for two reasons: the first one is that not all mobile agents will be available for a message delivery when the gateway would have finished the query processing. The other, and most important reason, is that complex data structures are more easily transferred from an agent to another via the blackboard, as it is possible to define our own objects to store on the blackboard. In our case, the type is a dataset, a structure of extracted results from a database. If this dataset had to be transferred via messages, it would have to be converted into a string, and then reconverted back to the original structure at reception of the message by the mobile agent.

Figure 7 resumes the situation, showing the sequence of messages and data transfers within the application on the server side. Note that messages are passed through the agent server in reality, which is not shown on this diagram.

Client side gateway agent
The client side gateway agent interacts directly with the end-user. It has a GUI, where the user provides the request (step 1 of Figure 8), launch the mobile agent by delegating a part of its job (step 2 of Figure 8). When the mobile agent is back, it posts its results to the blackboard and notify the gateway agent the data is ready to be processed (step 3 and 4 pf Figure 8), and the gateway agent reads data that has been found (step 5 of Figure 8).

The gateway agent is responsible for the creation of mobile agents, and for the final presentation of results returned by mobile agents before passing data to the application layer (step 6 of Figure 8).

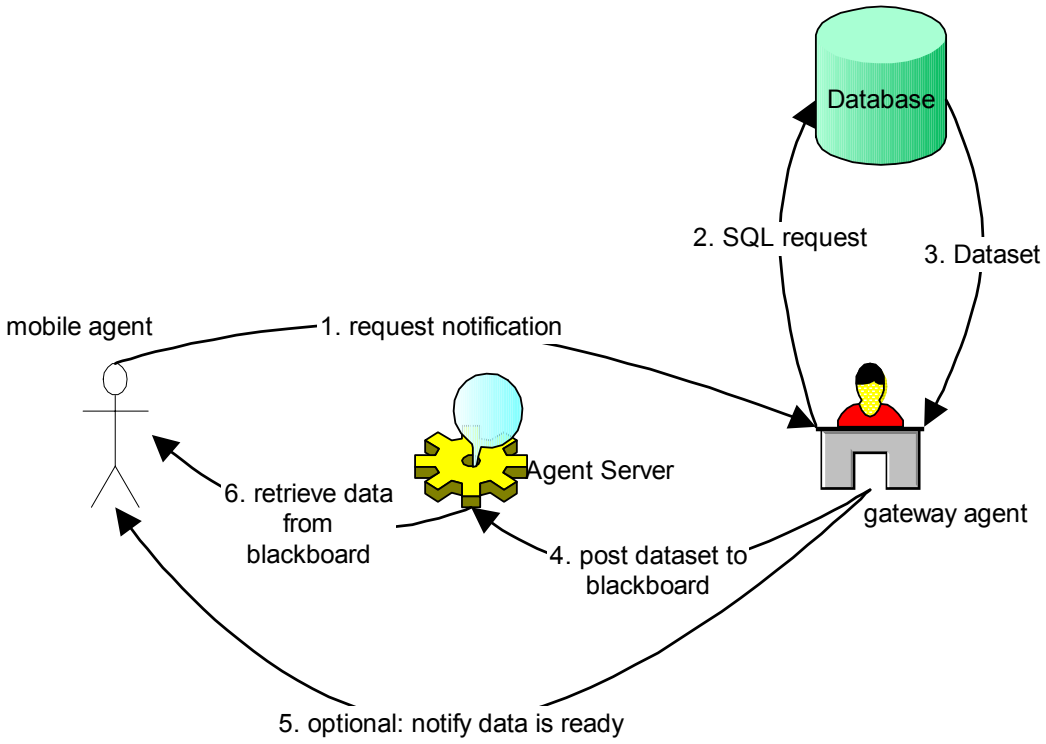


Figure 7: Interactions between entities on server side

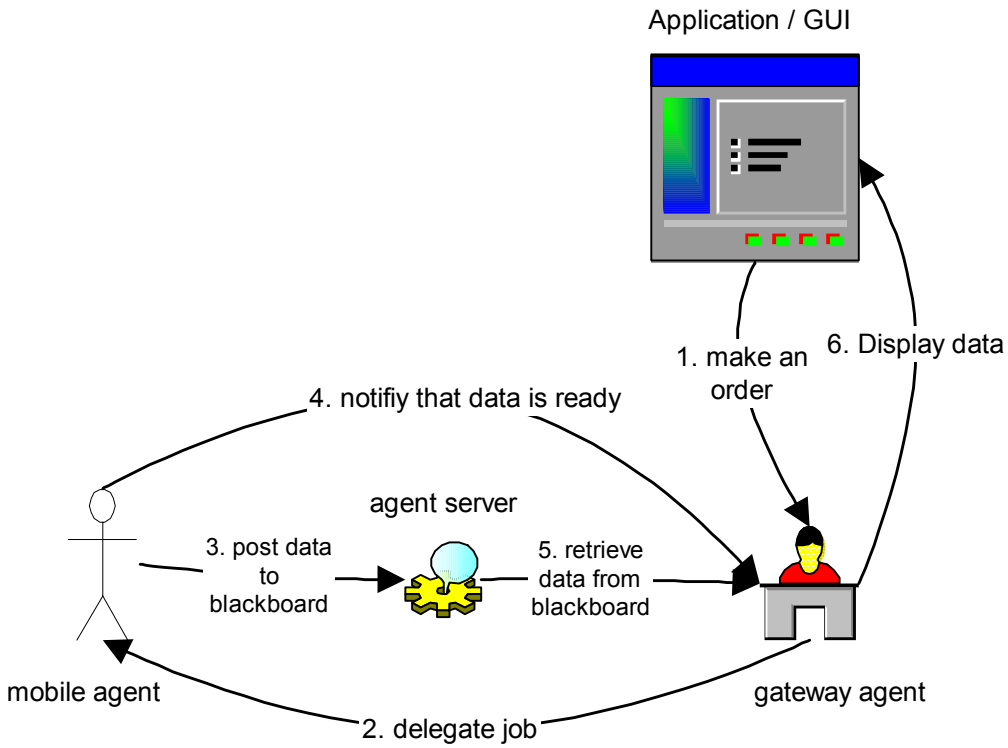


Figure 8: Interactions between entities on client side

3.2.3 Design of concurrent mobile agents

Many strategies for the design of mobile agents were conceivable. Two of them have been implemented to be tested against each other in a performance test.

Common part for both agent strategies:
At their creation, the gateway agent, client side, passes arguments to the mobile agent to define the job of the mobile agent.
Then, after the mobile agent migrated to a host, it tries to send a message to the gateway agent, server side (step 1 on Figure 7), and ask for the data its wants. If no gateway agent is found, the mobile agent continues its route.

> **First agent strategy: Agent A**
> If a gateway agent is found, which means the message was successfully delivered, the mobile agent continue its message delivery to all gateway agent until it did all nodes of the network and went back home. It then goes for the same tour, in the same order as the first one, checks the blackboard (step 6 on Figure 7) of each node for the result of the query it previously sent. It takes the result if there is one and go on until back home, where it posts all the results to the blackboard and notify the client side gateway agent that it finished its job.
>
> **Second agent strategy: Agent B**
> If a gateway agent is found, which means the message was successfully delivered, the mobile agent waits for the query it sent to be processes. When the gateway agent has processed the query, it posts the result on the blackboard and sends a message to the mobile agent to wake it up (step 5 on figure 7). At reception, the mobile agent checks the blackboard, takes the result and goes to the next node. At the end of the network tour, the mobile agent goes back home, posts all the results to the blackboard and notifies the client side gateway agent that it finished its job.

Mobile agents post their results to the blackboard for the same reasons the server side gateway agent posts its results to the blackboard: we deal here with complex data structure. Message communication is only used to notify agents of a predefined event.

Agent A would, in a distributed system, be the best choice in theory: it is less prone to be delayed or even be stacked at a node which does not response quickly, due to an overload or a malfunction. But this first agent has the disadvantage of doing two network tours instead of one for agent B, and it may take longer to complete the overall process in a small and fast processing network. We will see later which agent is the most effective depending on the size of the network.

4 Implementation details and tests

This part present the points were choices had to be done, followed by benches of mobile agents created for the application.

4.1 General considerations on Tracy

There is not a wide choice concerning implementation: agents have to be coded in a very strict way, as an agent extends a class already defined. Abstract methods must be defined; states have to be carefully control to avoid agents waiting for resources they will never get. It sometimes gets close to critical system programming. A mistake in the code of an agent can severely affect the agent server itself: robustness is not a characteristic of Tracy. Tracy is an alpha version, and things are far to be perfect. It happens that the manager does not reference a part of the network or loose a part of it while agents are travelling, resulting in agents trapped on hosts and even agent server to crash.

4.2 Database components and access

One of the advantages of the Tracy environment is that it is platform independent. Agents are supposed to be able to travel around a heterogeneous environment, and everything has been done to keep this real for the database connection.
First of all, the language used to query the database is the SQL language. It is the language supported by all databases.

The tests were performed with a Microsoft Access database, but the gateway agent developed to bridge the database to the mobile agent uses an ODBC driver, so any ODBC compliant database would do the job. Moreover, the Java database connectivity is done with JDBC, and many drivers are available for non ODBC compliant database on the Java web site [9]. A simple update of the driver would allow any specific database to be accessed by the gateway agent. Mobile agents are not concerned by the database type or platform operating system.

The server side gateway agent was connecting the database with the java.sql package, released with the JDK. It appears soon after that it would be easier to organize database's data with Jbuilder components. Jbuilder provide many containers for databases' results, for data representation, and methods for database connectivity and data organization.

The data extracted from the database is stored in a Borland QueryDataSet component. This component has many advantages, and makes data manipulation easier. The problem of this object is that its content is not serializable, only its structure is. When a mobile agent migrates, its objects are serialized, and if the mobile agent contain a QueryDataSet, the references to the local host will not be found on the host it is migrating to, and content will be lost.

To overcome this problem, the QueryDataSet is saved to a text file by the server side gateway agent. This operation generates two files: a file with the raw data, and a file specifying the structure of the dataset. These files are stored as an array of bytes, which is serializable. The mobile agent stores the files as an array of bytes in two distinct Java Vectors. At each node, an element is added to each Vector.
When the mobile agent is back on its home platform, it posts the Vector themselves to the blackboard of the agent server, and the client side gateway agent is in charge

of the reconstitution of the files: files are saved to the local host, then they are reconverted back to QueryDataSets (the ones from the servers), and then all QueryDataSets are concatenated to form only one QueryDataSet which is displayed to the GUI as the result of the request over the distributed system. The end-user cannot see in the result table which information comes from each host: it appears like if information was coming from only one host.

4.3 Blackboard content types

Any kind of data can be posted to the blackboard. But their type must be defined, so that agents which retrieve this data can know what its type is and handle retrieved data correctly. Any object cannot be posted straight to the blackboard: a class, implementing the existing BlackboardContent class, must be created to define a new blackboard content type.

Two new blackboard content types had to be defined for the application. The first one is used on the server side of the application: it is the array of bytes representing a binary file. The second one is used on the client side of the application: it is the Vector of arrays of bytes. These classes are shown in Appendix 2.

4.4 Migration properties

For the migration implementation, a fixed route is not set: the mobile agent first identifies which host is the domain manager, and migrates to it. Then the mobile agent asks the manager what are all agent servers connected to itself. From there, the mobile agent visits these nodes sequentially. When all nodes are checked, the mobile agent returns on the home platform.

4.5 Migration strategy

Many networks, among others enterprise's networks, have a policy to block incoming connection from the external network. The migration strategy chosen here is a push-to-all strategy: code and data is pushed to the destination host at once. In this case, an agent server must be allowed to accept incoming connections: the destination agent server listens for incoming connection, waiting for mobile agents to be pushed by other hosts to it. A user has to ensure that an agent leaving the network can come back, otherwise agent technology cannot be deployed.

4.6 Test results

All the tests were executed over a single switched 10Mbps subnetwork of P4 1GHz computers with 256 of RAM under Windows NT4. The test results measures the difference of time between agent's tour start (agent leaving home host) and agent's tour end (agent re-entering home host). The tests were performed with different number of servers, distinct from the client which generates mobile agents.

4.6.1 Agent strategy and network size influence

The first sets of results allow the performance comparison between agent strategies that were introduced in 3.2.3 in function of the network size. The data carried here by agent is what the application was designed to carry: request to be distributed, and result of the request of each node.

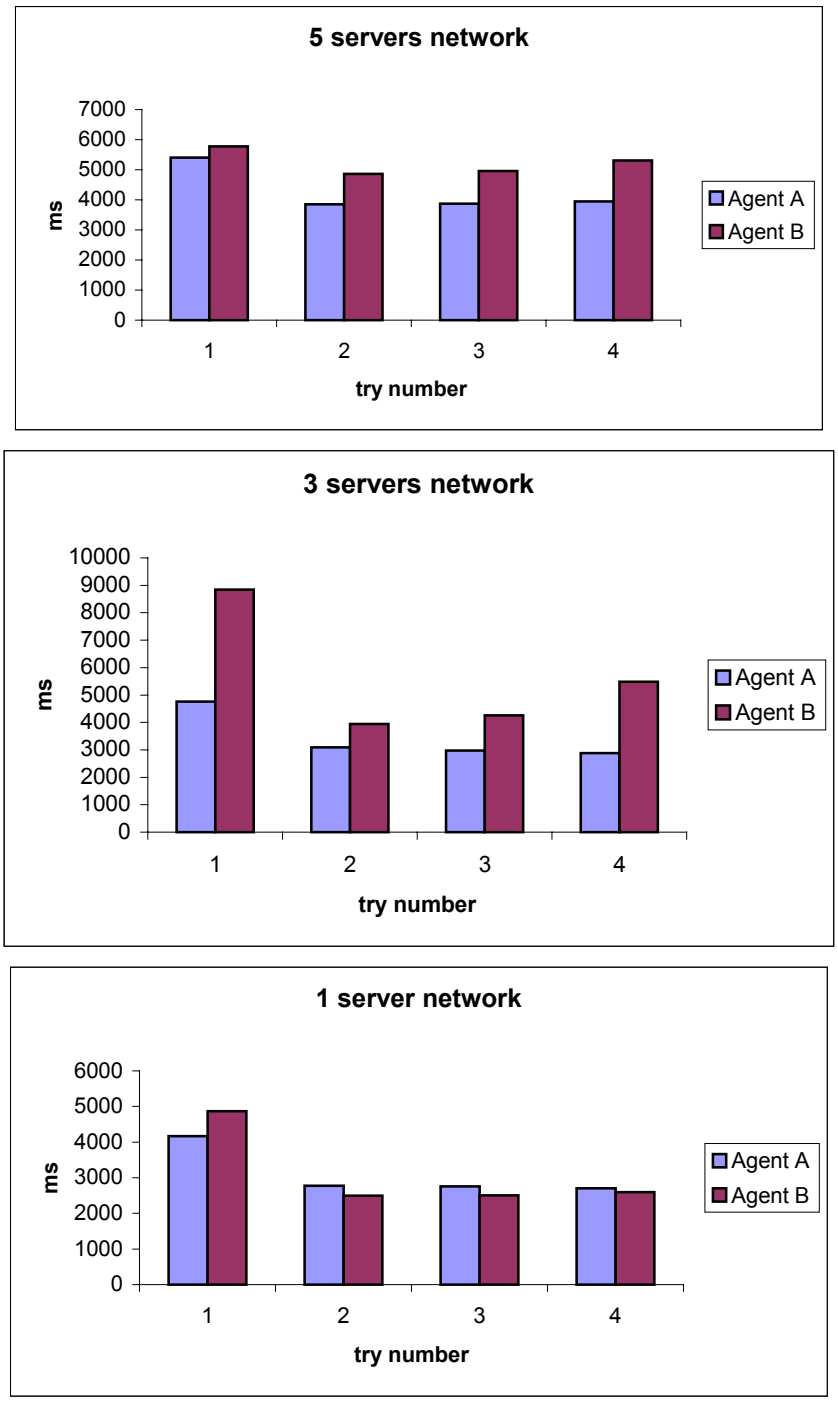


Figure 9: Size of network and agent strategy influence

Note that the first try is always much longer than the others. This can be explained by the fact that some variables in the agent server may not have been yet initialised.

We can see from Figure 9 results that, as it was suggested in 3.2.3, agent A, which does not wait for the host to process the request, operate faster that agent B, even if agent A travels twice as agent B. Transmission time is therefore less important than processing time for agent. This can be understood by the small size of agents: they are fast at migrating. Agent B can go even worse if the server is busy

at processing other requests. In our tests, the gateway agent was the only user process to run, but if it is not, the performance can really be affected. The first versions of agent B were hogging the CPU in a while loop while waiting for the results to be posted by the gateway agent, instead of being set to waiting mode. This resulted in a delay of about 15 seconds before the NT scheduler gives the gateway agent a chance to run.

It was suggested as well in 3.2.3 that agent B would maybe be faster over small network. It is the case for the 1 server network test, but the difference between both agents' timing is not really significant (less than a half millisecond), and we may not consider a client and a server alone as a network.

4.6.2 Agent load influence

We can see in figure 10 that agent can be very penalized by their size. Data has to be stored in objects. This is a serious limitation, Java very quickly generates an OutOfMemory exception when trying to create large objects. To reach such a high amount of data to transfer with a database request, the request has to be really wide and the database or the network very large. Nevertheless, there is a solution in this case, which will keep the advantage of using the agent architecture for distributing a request, and take the advantage of a client/server architecture to get the result. The solution would be to keep agent A for distributing the query, to open a listening port on the client, and for the server side gateway agent to make a connection to the client to transfer data instead of posting data and having a agent to retrieve it. Therefore, data size is not anymore a constraint. It could as well speed up the process, as as soon as the data is available, it is transmitted to the client. This could be used for online peer to peer file sharing.

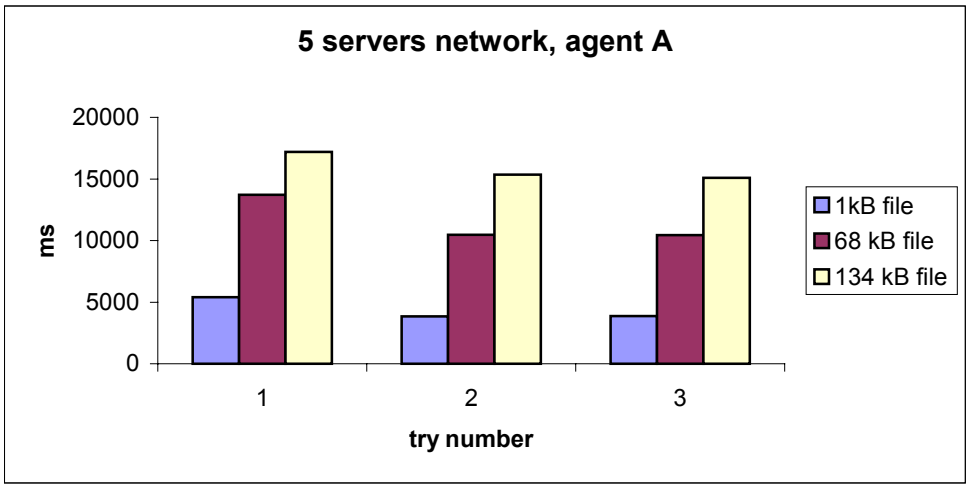


Figure 10: Agent load influence

5 Conclusion

5.1 Evaluation of achievement

Tracy is a good environment for research, but is not robust enough to allow any serious application based on her. The agent server crashed quite often during the tests. The logical network convergence time is also sometimes very slow, notably when recovering from an agent server crash; and the best recovery procedure is to kill all agent servers and restart them all. But we cannot blame the environment, its aim is not to guaranty stability. Tracy is for the moment only an alpha version.

However Tracy looks promising. A lot of functionalities are available, and her developers are willing to enhance actual features and extend functionalities in future versions, such as extended mobility strategy and allow larger domain management.
Even if programming for a given environment can be constraining, Tracy offers as well some liberties to the programmer, such as transmission strategy, migration strategy, definition of types for data on blackboard.

The goals of the application have been achieved, the application is capable of collecting and filter database content, and present this information as if it was coming from only one host. The distributed system is totally invisible to the end-user, who sees the distributed system as a mainframe computer containing all data that can be found on the network.

The three domains specified in the requirements have been covered by this application:

- Transfer data over a distributed system: distribution to all online hosts hosting the specific database (directed broadcast) of one same request.
- Data collection: the result of the request is taken back to the source.
- Data filtering is made by the request itself: the entire database is not taken back to its originator, only data that has been extracted from the database according to the request is. The SQL language is the filter for database content.

In terms of implementation, it can be seen that the advantage of mobile agents over traditional techniques for some application, including the one that has been studied here with data collection over a distributed system, is absolutely incontestable.
Mobile agents puts fewer burdens on both developer and end-user, makes transactions faster and more reliable with the management of a dynamic network infrastructure. The perspective of the underlying network is totally changed: the developer does not deal anymore with static addressing, and the administrator of a network does not have to reconfigure application if a change in the topology is made. A similar design to the one proposed over the traditional client/server architecture would request a static server to act as a manager and refer all nodes available for a request. Static addresses would have to be entered at each node for them to connect this central server or one of its backup.

Agent technology also provide a more secure design in comparison to a remote database access: server data and database are only accessible by the gateway agent hosted by the server. End-user don’t have any direct view and access to the host content, and the manager of the host can apply specific restriction by modifying the gateway agent hosted.

It has been seen as well those mobile agents have to keep their small size to stay efficient.

5.2 Directions for future work

Gathering information from a distributed database was a way to keep the design neutral, so it can be used in any context on any system. The application developed in this way can be used as a middleware itself. Everything has been done for the application to be compatible over a heterogeneous network, like Tracy and mobile agent systems in general: ODBC databases, SQL language, easy upgrade of the database driver on the gateway agent would allow anyone to use this application.
The SQL language allows retrieval of data from a database, but if the rights of the database are set up to accept modification, the SQL language can be use to post or modify data from the database.

Migration strategy could be improved and automated to go through firewalls, by implementing a pull strategy instead of the current push strategy, which would allow agents to reach hosts protected by firewalls.

Another problem which has not been taken into consideration is that an agent can be lost, if for example the node on which the mobile agent is goes offline without warning. This can be detected with a timer on the client, which can send another agent after a given time if the first one didn’t come back

References

[1] Mobile Agents and the Future of the Internet, David Kotz, Robert S. Gray.

[2] Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. By Stan Franklin and Art Graesser, Institute for Intelligent Systems, University of Memphis.

[3] Concepts and architecture of the Mobile Agent Tracy, Braun, Erfurth, Rossak.

[4] An introduction to the Tracy Mobile Agent System, Braun, Ertfurth, Rossak.

[5] Mobile Agent Systems: What is Missing? K. Rothermel, F. Hohl, N. Radouniklis.

[6] Migration of Mobile Agents in Java: Problems, Classification and Solutions. Illmann, Kargl, Weber, Kruger.

[7] Mobile Objects and Mobile Agents: The Future of Distributed Computing? Danny B. Lange.

[8] IEEE Conference and Workshops on Engineering of Computer-Based Systems. http://www.cigital.com/conferences/ecbs02/index.html

[9] List of JDBC drivers: http://industry.java.sun.com/products/jdbc/drivers

[10] Tracy home page: http://tracy.informatik.uni-jena.de

[11] Mobile Agents in Network Management Applications, MSc thesis, Markus Naylor, Napier University, November 2000.

[12] Buchanan WJ, Naylor M, Scott AV, "Enhancing network management using mobile agents", Proceedings Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000). IEEE Comput. Soc. 2000, pp.218-226.

[13] Aglets, http://www.trl.ibm.com/aglets/

[14] JATLite, http://java.stanford.edu/.

Appendix 1: User manual

Requirements

The only external software requirement to run the application is to have several workstations over the same subnetwork, all having a JRE 1.3 or later installed. Mobile agents extend an example file given with the environment named RunningTour, which visits all nodes of the network. This file is in the directory **/examples/agents/traversenet**

Access control

The program includes my login and password for Tracy, but it should expire sometimes in 2002. The root password is "root", and in case my login is expired, the root access should be use to create new accounts. The files containing login and password are: **gtwAgent.java** and **requestAgent.java**, so they may need a few modifications for another account holder.

To access the User Manager screen, the classpath need to be loaded (file **class.bat**) and the following command be entered: **java de.unijena.tracy.usermgmt.Gui**. Alternatively, the file **tracy.user** (in the Tracy root directory) can be replaced by the user's one if a user database has already been configured.

Property files

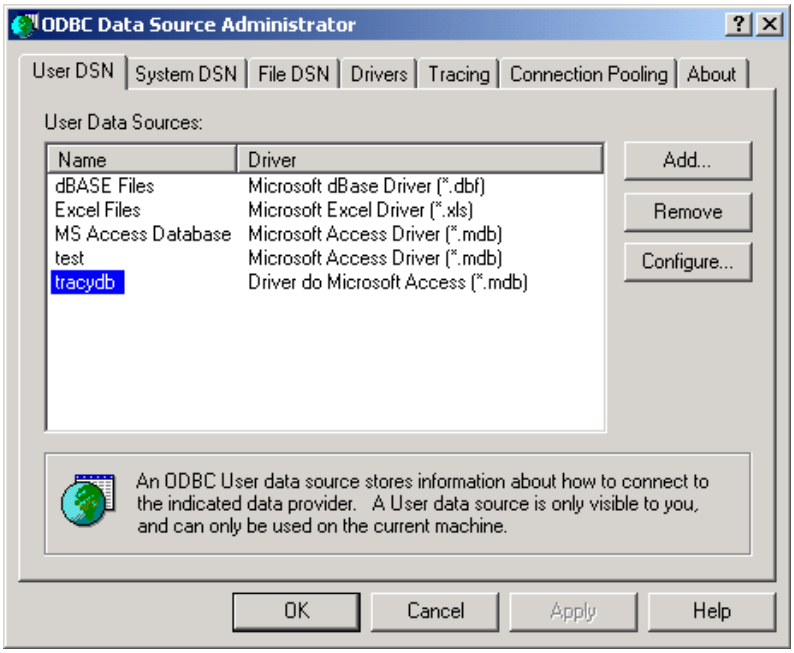
The name of the property file on the server side is **gtwprop.ini**, and is **clientprop.ini** on the client side. These two files need to be configured for the workstation the application runs under. The most important property that will not allow the application to start is the **agentserver.fqdn** property which must be set to the host name of the workstation (which is given by an *nslookup* on the IP of the host). This property must be set with the same string the station is referenced by the current DNS server. If the hostname is wrong or no DNS servers are found, Tracy fails to start.

I developed parts of the project on a standalone station (not connected to the Internet) when I was not yet dealing with agent migration, and I used Simple DNS Plus to trick the reverse DNS resolution mechanism of Tracy. Other properties can be changed by advanced users, but it should not be a requirement for the application to run.

Data source

The database must be referenced on the server side. In windows, an ODBC data source must be added. To add a source, go in the Control Panel, then Administrative tools, then Data Sources (ODBC), and here add the database you wish to access. An MS Access database example is given in the /base directory. All databases over the distributed system must have the same source name (name that you can specify for the request when entering the SQL command to perform over the distributed system) and the same data structure (number of columns and columns name). This

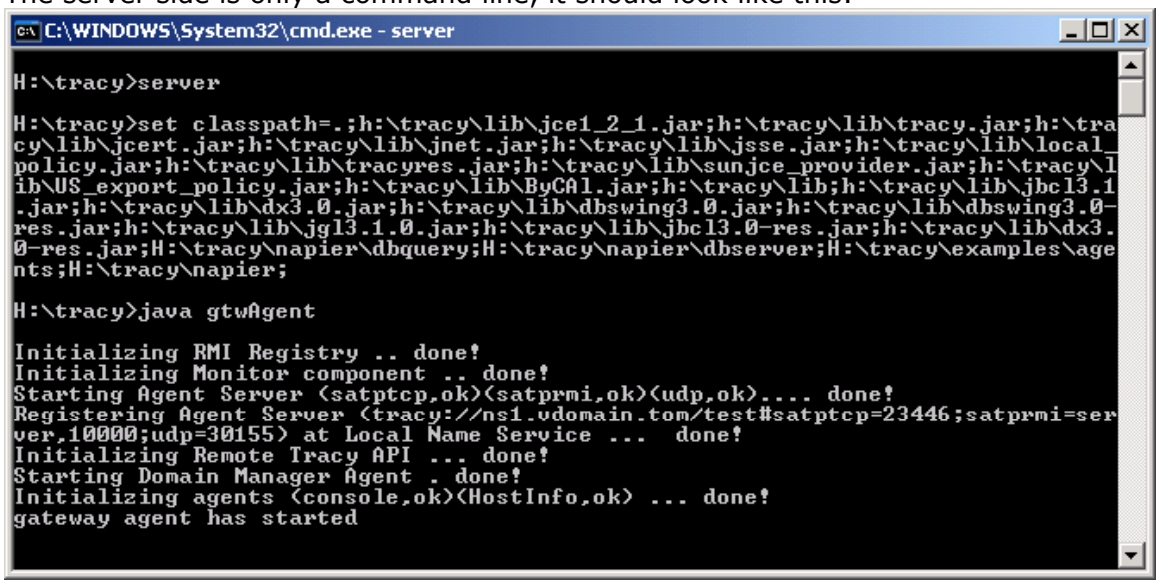
is an absolute condition for the application to work! If databases don't have the same structure, the client side gateway agent will be unable to merge results.



Run the application on server side

A batch file has been created, containing the classpath and command line. This file is **server.bat** in the Tracy root directory.

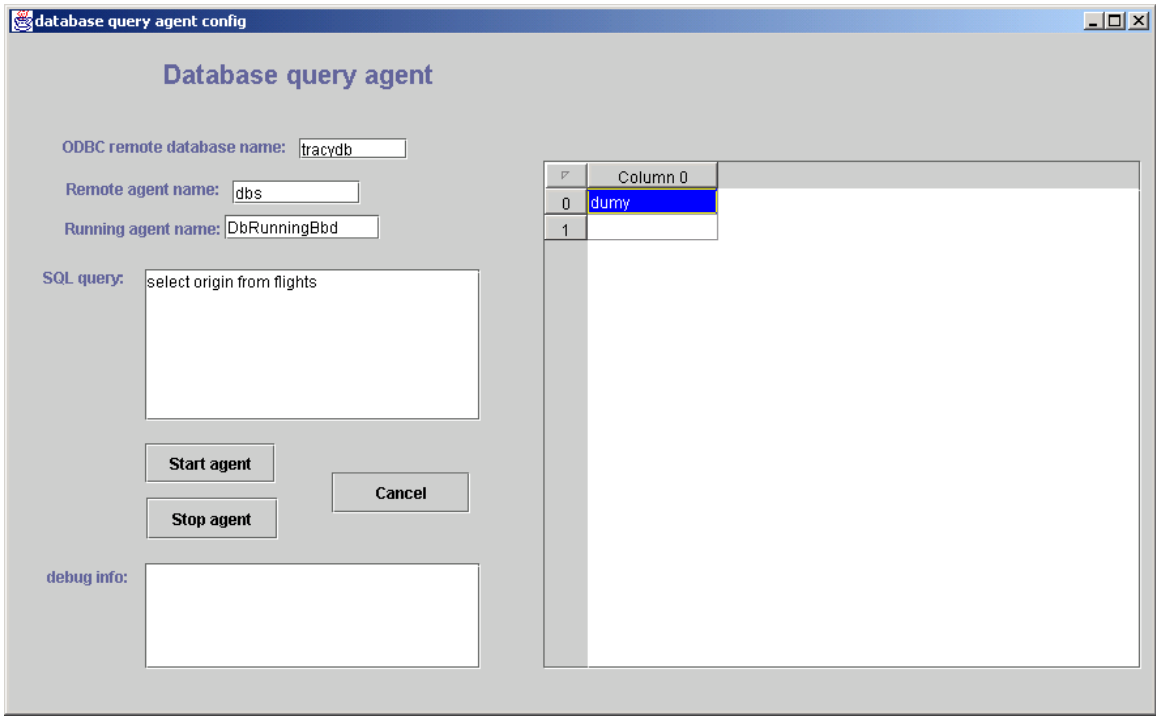
The server side is only a command line, it should look like this:



Run the application on client side

A batch file has been created, containing the classpath and command line. This file is **client.bat** in the Tracy root directory.

The GUI should look like this:



The first field (ODBC remote database name) is the name specified when adding a database source for the targeted database. The next field is the name of the gateway agent server (by default, "dbs", but which can be changed by modifying the file **/gtwAgent.java** which starts the server side gateway agent). The third field to complete contains the name of the mobile agent to use. Two of them have been implemented: **DbRunningBbd** (agent A) and **DbRunningMsg** (agent B). The fourth field is the SQL request to be distributed. Only the button "Start agent" works. It starts the mobile agent specified in the *Running agent name* field.

When the mobile agent comes back, the timing information appears in the left bottom box, and the results in the right array.

Libraries

The /lib directory contains all files needed that are not agents but are part of the application but can be used by other application. This directory contains:

- Tracy libraries
- Sun JSSE and JCE libraries
- Some Borland JBuilder3 libraries
- Blackboard content types defined for this specific application
- A timer (called by agents for bench purposes)

- An object which stores files as byte array.

Note that one of the Borland libraries had to be modified and recompiled to be compatible with the JDK 1.3. The file is **jbcl3.1.jar**, originally named jbcl3.0.jar. Be careful not to use the original Borland file, or use a compatible JDK 1.3 library (which should be the case for new releases of for patched version of JBuilder 3).

The classpath must include all material in the lib directory, as well as path to the class used by the main program. The complete classpath is set in the two batch files, client.bat and server.bat. This classpath is set for the root directory of the application to be h:\tracy. If the application is installed somewhere else, the drive letters must be changed (and if necessary the directory names).

Content of CD

This section gives the content of each directory

- Root directory

Property files. Policies files. User management file. Batch files to run the application. Note that the root directory needs disk write access. The application cannot be executed from CD. Data extracted from databases is saved to files both on server and client. These files are stored in the root directory (*.txt and *.schema), and they are not automatically deleted (but they can be deleted, the good operation of the application does not depend on it).

- /base

Microsoft Access database sample (for use within the application).

- /bin

- /doc

Java doc of Tracy (given with the environment).

- /examples

Examples given with the environment.

- /napier

Developed agents code.

- /lib

Tracy libraries. Sun libraries. Borland libraries. Classes of general purpose developed during the project.

- /ref

Useful publications on Tracy and on mobile agents.

Appendix 2: Project diary and Gantt chart

Week 1-2:

- Contact Bill Buchanan, get information on the project and on what is expected.

Week 3-5

- Go to the Tracy web site, download the environment and the papers.
- Reading of the documentation.
- Initial presentation on week 5.

Week 6

- Meet Aleem, a master student that study on Tracy and who present me the environment, how to start an agent server, the importance of classpath and jars.
- Download of Sun libraries
- Configuration of the environment on the H:/ drive
- Configuration of the user access

Week 7-9

- Configuration of property files
- Start agent servers
- Start agents given as samples with the environment.
- Test of the samples

Week 10-11:

- Meeting with Bill to talk about what I think about for the application and what he wishes to see achieved with the application.
- Finalize the design in accordance with the latest investigation results on Tracy.

Week 12 to 14:

- Exams period, quite busy, no time to start development, but thinking about how to implement the design.

Week 15:

- Development of the database connection, and "server side" of the application with java.sql classes
- Development of the blackboard content type for database resultsets (Java.sql)

Week 16:

- Start the development of the client side of the application

- Realize that it is quite difficult to perform operation on resultset with java.sql libraries, decide to migrate from java.sql to Borland jbuilder3 component for database components.

Week 17:
- Re-implementation of server side gateway agent with Borland components.
- Start the development of the first mobile agent (agent A).
- Development of the graphical interface, client side, with Borland component.
- Blackboard content type upgraded to handle jbuilder3 components.

Week 18
- Project review: the application is not fully functional, both client and server side are finished but mobile agents are not ready yet.

Week 19-21
- Go on with the development of Agent A
- Start development of agent B.
- Problem: Borland component's content does not go through network, try to find why.

Week 22
- Group project review with Bill
- try to find a database resultset container that serialize its content

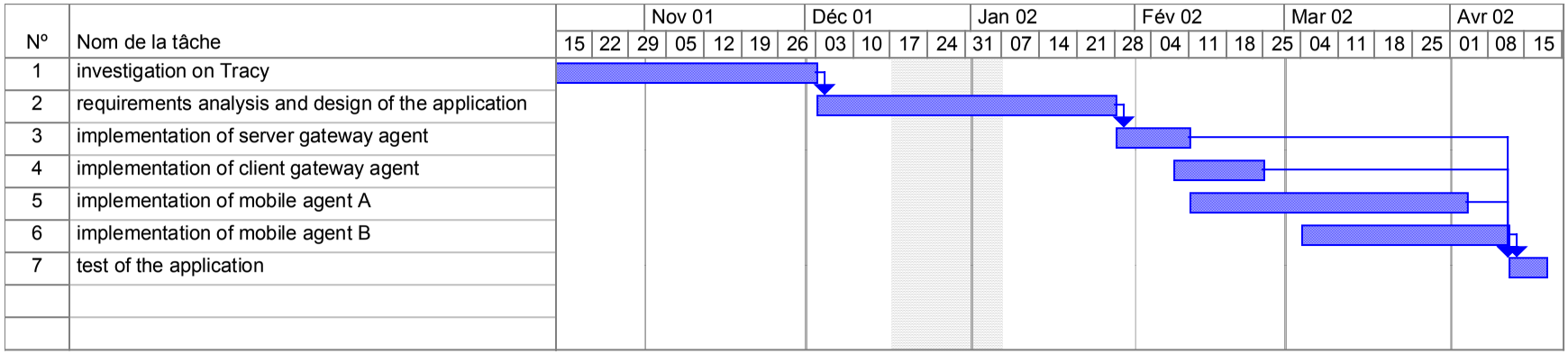
Week 23:
- Development of the conversion from datasets to binary files to byte array which are serializable. Serious modification to all agents to do conversions.
- Upgrade of blackboard content type

Week 24:
- Agent A finally works.
- Contact Peter Braun from the developer's team of Tracy to know how to set Tracy to run over multiple subnets.

Week 25:
- Agent B works
- Jan Eismann from the developer's team of Tracy sends me a new version of Tracy not release that is supposed to work over multiple subnet, but tests are unsuccessful, so final tests will only be performed over one subnet.

Week 26:
- Test of the application and of mobile agents
- Instantiation of Tracy servers and agent in class files to get rid of the Tracy GUI and have an application which is easy to run.

Gantt chart



N°	Nom de la tâche
1	investigation on Tracy
2	requirements analysis and design of the application
3	implementation of server gateway agent
4	implementation of client gateway agent
5	implementation of mobile agent A
6	implementation of mobile agent B
7	test of the application

Appendix 3: Program listings

The listed programs are the following ones:

- Gateway agents:
  - DbMiner.java (client side).
  - DBgwBbd.java (sever side).
- Mobile agents:
  - DbRunningBbd.java (Agent A).
  - DbRunningMsg.java (Agent B).
- DBFile.java : defines the Object DbFile, which is a file stored as a byte array (which is serializable).
- Blackboard content types:
  - BBDBFileContent.java
  - BBFileVectContent.java
- Timer.java (used by mobile agents to time the trip).
- gtwAgent.java : the application, server side. Instantiate the server according to the specified property file and start the gateway agent.
- requestAgent.java : the application, client side. Instantiate the server according to the specified property file and start the gateway agent.