Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche

researchportal.unamur.be

University of Namur

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Intelligent agents using belief revision

De Coster, Arnaud

Award date:
2000

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 03. Jul. 2025

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique.
Année académique 1999-2000

Intelligent Agents using Belief Revision

Arnaud De Coster

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique.

Acknowledgements

I would like to thank Pierre-Yves Schobbens and Mary-Williams for letting me discover the magic and wonderful world of intelligent agents. Thanks also to my favourite english native secretary, Régis Francart. And thank you Luigia for being so patient !

Résumé

Le concept d'agent intelligent émergea de l'intelligence artificielle à la fin des années septantes. Les recherches étaient alors principalement orientées vers les agents utilisant des representations symboliques de leurs environnements. Ce courant scientifique, toujours actif, étudie aussi les aspects sociaux des systèmes d'agents. Les années nonantes virent apparaître un nouveau courant dans lequel le concept d'agent est défini plus largement. Cette nouvelle vision fût, entre autres, récupérée par le génie logiciel.

Ce travail vise à présenter le monde des agents intelligents et à déterminer la mesure dans laquelle la 'Belief Revision', une discipline rattachée à la vision originelle du concept d'agent, peut trouver sa place dans le paradigme de programmation orientée agents du génie logiciel. Pour réaliser ces objectifs, ce travail comprend une présentation du monde des agents intelligents, une introduction à la 'Belief Revision' et la description d'une implementation d'un système d'agents capable de 'Belief Revision'.

Mots clefs : intelligence artificielle, agents intelligents, Belief Revision, génie logiciel.

Abstract

The intelligent agent concept emerged from the artificial intelligence area at the end of the seventies. Researches were then mainly oriented towards deliberative agents with symbolic internal models. This scientific stream, still active, also studies social aspects of agents systems. In the early nineties, appears a new stream in which the concept of agent is more broadly defined. This new vision has been used by software engineers.

This work's aim is to overview the intelligent agents world and to see in which measure, Belief Revision, a topic linked to the original agent conception, could find its place in the software engineering agents oriented programming paradigm. In order to reach this goal, this work is composed of an intelligent agents world overview, a Belief Revision presentation and the description of an implementation of an agents system using Belief Revision.

Keywords: artificial intelligence, intelligent agents, belief revision, software engineering.

Contents

Introduction

The intelligent agent concept finds its roots in the artificial intelligence area, in the late seventies. Researches were then oriented towards deliberative agents with symbolic internal models. This scientific stream, still active, studies macro issues such as negotiation, co-ordination and communication in agents systems.
In the early nineties emerged a new stream of agent researches more oriented towards practical issues. These researchers enlarged the concept of agent definition such that everybody began to call everything an agent. This broad view lead to a spreading of the intelligent agent concept to other computer science domains of research. User interfaces designing and software engineering may be the disciplines which took the most out of this new vision.

This work's aim is to overview the world of software agents and to see in which measure, belief revision, a 'more classical' artificial intelligence topic, could find its place in a system of agents. This system of agents being used for software engineering purposes. Roughly said, the question underlying this work could be reformulated as 'Where the old agent research stream can find its place in the software engineering use of agents?'

Firstly, an overview of the world of intelligent agent is achieved. The concept of intelligent agents is encompassed through the presentation of different definitions. Another way of approaching the concept is to establish a typology of existing so-called agents. Situating artificial intelligence techniques in the agent world is also interesting. Tools in order to facilitate the implementation of agents exist. They are then introduced. It is also the opportunity to compare the agent oriented programming paradigm with the object oriented programming paradigm.
Secondly, an overview of Belief Revision is presented through an axiomatic approach and a constructive approach.
Thirdly, an implementation of a multi agents system using Belief Revision is presented.
A conclusion that presents the results of the analysis ends this work.

First Part: Intelligent agents

1. Introduction

"The buzzword agent has been used recently to describe everything from a word processor's help system to mobile code that can roam networks to do our bidding. The metaphor has become so pervasive that we're waiting for some enterprising company to advertise its computer power switches as empowerment agents."

Peter Wayner in [Wayner, Joch 1995]

The situation still isn't clear five years later. Development in agent research has spread ever since what made the word 'agent' fuzzier and it is still not clear what constitutes an agent in the essence. And when it comes to a matter of 'intelligent' agent, opinions differ even more.

The fact that the concept of 'agent' is used by a large number of persons, all working on a lot of different areas, can in a way show the importance of this concept in plenty of different scientific areas. (From computer science to engineering sciences passing by sociology and psychology)

In computer sciences, the concept is used in a variety of areas such as agent oriented programming in software engineering (as an enlargement of the object oriented programming) [Jennings, Wooldridge 2000], in intelligent assistant agents systems (such as Microsoft office assistants), agent used to reduce the information overload [Maes 1994], or Mobile software agents as part of portable computers. (Laptops, Personal Digital Assistants) [Chess 95]

This enumeration is far from complete. But a further diversification of the term would lead to unclear concepts of what constitutes an agent and what are its characteristics.

A possible motivation to use agents emerged from the paradigm of the direct manipulation, which is prevailing at the present time in information technology. The user is directly manipulating the objects with which he operates.

By the never-ending increase of "information overload" and "functions overload", of which computer users are increasingly dependent, the approach of the direct manipulation is becoming an inadequate way of communication between computers and humans. The fact that more and more people want to, or have to deal with computers, and don't have a global understanding of the information and communication basic techniques aggravates this discrepancy between the user interface and their requirements.

Using agents as user interface is leading to a paradigm change for indirect manipulation. The user doesn't operate any longer directly with objects which he would like to modify, but he assigns agents (or assistants) to this function. These have an elaborate user model and knowledge over the field, in which they have to act, so that they can execute independently the job. They may also possess means and tools to procure themselves this knowledge. This paradigm change is only at its premises.

2. Different approaches and definitions of intelligent agents

There are almost as many definitions of the term " agent " than there are scientists, developers and journalists, who deal with this topic. Hereafter are introduced some definitions of the term " agent ". In each case another aspect of the term and different characteristics that constitute an agent are approached.
This will give the opportunity to discover where agents differ from normal programs.

The Britannica encyclopaedia [Britannica 2000] defines the concept of « agency » as follow:

« In law, the relationship that exists when one person or party (the principal) engages another (the agent) to act for him--e.g., to do his work, to sell his goods, to manage his business. »

Common examples of agent that respects this definition are commercial agents and insurance agents. It is obvious that this definition, although it already outlines the nature of an " agent " is not suitable in the core to define the term in the context of this work.

a) Delegation and authority

"An agent is a piece of software that does something for you"

[Dragan 1997]

"Agents can be considered personal software assistants with authority delegated from their users."

[Cheong 1996]

It is obvious that these definitions aren't sufficient either to define the term 'agent'. The prerequisite that an agent has to do something for the user is also true for every tool. In his article, the author of the first definition refines it by saying that agents are expected to automate repetitive tasks.
The second definition requires a type of authority or power, which is transferred to an agent from the user.

What exactly constitutes this power and what the agent is able to do remains unclear.

b) Communication

" an agent is a computer program whose purpose is to help a user perform some task (or set of tasks). To do this, it contains persistent state and can communicate with its owner, other agents and the environment in general."

[Lingnau, Drobnik, Dömel 1995]

This definition adds an important aspect to the first definitions: In order to achieve its goal, an agent must communicate with its owner, with other agents and its environment.

Communication with the owner leads to questions about the design of an adequate user interface. In which manner is the owner or the orderer of an agent going to indicate the best way possible his needs ? Is this to be done through a special programming language, or is it the agent that has to have mechanisms of natural speech recognition?. The agent must be instructed or he can become active on its part because it posses means of deducing its task.

The question about the best user interface is almost as old as computers themselves. Communication with other agents and the environment raises however a set of new questions. In which language do agents exchange information? How is it guaranteed that all parties taking part use the same means of communication ?

c) Perception ability

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."

[Russell, Norvig, 1995]

"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realise a set of goals or tasks for which they are designed."

[Maes 1995]

Pattie Maes, director of the software Agents Group at the MIT Media lab, defines agents like entities able to act in complex and dynamic environments. Additionally agents must be autonomous, that means that they execute the majority of their functions without direct human intervention and that they can check their own status. For example they have to start and stop themselves autonomously. Agents must also be autonomous in the selection of the means with which they are going to execute their function.

d) A set of characteristics

In [Woolridge, Jennings 1995] two distinctive developments of the term " agent " are presented according to a " weak " and a " stronger " meaning.

The weak meaning presupposes the following characteristics of an agent:

- *autonomy*: Agents operate without being directly influenced by any humans or other control systems, they can check their own status and have some kind of control over their actions ;

-*social ability*: Agents communicate with other agents (and perhaps humans) via some kind of agent communication language [Genesereth, Ketchpel 1994] ;

- *reactivity*: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it. This may lead to the fact that an agent spends most of its time in a kind of sleep state from which it will awake if certain changes in its environment (like the arrival of new e-mail) give rise to it;

-*proactivity*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative;

In computer science this meaning of an agent is regarded as independent, parallel running entity, which communicates its own encapsulated status with other entities, it is considered as a natural extension of the object programming orientation.

In the field of artificial intelligence, a " stronger " meaning is usually linked to the term ‘ agent ‘. To the characteristics explained above, is added that an agent has to carry more « human » behaviour. It has to use mentalistic notions such as knowledge, conviction, intention and responsibility or can even express feelings. [Bates 1994].

Another thing to give software agents more a strongly anthropomorphic characteristic(a basic condition for an agent already placed by [Foner 1993]) is to represent them with an animated persons or faces [Maes 1994].

Other characteristics of agents, linked to a « stronger » meaning, which are stated in [Woolridge, Jennings 1995], are :

-mobility: the ability of an agent to move around an electronic network;

-benevolence: is the assumption that agents do not have conflicting goals, and that every agent will therefore always try to do what it is asked to do;

-rationality: is the assumption that an agent will act in order to achieve its goals and will not act in such a way as to prevent its goals being achieved.

e) Intelligence

Intelligence is difficult to define and therefore it is also difficult to attribute it to an agent. The estimation of an artificial system as " intelligent" is based on the impression that an intelligent observer or user could have in front of the system. Intelligence is to be seen therefore rather as consequence from the behaviour of a system due to the combination of different characteristics and not as a characteristic that one could implement in the system [Petrie 1996].

Intelligence continues to lose importance as a criteria for the distinction between agents and conventional software. A large, complex expert system may perhaps supply intelligent suggestions on the solution for certain problems, but is however not automatically an agent. Intelligence is not a sufficient criteria for agents.

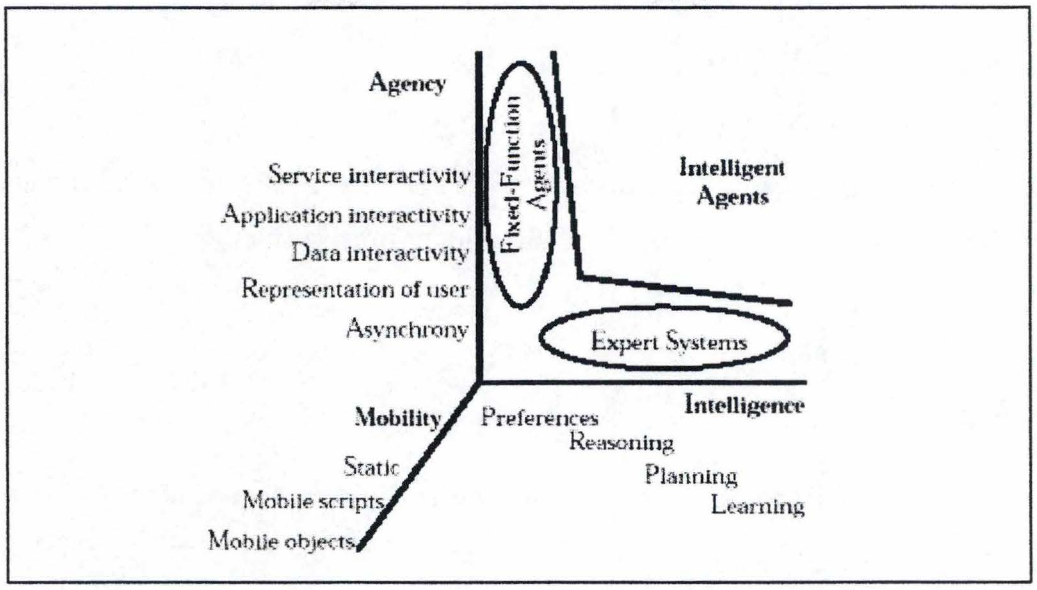

Figure 1.1:From [Gilbert et al.95]

Nevertheless, as shown in figure 1.1, [Gilbert et al.95] uses intelligence as an agent characteristic that can even be evaluated. In this paper , different sample applications are placed on a cube designed by the « mobility », « agency » and « intelligence » axles.
A boundary line separates the area of the intelligent agents from other programs.

3. Classifications of intelligent agents

It is impossible to create a general classification for intelligent agents due to the complexity of the term. The only way to approach it is in a multidimensional system.

a) Classification levels

- a first dimension is the partitioning according to the degree of independence of an agent.

At the lower end of the independence rank are scripts, which must explicitly be called by the user. Asynchronously started programs indicate a higher degree of independence. Asynchronously started can mean either in regular time intervals or either explicitly by certain events. At the upper end of the scale of independence, can be found programs that are nearly constantly active and that become really independently active by observation of their environment. (if an event or a change of status on which they want to react occurs.)

- a further dimension of classification of agents is how far is the user represented and under which degree of autonomy can the agent act on his behalf.

At the lower rank, we can find agents that explicitly include the user within each step of their activity. Conversely, some agents don't need any user interaction to perform their task. Between these two extremes, agents with various degrees of autonomy exist.
For example an agent that looks for documents independently on the base of a user profile doesn't request them automatically (this leading to useless connections costs), but this decision is left to the user. Another example : an agent that can delete and answer automatically to certain inquiries, can detail automatically the checked emails, but can also delete some of them inadvertently. The confidence that the user gives to an agent, is to be seen also on other side of the same medal. The user is only ready to grant to an agent as much autonomy as confidence he puts into it.

- Agents, in order to correctly achieve a task for the user, must know what they are supposed to complete.

At the lower end of the scale, the user model is called via procedural approach. The user ,via a direct user interface, specifies at a low level the exact activities that have to be completed for him.
There is a more elaborate approach that lets the user create a profile. For example using rules, from which the system draws conclusions.
More highly developed approaches allow the system to create and understand a user model, and from there what the user wants to achieve.
Further approaches allow an independent planning of actions, in order to achieve a certain target, by learning both the behaviour of users and resources from the environment on which he acts. Such a system can discover independently new means and apply these, in order to be able to better meet the user desires.

- At the lower end of the organisation scale (according to the ability of an agent to communicate) are agents, which don't communicate and have no interaction with other agents but communicate exclusively with the user and/or resources. There are large quantities of programs, which fall into this category and are called " Believable Agents".
There are systems that can communicate to one or more resources or applications and with the user. " Interfaces Agents" as a new form of user interface or assistance systems falls into this category. The possibility of an additional communication with other agents is a fundamental characteristic of multi agent systems.

- In networked environments agents can execute their function locally on a node or move by the network. This characteristic is defined as mobility. An agent can achieve its task either statically or either mobilely.
Mobile Agent systems are characterised by the fact the agents move by the network (that means with their program code and all their data's) and go on different systems in order to achieve their tasks.
Static agents are either local or either remote.
Static and local: The agent resides on the user side of the network, i.e. on the local computer (Client). He may assume its environment on local and distant systems via sensors. However it does not move by the network. Multi agents systems that operate locally and stationarily are also conceivable.

Static and remote: Agents reside in a fixed place outside the local computer system (on servers). The user communicates over the network with them and they complete their tasks for him from their positions.

- The architecture of the agents (deliberative, reactive, hybrid, heterogeneous (see further)) can also be consulted as classification level.

While independence, autonomy, user model and ability to communicate surely represent key aspects of an intelligent agent, characteristics such as mobility or architecture are to be seen as neutral characteristics. As described earlier " intelligence " was not to be regarded as a discretely modelled characteristic that can be measured, but as the user perception of the combination of the key characteristics.

The importance of the individual characteristics must be differently weighted in dependency of respective application. For example for an interface agent, an elaborate user model can be more important than the ability to communicate with other agents

b) An example of classification

Next, some types of agents are presented like it arises frequently in the literature. The aforementioned types correspond to a more strongly functional classification of agents.

(1) Believable agents

According to their appellation of « Believable Agents », these have to act as naturally as possible. Eliza, a simple program, which imitated human speech habits, is one of the earliest examples of a Believable agent.

Driving strength for this type of agents today is the entertainment industry. In addition, applications in the business area are conceivable, for example in the field of virtual offices or virtual business.

In [Bates 1994] the importance of the feelings modelling is put forward for the reliability of an agent character. In the context of the Oz-project of the Carnegie Mellon university [Oz

2000] animated figures by computers, that are called Woggles, can act in artificial worlds independently.

Another examples are the Extempo characters. This company [Extempo 2000] develops anthropomorphic agents, who can communicate with humans.

Extempo sees as a possible application of their agents the on-line presentations of enterprises in the WWW.

A high degree at independence and autonomy are important aspects of this type of agent. The architecture of the agent system is not important and the required ability to communicate depends on the respective task. The presented examples are not mobile. An elaborate user model is however the key factor for Believable Agents.

(2) Collaborative agents

Agents, who operate together on the achievement of a task, result directly from the motives for research in the area of the distributed artificial Intelligence: They are together to solve problems, which would be too complex for a single agent or which requires distributed authority.

The " Pleiades project " [Pleiades 1996] developed at the School of computer Science of the Carnegie Mellon University is to be stated as example of the architecture of such systems. In this system, "task-specific" and "information-specific" agents cooperate in order to create and manage a visitor's schedule to CMU.

As shown in figure 1.2, two abstraction levels of agents interact together. One, the " task assistants " performs tasks for the user, for example, they agree on dates or collect interesting news articles. These agents co-operate with other agents on the same abstraction level, in order to negotiate, for example, dates. Agents of a second abstraction level, the "information assistants", supply the information, which they needs for their work. These work out different sources of information and can cooperate with other agents of the same abstraction level.

Task Assistants that were developed in the context of the project Pleiades, are a calendar agent ("calendar apprentice"), who can not only co-ordinate dates with other calendar agents, but can also learn the special preferred date from its user. They automatically submit suggestions of date [Mitchel et al. 1994], a newsreader (" NewsWeeder ") and a Webbrowser ("WebWatcher"). They learn the user behaviour.

Can also be found a "Person finder", that analyses different sources of information, and in order to find the addresses of persons communicate with other agents, as well as an agent, who organises attendance programs ("Visitor host").

The Visitor host determines first the name, interests, current projects as well as the organisation of the visitor. Afterwards it seeks persons that he would probably like to meet, and agrees on a date for these meetings thanks to the calendar agents of these persons [Nwana 1996].

Collaborative Agents are independent and act autonomously. They can structure their user model in different ways. Communication takes place between agents, users and applications, they don't have to be mobile.

(3) Interface Agents

The best metaphor for interface agents is a personal assistant, which co-operates with the user in its working environment. This working environment can be a simple program or a complete operating system. The agent observes the actions of the user and makes suggestions for him on how he could perform the tasks better, or it executes these tasks automatically and independently. Apart from the direct observation of the user behaviour, interfaces agents can also assume tasks. They learn to be able by positive or negative feedback of the user from the actions they suggested and possibly also ask different agents for advice, if they don't know a situation.

From the use of interface agents results, according to [Maes 1994], a new type of use of computers interface, in which direct manipulation, delegation and co-operation do not separate the relation between humans and machine any longer.

Maes put into evidence two central problems on interface agents:

The first problem is the question about the ability of the agent: How does he gain the knowledge in order to be able to help the user?

The second problem consists of the question of the trust that the user would put in the agent, does it gives him pieces of advice or performs tasks for him.

The approach of an independently learning architecture represents the solution of the problem for Maes. The agent gains competence and trust from the user by observing him and by removing only gradually ever more tasks from him. This slow taking over of tasks occurs for the user transparently; that means, he can always get details on what the agent does, and can easily ask it not to do this anymore in the future.

[Pearson 1996] distinguished different functions from interface agents: they can be advisory or training active, help the user during the execution of tasks, running as deputies, or providing a search and filtering information support.

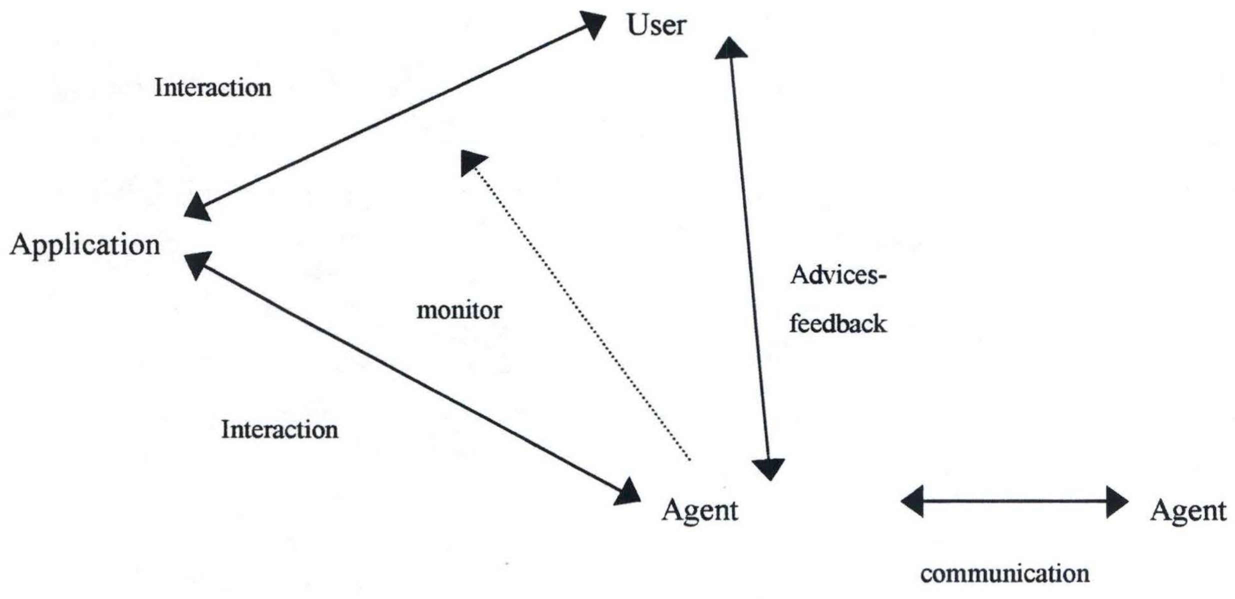

Figure 1.3: How Interface Agents Work from [Maes, 1994]

The assistants that Microsoft places in its Office software, were the first examples of the use of interface agents in a common commercial product. Even if the assistants do not act so far yet very intelligently, the development continue to advance.

The technical facilities guarantying the success of the assistants seem to be existing, however conceptions of a computer program observing the user leads, like Maes foresaw, to associations of a monitoring by the machine.

Whether the assistants gain the confidence of theirs human users remains an interesting question.

The substantial aspects of interface agents are the user model, which must be structured in the course of the time, and their independence and autonomy that they must also win in the course of the time. Communication with other agents can be important, but is not a key factor. Interface agents are usually single agents.

(4) Information Agents

A primary function of information Agents is collecting, adapting and administrating information from different sources. In a way, this function is also a major function of agents of all kinds and can thus also apply to the Pleiades for example. The strongly growing importance of the WWW as source of information induces agents specialised in information retrieval called information Agents.

The independence and autonomy of information Agents is less important than an applicable user model. Information Agents must be able to primarily communicate with the information suppliers. They can operate locally, remotely or mobily.

(5) Mobile agents

A last class of agent, that has to be presented, is mobile agents, which roam in a computer network. They communicate with servers and possibly also with different agents and complete tasks for the user in this way.
According to [Nwana 1996], motivations for the use of mobile agents are:

- Smaller communication costs: Instead of transferring a large quantity of information from a distant server, in order to compress it locally or to filter the relevant sections, a mobile agent can process the data on behalf of the user on the server side and then return only with the relevant information. In addition the user must not constantly be connected to the network, while his agent completes functions for him.

- Limited local resources: On the side of the user, resources cannot be sufficient for the execution and storage of large data base inquiries. A small mobile agent can however, with the necessary provided information, execute these functions on a server and then return with the results to the local computer.

- A flexible approach for distributed systems; new agents as service providers and information collectors, can easily be added, without harming the system.

The fundamental difference between the well-known Client server paradigm of distributed applications and the mobile agent approach is shown in figure 1.3.

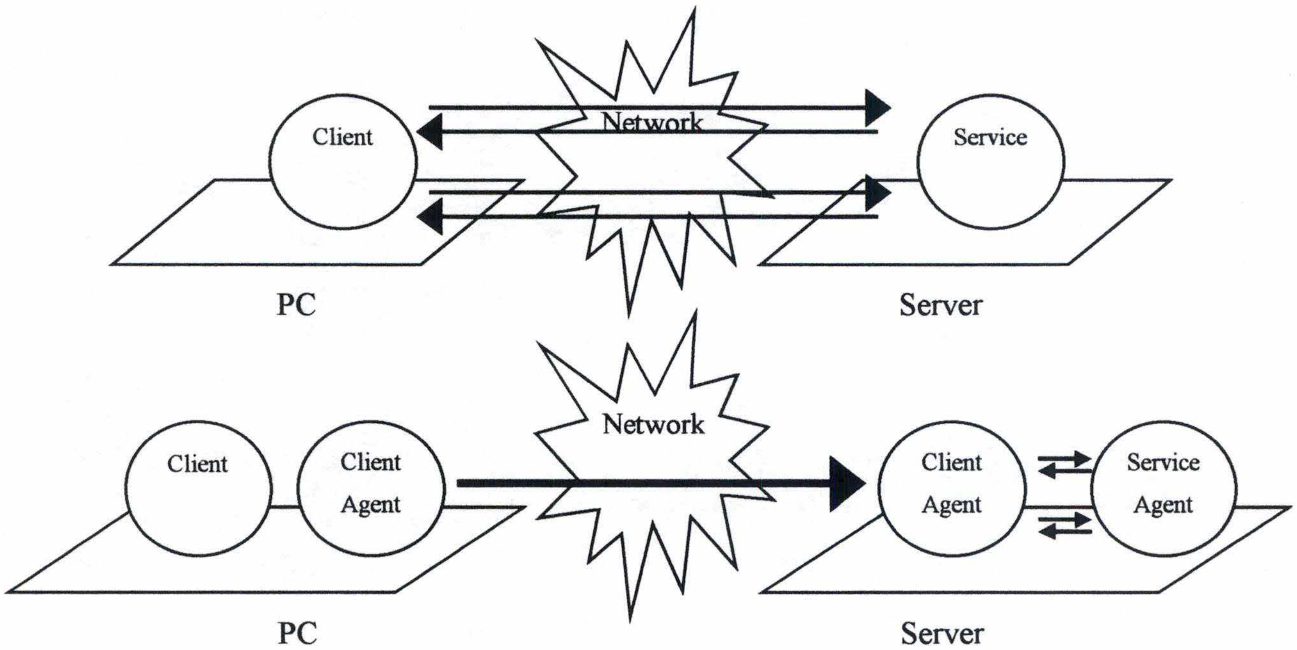

Figure 1.3: Client server vs. mobile agents [White 1996]

Mobile agents are objects with behaviour, a status and a place [Sommers 1997].

The task of a mobile agent defines its behaviour. The behaviour for a mobile agent is its movements from place to place. Additionally messages between agents are a substantial constituent of the behaviour of mobile agents.

The status of a mobile agent, thus its program code and its data, must be totally enclosed (protected and independent from its environment) so that they can be sent in a simple way to a new place, in order to get there and resume again their execution. The substantial events in the life cycle of a mobile agent, by which its status is modified, are its creation, its end (or disposal), the departure to a new place (or dispatch), the arrival at a new place and the communication with other agents.

Places are the substantial concept of mobile agents. A place is defined as an environment in which an agent can arrive and execute. Several logical places, for example several providers of information, can be linked to a physical place, for example a network server. Places have a similar function as the virtual machine of the programming language Java: They make all services that the agent needs for its execution available. They co-ordinate the flow of several agents and their communication among themselves and with the system and protect the system against non-authorised accesses.

A further substantial aspect of mobile agents is security. The security of the environment is to be ensured by the concept of places. However the protection of the agents must also be ensured in some way against the system environment that could have an interest in getting or changing the information of the agents [Green and al 1997].

As an example of the application of mobile agents the Stormcast project at the University of Tromso, Norway can be mentioned [Johansen 1997b]. In the context of this project, which has been running for some years, weather information (satellite photographs) are collected and made available. At the base of TACOMA [Johansen 1997a] the system is now developed so that the users produce an agent over their WWW Browser, and send these to the Stormcast server. There it selects the only necessary information from the quantity of data and finally returns with these to the user. The quantity of data that can be transmitted is to be reduced by the use of agents.

Mobility and communication are the substantial characteristics, which constitute a mobile agent. Other characteristics, like independence and autonomy play however an important role with this type of agents.

4. Software agents and Artificial Intelligence

The place of the software agent in the area of the research of artificial intelligence is overviewed in [Nwana 1996].

Software agent as multi-agents systems (MAS) together with the areas of distributed problem solving and parallel artificial intelligence (" parallel AI ") form the area of the distributed artificial intelligence (DAI).

According to [Green et al 97], the motivations for the increasing interest in DAI and MAS research includes their ability:

- to solve problems that are too large for a centralised single agent to do due to resource limitations or the sheer risk of having one centralised system;
- to allow for the interconnecting and interoperation of multiple existing legacy systems, e.g., expert systems, decision support systems, etc.;
- to provide solutions to inherently distributed problems;
- to provide solutions which draw from distributed information sources;
- to provide solutions where the expertise is distributed, e.g., in health care provisioning;
- to enhance speed (if communication is kept minimal), reliability (capability to recover from the failure of individual components, with graceful degradation in performance), extensibility (capability to alter the number of processors applied to a problem), the ability to tolerate uncertain data and knowledge;
- to offer conceptual clarity and simplicity of design.

Carl Hewitt's Concurrent Actor Model (according to [Nwana 1996]) is considered as the starting point of the research in software agents. He suggests an independent, interactive and parallel running object, which he called "Actor". It has a state protected from the environment and can communicate with other objects.

An "Actor" is a computational agent which has a mail address and a behaviour. Actors communicate by message-passing and carry out their actions concurrently.

DAI is now concerned with deliberative agents who use a symbolic internal model for the knowledge representation. Within this area, aspect of macro issues such as interaction and communication between agents or strategies for co-ordination, co-operation and conflict resolution are treated.

Parallel to research and development in theory, more practical research in agents architecture and language have been achieved. This domain of research is overviewed in [Woolridge, Jennings 1995] and [Nwana, Woolridge 1997] and is presented later in this work.

Since 1990 the research in the area of the agents has expanded rapidly. The concept, which before was operated under theoretical criteria as a rather abstract research project, spread.

Starting from 1993 the rapid spreading of the Internet and the WWW led, as a global information system, to the development of a new type of "agent", who led again to an extension of the term "agent".

a) Co-ordination in multi agents systems

The following short overview of co-ordination mechanisms comes out of [Nwana, Lee and Jennings 1996].

The central problem of multi agent systems is the co-ordination of individual agents. The work of the single agents must be co-ordinated in order to achieve a global target thanks to the different abilities and desires of the individual agents and in order to resolve the dependencies between the agents. Without co-ordination, systems with many autonomously operating single agents lead to chaos.

Coordination encompasses the allocation and distribution of tasks between different agents, the coordination and cooperation of the agents among themselves and the conflict resolution or negotiation strategies; it can take place in different types: given structures, contract networks, multi-agent planning:

- The easiest possibility consists of a given fixed structure ("organisational structuring") of the system, in which the individual roles and functions are fixed in the system. For example the agents community can be divided into "master agents" and "slave agents". The master agents delegate functions to the slave agents. Another type of structure can be a so-called "Blackboard" that is administered by master agents (that schedule the agents' reads/writes to/from the blackboard) and is used as central start place for the distribution of tasks in the system. A weak point of this approach is the danger of bottlenecks that can appear because of the lack of central control instances.

- In contract networks ("contract nets"), similarly to the approach of the fixed structured systems, "manager agents" divide the task into sub-tasks and look up for " contractor agents " which can execute these tasks. The difference is situated in the fact that contractor agents can also recursively become manager agents that can further divide the task and distribute the subtasks. Another difference is that the structure of the system is not fixed. Several contractor agents can compete to achieve a task, so that the manager agent can select the most favourable one to achieve a task. The approach of the contract networks offers many advantages in comparison to the fixed structure: it doesn't lead to bottlenecks since several agents have to compete for a task.. However, it doesn't treat the conflicts between individual agents and assumes that all agents cooperate. This approach is best used when the application task has a well-defined hierarchical nature, when the problem has a coarse-grained decomposition and when there is minimal coupling among subtasks.

- multi-agent planning means that each agent plans a methodology for the execution of the central task. Then they are either submitted to a central instance ("centralised multi-agent planning"), in order to eliminate inconsistencies and conflicts or submitted to the agents community so that the adjustment is made by each agent with all the others ("distributed multi-agent planning"). This approach presupposes a larger quantity of information and agent communication. However the bottleneck problem occurs again with a central instance that checks the plans.

If agents, which are not working for a common interest, have to coordinate themselves, antagonisms will arise. This can be solved by negotiation methods.

Negotiation is understood as:

"the communication process of a group of agents in order to reach a mutually accepted agreement on some matter."

Bussman and Muller according to [Nwana, Lee, Jennings 1997]

According to the "game theory based negotiation" approach, all possible outputs of the negotiation process are evaluated via an utility function based on the different requirements of the different negotiators involved. All the negotiators must admit this evaluation. Negotiators try to draw the largest advantage over an iterative negotiation protocol. Cases where agents, who lie or withhold information, in order to achieve a larger advantage, represent possible extensions of this approach. The fact that all agents that take part in the negotiation must know all the evaluations, is the main point of criticism of this approach. Beyond that, the number of evaluations can lead to a problem for negotiations involving many agents and outcomes.

Beside this approach a lot of further considerations for automatic negotiations still remain. According to [Nwana, Lee, Jennings 1997] independent third mediators can be placed between the parties in order to submit negotiation suggestions. A practical approach is [Chavez et al. 1997] Kasbah, a " market place for agents ", on which agents deal with products. The most important parameter, by which the negotiation is determined, is the required price for a product, which, on the basis of different strategies falls up to a certain minimum. If an agent is ready to pay the required price, the deal, after an acknowledgement of the user, can take place. It is thus a kind of auction. Auctions are a very formalised form of negotiation about a certain product attribute, usually the price, and can be automated relatively easily.

b) The user model

The construction of a user model, thus information about the interests, targets and general preferences of a user, also plays a central role for intelligent agents. It must learn what are the desires of the users.

Machine learning traditional techniques are, according to [Green et al 97], the symbolic and subsymbolic classification.

- In the symbolic classification all possible status of the system are divided into classes on the basis of a finite number of attributes, these forming a decision tree. Then a status (a leaf of the tree) of the system can be determined by matching the nodes of the decision tree with the observed attributes.

- The subsymbolic classification uses neural networks and detects samples of attributes and not the attributes themselves. 'Supervised neural networks' are trained before on the basis of well-known categories, where false conclusions are eliminated gradually. "Unsupervised nearly networks" are able to form independently different classes on the basis of samples and to arrange the samples in these classes.

A disadvantage of traditional techniques of the machine learning is that, except for the unsupervised neural networks, learning before the use of the system must take place and that no possibility of learning during the operation exists.

The approach of directly learning from the environment [Green et al 1997] wants to achieve this independent learning. Possible procedures are " Reinforcement Learning ", " Learning by Observation" and learning by statements also called " Instructional Learning ".

- Reinforcement Learning is based on the approach that on the basis of a function, from each state of the system, a certain action can be triggered. When several actions can be triggered from a particular state, a selection strategy is used to decide which one is eventually executed. The function is automatically adapted to the user behaviour on the basis of a learning algorithm in the course of the time. Substantial aspects of this approach are thus the learning algorithm and the selection strategy.

- Learning by observation means that individual agents imitate and thus learn their behaviour by observation of the behaviour of more experienced agents. It is not necessary to have an active teacher; the agents can learn and observe independently.

- Another learning method is learning by statements. Agents are explicitly instructed a certain behaviour. This can take place via the user, or via an experienced agent.

Another approach that achieves the same results is the " Case Based Reasoning " [Green et al 1997]. Here, system users are divided into classes and each individual is assigned to a class on the basis of his actions steps. Then stereotyped behaviours, which the agent exercises, are assigned to this class. As soon as a user is assigned to a class, his behaviour can modify the stereotyped behaviour of the class. The advantage of this approach consists of the fact that individual user profiles can be formed during a short period of time, this is not the case with the interaction of only one user.

c) Theory of intelligent agents

Starting point for considerations to formal theories of agent systems is the thesis that the behaviour of agents seems to be determined by faith, desires, etc... [Woolridge, Jennings 1995]. The philosopher Daniel Dennett shaped the term "Intentional system" for such systems.

However simple systems such a light switch can also be described as intentional systems: A light switch is a simple, cooperative agent, whose function consists of letting current flow if it believes that we require it. The way to communicate with it is simply the pressure put on it.

It becomes clear that such a description does not make sense for a light switch. Intentional descriptions are however useful in systems with complex functionality that can not be guessed easily. Here an intentional description serves as tool for abstraction in order to predict, describe and explain the behaviour of complex systems. For example (partly even unconsciously) the behaviour of a computer will often be described by this way in place of describing it via its technical components.

Intentional characteristics that describe an agent, can be divided according to [Woolridge, Jennings 1995] in two categories: " information attitudes " that are related to the information that the agent has about its environment, and " pro attitudes ", by which actions of the agent are controlled. Information attitudes are faith and knowledge. Desires, intention, obligation, selected among other things, define pro-attitudes.

Intentional attitudes do not permit a semantically unique description with classical formal logic means (for example, is " faith " to be formalised within a boolean logic?). The most common approach for the formal representation of intentional characteristics is based on the " Possible Worlds" model after Hintikka, which is here only roughly sketched following [Woolridge, Jennings 1995].

In this logic, an agent's beliefs can be characterised as a set of possible worlds. Each world represents a state that can be considered possible given what the agent knows. These worlds are denoted epistemic alternatives. The agent believes whatever is true in all epistemic alternatives. This type of representation seems simple at first sight, however a practicable formal description from this model can be derived over modal logic.

In this model, an agent believes all valid formulas (including all possible tautologies, which are infinite) and all the consequences of its beliefs (also infinite). These two problems constitute the logical *omniscience problem*. Since any real system is resource bounded, it seems that the possible worlds model is inappropriate for representing resource bounded believers This ‘ Logical Omniscience problem ‘ is not solved yet and represents the starting point of further research in this area.

d) Theories of Agency

Combinations of the different components of information attitudes and pro-attitudes have to be refined in order to have a more practical approach.

Several theories of putting these attributes in relation are presented in [Woolridge, Jennings 1995]:

-Moore's theory matter is the question of what an agent needs to know in order to be able to perform some action. He formalised a model of ability in a logic containing a modality for knowledge, and a dynamic logic- for modelling action

-According to Cohen and Levesque's theory, two basic attitudes are used: beliefs and goals. Others attitudes, such as intention, are defined in terms of these.

-Rao and Georgeff have developed a logical framework for agent theory based on three primitive modalities: beliefs, desires, and intentions. A rational agent has bounded resources, limited understanding and incomplete knowledge of what happens in the environment in which it lives. Such an agent has beliefs about the world and desires to satisfy, driving it to form intentions to act. An intention is a commitment to perform a plan. In general, a plan is only partially specified at the time of its formulation since the exact steps to be performed may depend on the state of the environment when they are eventually executed. The activity of a rational agent consists of performing the actions that it intended to execute without any further reasoning ; until it is forced into a revision of its own intentions by changes of its beliefs or desires.

-Singh has developed an interesting family of logics for representing intentions, beliefs, knowledge, know-how, and communication in a branching-time framework

There is still no clear consensus in either the AI or philosophy communities about precisely which combination of information and pro-attitudes are best suited to characterising rational agents.

e) Architectures of intelligent agents

In this paragraph based on [Woolridge, Jennings 1995] and [Nwana, Ndumu 1997] ,general concepts of architecture of agent system are introduced. This constituting a kind of practical approach of the agents theories.

(1) Deliberative architecture

The classical, deliberative architecture of an agent system is based on the symbolic approach of artificial intelligence. In a reasoning system, logical conclusions are made on the basis of an explicit symbolic picture of the reality, these determine the further behaviour of the system. The main problems with this approach are the complexity of most symbol manipulation algorithms and the difficulty of theorem proving in even simple logic.
Others problems are related to the deliberative architectures. How can the real world be transferred into a symbolic representation? How can complex connections of the real world be represented symbolically and how new conclusions can be pulled from it? The first question leads to the research in the areas of pattern recognition, voice recognition, automatic learning, the second to the areas of knowledge representation, automatic reasoning and planning. These problems are still not solved despite the immense volume of works generated by them.

Here follow some example of deliberative architectures :

-Planning agents are agents that have the characteristic of being able of dynamically programming a sequence of execution in order to reach a goal. An early planning system is STRIPS. This system uses a symbolic description of both the world and a desired goal state, and a set of actions descriptions, which characterise the pre- and post- conditions associated with various actions. It then attempts to find a sequence of actions to reach the goal by matching the posts-conditions of actions against the desired goal.

- As mentioned earlier, Rao and Georgeff have developed an agent theory based on the information attitude of belief and the pro-attitudes of desires and intention. Some researchers have also developed agent architectures based on these attitudes. These architectures constitute a kind of refinement of the Beliefs Desires Intentions agent theory. As examples, the IRMA, PRS and COSY are systems based on the BDI architectures.

(2) Reactive architectures

Brooks pointed new ways out in AI and also for the agent research with its criticism at the symbolic approach .

In place of applying global pictures of the reality into a symbolic representation and applying mechanisms for automatic reasoning on this knowledge base, he suggests an architecture that gets along completely without a symbolic representation. Many independent and very simple agents operate hierarchically on the solution of a problem. They constantly are in contact with the real world thanks sensors and react immediately to modifications. On base of the " subsumption architecture " he has developed controls for mobile robots [Woolridge, Jennings 1995].

Maes states the following as main arguments for a reactive architecture of agent systems: They are relatively simple and interact with other agents easily in basics ways. A complex behaviour develops nevertheless from this simplicity. Moreover they permit a stronger form of the division of responsibilities: different modules operate autonomously in each case on one function. There is no model for a global behaviour. Thirdly reactive systems operate very much more near the reality, since they work without complex models of the world but directly with it. [Maes 1991] quoted in [Nwana, Ndumu 1997]

The problem of reactive agent systems is the difficulty to predict their behaviour.

(3) Hybrid architecture

Hybrid agents as the mixture of deliberative and reactive agent systems, is a suggestion to combine the respective advantages of both architectures. They usually consist of subsystems, of which the deliberative sections are responsible for planning and decision making, while the reactive section can react faster to occurrences of the real world, without being dependent on the deliberative system. Reactive systems bring characteristics such as robustness, faster times of response and adaptability into hybrid architectures. Deliberative systems add aspects of the long-term goal orientation.

(4) Heterogeneous architecture

The approach of a heterogeneous architecture emerged from the consideration that in a real system many agents of different natures have to cooperate together, in order to achieve a more increased value than if each agent operates individually.

According to [Genesereth, Ketchpel 1994] a basic condition of existence for such a system is a common language. An agent has to be able to communicate, similarly to the paradigm of object orientation, via messages with other agents. In contrast to the object-oriented programming in which messages can have a different meaning from object to object, agents have to speak an uniform language with other agents.

A second problem of heterogeneous agent systems and of multi agent systems is generally the question of how the communication of the agents can be organised among themselves. Does have each agent to be able to communicate directly with every different one or should a special mediator (" broker ") organise the communication. Such systems are called " federated system ". The immense administration effort is a substantial argument against direct communication. Each agent must know every different one, in order to be able to communicate with it. In open, dynamically expandable systems, this is often also not possible at all. Therefore the approach of a mediator, as shown in figure 1.4, which provides the contact between the partners, is more common.

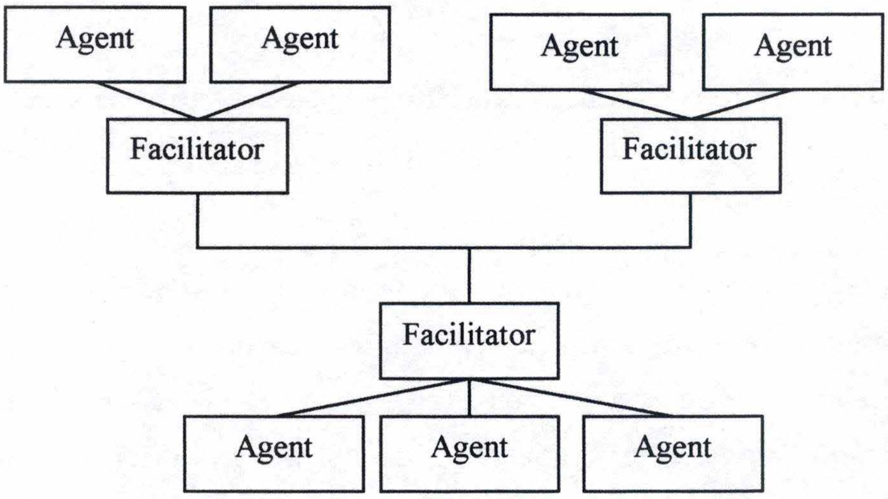

Figure 1.4: Federated system [Genesereth, Ketchpel 1994]

In a federated agent system the agents indicate an agent language, their desires and abilities to the facilitators. The facilitators look up then suitable partners and open the communication connection between them. Mediators can have also functions, which go beyond the pure switching, as for example translations of the agent language or the allocation of a demand for several providers.

5. Agent communication

a) Agent Communication Languages

A reason for the application of multi agent systems is the acceptance that a group of agents can obtain an increase in value compared to an individual agent. [Nwana, Woolridge 1997]

The increase in value of multi agents system can usually be achieved only by communication of the individual agents among themselves. When the communication among agents is made implicitly, i.e. if each agent knows always exactly what the other agents would ask it just by observing the environment, a system of multi agent could get along without communication. However, this situation can only be reached only in very small trivial systems.

Communication is necessary for systems in which agents cooperates in order to achieve a common target and in environments where agents compete with different targets.

Purpose of communication can be the exchange of information, intentions or targets between two or more agents. In addition, the delegation of tasks and the monitoring of their execution, assuming or rejecting a task or displaying the progress, success or failure can be possible contents.

The " Knowledge Query and Manipulation Language " (KQML, [Finin et al 1994]) represents an approach to standardise the language with which agents communicate together.. KQML is a result of the DARPA project " Knowledge Sharing Effort " (KSE) [Neches 1994].

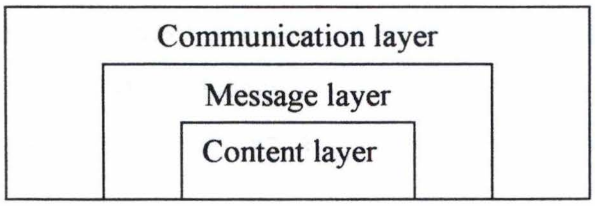

Figure 1.5: An abstract view of the KQLM language from [Nwana, Woolridge 1997]

For example the record in the figure 1.7 states that a chip is larger than another.

```
(ask-if
:language KIF
:ontology Chips
:content (> (* (width chip1) (length chip1))
* (width chip2) (length chip2))))

(reply
:language KIF
:ontology Chips
:in-reply-to Agent1
:content (true))
```

Figure 1.8: An example of KQML performatives embedding KIF.

In the figure 1.8, is an example of KIF embedded in KQML as part of communication between two agents. Agent 1 asks first whether chip 1 is larger than chip 2 and agent 2 acknowledges this afterwards.

The " chip " ontology, which is used in the example must be determined : what the terms " chip1 ", " chip2 ", " width " and " length " mean in this special context, must be available to both communication partners.

5. Agent communication

a) Agent Communication Languages

A reason for the application of multi agent systems is the acceptance that a group of agents can obtain an increase in value compared to an individual agent. [Nwana, Woolridge 1997]

The increase in value of multi agents system can usually be achieved only by communication of the individual agents among themselves. When the communication among agents is made implicitly, i.e. if each agent knows always exactly what the other agents would ask it just by observing the environment, a system of multi agent could get along without communication. However, this situation can only be reached only in very small trivial systems.

Communication is necessary for systems in which agents cooperates in order to achieve a common target and in environments where agents compete with different targets.

Purpose of communication can be the exchange of information, intentions or targets between two or more agents. In addition, the delegation of tasks and the monitoring of their execution, assuming or rejecting a task or displaying the progress, success or failure can be possible contents.

The " Knowledge Query and Manipulation Language " (KQML, [Finin et al 1994]) represents an approach to standardise the language with which agents communicate together.. KQML is a result of the DARPA project " Knowledge Sharing Effort " (KSE) [Neches 1994].

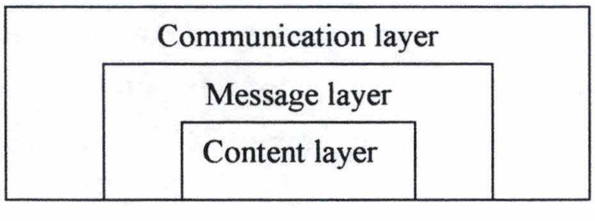

Figure 1.5: An abstract view of the KQLM language from [Nwana, Woolridge 1997]

As shown in figure 1.5, KQML is structured in three layers. In the content layer is transmitted the actual message. About the contents of this message KQML does not state anything. It could be anything, as long as it is ASCII-representable. The " message layer " defines different categories of messages (illustrated by the figure 1.6) as well as the protocol, according to which these are to be exchanged. By protocol, it is meant the rules that agents must use when initiating and maintaining an exchange. KQML specifies several protocols, including synchronous (where a blocking query waits for an expected reply) and asynchronous (which involves non-blocking messages). The 'Communication layer' provides the security for the transfer of the messages between the agents.

Category	Reserved performative names
Basic informational performatives	Tell, deny, untell, cancel
Basic query performatives	Evaluate, reply, ask-if, ask-about, ask-one, ask-all, sorry
Multi-response query performatives	Stream-about, stream-all
Basic effector performatives	Achieve, unachieve
Generator performatives	Standby, ready, next, rest, discard, generator
Capacity definition performatives	Advertise
Notification performatives	Subscribe, monitor
Networking performatives	Register, unregister, forward, broadcast, pipe, break
Facilitation performatives	Broker-one, broker-all, recommended-one, recommend-all, recruit-one, recruit-all

Figure 1.6:Some KQML performatives from [Nwana, Woolridge 1997]

Although the use of a general agent language for heterogeneous agent systems is obvious, there is number of other approaches for communication between agents beside KQML. Usually only one subset of the KQML message is implemented, often also in another meaning and with other protocols. Many today's prototypes use an owned agent language.

b) Ontology

By an ontology one understands the background knowledge (" world knowledge ") over facts and connections between them, which is essential for the execution of successful communication. Ontology extends the syntax of a language by a necessary semantics. One can compare an ontology with the controlled vocabulary of a database.

Similarly to the agent languages, it is also necessary that all the partners that take part in the same communication , if they want to inform themselves, must use the same ontology. For many limited domains of knowledge there are ontologies usually integrated in prototypes. As it is the case of owned languages it comes also here to problems, if the systems must communicate. Also the extension of an existing ontology, in order to cover for example a new field of knowledge, becomes difficult with an implicit integration.

A little practicable solution would be the creation of a global ontology with a all global " world knowledge ". There are nevertheless attempts to manage this venture. The CYC project is by far the most ambitious project in this area.

The " Knowledge interchange format " (KIF) tries to take another way, in order to solve the problem of the semantics of messages. Like KQML from the KSE project, the DARPA developed KIF a Meta pattern for different knowledge bases. By a common Meta pattern different knowledge bases becomes interoperable and modular.

KIF is a prefix version of first order predicate calculus with extensions to support non-monotonic reasoning and definitions.

```
(> (* (width chip1) (length chip1))
   (* (width chip2) (length chip2)))
```

Figure 1.7: Example of KIF from : [Genesereth, Ketchpel 1994]

For example the record in the figure 1.7 states that a chip is larger than another.

```
(ask-if
:language KIF
:ontology Chips
:content (> (* (width chip1) (length chip1))
* (width chip2) (length chip2))))

(reply
:language KIF
:ontology Chips
:in-reply-to Agent1
:content (true))
```

Figure 1.8: An example of KQML performatives embedding KIF.

In the figure 1.8, is an example of KIF embedded in KQML as part of communication between two agents. Agent 1 asks first whether chip 1 is larger than chip 2 and agent 2 acknowledges this afterwards.

The " chip " ontology, which is used in the example must be determined : what the terms " chip1 ", " chip2 ", " width " and " length " mean in this special context, must be available to both communication partners.

6. Agents Programming

Generally an agent can be implemented in any programming language. In order to simplify the development of agents, many extensions of conventional programming languages exist. Also a list of many of these programming languages with which special aspects of an agent can be more easily developed can be found at [AgentBuilder 2000].

Agents, who can communicate with their system environment or move as mobile agent from a system to another, need interfaces for these systems, so they can interact on standardised run time environments in different systems, from which they can migrate to another.

In particular for mobile agents different systems were conceived by many companies. General Magics Telescript was one of the first commercially available environments for mobile agents. Innumerable other architecture approaches exists. Among the most popular, can be found Aglets [Aglets 2000], Concordia [Concordia 2000] and Grasshopper [Grasshopper 2000].

By the important use that the programming language Java gained for distributed applications it is not a surprise that Java based architectures for mobile agents are increasingly more common.

An interoperability between the systems of different manufacturers is indispensable for the success of mobile agents. Several bodies work in order to create agents standards for a better interoperability [The agent society 2000] : the Agent Society via its Agent Interop Working Group, the Active Group, the Foundation for Intelligent Physical Agents (FIPA), the IETF, the Object Management Group via its Mobile Agent Facility Specification and the WWW Consortium.

a) An agents programming language: Aglets

The Aglets system lays on Java as programming language. Aglets are Java objects, which can transport themselves from host to host, that means that they can stop their execution at any time, transport themselves to another host and carry on again their execution once they arrive there. The Java Aglet API (J-AAPI; " Java Aglet Application Programming Interface ") [Lange 1997] as a fundamental interface description between Aglets and their system environment follows these development principles:

-simplicity and expandability: The Aglet development should be as simple as possible for Java programmer.
-platform independent: Aglets should be executable on all host, which support the J-AAPI.
- industry standard: The J-AAPI is to become an industry standard.
- security: Non-trusted aglets shouldn't become a risk for the hosts.

Aglets transfer from host to host and aglet communication uses an agent transfer Protocol ("ATP" [Lange, Aridor 1997]). This application level protocol defines the syntax for the designation and identification of agents in the Internet as well as a transmission protocol for transferring agents from host to host. ATP is not only used by aglets. According to [Lange, Aridor 1997]: « While mobile agents may be programmed in many different languages and for a variety of vendor specific agent platforms (consisting of virtual machines and libraries), ATP offers the opportunity to handle agent mobility in a general and uniform way. »

The Java –Aglet API defines the following classes of the Aglet system [Lange 1997]:

- An **aglet** is a mobile Java object that visits aglet-enabled hosts in a computer network. It is autonomous, since it runs in its own thread of execution after arriving at a host, and reactive, because of its ability to respond to incoming messages.

- A **context** is an aglet's workplace. It is a stationary object that provides a means for maintaining and managing running aglets in a uniform execution environment where the host system is secured against malicious aglets. One node in a computer network may host multiple contexts.

- A **proxy** is a representative of an aglet. It serves as a shield for the aglet that protects the aglet from direct access to its public methods. The proxy also provides location transparency for the aglet. That is, it can hide the real location of the aglet.

- A **message** is an object exchanged between aglets. It allows synchronous as well as asynchronous message passing between aglets. Message passing can be used by aglets to collaborate and exchange information in a loosely coupled fashion.

- A **message manager** allows for concurrency control of incoming messages.

- An **itinerary** is an aglet's travel plan. It provides a convenient abstraction for non-trivial travel patterns and routing.

- An **identifier** is bound to each aglet. This identifier is globally unique and immutable throughout the lifetime of the aglet.

"Aglets are possibly the most applicable mobile agent technology at the present moment in time."

[Green et al. 1997]

b) An agents programming language: Jack

JACK Intelligent Agents is a framework in Java for multi-agent system development built by the company Agent Oriented Software Pty. Ltd.

According to [Jack 1999] : « The company's aim is to provide a platform for commercial, industrial and research applications. To this end, its framework supplies a high performance light-weight implementation of the BDI architecture and can be easily extended to support different agent models or specific application requirements. »

Other products from universities and different companies supply implementations of the BDI architecture [AgentWeb 2000] : dMARS, UMPRS Agent, JAM and Tileworld.

'JACK Intelligent Agents' is developed in Java and Uses JACK Agent Language, JAL, which follows the standard JAVA/Object Oriented paradigm.

Jack agents have been designed mainly for use as components of larger environments. On one hand, an agent must coexist and be visible as simply another object by non-agent software. On the other hand, a Jack programmer must be allowed to easily access any other component of a system.

For similar reasons, Jack agents are not linked to any specific agent communication language. Jack provides a native lightweight communication infrastructure. Nothing prevents the adoption of a high level symbolic protocol such as KQML. However Jack has been oriented towards industrial object-oriented middleware and message passing infrastructures.

As mentioned earlier, JACK Intelligent Agents is an implementation of the BDI architecture. This is reflected by the relation between the JACK Agent Language components and the BDI information attitude and pro-attitudes as shown in the figure 1.9.

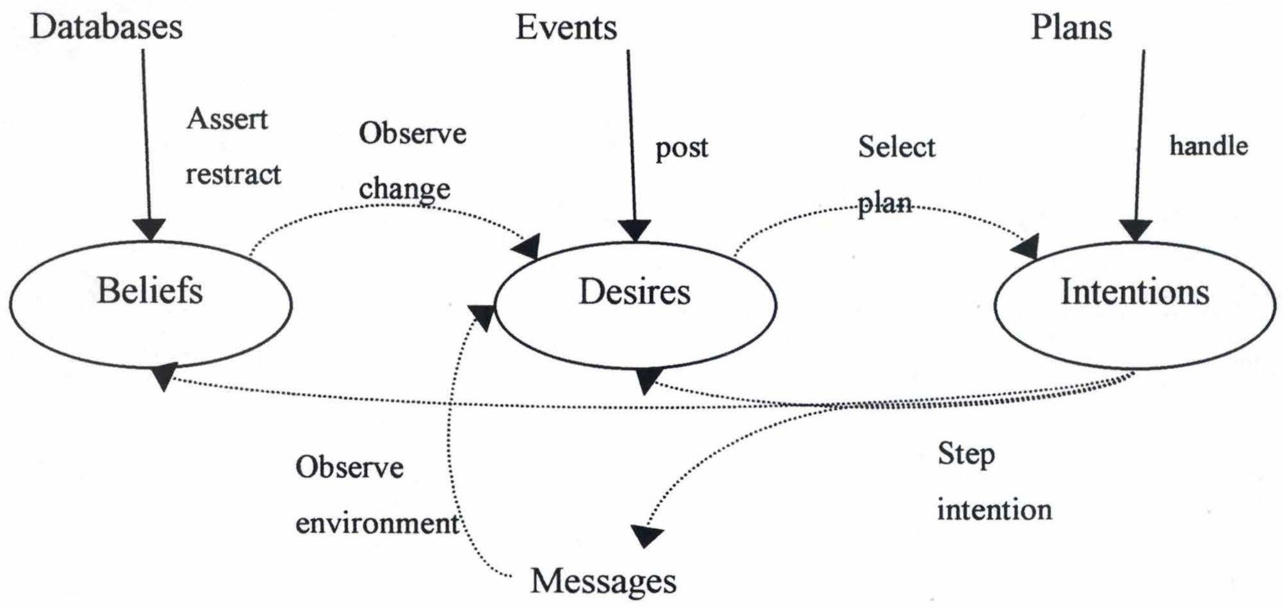

Figure 1.9: BDI attitudes and Jack components

JACK base components and methods can be described as follows [JACK 2000]:

- **Agent**, which models the main reasoning entities in JACK.

- **Event**, which models occurrences and messages that these agents must be able to respond to. Events may arise externally from messages from other agents, or internally as a consequence of one of the agent's own actions, or in response to one of its internal goals.

- **Plan**, which models procedural descriptions of what an agent does to handle a given event. All the action that an agent takes is prescribed and described by the agent's plans. Plans possess methods to determine their relevancy to particular kinds of events or to a particular events occurrences.

- **Database**, which models an agent's knowledge as beliefs that follow either a closed world or open world semantics. Databases represent an agent's beliefs as first order relational tuples, and maintain their consistent logical and key constraints. Databases can be defined to automatically send events when particular beliefs modifications occur.

- **Capability**, which aggregates functional components (events, plans, databases and other capabilities) that agents can use.

7. Areas of application of intelligent agents

The use of intelligent agents is judged useful in many ranges of application. Many approaches of the term have been pointed out earlier under many different definitions, but other approaches can be seen. Enumeration of the areas of application partly follows [IBM 1995] and [Nwana 1996].

a) System and network management

In the area of system and network management, the agent term emerged early. The expenditure guarantying always the availability of the systems continues to increase in the measure, in which the information and communication technology at the basis become more complex. Here, intelligent agents help the system operators and administrators to detect ,seek out and recover.

In a complex system, there are agents who monitor in each case several items of the system. According to this model, agents recover simple errors or inform superordinate instances if they cannot solve it.

b) Mobile computers

Computers become ever smaller and more mobile. The use of " Personal Digital Assistants " (PDA) is increasing. Intelligent agents, who operate within a network, can execute functions for the user independently, without that he must be constantly connected with the network possibly via an expensive communication medium. They can receive and sort for example emails, look for information or coordinate themselves with other agents. [Chess 95]

c) Information management

The intelligent organisation of detailed information is already an active area of research. Agents could help to sort information by learning the user behaviour or via a type of ' social filtering '. 'Social filtering' means that the user agent determines the relevance of certain information by cooperation with others agents which users potentially share the same interests.

Agents can adapt over the time to the needs and tasks of a certain user, by becoming familiar step by step with their interests, preferences and habits. These agents, like a human assistant, gradually replace their users to perform some functions and to take some decisions.

Also the active search for information can be simplified by the use of agents. By the access to ever more sources of information and the increasing importance of the WWW, agents are needed to sort informations in order to find relevant ones.

d) Workgroups

In workgroups, agents, who are active for the respective members of the group, can solve problems by cooperation without effort of their human orderers. For example intelligent agents can maintain a department calendar and react independently to modifications, by informing other agents and postponing dates. Several agents do not have to exclusively cooperate. They can also compete in a certain measure. For example, if it is question of the assignment of limited resources.

e) Electronic Commerce

Electronic commerce is a growing with the popularity of the Internet. On one hand, buyers need to find sellers of products and services. They also need to find product information and expert advice. On the other hand, sellers need to find buyers and they need to provide expert advice about their product or service. Both buyers and sellers need to automate the handling of their electronic business.

Intelligent agents can be useful in the electronic commerce in a number of ways. Once they have the specifications of a product, agents can "go shopping" for a user. They can also act as "salespeople" for sellers by providing product or service sales advice and helping them to solve their problems.

f) User Interfaces

Although the user interface has been improved thanks graphical user interfaces (GUIs), for many, it remains difficult to learn and use computers. As applications of computers improve, the user interface needs to accommodate the increase in complexity. As user populations grow and diversify, computer interfaces need to learn user habits and preferences and adapt to individuals.

Intelligent agents can help to solve both these problems. Intelligent agent technology allows systems to monitor the user's actions, develop models of user abilities in order to automatically help when problems arise.

g) Software engineering

As already mentioned, agents are considered as a natural extension of the object programming orientation. Being so, agents oriented programming inherits of all the benefits of the object oriented programming A similarity of both these approaches is that both emphasise the importance of interactions between entities. However, according to [Jennings, Woolridge 1998] there are also a number of important differences:

- Objects are generally passive in nature: they need to be sent a message before they come alive.
- When one of his methods is invoked, an object doesn't have any choice about refusing to perform the invoked method. This approach may suffice for smaller applications in co-operative and well-controlled environments, it is not suited to either large or competitive and open environments.
- Object-orientation fails to provide an adequate set of concepts and mechanisms for modelling complex systems. Complex systems require richer problem solving abstractions. Individual objects represent too fine a granularity of behaviour and method invocation is a too primitive mechanism for describing the types of interactions that take place.
- Object- oriented approaches provide minimal support for structuring collectives (basically inheritance class hierarchies define relationships). Complex systems involve a variety of organisational relationships (of which "part-of" and "is-a" are but two of the simpler kinds).

h) Management of business processes

The execution of business processes is for a long time supported by the information and communication technology. Autonomous software agents undertake the monitoring of business processes and permit a decentralised administration. So, autonomous agents can adjust faster to new situations.
Also, the use of agents permits a more exact modelling of the reality, since the organizational structure of the enterprises can be illustrated more exactly. Strongly distributed enterprises, in the extreme completely virtual enterprises, profit from the application of agents for the support of business processes.
Part of business processes can be executed by an agent wich knows the necessary steps for the completion of the job. This one will divide the job into subtasks and will send these subtasks to other more specialized agents owned by morte specialized company departments.

According to [Woolridge, Jennings 1995], the use of agents is a natural way to face the distribution of data and resources.

Second Part: Belief Revision

1. Situation

Classical logics have played an important role in computer science. It has been an important tool for developing architecture and software. According to the logicians, the human way of reasoning is also amenable to analysis using classical logic. However, artificial intelligence researchers have shown that classical logic is not sufficient and robust enough to make computers reasoning like humans. Humans do not reason as a classical reasoning system. They jump to conclusions based on commonsens reasoning. By commonsense reasoning, humans refer to such statements as " ..it is my experience that 'this' must be the case." Or "... there is no good reason not to believe 'this'".

So the goal of nonmonotonic reasoning is to model the way in which commonsense is used by humans. So, nonmonotonic reasoning must be able to jump to conclusions and be sufficiently robust so that when a conclusion reached by nonmonotonic reasoning is shown to be wrong, it may be revised. Nonmonotonic reasoning is a new logic developed by artificial intelligence searchers and is based on classical logic. Roughly said, their property of nonmonotonicity means that the amount of conclusions that can be drawn decreases when the amount of premisses increases

Most well known nonmonotonic logics are: Circumscription developed by John McCarthy, Default Logic developed by Reiter, autoepistemic developed by Moore and Belief Revision developed by Carlos Alchourron, Peter Gärdenfors and David Makinson. [Alchourrón et al., 1985]

Most nonmonotonic logics are processes that allow a reasoner to make plausible conjectures in the absence of complete information. Belief revision is differentiated from them by the fact that in place of dealing with incomplete informations, it deals with the dynamics of belief systems. It models how an agent or a computer system updates its beliefs when receiving new information.

2. Introduction

Dynamically, a reasoned can updates its belief system (a set of sentences called a theory and noted T) on three main ways:

(i) *Expansion*: A new sentence together with its logical consequences is added to a theory T. The belief system that results from expanding K by a sentence a will be denoted $T^{+}a$.

(ii) *Revision*: A new sentence that is inconsistent with a theory T is added, but in order that the resulting belief system is consistent some of the old sentences in T are deleted. The result of revising T by a sentence a will be denoted $T^{*}a$.

(iii) *Contraction*: Some sentence in T is retracted without adding any new facts. In order that the resulting system is consistent some other sentences from T may have to be given up. The result of contracting T with respect to the sentence a will be denoted $T^{-}a$.

Expansions of belief systems can be handled easily. $T^{+}a$ can simply be defined as the logical closure of T together with a:

(Def^{+}) $T^{+}a = \{b: T \cup \{a\} \mid\!- b\}$

On the other hand, the revision and the contraction of a theory with respect to a sentence are not so trivial.

A well-known example can illustrate this:

An reasoning system beliefs that 'all birds fly' and that 'Tweety is a bird', consequently it also believes that 'Tweety can fly'. But later, it discovers that it was mistaken because it appears that 'Tweety cannot fly' at all. How should it modify its knowledge ? Off course, it has to remove ‘Tweety can fly’ but it has to remove other beliefs that are in conflict with its recent discovery that ‘Tweety cannot fly’. It appears that it has several possible courses of action to follow in this regard: it can retract the information that ‘Tweety is a bird’, it can retract that ‘all birds fly’ or it can retract both of these beliefs.

So it appears that it is not possible to easily give an explicit definition of revisions and contractions as for expansions.

Nevertheless, some intuitive basics principles are useful for the research of such definitions. Modifications to the knowledge base have to be rational and guided by the principle of minimal changes. By ‘rational’ it is meant that the reasoner realises that inconsistencies are problematical and thus seek to avoid them. The ‘principle of minimal change’ means that as much information should be conserved as is possible in accordance with an underlying preference relation. The underlying preference relation is used to capture the information content of the knowledge base, the reasoning agent’s commitment to this information, and how the information should behave under change. This concept of minimal change is not to be evaluated by cardinality measures but rather by the interdependencies among the knowledge.

When approaching the problem of belief revisions (and contractions) there are two approaches: the axiomatic and the constructive. The axiomatic approach consisting in defining some constraints, some postulates that revision and contraction functions have to respect, so this approach defines a class of functions for revisions and a class of functions for contractions. The constructive approach consisting in defining a particular function for each class of contraction and revision functions.

Firstly, the axiomatic method will be presented. Then, it will be shown that a preference relation called an epistemic entrenchment ordering contains sufficient information to construct these change functions.
After the presentations of the postulates and the proposals for revision methods, the two approaches will be connected via theorems.

This presentation is mainly based on [Williams, 1997] and [Williams, 1998].

3. Preliminaries

A language L which is based on first order logic will be used. The set of all logical consequences of a set $T \subseteq L$, that is $\{a{:}T \mid\!- a\}$, is denoted by Cn(T). A theory of L is any subset of L close under |-. A consistent theory of L is any theory of L that does not contain both a and $\neg a$, for any sentence a of L. A complete theory of L is any theory of L such that for any sentence a of L, the theory contains a or $\neg a$. We shall denote by K_L the set of all theories of L.

4. Rationality postulates for belief revisions

As seen earlier, expansion models the simplest change to a knowledge base i.e. the acceptance of information without the removal of any previously accepted information.
The expansion of a theory T with respect to a sentence a is the logical closure of T and a.
More formally, an expansion function $^+$ is a function from $K_L \times L$ to K_L, mapping (T,a) to T^+a where $T^+a=Th(T \cup \{a\})$. Expansion is a monotonic operation: $T \subseteq T^+a$. If $\neg a \in T$, then T^+a is inconsistent. If $\neg a \notin T$, the principle of minimal change is respected since T^+a would be the smallest change we can logically make to T in order to incorporate a.

A contraction of T with respect to a sentence a consists of the removal of a set of sentences from T so that a is no longer implied, provided a is not a tautology. Formally, a contraction function $^-$ is any function from $K_L \times L$ to K_L, mapping (T,a) to T^-a which satisfies the following postulates. For any a,b $\in$ L and any T $\in$ K_L:

($^-$1) $T^-a \in K_L$
($^-$2) $T^-a \subseteq T$.
($^-$3) *If* $a \notin T$, *then* $T \subseteq T^-a$
($^-$4) *If not* $\vdash a$, *then* $a \notin T^-a$.
($^-$5) $T \subseteq (T^-a)^+a$
($^-$6) *If* $\vdash a \leftrightarrow b$, *then* $T^-a = T^-b$.
($^-$7) $T^-a \cap T^-b \subseteq T^-a{\wedge}b$
($^-$8) *If* $a \notin T^-a{\wedge}b$ *then* $T^-a{\wedge}b \subseteq T^-a$

These postulates embody the principle of minimal change and act as integrity constraints for change functions.

($^-$1) says that the result of a contraction is a theory.
($^-$2) says that contracting a theory only involves the removal of old information and never the incorporation of new information.
($^-$3) says that when information a is not in believed then taking it away should have no effect
($^-$2) and ($^-$3) say that if a is not in T, then $T^-a=T$.

($^-$4) says that if a is not a tautology then it must be removed in the contraction T^-a.

($^-$5) says (with the previous four) that if $a \in T$ then $T=(T^-a)^+a$. Intuitivelly, it means that a minimal amount of information is lost during the contraction.

($^-$6) says that the same result is obtained when a contraction is made with respect to equivalent logical sentences.

($^-$7) says that the theory that results from contraction with respect to $a \wedge b$ sould never be smaller than taking the intersection of T^-a and T^-b.

($^-$8) says that if a is not contained in the contraction with respect to the conjunction $a \wedge b$ then the contraction with respect to this conjunction cannot be larger than the theory obtained by contraction a alone.

Given ($^-$1) and ($^-$6), ($^-$7) and ($^-$8) imply that T^- $a \wedge b$ is equivalent to either T^-a, T^-b or $T^-a \cap T^-b$.

Revision attempts to change a knowledge base as little as possible in order to incorporate newly acquired information. This new information may be inconsistent with the knowledge base. In order to maintain consistency, some old information may need to be retracted. Thus revision functions are nonmonotonic in nature.

Formally, a revision function * is any function from $K_L \times L$ to K_L, mapping (T,a) to T^*a wich satisfies the following postulates. For any $a,b \in L$ and any $T \in K_L$:

(*1) $T^*a \in K_L$

(*2) $a \in T^*a$.

(*3) $T^*a \subseteq T^+a$

(*4) *If* $\neg a \notin T$ *then* $T^+a \subseteq T^*a$

(*5) *If* $T^*a = \bot$ *then* $|\!- \neg a$

(*6) *If* $|\!- a \leftrightarrow b$, *then* $T^*a = T^*b$.

(*7) $T^*a \wedge b \subseteq (T^*a)^+b$

(*8) *If* $\neg a \notin T^*a$ *then* $(T^*a)^+b \subseteq T^*\, a \wedge b$

(*1) says that revising a theory results in a theory.

(*2) says that the information to be added by a revision is always successfully incorporated.

(*3) says that revising a theory can never incorporate more information than an expansion operation.

(*4) says that if $\neg$a is not in T then the expansion of T with respect to a is contained in the revision with respect to a.

(*3) and (*4) say that if a is consistent with T then $T^*a=T^+a$.

(*5) says that the only way to obtain an inconsistent theory is to revise with an inconsistent sentence a.

(*6) says that revision functions are syntax independent.

(*7) says that the theory that results from revising with respect to a$\wedge$b should never contain more than the revision with respect to a followed by the expansion with respect to b.

(*8) says that if $\neg$a is not contained in the revision with respect to a then the revision with respect to a followed by the expansion with respect to a should not contain more information than the theory that results from revising withrespect to a$\wedge$b.

If the revision function* satisfies the first six postulates, then the postulates (*7) and (*8) imply that $T^*a\vee b$ is equivalent to T^*a,T^*b or $T^*a \cap T^*b$.

The following theorems gives the relation between the various change functions:

If $^-$ is a is a contraction function and $^+$ the expansion function, then * defined by the Levi Identity below defines a revision function.

$$T^*a = (T^- \neg a)^+ a$$

*If * is a revision function, then $^-$ defined by the Harper identity below defines a contraction function.*

$$T^-a = T \cap T^* \neg a$$

5. The constructive approach.

As it has been mentioned earlier, the above postulates do not provide a mechanism for defining a particular function but they describe classes of functions. To isolate a unique function, additional structure is necessary. This structure is a preference relation such as an epistemic entrenchment ordering.

Intuitively, this preference relation can be used to determine a change function by providing a selection criteria that can be used to identify those sentences to be retracted, those to be retained, and those to be acquired during changes.

Given a theory T of L, an epistemic entrenchment related to T is any binary relation $\leq$ on L satisfying the conditions below:

(EE1) *If $a \leq b$ and $b \leq c$, then $a \leq c$.*

(EE2) *For all a, $b \in L$, if $a \mid\!- b$ then $a \leq b$.*

(EE3) *For all a, $b \in L$, $a \leq a \wedge b$ or $b \leq a \wedge b$.*

(EE4) *When $T \neq \perp$, $a \notin T$ iff $a \leq b$ for all $b \in L$.*

(EE5) *If $b \leq a$ for all $b \in L$, then $\mid\!- a$.*

If $a \leq b$, b is said being at least as entrenched as a.

$a < b$ if $a \leq b$ and not $b \leq a$.

If $a \leq b$ and $b \leq a$, a and b are said being aqually entrenched.

(EE1) says that an epistemic entenchment is transitive.

(EE2) says that if a is logically stronger than b, then b is at least as entrenched as a.

(EE3) with (EE1) and (EE2) implies that a cunjunction is ranked at the same level as its least ranked conjunct.

(EE4) says that sentences not in the theory T are minimal.

(EE5) says that tautologies are maximal

The set cut≤(a) contains all the sentences that are at least as entrenched as a.

More formally:

For an epistemic entrenchment ordering ≤ and a sentence a, define $cut_{\leq}(a)=\{b : a \leq b\}$.

An important property of an epistemic entenchement is that it is a total preorder of the sentences in L such that the following theorems holds:

If ≤ is an epistemic entrenchment, then for any sentence a, cut≤(a) is a theory.

A subtheory of a finitely axiomatizable theory may not be finitely axiomatizable. A finite description of an epistemic entrenchment ordering is useful for developping a computational model for change functions:

An epistemic entrenchment ordering ≤ is finitely representable if and only if it has a finite number of natural partitions, and for all a ∈ L, cut≤(a) is finitely axiomatizable.

The next theorem from Gärdenfors and Makinson provides a constructive method for building a contraction function from an epistemic entrenchment ordering:

Let T be a theory of L. For every contraction function ⁻ for T there exists an epistemic entrenchment ≤ related to T such that (E⁻) is true for every a ∈ L. Conversely, for every epistemic entrenchment ≤ related to T, there exists a contraction function ⁻ such that (E⁻) is true for every a ∈ L.

$$(E^{-}) \qquad T^{-}a = \begin{cases} \{b \in T : a < a \vee b\} & \text{if not } \vdash a \\ T & \text{otherwise} \end{cases}$$

An analogous result for revision is provided in the theorem below:

*Let T be a theory of L. For every revision function * for T there exists an epistemic entrenchment ≤ related to T such that (E*), below, is true for every $a \in L$. Conversely, for every epistemic entrenchment ≤ related to T, there exists a revision function * for T such that (E*) is true for every $a \in L$.*

$$\text{(E*)}\ T{*}a = \begin{cases} \{b \in L: \neg a < \neg a \vee b\} & \text{if not} \vdash \neg a \\ \bot & \text{otherwise} \end{cases}$$

6. Implementing belief revision

Many difficulties arise when developping a computational model of belief revision.

First, appears a representation problem because the epistemic entrenchment ordering typically ranks an infinite number of sentences.

Secondly, the epistemic entrenchment is lost in the process of change, and then the iteration of change functions is not naturally supported.

In order to face this problem, a finite partial entrenchment ranking is defined as a finite representation of a finitely representable epistemic entrenchment ordering.

A finite partial entrenchment ranking grades the content of a finite knowledge base according to its epistemic importance. Inuitively, the higher the value assigned to a sentence the more firmly held it is, or the more entrenched it is.

A finite partial entrenchment ranking is a function B from a finite subset of sentences into the interval [0,1] such that the following conditions are satisfied for all a $\in$ dom(B):

(PER1) *{b $\in$ dom(B): B(a) $<$ B(b)} not |- a if a is not a tautology.*
(PER2) *If |- $\neg$a, then B(a)=0.*
(PER3) *B(a)=1 if and only if |-a.*

(PER1) says that sentences assigned a value higher than an arbitrary sentence a, do not entail a.
(PER2) says that inconsistent sentences are asigned zero.
(PER3) says that tautologies are assigned 1.

A finite partial entrenchment ranking can be used to represent a finitely representable epistemic entrenchment ordering.

The numerical assignment can be viewed in two distinct ways:

- qualitatively, where the relative ordering of sentences is used.
- quantitavely, where the numerical assigned to sentences possesses some extra meaning, such as probability, and a calculus based on their numerical value adopted.

The family of all finite partial entrenchment ranking is denoted by β.

In a finite partial entrenchment ranking, sentences mapped to numbers greater than zero represent explicit beliefs. Their logical closure represents the implicit beliefs.

Define the explicit information content represented by B ∈ β to be { a ∈ dom(B): B(a) > 0}, and denote it by exp(b). Similarly, define the implicit information content represented by B ∈ β to be Th(exp(B)), and denote it by content(B).

A finite partial entrenchment ranking usually represents an incomplete specification of a reasoning system preferences from wich an epistemic entrenchment ordering can be generated. In order to describe this generated epistemic entrenchment ordering, it is necessary to rank implicit information.

Let a be a nontautological sentence. Let B be a finite partial entrenchment ranking. We define the degree of acceptance of a to be

$$degree(B,a) = \begin{cases} \textit{largest } j \textit{ such that } \{b \in exp(B): B(b) \geq j\} \vdash a & \textit{if } a \in content\ (B) \\ 0 & \textit{otherwise} \end{cases}$$

A procedure for adjusting a finite partial entrenchment ranking can be described.
For this adjustment, not only the new information is needed but also its ranking.

Adjustments use a policy for change based on an absolute minimal measure; they transmute a finite partial entrenchment ranking so as to incorporate the desired new information using an absolute minimal measure of change.

a is the new information to be incorporate. (contingent sentence.)
The number i ($0 \leq i < 1$) is its degree of acceptance.
The set of contingent sentences is denoted L^{Φ}.

*Let $a \in L^{\Phi}$ and $0 \leq i < 1$. We define the adjustment of a finite partial entrenchment ranking B to be a function * such that*

$$B^*(a,i) = \begin{cases} (B^-(a,i)) & \text{if } i \leq degree(B,a) \\ (B^-(\neg a,0))^+(a,i) & \text{otherwise} \end{cases}$$

where

$$B^-(a,i)(b) = \begin{cases} i & \text{if } degree(B,a)=degree(B, a \vee b) \text{ and } B(b) > i \\ B(b) & \text{otherwise} \end{cases}$$

for all $b \in dom(B)$, and

$$B^+(a,i)(b) = \begin{cases} B(b) & \text{if } B(b)>i \\ i & \text{if } a \leftrightarrow b \text{ or } B(b) \leq i < degree(B, a \rightarrow b) \\ degree(B, a \rightarrow b) & \text{otherwise} \end{cases}$$

for all $b \in dom(B) \cup \{a\}$

Two processes are at work in an adjustment. Sentences migrate up or down the ranking. Migration downwards is related to contraction, whilst movement upwards is related to expansion.

Another more practical transmutation is maxi-adjustment. It allows the user to specify dependencies among beliefs using Spohnian Reasons wich is defined as follows:

If a,b in exp(B) and we preserve the properties of a partial entrenchment ranking, then b is a reason for a iff $degree(b \rightarrow a) > degree(a)$.

A maxi-adjustment involves the absolute minimal change of a partial entrenchment ranking required to incorporate the desired new sentence such that currently held information is retained unless there is an explicit reason to retract it.

Let $B \in \beta$ be finite. We enumerate the range of B in ascending order as $j_0, j_1, \dots, j_{Rmax}$. Let a be a contingent sentence, j_m=degree(B,a) and $0 \leq i \leq j_{Rmax}$. Then the (a,i) maxi-adjustment of B is $B^*(a,i)$ defined by:

$$B^*(a,i) = \begin{cases} (B^-(a,i)) & \text{if } i \leq degree(B,a) \\ (B^-(\neg a,0))^+(a,i) & \text{otherwise} \end{cases}$$

where for all $b \in dom(B)$, we define $B^-(a,i)$ as follows:

1. *For b with $B(b) > j_m$ we have $B^-(a,i)(b) = B(b)$.*
2. *For b with $i < B(b) \leq j_m$, suppose we have defined $B^-(a,i)(b)$ for b with $B(b) \geq j_{m-k}$ for k = -1,0,1,2,...,n-1, then for b with $B(b) = j_{m-n}$ we have*

$$B^-(a,i)(b) = \begin{cases} i & \text{if } a \mid\!- b \text{ or} \\ & a \text{ not } \mid\!- b \text{ and } b \in \Gamma \\ & \text{where } \Gamma \text{ is a minimal subset of} \\ & \{b : B(b) = j_{m-n}\} \text{ such that} \\ & \{b : B^-(a,i)(b) > j_{m-n}\} \cup \Gamma \mid\!- a \\ B(b) & \text{otherwise} \end{cases}$$

3. *For b with $B(b) \leq i$ we have $B^-(a,i)(b) = B(b)$.*

And for all $b \in dom(B) \cup \{a\}$ we define $B^+(a,i)$ as follows:

$$B+(a,i)(b) = \begin{cases} B(b) & \text{if } B(b) > i \\ i & \text{if } a \equiv b \text{ or } B(b) \leq i < degree(B, a \rightarrow b) \\ degree(B, a \rightarrow b) & \text{otherwise} \end{cases}$$

Both adjustment and maxi-adjustment can be described by procedural algorithms. Their implementations require the support of a theorem prover.

Third Part: An Application

1. Introduction

In this part, an application consisting of an agents system using Belief Revision is described. This implementation underlines the software engineering advantages of the agents oriented programming paradigm.

This application consists of a system of co-operative agents working together in order to extract a consistent set of beliefs from several sources.

2. Tools used in this application

In order not to reinvent the wheel and to facilitate the implementation, several tools are used.

As seen earlier, agent-programming languages exist in order to facilitate the development of agents systems. 'JACK Intelligent Agents' and its programming language 'JACK Agent Language' being at disposition, they have been used for this application. A description of this tool can be found in the first part of this work.

In order to extract a consistent set of beliefs, the system uses SATEN [SATEN 97] and VADER [VADER 97]. Both have been developed in Java, by the CIN project at the Business Technology Research Laboratory of the University of Newcastle, Australia. VADER is a first order logic theorem prover used by SATEN, a system for first order logic theory extraction and revision.

The integration of these logic engines with JACK agents doesn't present any problems. JACK being developed for facilitating the integration of JAVA objects.

In order to communicate, the agents use the KQML standard language which description can be found in the first part of this work.

3. Goal and situation

The system has been design to solve the following problem:

At a time given by the user, two agents having different beliefs on the same area have to put in common their beliefs about this area in order to enjoy each other's beliefs. Inconsistencies may appear in this common set.

This application consists of a multi benevolent agents system. These agents are co-operative and the system is co-ordinated via a given structure. They communicate via the KQML language and use their own ontology.

4. System architecture

Before presenting the architecture of this system, it can be interesting to present the underlying principles that lead its design:

- This multi-agents system has to be considered as a part of a larger system where others agents, applications and users can also interact with the already existing agents. Among other characteristics of this system reflecting this principle, the capability of communicating via the standard KQML protocol can be mentioned. A possible extension of this architecture is presented at the end of this part.

- In this architecture design, the accent is put on the software engineering advantages presented by the use of agents. As seen earlier, agents oriented programming paradigm is considered as an extension of the object oriented programming paradigm. So the re-use of agents, the grouping of different functions of the same nature in the same agent and a maximum modularity are taken into account.

The intentional global behaviour of this system shown in figure 3.1 can be described as follow:

When the user wants that Agent1 and Agent2 put their beliefs together, he launches the agent being the interface between him and the system. Then this Interface Agent orders the Gathering Agent to begin its job. This one requests the desired beliefs and their rankings to Agent1 and Agents2 which give them it. Once both beliefs sets are gathered into one set, the Gathering Agent add rules with the highest ranking to this set and sent it to the Belief Revision Agent. The added rules can be considered as integrity constraints on these beliefs. The Gathering Agent orders him to make an extraction on the given sentences. When it is done, it sends back the inconsistent-safe beliefs to the Gathering Agent which, after removing the rules, dispatch them to both Agent1 and Agent2.

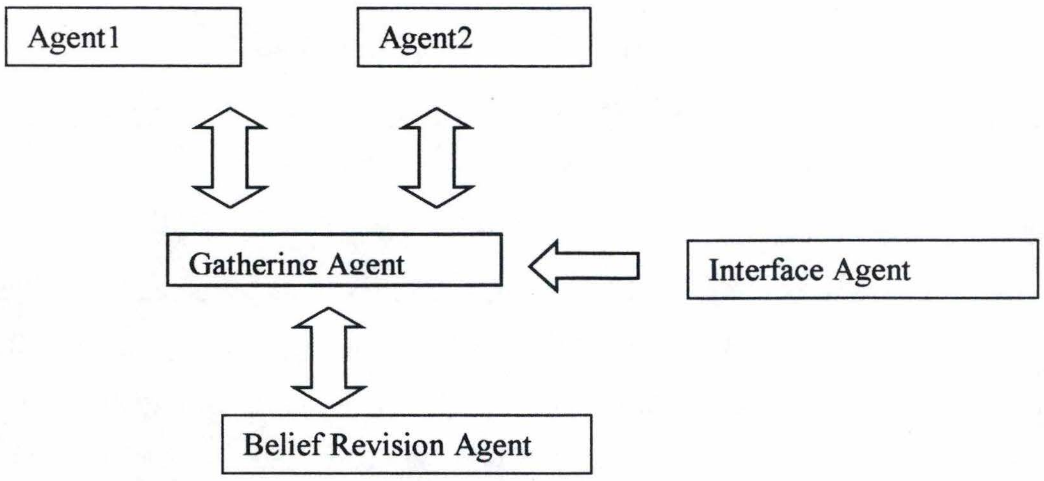

Figure 3.1: Agents and communication channels

5. Knowledge representation

As seen in the JACK overview chapter, agent's beliefs are implemented through instances of the Database class. To be used, an instance of a Database has to have predefined fields. So, in order that an agent possesses a meaning of what it beliefs, and to profit what is offered by the Database class, the content of a database has to be considered as a set of arguments of the same predicate. (I.e. an agent will possess as many database objects as it beliefs different class of things)

This implies that a JACK agent won't be able to dynamically learn new classes of things, as its database formats are pre-defined.
It also implies that JACK doesn't naturally support the representation of complex formulas.

6. Communication

Some KQML performatives are used for the agent's communication. It is clear that others performatives may be used to fulfil the same task. Nevertheless, the presented set seems to be the most efficient one.

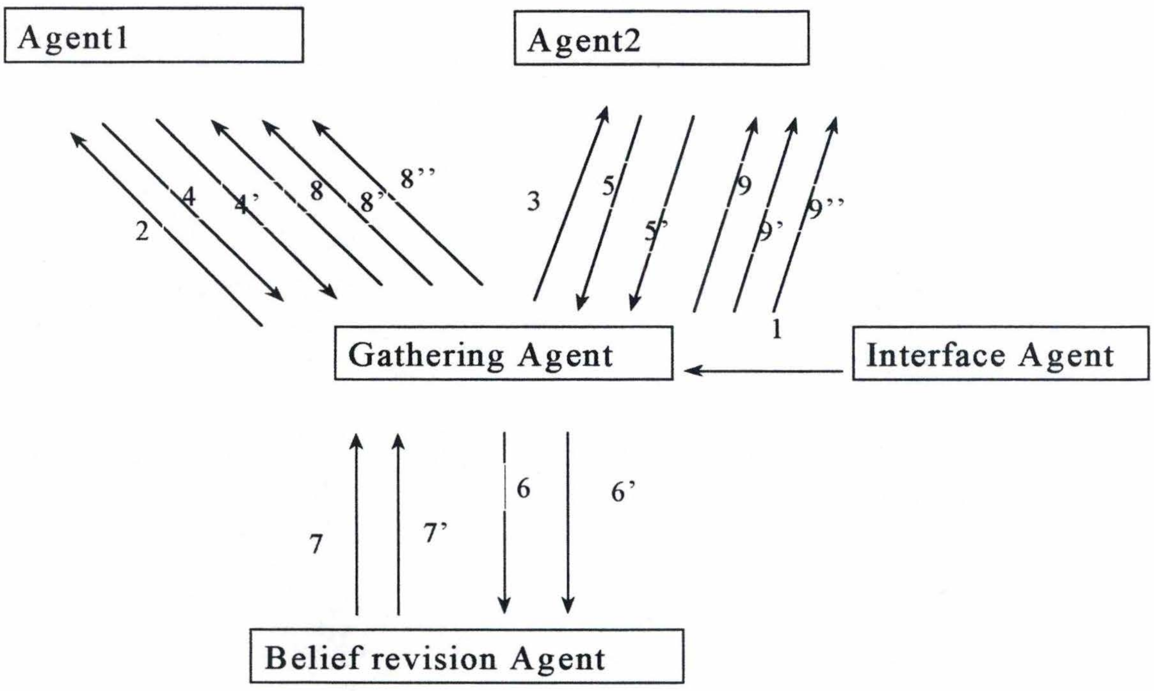

Figure 3.2: Communication via KQML performatives

To be as much clear and generic as possible and because the ontology is not standard, messages contents are not presented. Exception is made for the 'achieve' performative where contents are unique.

The different performatives used in the system are (with numbers referring to figure 3.2):

8 achieve:go

The 'achieve' performative is an order to act. In this case, it is ordered to begin the task.

2 and 3) stream-all

The requested beliefs being about selected areas of interest, these are gathered in particular database tables. This performative contains two kind of SQL message both requesting all the data's of a table.

4 and 5) reply

The 'stream all' performative being a request for several replies, each 'reply' performative contains one table item.

4' and 5') eos

The gathering agent doesn't know the amount of beliefs that it is receiving. So this predicate indicate the end of the beliefs stream

6) insert

The Belief Revision agent is ordered to insert one knowledge per 'insert' performative.

6') achieve: extract

Being confronted to an unknown number of 'insert' messages, this performative signals the end of the stream and order to extract what has been received.

7) insert

Each 'insert' performative is accompanied by the knowledge to insert.

7') achieve: dispatch

The Gathering Agent is warned that the stream of beliefs to be inserted in its temporary database ends. It is then ordered to dispatch the beliefs to both Agent1 and Agent2.

8 and 9) delete-all

During, all these operations, Agent1 and Agent2 had to carry on their activities with their own non-updated beliefs. Before receiving the common beliefs, they have to delete all their old beliefs related to the particular areas in order that there are no contradictions.

8' and 9') insert

Agent1 and Agent2 receive the common beliefs. Each insert performative being accompanied by one of the common beliefs to be inserted.

8'' and 9'') achieve: ok

Via this performative, Agent1 and Agent2 are warned that the stream of beliefs is finished and that they can carry on their tasks.

It can be noticed that every time the set of beliefs has to be sent, it is sent knowledge after the other in order not to have too large messages on the network.

As mentioned in its description chapter, JACK only offers a lightweight communication mechanism: messages are objects of the class MessageEvent, an extension of the class Event. They are sent by an agent to another known agent. These messages are treated via a plan by the repository as normal events. So, originally, Jack agents don't posses any way of situating a message in a dialog.

For each agent, the use of a finite automate corresponding to each dialog it has with another agent is a solution to this problem. This can easily be implemented via a variable for each dialog. Its content corresponding to the current state of the dialog. When receiving a message, an agent has to check if it is expected in the dialog with its sender. So, the repository can switch the automate to another state by changing the variable content. When sending a message, an agent has also to update its dialog automate.

For an agent, these variables (each variable corresponding to a dialog with a particular agent) are part of its beliefs. So, the set of these variables has to be in one of its database.

This mechanism ought an agent to have several concurrent dialogs at the condition that they don't interfere on the same databases.

This mechanism leads to the fact that agents communication intelligence, the way of it leads a dialog is dependent of the sophistication of its dialog automate. Moreover, in this case, the dialog automate being implemented in the agent code, an agent cannot modify its code by himself in order to better lead a dialog.
A better way of handling dialogs would be to entirely represent them as beliefs.

With what Jack offers, handling of messages is the next one:

Messages are events. If a message is unexpected (i.e. shouldn't appear at this moment in the dialog, or its performative is unknown), its repository has to treat it. (At least to reply that it doesn't understand the message).
So, an agent has to treat any messages it receives. In the JACK Agent Language, a plan can only treat a particular class of event. Trough both these consideration, it can be deduced that different KQML performatives have to be of the same class. Otherwise if an agent receive a message it doesn't know (the event class isn't treatable), it wouldn't be able to reply that it doesn't understand it.
Using JACK, the best way to implement the capability of handling KQML performatives would be to make agents able of treating only one class of event thanks different plans. Each performative is treated via one plan. All these plan treating the same class of event. The checking of the matching plan-performative takes place in the 'context' section of the plan. There is also one plan to treat unknown performatives.
When receiving a message, an agent will try its plan one after the other up to the moment it find the right one dedicated the received performative.

7. Agents descriptions

In this section, the different agents used in the system are described. For more clarity, these descriptions don't reference the codes used at the implementation. The agent's code can be found in the appendix of this work. The agents' architectures are presented via diagrams. Agents are denoted by a bold lined rectangle. Inside this rectangle, databases are denoted by rectangles, plans are denoted by hexagons and messages are denoted by rounded rectangles. Java objects are denoted by trapezoids. The situation of an object being a component of another object is represented by a trapezoid inside a trapezoid. An arrow from a message rectangle to a plan hexagon means a 'is handled by' relation. An arrow from a plan to an event means a 'sends' relation. An arrow leaving a rectangle and heading hexagon means a 'is consulted by' relation. An arrow from a plan to a database denoted a 'modifies' relation.

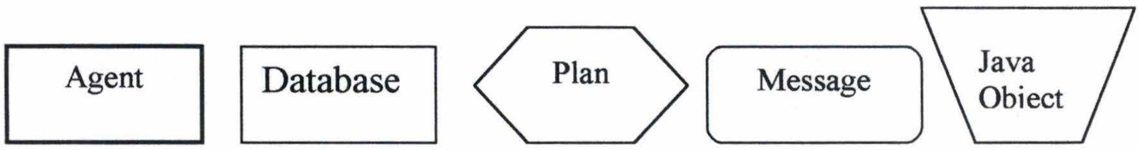

Figure 3.3: Diagrams notation

It can be noticed that in the implementation, each agent is running on its own process, so the system can be spread on different machines.

All agents, when created, send themselves a message to initialise some of their knowledge.

All these agents, except the Interface Agent, are deliberative. They possess a kind of representation of their possible dialogs and adapt their behaviour according to them. The Gathering agent also has the representation of what it has to send to the others. It is clear that at this state of the system their deliberative feature can be dubious.

In order to be able to respond to the sender of a message that is not understood, normally every agent should possess a plan to treat this message. This plan has not been implemented and is not present in the architecture diagrams.

a) The Interface Agent

This agent is reactive. It doesn't possess any symbolic representation of the world and only act by reflex.

Because agents can only communicate with the world via messages, a user cannot directly interact with an agent. So this agent constitutes the interface between the user and the system of agents. It sends the' achieve go' performative when its process is created.

Because of its simplicity, its architecture is not presented.

b) Agent1 and Agent2

Their architectures can be represented as follow:

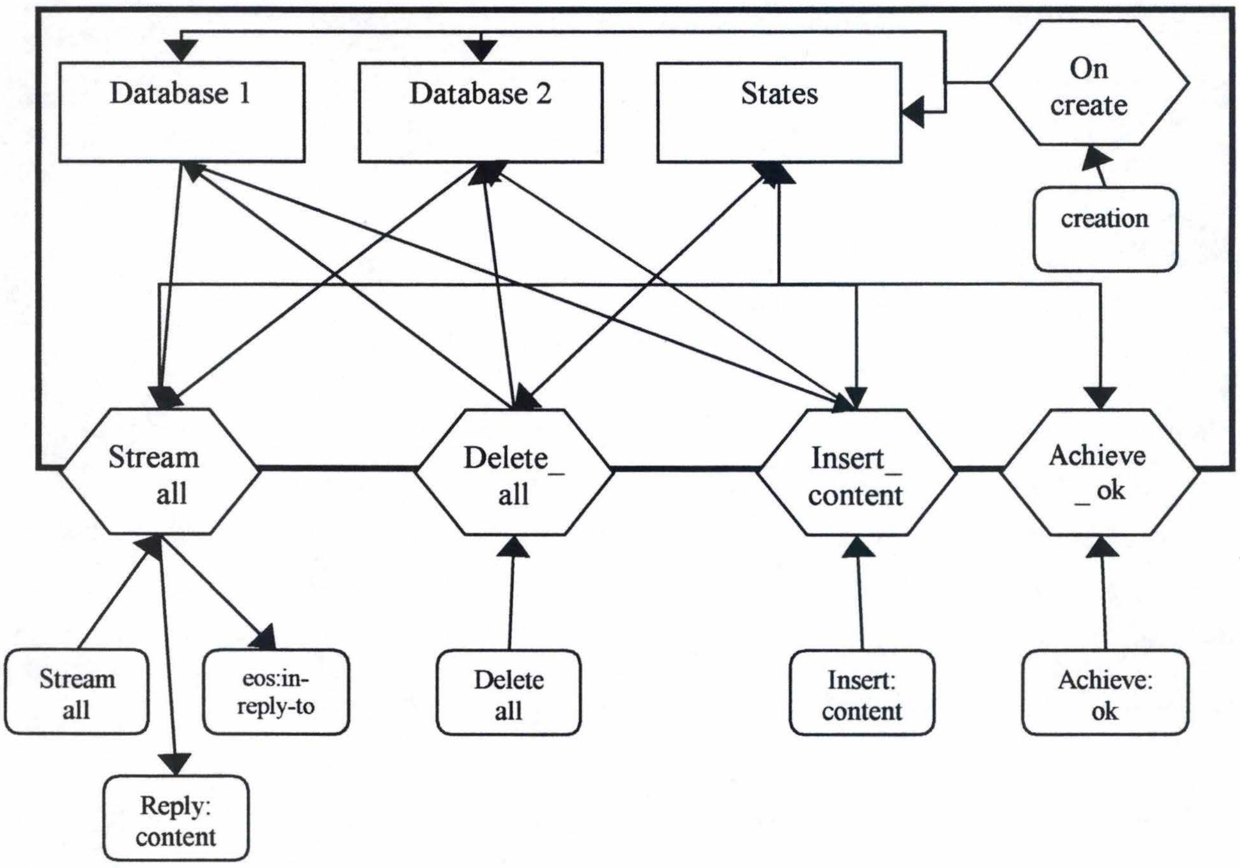

Figure 3.5: Agent1 and Agent2 architecture.

For both agents, knowledge about a particular area that has to be gathered are located in database1 and in database2. Because of the JACK way of representing beliefs, each database is a set of arguments of the same predicate. Each item of a database is accompanied by its ranking. Originally, all rankings are the same for all the items of the same database for each agent. The particularity of these agents, is that, before the gathering of beliefs, they posses different rankings for the same database. For example, Agent1 is specialised in contents of database1 and Agent2 is specialised in contents of database2. Another characteristic is that these databases are semantically linked.

c) The Gathering Agent

Its architecture is the following one:

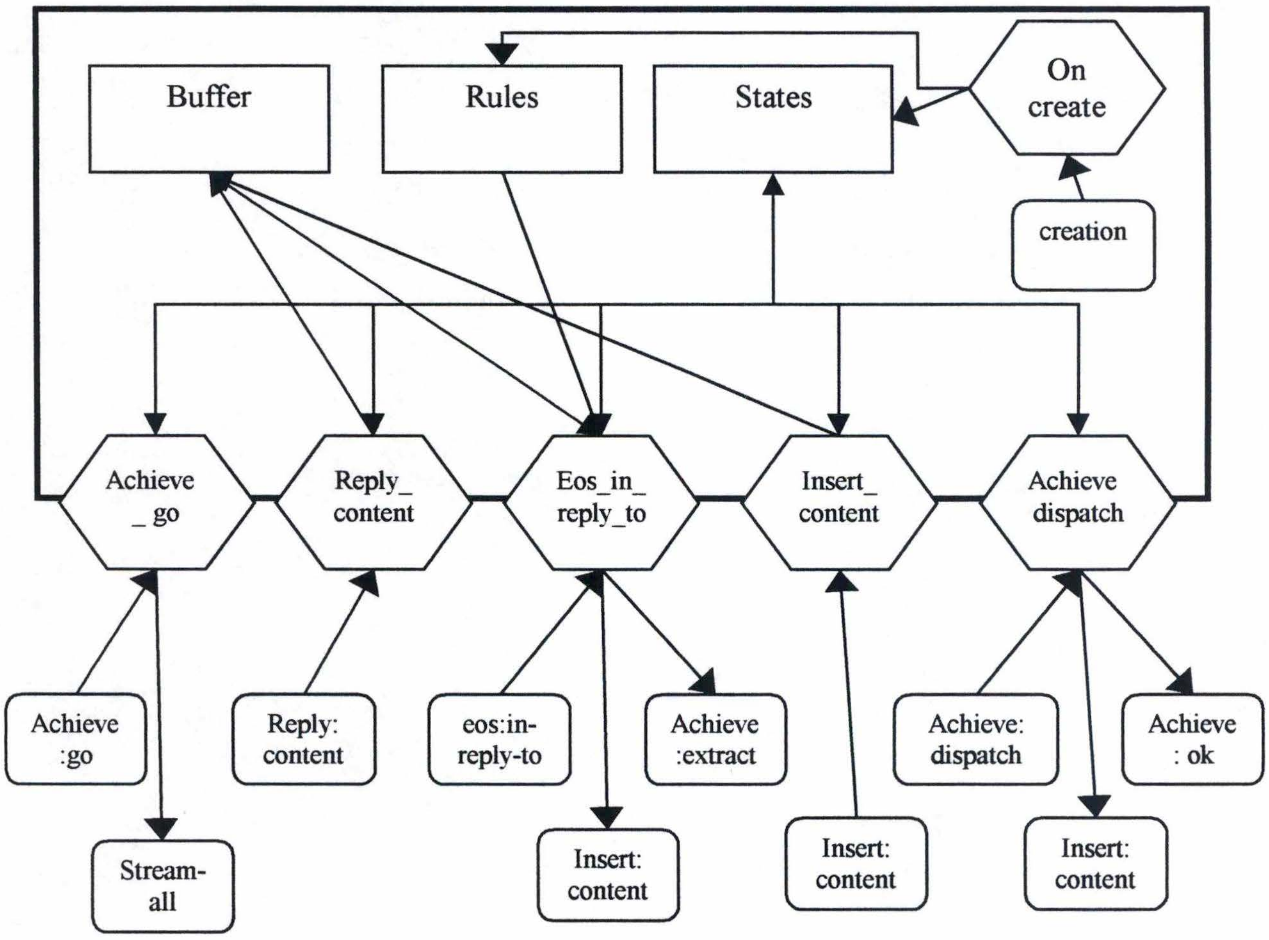

Figure 3.4: Gathering agent architecture

The existence of this agent is justified by modularity requirements. Thanks this agent, the whole system is more flexible and can easily be updated. For example, if a new belief revision agent is more efficient than the existing one, the Gathering agent has just to redirect the messages to this new one. The belief revision agent replacement is made transparently for the user and for Agent1 and Agent2.

This agent explicitly possesses the rules, the integrity constraints that databases to be gathered have to respect. Once received in its temporary database, beliefs from both Agent1 and Agent2, this agent add the rules with the highest ranking and then send the database to the belief revision agent.

Once the set of consistent beliefs is sent back from the Belief Revision Agent, the Gathering Agent removes the rules and dispatches the set of common consistent beliefs to Agent1 and Agent2.

d) Belief Revision Agent

That is this agent that uses parts of the belief revision system SATEN.
So, apart from its communication ability, this agent is able of extracting theories and revising information. Although its hasn't been yet programmed to handle dialogs of revision request.

Its architecture can be described as follow:

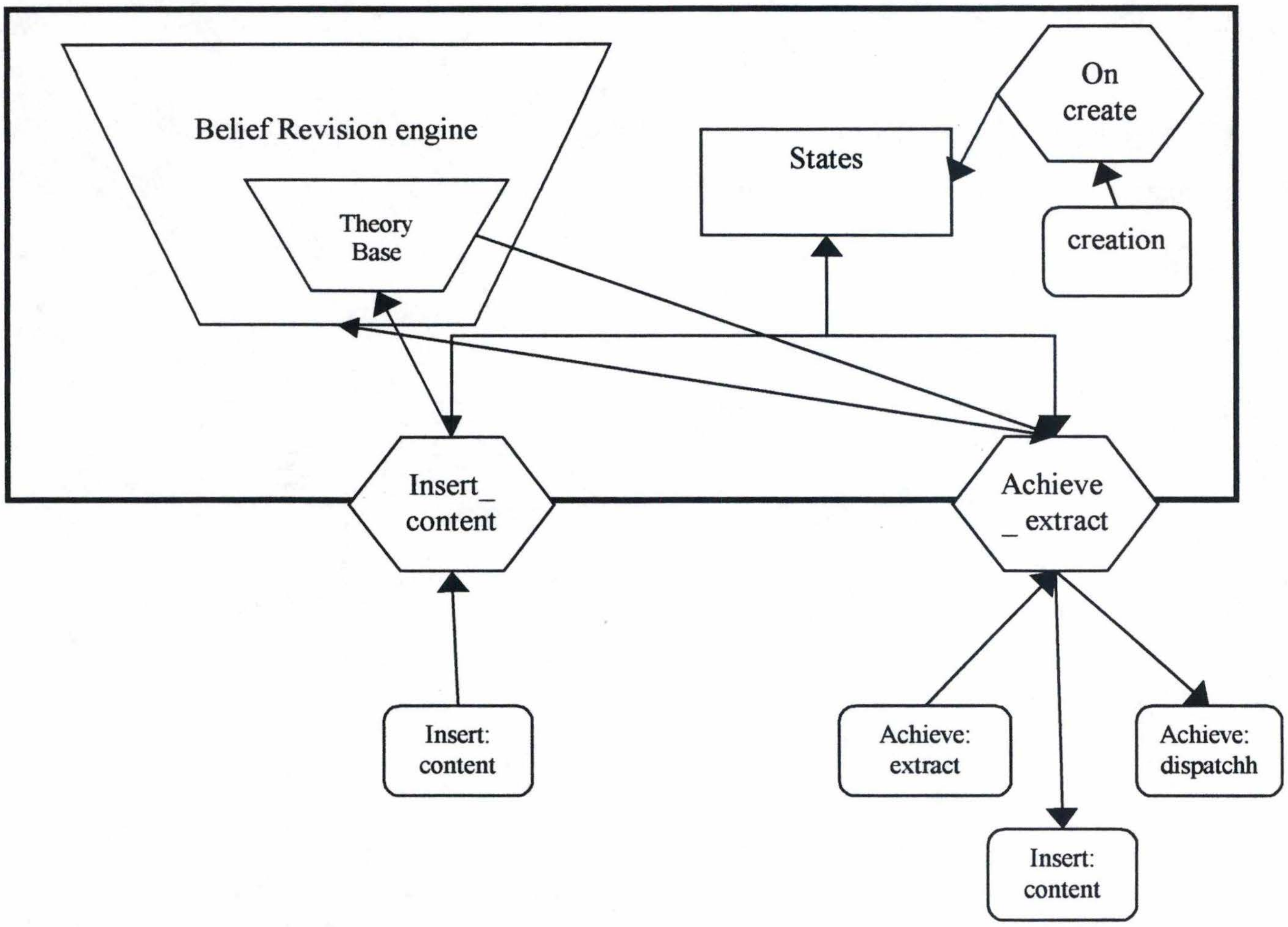

Figure 3.6: Belief Revision agent architecture

This agent extracts the set of beliefs it received from the gathering agent. The result of the extraction is a set of beliefs respecting the integrity constraints.

8. Possible extensions

As mention earlier, the architecture design of this system has been directed by the idea that this agents system has to be seen as a part of a larger system.

A possible extension would be the next one:

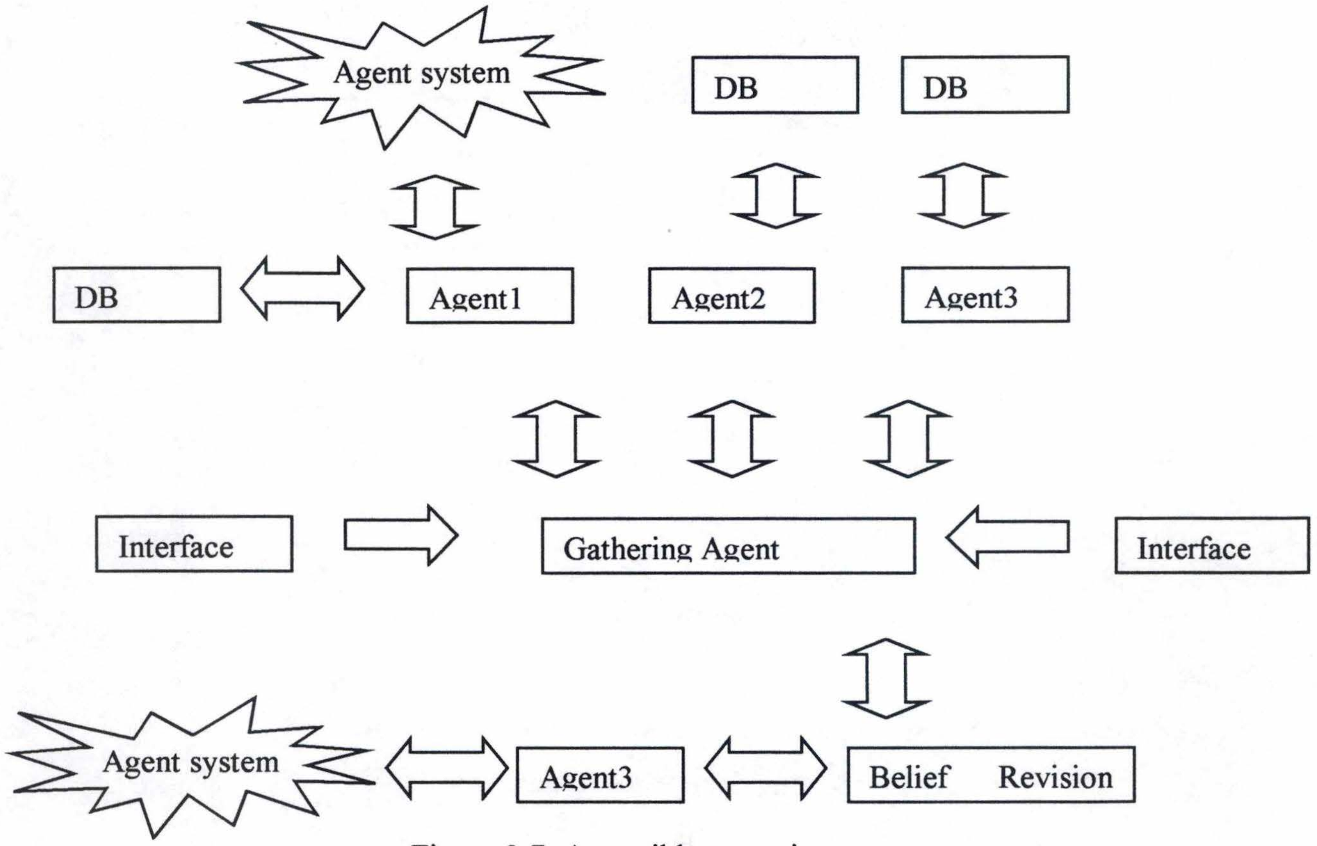

Figure 3.7: A possible extension

As shown in figure 3.7, databases which tables need to be gathered can be linked to the system through new or existing agents foreign agents system could ask belief revision services trough a negotiator agent. Another authorised user could log to the system through his interface agent. Another foreign agents system could request a database through a negotiator agent, which check the accesses.

This extension would rise new problems to be solved.

Among improvements to be made in order to face this new situation:

Negotiator agents should possess sophisticated communication abilities. Agents having many concurrent dialogs should have systems to prevent interference. Agents in contact with foreign systems should have beliefs about authorised accesses.

Conclusion

After having introduced both intelligent agents and Belief Revision, this work presented an approach of using belief revision in an application consisting of a system of agents. It has been shown that the use of agents has been particularly useful in order to have a flexible system. However, the use of belief revision has not been of a great importance. Its usefulness has not been demonstrated.

Several reasons to this fact can be put forward:
First of all, it can be mentioned that the application hasn't been well chosen for this demonstration. The nature of the application doesn't let a great part to the belief revision functionality. Communication among agents has been more underlined.
A second reason that can be linked to the first one is that belief revision is more useful in dynamic situations. In its present state, this application is really static one. Moreover, the use of JACK, a not enough flexible tool used for implementing the system, reinforced this static feature.
A third reason that has also to be linked to the first one is that the belief revision ability is not situated on the good level. In fact, it shouldn't be use as a tool for a system of agents but as an individual tool for agents. Its importance may have been demonstrated if the whole system was considered as a unique agent by agents outside the system.

Nevertheless, as seen in this work, in order to present more software engineering advantages than objects, agents have to communicate. The application demonstrated this communication importance. It also demonstrated the necessity of handling communication as being part of an agent's beliefs. The agents need to have symbolic representations of the handled dialogs.

Maybe the place of belief revision would be there, in communication, in the agent oriented programming paradigm. The first stream of research about intelligent agents would have its place in the macro-issues of agent-oriented software engineering.

Bibliography

[AgentBuilder 2000]

http://www.agentbuilder.com/AgentTools/index.html

[AgentWeb 2000]

http://www.cs.umbc.edu/agents

[Aglets 2000]

http://www.trl.ibm.co.jp/aglets/

[Alchourrón et al., 1985]

Alchourrón C. Gärdenfors P. and Makinson D., *On the logic of theory change: Partial meet contraction functions and their associated revision functions*, Journal of Symbolic Logic 50, pp. 510-530. (1985)

[Bates 1994]

Bates, J, *The Role of Emotion in Believable Characters*. Communications of the ACM (37) 7. S. 122ff. ACM-Press, New York, NY, USA (1994).

[Britannica 2000]

http://www.britannica.com

[Chavez et al.1997]

Chavez, A; Dreilinger, D.; Guttman, R.; Maes, P., *A Real-Life Experiment in Creating an Agent Marketplace* in *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97)*. London, UK (1997).

[Cheong 1996]

Cheong, F.-C., *Internet Agents: Spiders, Wanderers, Brokers, and Bots*. New Riders Publishing, Indianapolis, IN, USA (1996).

[Chess 95]

Chess D., Grosof B., Harrison C., Levine D., *Itinerant Agents for Mobile Computing*, Number RC 20010, IBM T.J. Watson Research Center, February 1995.

[Concordia 2000]
http://www.meitca.com/HSL/Projects/Concordia

[Dragan 1997]

Dragan, R.V., *Looking Forward: Agent Software*. PC Magazine 25th of .March (1997).
http://www.pcmag.com/special/anniversary/forward/lfr6.htm

[Extempo 2000]

http://www.extempo.com.

[Finin et al. 1994]

Finin, T.; Fritzson, R.; McKay, D.; McEntire, R., *KQML as an agent communication language* in Proceedings of the 4rd International Conference on Information and Knowledge Management (CIKM). ACM Press, New York, NY, USA (1994).

[Foner 1993]

Foner, L.:*What's an Agent, Anyway? A Sociological Case Study*. Agents Memo 93-01, MIT Media Lab, Cambridge, MA, USA (1993).

[Genesereth, Ketchpel 1994]

Genesereth, M. R.; Ketchpel, S. R., *Software Agents*. Communications of the ACM (37) 7. S. 48ff. ACM-Press, New York, NY, USA (1994).

[Gilbert et al.95]

Gilbert, D.; Aparicio, M.; Atkinson, B.; Brady, S.; Ciccarino, J.; Grosof, B.; O'Connor, P.,Osisek, D.; Pritko, S.; Spagna, R.; and Wilson, L., *IBM Intelligent Agent Strategy*,
IBM Corporation. (1995)

[Grasshopper 2000]
http://www.ikv.de/products/grasshopper/

[Green et al. 1997]

Green, S.; Hurst, L.; Nangle, B.; Cunningham, P.; Somers, F.; Evans, R., *Software Agents: a review*. Report of the Intelligent Agent Group am Trinity College Dublin, IE (1997).

[IBM 1995]

The Role of Intelligent Agents in the Information Infrastructure. IBM Corporation Research Triangle Park, NC, USA (1995).

[Jack 1999]

Busetta P., Rönnquist R., Hodgson A., Lucas A., *JACK Intelligent Agents- Components for Intelligent Agents in Java*, Agent Oriented Software Pty Ltd. (1999)

[Jack 2000]

Coburn M., *Jack Intelligent Agent User Guide*, Agent Oriented Software Pty Ltd. (2000)

[Jennings, Woolridge 1996]

Jennings, N.; Woolridge, M., *Software Agents* in IEEE Review (42) 1. S.17ff. Los Alamitos, CA, USA (1996).

[Jennings, Wooldridge 98]

Jennings N.R., Wooldridge M., *Agent-Oriented Software Engineering*.(1998)

[Johansen 1997a]

Johansen, D., *The Tacoma Project*. University of Tromso, NO und Cornell University, Ithaca, NY, USA (1997).

[Johansen 1997b]

Johansen, D., *The StormCast Project*. University of Tromso, NO (1997).

[Lange 1997]

Lange, D. B., *Java Aglet Application Programming Interface (J-AAPI) White Paper - Draft 2*. IBM Tokyo Research Laboratory, JP (1997).

[Lange, Aridor 1997]

Lange, D. B.; Aridor, Y., *Agent Transfer Protocol -- ATP/0.1*. IBM Tokyo Research Laboratory, JP (1997).

[Lingnau, Drobnik, Dömel 1995]

Lingnau, A.; Drobnik, O.; Dömel, P.: *An HTTP-based Infrastructure for Mobile Agents*. World Wide Web Journal – Fourth International World Wide Web Conference Proceedings, Boston, MA, USA (1995)

[Maes 1991]

Maes, P. (Hrsg.), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. MIT-Press, London (1991).

[Maes 1994]

Maes, P., *Agents that reduce work and information overload* in Communications of the ACM (37) 7. S.30ff. ACM-Press, New York, NY, USA (1994).

[Maes 1995]

Maes, P., *Artificial Life Meets Entertainment: Life like Autonomous Agents* in Communications of the ACM (38) 11. S. 108ff. ACM Press, New York, USA (1995).

[Mitchel et al. 1994]

Mitchel, T.; Cartiana, R.; Freitag, D.; McDermott, J.; Zabowski, D. *Experience with a Learning Personal Assistant* in Communications of the ACM 37 (7). S. 81ff. ACM-Press, New York, NY, USA (1994).

[Neches 1994]

Neches, R., *Overview of the DARPA Knowledge Sharing Effort*. (1994).

[Nwana 1996]

Nwana, H. S., *Software Agents: An Overview*. Knowledge Engineering Review (11) 3. S. 205ff. Cambridge University Press (1996).

[Nwana, Ndumu 1997]

Nwana, H. S.; Ndumu, D. T., "An Introduction to Agent Technology" (1997)

[Nwana, Lee, Jennings 1997]

Nwana, H.S.; Lee, L.; Jennings, N. R., *Co-ordination in Multi-Agent Systems* (1997).

[Nwana, Woolridge 1997]

Nwana, H.S.; Woolridge, M., *Software Agent Technologies* (1997).

[Oz 2000]

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/oz.html

[Pearson 1996]

Pearson, S., *An Overview of Agent Technology*, HP Technical Paper HPL-96-40. Hewlett-Packard Company, Palo Alto, CA, USA (1996).

[Petrie 1996]

Petrie, C. J., *Agent-Based Engineering, the Web, and Intelligence* in IEEE Expert (11) 6. S.24ff. Los Alamitos, CA, USA (1996).

[Pleiades 1996]

http://www.cs.cmu.edu/~softagents/pleiades/index.html

[Russell, Norvig, 1995]

Russell, Stuart J. and Norvig P., *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall (1995)

[SATEN 97]
http://infosystems.newcastle.edu.au/webworld/saten/

[The agent society 2000]

http://www.agent.org/

[VADER 97]

http://infosystems.newcastle.edu.au/webworld/vader/

[Wayner, Joch 1995]

Wayner, P.; Joch, A., *Agents of Change* in Byte (20) 3. McGraw-Hill, New-York, USA (1995).

[White 1996]

White, J., *Mobile Agents White Paper*. General Magic, Sunnyvale, CA, USA (1996).

[Williams, 1997]
Williams M-A., *Belief Revision* in *Non-monotonic Reasoning*, G. Antoniou ,The MIT Press. (1997)

[Williams, 1998]
Williams M-A., *Applications of Belief Revision*. (1998)

[Woolridge, Jennings 1995]

Woolridge, M. J.; Jennings, N. R., *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review 10 (2). Cambridge University Press, USA (1995).

Appendix

Source Code

Interface Agent

Interface_agent.agent

```
import      aos.jack.jak.core.*;
import      java.io.*;


agent interface_agent extends Agent {


     #sends event messaj mes;
     #handles event ecrire;
     #uses plan ecriture;

     public interface_agent(String n)     {
      super(n);

     }


     public void launch()
     {
      send("gathery@Carlton",mes.messaj("lancement"));

     }


}
```

messaj.event

```
event messaj extends MessageEvent {

    public String content;


    #posted as
    messaj(String what)
      {
       content=what;

      }
}
```

Gathering Agent

Gathering_agent.agent

```
import      aos.jack.jak.core.*;
import      java.io.*;


agent gathering_agent extends Agent {

     #private database states st();
     #private database buffer buf();
     #private database rules ru();

     #handles event messaj;
     #uses plan achieve_go;
     #uses plan reply_content;
     #uses plan insert_content;
     #uses plan eos_in_reply_to;
     #uses plan achieve_dispatch;

     #posts event oncreate oc;
     #handles event oncreate;
     #uses plan creation;


     public gathering_agent(String n)
     {
      super(n);
      postEvent(oc.oncreate());

     }


}
```

Buffer.db

```
database buffer extends OpenWorld {
   // the db where the gathering agent stock formulas+ranking
   // before sending it to the extractor

    #key field String formula;

    #indexed query get_any(logical String f);
    #indexed query get_element(String e);

}
```

states.db

```
database states extends ClosedWorld {


    #key field String name;
    #value field String state;
    #indexed query get_ifstate(String n,String s);
    #indexed query get_state(String n,logical String s);


}
```

rules.db

```
database rules extends OpenWorld {
  // rules to use, with their rank=0.99

    #key field String rule;

    #indexed query get_any(logical String r);


}
```

creation.plan

```
import java.lang.System;

plan creation extends Plan {

    #handles event oncreate oc;
    #modifies database states st;
    #modifies database rules ru;

    body()
    {
     st.assert("interfacia@Melbourne","ready",Cursor.TRUE);
     st.assert("gathery@Carlton","ready",Cursor.TRUE);
     // a revoir car y  manque init. de agency1


     ru.assert("*X(*V(*W((-Equal(V,W)&Isaresidentof(X,V))->
     -Isaresidentof(X,W))))&(-Equal(wavre,namur)&
     -Equal(namur,wavre)&Equal(namur,namur)&Equal(wavre,wavre)),0.999");
     ru.assert("*A(*B(!C(Ismarriedwith(A,B)
     ->(Isaresidentof(A,C)&Isaresidentof(B,C))))),0.998");

    }
}
```

achieve_go.plan

```
import java.lang.System;

plan achieve_go extends Plan {

    #handles event messaj mes;
    #modifies database states st;

    context()
    {
     mes.contenu.startsWith("achieve_go") && st.get_ifstate(mes.from,"ready");
    }

    body()
    {

    System.out.println(""+mes.from + " traitement de lancement");

    st.assert("agency1@Hamilton","waiting_stream",Cursor.TRUE);
    @send("agency1@Hamilton",mes.messaj("stream-all:content ismarriedwith,*
    isaresidentof,*"));

    st.assert("agency2@Agency2land","waiting_stream",Cursor.TRUE);
    @send("agency2@Agency2land",mes.messaj("stream-all:content ismarriedwith,*
    isaresidentof,*"));

    }
}
```

insert_content.plan

```
import java.lang.System;
import java.lang.Double;

plan insert_content extends Plan {

    #handles event messaj mes;

    #modifies database states st;
    #modifies database buffer buf;

    context()
    {
     mes.contenu.startsWith("insert:content");
    }

    body()
    {
     String formula=mes.contenu.substring(mes.contenu.indexOf(" ")+1);
     System.out.println("ai recu formula: "+formula);
     if (formula.startsWith("Ismarriedwith")|
     formula.startsWith("Isaresidentof"))
     buf.assert(formula);
    }
}
```

reply_content.plan

```
import java.lang.System;
import java.lang.Double;

plan reply_content extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database buffer buf;

    context()
    {
     mes.contenu.startsWith("reply:content")
     && st.get_ifstate(mes.from,"waiting_stream")   ;
    }

    body()
    {
     String dbname=mes.contenu.substring(14,mes.contenu.indexOf(","));
     System.out.println(""+dbname);
     String content=dbname+"("+mes.contenu.substring
     (mes.contenu.indexOf(",")+1,mes.contenu.lastIndexOf(","))+")";
     double rank=new
     Double(mes.contenu.substring(mes.contenu.lastIndexOf(",")+1))
     .doubleValue();
     buf.assert(""+content+","+rank,Cursor.TRUE);
     }


}
```

oncreate.event

```
event oncreate extends Event {

    #posted as
    oncreate()
      {

      }
}
```

eos_in_reply_to.plan

```
import java.lang.System;

plan eos_in_reply_to extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database buffer buf;
    #modifies database rules ru;

    context()
    {
     mes.contenu.startsWith("eos:in-reply-to")
     && st.get_ifstate(mes.from,"waiting_stream");
    }

    body()
    {
     st.assert(mes.from,"ready",Cursor.TRUE);
     logical String value;
     st.get_state("gathery@Carlton",value);
     if (value.as_string().equals("ready"))
       {
       st.assert("gathery@Carlton","received1",Cursor.TRUE);
       }

     if (value.as_string().equals("received1"))
       {
        st.assert("gathery@Carlton","received2",Cursor.TRUE);
        int limite=buf.nFacts();
        for (int i=1;i<=limite;i++)
         {
         logical String f;
         buf.get_any(f);
         @send("extracty@Extractland",mes.messaj("insert:content
         "+f.as_string()+""));
         buf.get_element(f.as_string()).retractAll();
         }

        logical String r;
        for (Cursor c=ru.get_any(r);c.next(); )
         {
          @send("extracty@Extractland",mes.messaj("insert:content
          "+r.as_string()+""));
         }
        @send("extracty@Extractland",mes.messaj("achieve:extract"));
       }
    }

}
```

achieve_dispatch.plan

```
import java.lang.System;

plan achieve_dispatch extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database buffer buf;

    context()
    {
     mes.contenu.startsWith("achieve:dispatch");
    }


    body()
    {
     @send("agency1@Hamilton",mes.messaj("delete-all:content
     Ismarriedwith,*"));
     @send("agency1@Hamilton",mes.messaj("delete-all:content
     Isaresidentof,*"));
     @send("agency2@Agency2land",mes.messaj("delete-all:content
     Ismarriedwith,*"));
     @send("agency2@Agency2land",mes.messaj("delete-all:content
     Isaresidentof,*"));
     int limite=buf.nFacts();
     for (int i=1;i<=limite;i++)
        {
        logical String f;
        buf.get_any(f);
        String dbname=f.as_string().substring(0,13);
        String arg=f.as_string().substring(14,f.as_string().indexOf(")"));
        String ranking=f.as_string().
        substring(f.as_string().indexOf(")")+2);
        System.out.println("J'envoie: "+dbname+","+arg+","+ranking);
        @send("agency1@Hamilton",mes.messaj("insert:content
        "+dbname+","+arg+","+ranking));
        @send("agency2@Agency2land",mes.messaj("insert:content
        "+dbname+","+arg+","+ranking));

        buf.get_element(f.as_string()).retractAll();
        }

    }

}
```

messaj.event

```
event messaj extends MessageEvent {

    public String content;


    #posted as
    messaj(String what)
      {
       content=what;

      }
}
```

Belief Revision agent

extracteur.agent

```
import      aos.jack.jak.core.*;
import      java.io.*;

agent extracteur extends Agent {

     #private database states st();
     #handles event messaj;
     #uses plan insert_content;
     #uses plan achieve_extract;
     #posts event oncreate oc;
     #handles event oncreate;
     #uses plan creation;
     #sends event messaj;

    public static TheoryBase formulas= new TheoryBase();
    public static MaxiAdj reviseur= new MaxiAdj(formulas);

     public extracteur(String n)
     {
      super(n);
      postEvent(oc.oncreate());
     }
}
```

states.db

```
database states extends OpenWorld {

    #key field String name;
    #value field String state;
    #indexed query get_ifstate(int i, String s);


}
```

messaj.event

```
event messaj extends MessageEvent {

    public String content;


    #posted as
    messaj(String what)
      {
       content=what;

      }
}
```

oncreate.event

```
event oncreate extends Event {

    #posted as
    oncreate()
      {

      }
}
```

creation.plan

```
import java.lang.System;

plan creation extends Plan {

    #handles event oncreate oc;
    #modifies database states st;

    body()
    {
     st.assert("gathery@Carlton","waiting_insert",Cursor.TRUE);
    }

}
```

achieve_extract.plan

```
import java.lang.System;
import java.lang.Double;

plan achieve_extract extends Plan {

    #handles event messaj mes;
    #modifies database states st;

    context()
    {
     mes.contenu.startsWith("achieve:extract")
     && st.get_ifstate(mes.from,"waiting_insert")   ;
    }

    body()
    {
     extracteur.reviseur.doExtract();
     System.out.println("Resultat ");
     System.out.println(""+extracteur.reviseur.base.toString());
     int limite=extracteur.reviseur.base.beliefs.mod();
     for (int i=0;i<limite;i++)
      {
          String formula=extracteur.reviseur.base.beliefs.contents[i];
          double ranking=extracteur.reviseur.base.rankings.contents[i];
          @send("gathery@Carlton",mes.messaj("insert:content
          "+formula+","+ranking));

      }
     @send("gathery@Carlton",mes.messaj("achieve:dispatch"));
     extracteur.reviseur.base=new TheoryBase();

    }

}
```

insert_content.plan

```
import java.lang.System;
import java.lang.Double;

plan insert_content extends Plan {

    #handles event messaj mes;
    #modifies database states st;

    context()
    {
     mes.contenu.startsWith("insert:content");
    }
     body()
    {
     String formula=mes.contenu.substring(mes.contenu.indexOf("
     ")+1,mes.contenu.lastIndexOf(","));
     double rank=new Double(mes.contenu.substring(mes.contenu
     .lastIndexOf(",")+1)).doubleValue();
      extracteur.reviseur.base.addBelief(formula,rank);
     }
}
```

Agent1

agency_agent.agent

```
import      aos.jack.jak.core.*;
import      java.io.*;

agent agency_agent extends Agent {

     #private database isaresidentof resi();
     #private database ismarriedwith mari();
     #private database states st();
     #handles event messaj;
     #uses plan stream_all_content;
     #uses plan delete_all_content;
     #uses plan insert_content;
     #posts event oncreate oc;
     #handles event oncreate;
     #uses plan creation;

     public agency_agent(String n)
     {
      super(n);
      postEvent(oc.oncreate());
     }
}
```

states.db

```
database states extends OpenWorld {

    #key field String name;
    #value field String state;
    #indexed query get_ifstate(int i, String s);

}
```

isaresidentof.db

```
database isaresidentof extends OpenWorld {
    #key field String name;
    #value field String city_name;
    #value field double ranking;
    #indexed query
    get_any(logical String n,logical String c,logical double r);
    #indexed query get_element(String n,String c, double r);

}
```

ismarriedwith.db

```
database ismarriedwith extends OpenWorld {
    #key field String name_man;
    #key field String name_woman;
    #value field double ranking;
    #indexed query
    get_any(logical String m,logical String w,logical double r);
    #indexed query get_element(String m,String w,double r);
}
```

messaj.event

```
event messaj extends MessageEvent {

    public String content;


    #posted as
    messaj(String what)
      {
       content=what;

      }
}
```

oncreate.event

```
event oncreate extends Event {

    #posted as
    oncreate()
      {

      }
}
```

creation.plan

```
import java.lang.System;

plan creation extends Plan {

    #handles event oncreate oc;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;

    body()
    {
     st.assert("gathery@Carlton","ready",Cursor.TRUE);
     resi.assert("jacqueline","wavre",0.7);
     resi.assert("mireille","namur",0.7);
    }
}
```

stream_all_content.plan

```
import java.lang.System;

plan stream_all_content extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #reads database isaresidentof resi;
    #reads database ismarriedwith mari;

    context()
    {
     mes.contenu.startsWith("stream-all:content") &&
     st.get_ifstate(mes.from,"ready");
    }

    body()
    {
     st.assert(mes.from,"preparation_stream_all",Cursor.TRUE);
     String dbname=mes.contenu.substring(19,mes.contenu.indexOf(","));
     dbname="Isaresidentof";
     String nom1="Isaresidentof";
     if (dbname.equals(nom1))
        {
         int dbnfacts=resi.nFacts();
         logical String n;
         logical String v;
         logical double r;

         for (Cursor c=resi.get_any(n,v,r);c.next(); )
          {
           @send(mes.from,mes.messaj("reply:content
           "+dbname+","+n+","+v+","+r));
           System.out.println("je renvoie "+"reply:content
           "+dbname+","+n+","+v+","+r);
           }
          }
      String dbname="Ismarriedwith";
      String nom2="Ismarriedwith";
      if (dbname.equals(nom2))
        {
         int dbnfacts=mari.nFacts();
         logical String m;
         logical String w;
         logical double r;
         for (Cursor c=mari.get_any(m,w,r);c.next(); )
           {
            @send(mes.from,mes.messaj("reply:content
            "+dbname+","+m+","+w+","+r));
            System.out.println("je renvoie "+"reply:content
            "+dbname+","+m+","+w+","+r);
           }
          @send(mes.from,mes.messaj("eos:in-reply-to"));
          st.assert(mes.from,"ready",Cursor.TRUE);
        }
     }
}
```

delete_all_content.plan

```
import java.lang.System;

plan delete_all_content  extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;


    context()
    {
     mes.contenu.startsWith("delete-all:content");// &&
     st.get_ifstate(mes.from,"ready");
    }

    body()
    {
    String dbname=mes.contenu.substring(19,mes.contenu.indexOf(","));
    String nom1="Isaresidentof";
    if (dbname.equals(nom1))
        {
        int limite=resi.nFacts();
        for (int i=1;i<=limite;i++)
         {
         logical String n;
         logical String c;
         logical double r;
         resi.get_any(n,c,r);
         resi.get_element(n.as_string(),c.as_string(),r.as_double())
         .retractAll();
         }

        }

      String nom2="Ismarriedwith";
      if (dbname.equals(nom2))
        {
         int limite=mari.nFacts();
         for (int i=1;i<=limite;i++)
          {
           logical String m;
           logical String w;
           logical double r;
           mari.get_any(m,w,r);
           mari.get_element(m.as_string(),w.as_string(),r.as_double())
           .retractAll();
          }
        }

    }
}
```

insert_content.plan

```
import java.lang.System;
import java.lang.Double;

plan insert_content extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;


    context()
    {
     mes.contenu.startsWith("insert:content");
     && st.get_ifstate(mes.from,"waiting_insert")   ;
    }


    body()
    {
     String dbname=mes.contenu.substring(15,mes.contenu.indexOf(","));
     String first=mes.contenu.substring(mes.contenu.indexOf(",")+1
     ,mes.contenu.indexOf(",",mes.contenu.indexOf(",")+1));
     String second=mes.contenu.substring(mes.contenu.indexOf
     (",",mes.contenu.indexOf(",")+1)+1,mes.contenu.lastIndexOf(","));
     Double temp=new Double(mes.contenu.substring
     (mes.contenu.lastIndexOf(",")+1));
     double ranking=temp.doubleValue();
     if (dbname=="Ismarriedwith")
        mari.assert(first,second,ranking);
     if (dbname=="Ismarriedwith")
        mari.assert(first,second,ranking);
    }


}
```

delete_all_content.plan

```
import java.lang.System;

plan delete_all_content  extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;

    context()
    {
     mes.contenu.startsWith("delete-all:content");// &&
     st.get_ifstate(mes.from,"ready");
    }

    body()
    {
    String dbname=mes.contenu.substring(19,mes.contenu.indexOf(","));
    String nom1="Isaresidentof";
    if (dbname.equals(nom1))
        {
        int limite=resi.nFacts();
        for (int i=1;i<=limite;i++)
         {
         logical String n;
         logical String c;
         logical double r;
         resi.get_any(n,c,r);
         resi.get_element(n.as_string(),c.as_string(),r.as_double())
         .retractAll();
         }
        }

      String nom2="Ismarriedwith";
      if (dbname.equals(nom2))
        {
         int limite=mari.nFacts();
         for (int i=1;i<=limite;i++)
          {
           logical String m;
           logical String w;
           logical double r;
           mari.get_any(m,w,r);
           mari.get_element(m.as_string(),w.as_string(),r.as_double())
           .retractAll();
          }
        }
    }
}
```

Agent2

agency_agent.agent

```
import      aos.jack.jak.core.*;
import      java.io.*;

agent agency_agent extends Agent {

     #private database isaresidentof resi();
     #private database ismarriedwith mari();
     #private database states st();
     #handles event messaj;
     #uses plan stream_all_content;
     #uses plan delete_all_content;
     #uses plan insert_content;
     #posts event oncreate oc;
     #handles event oncreate;
     #uses plan creation;

     public agency_agent(String n)
     {
      super(n);
      postEvent(oc.oncreate());
     }
}
```

states.db

```
database states extends OpenWorld {

    #key field String name;
    #value field String state;
    #indexed query get_ifstate(int i, String s);

}
```

isaresidentof.db

```
database isaresidentof extends OpenWorld {
    #key field String name;
    #value field String city_name;
    #value field double ranking;
    #indexed query
    get_any(logical String n,logical String c,logical double r);
    #indexed query get_element(String n,String c, double r);

}
```

ismarriedwith.db

```
database ismarriedwith extends OpenWorld {
    #key field String name_man;
    #key field String name_woman;
    #value field double ranking;
    #indexed query
    get_any(logical String m,logical String w,logical double r);
    #indexed query get_element(String m,String w,double r);
}
```

messaj.event

```
event messaj extends MessageEvent {

    public String content;


    #posted as
    messaj(String what)
      {
       content=what;

      }
}
```

oncreate.event

```
event oncreate extends Event {

    #posted as
    oncreate()
      {

      }

}
```

creation.plan

```
import java.lang.System;


plan creation extends Plan {

    #handles event oncreate oc;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;

    body()
    {
     st.assert("gathery@Carlton","ready",Cursor.TRUE);
     mari.assert("jacqueline","georges",0.7);
     resi.assert("georges","namur",0.4);
     resi.assert("mireille","wavre",0.4);
    }
}
```

stream_all_content.plan

```
import java.lang.System;

plan stream_all_content extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #reads database isaresidentof resi;
    #reads database ismarriedwith mari;

    context()
    {
     mes.contenu.startsWith("stream-all:content") &&
     st.get_ifstate(mes.from,"ready");
    }

    body()
    {
     st.assert(mes.from,"preparation_stream_all",Cursor.TRUE);
     String dbname=mes.contenu.substring(19,mes.contenu.indexOf(","));
     dbname="Isaresidentof";
     String nom1="Isaresidentof";
     if (dbname.equals(nom1))
        {
         int dbnfacts=resi.nFacts();
         logical String n;
         logical String v;
         logical double r;

         for (Cursor c=resi.get_any(n,v,r);c.next(); )
         {
          @send(mes.from,mes.messaj("reply:content
          "+dbname+","+n+","+v+","+r));
          System.out.println("je renvoie "+"reply:content
          "+dbname+","+n+","+v+","+r);
          }
         }
       String dbname="Ismarriedwith";
       String nom2="Ismarriedwith";
       if (dbname.equals(nom2))
        {
         int dbnfacts=mari.nFacts();
         logical String m;
         logical String w;
         logical double r;
         for (Cursor c=mari.get_any(m,w,r);c.next(); )
          {
           @send(mes.from,mes.messaj("reply:content
           "+dbname+","+m+","+w+","+r));
           System.out.println("je renvoie "+"reply:content
           "+dbname+","+m+","+w+","+r);
          }
         @send(mes.from,mes.messaj("eos:in-reply-to"));
         st.assert(mes.from,"ready",Cursor.TRUE);
        }
      }
}
```

delete_all_content.plan

```
import java.lang.System;

plan delete_all_content  extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;


    context()
    {
     mes.contenu.startsWith("delete-all:content");// &&
     st.get_ifstate(mes.from,"ready");
    }

    body()
    {
    String dbname=mes.contenu.substring(19,mes.contenu.indexOf(","));
    String nom1="Isaresidentof";
    if (dbname.equals(nom1))
        {
        int limite=resi.nFacts();
        for (int i=1;i<=limite;i++)
         {
         logical String n;
         logical String c;
         logical double r;
         resi.get_any(n,c,r);
         resi.get_element(n.as_string(),c.as_string(),r.as_double())
         .retractAll();
         }

        }

      String nom2="Ismarriedwith";
      if (dbname.equals(nom2))
        {
         int limite=mari.nFacts();
         for (int i=1;i<=limite;i++)
          {
           logical String m;
           logical String w;
           logical double r;
           mari.get_any(m,w,r);
           mari.get_element(m.as_string(),w.as_string(),r.as_double())
           .retractAll();
          }
        }

    }
}
```

insert_content.plan

```
import java.lang.System;
import java.lang.Double;

plan insert_content extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;


    context()
    {
     mes.contenu.startsWith("insert:content");
     && st.get_ifstate(mes.from,"waiting_insert")   ;
    }


    body()
    {

     String dbname=mes.contenu.substring(15,mes.contenu.indexOf(","));
     String first=mes.contenu.substring(mes.contenu.indexOf(",")+1
     ,mes.contenu.indexOf(",",mes.contenu.indexOf(",")+1));
     String second=mes.contenu.substring(mes.contenu.indexOf
     (",",mes.contenu.indexOf(",")+1)+1,mes.contenu.lastIndexOf(","));
     Double temp=new Double(mes.contenu.substring
     (mes.contenu.lastIndexOf(",")+1));
     double ranking=temp.doubleValue();
     if (dbname=="Ismarriedwith")
        mari.assert(first,second,ranking);
     if (dbname=="Ismarriedwith")
        mari.assert(first,second,ranking);

    }

}
```

delete_all_content.plan

```
import java.lang.System;

plan delete_all_content  extends Plan {

    #handles event messaj mes;
    #modifies database states st;
    #modifies database isaresidentof resi;
    #modifies database ismarriedwith mari;

    context()
    {
     mes.contenu.startsWith("delete-all:content");// &&
     st.get_ifstate(mes.from,"ready");
    }

    body()
    {
    String dbname=mes.contenu.substring(19,mes.contenu.indexOf(","));
    String nom1="Isaresidentof";
    if (dbname.equals(nom1))
        {
        int limite=resi.nFacts();
        for (int i=1;i<=limite;i++)
         {
         logical String n;
         logical String c;
         logical double r;
         resi.get_any(n,c,r);
         resi.get_element(n.as_string(),c.as_string(),r.as_double())
         .retractAll();
         }
        }

      String nom2="Ismarriedwith";
      if (dbname.equals(nom2))
        {
         int limite=mari.nFacts();
         for (int i=1;i<=limite;i++)
          {
           logical String m;
           logical String w;
           logical double r;
           mari.get_any(m,w,r);
           mari.get_element(m.as_string(),w.as_string(),r.as_double())
           .retractAll();
           System.out.println("nbr faits ds boucle "+mari.nFacts());
          }
        }

    }
}
```