Vrije Universiteit Brussel – Belgium

Faculty of Sciences
In Collaboration with Ecole des Mines de Nantes – France
and
Monash University – School of Computer Science and Software Engineering – Australia
2002

An Agent-Based Platform for Assisting Repository Navigation and Administration (Part of the LEOPARD Project)

A Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
(Thesis research conducted in Australia in the EMOOSE exchange)

By: Olivier Constant

Promoter: Prof. Theo D'Hondt (Vrije Universiteit Brussel)
Co-Promoter: Dr. Annya Réquilé (Monash University)

Abstract

Repositories can be hard to navigate and administrate. Standard static access techniques like indexing have limited power and flexibility and their maintenance requires appropriate feedback. Users, administrators and content providers are all concerned by this issue. It has become all the more critical as the successful emergence of learning objects leads to the creation of very large and complex repositories.

We propose an approach strictly based on the dynamic collection of users' navigation pathways for allowing for the effective use of large repositories. The computation of pathways allows for the inference of virtual semantic links between resources of the repository. Hence users can be assisted in their navigation dynamically based on pathways from their personal profile but also from the whole community of users including experts. In addition, pathways are a potential source of feedback for repository administrators and content providers. Possibilities include the discovery of communities of users and categories of content.

A prototype has been designed as a multiagent system. To allow for its implementation, an agent infrastructure and agent framework have been developed for MS .Net. The prototype is an experimental platform that demonstrates how pathways can be collected and how assistance can be provided for the navigation of a repository of web pages. It also intends to be a research platform for a future extraction of feedback and the development of advanced algorithms for a better assistance.

Acknowledgements

My most sincere thanks are due to Annya Réquilé who supervised me during these past five months. She guided me with enthusiasm and without neglecting the human aspects of research.

I am also particularly grateful to Christine Mingins who is the initiator of the whole project and made this master's thesis possible.

I would then like to thank Brian Yap for his collaboration and Jan Miller and Hugo Leroux for a thousand reasons they know.

Finally, a special word of thanks is due to all the people at the CSSE who were responsible for such a good atmosphere at work.

Contents

Introduction

A shift has recently occurred in the e-learning community. Educational material starts being developed differently as classically integrated courses tend to be replaced by small reusable chunks of instructional data [1]. These chunks of data supporting learning are called learning objects [2]. More precisely, learning objects are defined by Wiley in [3] as "any digital resource that can be reused to support learning". Learning objects rely on the idea that since more and more learning content becomes available on-line through colleges, universities and companies, it is costly and senseless to produce new material that is similar to material that is already available. Instead, new courses can be built based on already-existing, reusable learning objects. The same learning object can even be used in totally different contexts: a video about Australian bushfires for example can be handled from both a biological and an economical point of view. Consequently, learning objects are often referred to as the Lego™ approach.

This new paradigm for building courses has quickly raised enthusiasm. Many different learning object repositories have appeared so far, for example the one proposed by Apple [4] or in California [5]. The IEEE Learning Technology Standards Committee currently works on standards for facilitating the interoperability of learning objects ([6]) so that, if the success is confirmed, there may soon be a virtual worldwide library of learning objects available for teachers and course builders.

Australia is getting particularly involved. The Le@rning Federation has been recently initiated by the government to incite a massive investment in the area. "In 2001-2006 all States, Territories and the Commonwealth of Australia are collaborating in this Initiative – The Le@rning Federation – to generate, over time, online curriculum content for Australian schools" [7].

This decision is bound to lead to the creation of very rich and complex repositories of learning objects. However, Edouard Lim, chief librarian of Monash University, noticed that "there is no credible research as to whether extant repositories meet the needs of course builders". He pointed out that current learning object discovery tools like SchoolNet [8] or Merlot [9] do not manage to satisfy the users. Some lead users to the website of the learning object's owner instead of the learning object itself. The more advanced repositories propose search engines relying on domain specification and keywords, which tends to provide a vast amount of non-sorted answers to queries, a part of them being irrelevant. This is all the more critical as the repositories get bigger and richer.

The issue of the effective use of complex repositories, although likely to become particularly crucial for learning objects, is general. A repository, in the general sense, is composed of a set of resources, where a resource is a chunk of data that can be accessed by users. Examples of repositories are a relational database, a web site, a set of multimedia documents. As the fast development of networks makes more and more information become available, resources of all types can be accessed inside repositories whose size is virtually unlimited. The bigger the size, the harder it is to make effective use of the resources.

Different actors are involved in the use of repositories:

(1) Users access the resources (navigate the repository) for satisfying an interest in a topic/domain, using access facilities. An access facility is a mean for resource

discovery, for example an index or a search engine. The resources are accessed sequentially: such a sequence is a navigation pathway.

(2) Administrators or "curators" (by analogy with museums) are responsible for organizing the resources and providing access facilities.

(3) Content providers elaborate the resources and, sometimes, provide embedded access facilities (for example web links when the resources are web pages).

From a user's point of view, making effective use of the repository consists in finding the resources that are relevant for the domain they are interested in. This task may be easy for experts who know both about the domain and the repository, but hard for the other users. This is where access facilities are crucial. Traditional access facilities include:

(1) Search engines. The advantage is that they do not usually require much maintenance: evolutions of the repository are automatically and periodically taken into account when the repository is swept by the engine. Search engines rely on the textual content of the resources when they are text-based, or on metadata. Despite all the advances that have been made like clustering [10] and relevance-based filtering algorithms, using a search engine is still often not efficient. Finding keywords for a search engine is a task that requires time and concentration, and it is all the harder as the user does not know much about the domain.

(2) Indexes. They provide direct access to certain resources. Thus an index only deals with a limited number of resources. It allows for the fast discovery of some key resources, however it does not allow for an effective use of the whole repository. The bigger the repository, the bigger the limitations of indexes.

(3) Web anchors. They are embedded in the resources and provide unidirectional links to other resources.

All these access facilities have in common that they rely on static data: indexes and web anchors are static, search engines use static textual content or metadata. This static nature brings its own drawbacks that get all the worse as the repository evolves quickly.

First, maintenance is tedious because it must be done by hand. For indexes, administrators have to take into account the evolution of their repository as well as the possible evolution of the interests of the users. It requires a close control of the repository, which is hard to achieve when the resources come from multiple sources. For web anchors, the pointed web pages may have been removed or new relevant ones may have appeared. This should be regularly checked by content providers so that they modify their web pages if needed. Therefore the problem of "dead links" is commonly spread throughout the web for example. For the same reasons, the metadata of resources is not always complete or up-to-date. For instance, the meta tags of web pages are seldom used effectively.

Secondly, it is provided for users by non-users: indexes by administrators, web links by content providers, metadata possibly by both (metadata can be provided by users in some rare cases but that requires an effort from them). Indexes, for example, rely on the administrator's judgment on 3 points: (1) which topics users should be interested in for structuring the index, (2) the selection of the resources to be indexed, and (3) the name of the index entries, that should be understood by the users. However, users, administrators and content providers may have different points of view. Static-based access facilities depend on the administrators' or

content providers' interpretation of the resources and understanding of the users' interests and level of knowledge. This ignores the diversity and heterogeneity of the users. A resource like a learning object can be considered differently in different contexts. A web page about Australian bushfires may have anchors to pages related to biology but not economy because the content provider is a biologist.

From these two problems, two main needs emerge:

(1) The need for feedback about the usage of the repositories. This is because administrators and content providers have a limited knowledge of the repository and the needs of the users whereas that knowledge is important for doing their job: maintaining, updating and improving access facilities as well as resources.
(2) The need for assistance in user navigation based on the experience of users instead of administrators or content providers.

The LEOPARD project (Learning EnvirOnment Platform for Agent-based Repository Discovery) [11] aims at addressing these needs. This project originates from Edouard Lim's propositions and it is conducted at the School of Computer Science and Software Engineering - Monash University. The LEOPARD project is at its beginning and this Master's Thesis work is part of it.

Chapter 1: State-of-the-Art

Recommender systems are first examined since they aim at assisting resource discovery. Then agents are studied as a widespread approach for handling assistance problems. Lastly, metadata is presented as a static technique for the better use of resources.

1 Recommender systems

1.1 Background

Recommender systems [12] are systems that learn about the preferences of users in order to help them find resources, sometimes called items, they are interested in. The items can be any resource: books, movies or web pages for instance. Such systems try to address the problem of information overload, particularly on the Web and in e-commerce, free the user from having to formulate explicit queries and make the access to desired items more efficient.

The preferences of users are gathered into 'user profiles'. A user profile is personalized information about what is known by the system of the interests of the user. These can be long-term or current, dynamically changing interests. Typically, a user profile is a collection of ratings indicating the user's interest on certain items.

There exist two main techniques for filtering and selecting items, both relying on the ability of the system to extract profile knowledge about what the user likes and dislikes:

- The content-based technique consists in selecting the items that are similar to what the user likes and dissimilar to what he dislikes. Such a technique is content-dependent because it requires to define what "similar" and "dissimilar" mean for items.
- Collaborative filtering [13] relies on the comparison of users instead of items. The profile of the user is compared with other user profiles in order to find users who have similar "tastes" or preferences. Thus the user is likely to like items that such users liked. So the items that most of such users found interesting are selected for recommendation.

Some recommender systems combine both techniques, in which case they are said to be hybrid [14]. Content-based recommendation requires feedback from the user in order to know what he likes and dislikes. In the case of hybrid systems, this information is shared in order to make collaborative filtering possible for other users. Hybrid systems present several advantages compared to "pure" systems. On one hand, pure content-based systems provide over-specialized recommendations as users are only recommended items that are similar to items they have already graded. In addition, content-based systems face the problem of the methods for content analysis: such methods are either imprecise or targeted to very specific items. On the other hand, collaborative systems cannot recommend new items since they have not been rated by any user. Furthermore, they cannot make recommendations for users who are similar to no one. Hybrid systems overcome these problems by the mean of additional complexity.

In all the cases, the user profiling strategy is an important issue. Such a strategy defines how profile information, i.e. likes and dislikes, is collected from the user. It can be done in an active or passive manner [15].

- Active profiling consists in asking the user to provide profiling information explicitly. Typically, the user is asked to rate several items. In the case of systems that are exclusively collaborative filtering systems, this strategy raises the problem of the reward of users for providing such a feedback. Indeed, a user who takes the time to provide feedback explicitly does not get any benefit from it.

- The passive strategy relies on the transparent observation of the user behaviour. Monitoring certain aspects of his behaviour allows the system to infer some implicit rating as a feedback. Although this strategy removes the burden of rating items for users, its effectiveness closely depends on the algorithms for inferring interest ratings. For example, the time spent in reading web pages, the number of mouse clicks on a page can be relevant indicators of interest. This point is currently an active research area [15-17].

1.2 Examples

1.2.1 MovieLens

MovieLens [18] is a classical collaborative filtering system that targets on recommendations of movies. Users are requested to rate some movies on a scale between 1 and 5. The more ratings, the more accurately the system can match the user with others and provide effective recommendations.

1.2.2 Fab

Fab [19] is a recommender system for the web. It is a hybrid system: recommendations come from both content-based comparisons with items in the user profile and high-rated items from the profiles of similar users. Internally, it relies on different kinds of agents. Collection agents are in charge of finding web pages related to given topics. On request, they send them to a central router that then forwards them as recommendations to users. Users are then requested to rate them explicitly. In addition, a selection agent is dedicated to each user. It is in charge of selecting the web pages received from the central router. This is achieved through the maintenance of a user profile that is built from the user's ratings.

1.2.3 MEMOIR and related

In [20], an evolution of a previous recommender system for the web called MEMOIR is described. It relies on the notion of Open Hypermedia which consists in managing links between documents separately from documents. This provides much flexibility in the nature and use of links. For example, the concept of generic link mentioned in [20] allows for content-based navigation. Links, for instance, can be followed by strings in the case of text documents: destination anchors are determined according to the string.

The system is agent-based. A user profile is built for knowing what the user finds interesting. Interest is inferred according to the activity the user is doing. Activities are monitored by a local User Interface agent. Such activities include navigating, bookmarking and rating Web pages. The user's navigation is observed thanks to a specific proxy. Also, users have the possibility to add annotations to the web pages.

Links are thus generated and stored in a "linkbase" where they are categorized into topics. Whenever a web page is browsed by the user, a context identifier helps check is there are links in the linkbase that match the context of the page. If so, then the links are recommended through the alteration of the web page.

1.2.4 Casper

Casper [21] is a system that provides recommendations for finding jobs on an online recruitment website. Recommendations rely on case-based reasoning: the system evaluates each possible new job comparing it with the jobs already evaluated. It then proposes jobs that are the most similar to jobs that have interested the user. Profiles are built passively by the system through a server log.

1.3 A profiling technique

This section describes a profiling technique that is interesting in the context of our project.

In Casper, user profiles are built passively from server logs [21]. Server logs simply contain information about accesses by users to a repository of web pages. Thus it is an interesting example on how to build user profiles with that kind of repository.

The recruitment website publishes job descriptions. When a user accesses a job description, he can ask the system to email the description to him for further examination or apply for the job online. Job descriptions are discovered through search queries. User profiles are built to automatically filter out irrelevant jobs returned by the queries.

User profiles contain a list of jobs that have been accessed. Jobs are rated via 3 "relevancy indicators":

- The number of visits made by the user. A filter detects irrelevant revisits that are due to "irritation clicks". Irritation clicks occur when the latency of the network irritates the user, who clicks repeatedly on the same anchor in the web page.
- The action performed. It can be, by increasing indication of relevancy, read, email to oneself or apply online. Obviously, this indicator is specific to the web site.
- The time spent by the user reading (read-time).

The calculation of the read-time is the more delicate. A general average read-time for all the jobs in all the profiles is calculated. The read-time is calculated when a user leaves a job description for reading another one. If the value is bigger than a threshold, for example because the user logged off, the value is replaced by the general average read-time. If the user already read the job description in the past, the new value is added to the previous one. Finally, the read-time is graded based on the comparison with other read-times in the user profile.

Such a method is an example of how to infer relevancy indicators using a passive profiling technique on simple server logs. However it is adapted to homogeneous resources like job descriptions: this is why calculating a general average read-time makes sense. For resources that are highly heterogeneous, the algorithm would have to be more sophisticated.

2 Agents

2.1 Overview

2.1.1 Definition of an agent

The notion of agent originally comes from the AI community and has been particularly dealing with the field of Distributed AI. Although such a notion is not new, finding a precise definition of an agent is a problem that has long been debated [22]. The term "agent" has been applied by many people to very different software entities in various contexts. According to loose definitions, it is just a system that encapsulates some artificial intelligence.

However, there exists a general agreement on some more precise characteristics. An agent is an entity that has a set of goals. These goals concern the environment within which the agent is situated. In order to reach such goals, the agent can act upon its environment over time and, usually, sense it.

Besides, an agent has ordinarily some degree of autonomy, which is the ability to have an activity without any external intervention of any human or program. To distinguish between levels of autonomy, some authors speak about "autonomous agents" or "semi-autonomous agents". In general, a significant consequence of autonomy is that agents are said to have control not only over their internal state but also over their behaviour (which is sometimes called encapsulation of behaviour). It means that they can be requested to perform some action or provide a service but the final decision about what to do is up to them.

A few agents are solitary in the sense that they interact with no other agent or with the human user only, who can be considered as a non-software agent. In many cases however, the ability to communicate with other agents is considered as an important feature.

A commonly admitted definition of an agent is given by Jennings and Wooldridge in the following statement:

> "an agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [23]

What is called flexibility is precisely what makes agents somehow intelligent. It means that they are:

- reactive: they sense their environment and respond by acting on it;
- proactive: they take initiative in order to satisfy their goals;
- socially able: they can interact with other agents.

Also, there exist some common properties that are crucial in some contexts. For instance learning agents have the capability of altering their behaviour with experience. Also, mobility can be fundamental in distributed systems: mobile agents are able to migrate from one host platform to another, in order for example to carry out some task locally.

2.1.2 Agents vs. objects

Comparing objects and agents is a way of providing a better understanding of the concept of an agent. Objects and agents present similarities: they both have behavioural capabilities, encapsulate an internal state and use message passing for communicating. Thus Shoham introduced in [24] the notion of Agent-Oriented Programming as a specialisation of Object-Oriented Programming with specific constraints. From this perspective, Bradshaw [25] characterizes an agent as "an object with an attitude". In other words an agent, as an abstraction mechanism, is an object with additional capabilities [26, 27].

First, the state of an object is just a set of attributes, while many agents have a more sophisticated structure. This is because of the requirements of the behaviour of an agent. For example, Shoham describes the state of an agent, called its mental state, as a composition of "mental components" such as beliefs, capabilities, choices, and commitments. Also, a successful kind of agent is built on the Belief, Desire and Intention (BDI) model, which separates between the information, motivational and deliberative states [28].

Then, unlike objects that receive indisputable orders through method invocation, agents only receive requests. It means that agents can actually decide what to execute, which may imply to ignore some requests. This property of control over the behaviour is a consequence of what Odell [27] calls unpredictable autonomy. It is summarized by the sentence "Agents can say "no"." The behaviour of an agent may not be predictable externally since it depends on the encapsulated states and goals of the agent.

Besides, an object is passive by default while an agent has to be active for pursuing its goals. This is what Odell calls dynamic autonomy. It is summarized by the sentence "Agents can say "go"." It can be characterized in degrees, from simple reactivity to an entirely proactive behaviour. A consequence is that an agent has its own thread of control. Thus the notion of active objects is probably the closest to agents in the object-oriented world. For example, Huhns and Stephens state in [29] that "Fundamentally, an agent is an active object with the ability to reason, perceive and act." Therefore, several works have illustrated ways of implementing agents from active objects [30, 31].

Lastly, the relationships between agents are more complex than in the case of objects. Because of both sorts of autonomy, agent communication involves event notification and is generally asynchronous. This involves parallel processing. In addition, the effective collaboration of different agents requires a good organization and brings the need for a social dimension within multi-agent systems.

2.2 Multi-agent systems (MASs)

2.2.1 Overview

Communicative agents are able to cooperate in order to satisfy their respective goals. A set of interacting agents is called a multi-agent system or agent-based system (groups of agents are also called communities or societies of agents). For the AI community, MASs are regarded as an interesting and still promising (although not new) approach for Distributed Artificial Intelligence (DAI) [32].

Sycara characterizes MASs as follows [33]:

(1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint;
(2) there is no system global control;
(3) data are decentralized;
(4) computation is asynchronous.

Despite their autonomy (2)(4), the agents take part of the achievement of the overall goals of the system by working on the realization of their simple goals with their limited data and capabilities (1)(3).

Considering all the characteristics described above, it is apparent that MASs are a very particular type of systems that are more suitable to certain types of problems or environments [33, 34].

MASs are particularly suited for conceiving complex systems. The abstraction and modularity they provide leads to the conceptualization of a system as "a society of cooperating autonomous problem solvers" [34]. Such a partition of the problem helps reduce complexity.

Then, such a metaphor fits well to some kinds of problems involving naturally autonomous entities. Examples include air-traffic control, manufacturing systems or virtual characters in computer games (see Applications).

Because of their decentralized nature, MASs are also a good mean for solving problems that involve distribution, in the general sense: either distribution of data, control, expertise or resources [34]. In particular, MASs can be used for modeling real-world entities with their own expertise and resources that need interact between them. In the case of distributed data, agents are a mean for making computation at the data sources, thus reducing (possibly distant) communication to exchanges of already-computed high-level information.

Furthermore, MASs are a good mean for implementing open systems. An open system is a system whose components can change dynamically and be highly heterogeneous, for instance in terms of authors or implementation languages and techniques. A typical example of an open system with heterogeneous components is the World Wide Web. Heterogeneity is not a problem since agents present strong encapsulation. The dynamic alteration of the composition of a MAS is made possible through certain types of MAS organizations like the Facilitator. As an example, the system described in [35] supports the dynamic addition and retraction of services by the mean of agents that register/unregister to the system.

Finally, MASs present the advantage of allowing for the interoperation of legacy systems. Again, because agents naturally present a strong encapsulation, a legacy system can be turned into an agent. Strategies for such a transformation include the use of a wrapper or a transducer [36]. Modifying legacy systems can be very expensive. Integrating them into a MAS is a way of making them able to collaborate with new systems without having to modify them.

2.2.2 Agent communication

While partitioning a system into agents helps reduce its complexity, complexity may then arise in the relationships between agents. Agent communication is at the heart of MASs, whether agents collaborate or compete, whether they communicate between them or with non-

agent programs. Concretely MASs are based on 3 elements for communication: an agent communication language, a content format and an ontology.

a) Communication languages and protocols

Agent communication languages (ACLs) define the semantics of the communication protocol. An ACL message is composed of several arguments that at least specify the sender, the intended receiver, the content of the message and for understanding that content, the content language and the ontology used. There exist two widespread ACLs:

KQML, the Knowledge Query Manipulation Language [37, 38], has been used in many projects for almost a decade and is thus a kind of *de facto* standard.

The FIPA ACL is intended to become a standard as well. FIPA is the Foundation for Intelligent Physical Agents. It was formed in 1996 to produce software standards for the interoperation of heterogeneous agents and the services that they can represent in agent-based systems [39]. At this time FIPA has not released any standard yet although experimental ones are provided, among them the FIPA ACL language.

b) Content formats

While an ACL defines how to exchange messages, a content format specifies a syntax and its associated semantics for defining how the content of the messages is represented. It can be seen as the "inner language" of an ACL [36]. The understanding of messages requires agents to use a parser for that inner language.

A commonly used content format is KIF, the Knowledge Interchange Format [40]. Examples of other formats supported by KQML are SQL, Prolog, Lisp. In addition to KIF, FIPA proposes experimental specifications for CCL (Constraint Choice Language, based on the representation of choice problems as Constraint Satisfaction Problems - CSPs) or RDF (the Resource Description Framework, designed for expressing machine-understandable metadata and supporting interoperability between applications, and that can be encoded in XML).

KIF is often used for illustrations because it is easily readable. Here is an example:

```
(ask-one (mug-price blue-mug ?price))
(reply (mug-price blue-mug (aud 10)))
```

The first line is a request for the price of a blue mug, the second one is the reply informing that the blue mug costs 10 Australian dollars.

c) Ontologies

An ontology defines a vocabulary that should be shared and known by the agents. It allows agents to agree on the meaning of the words used in the content of messages. An ontology is typically domain-dependent and thus defined for a particular MAS. However, there are some attempts of standardization. The main example is the Semantic Web project that aims at defining a standard ontology for the web in order to facilitate the use of internet agents [41].

2.2.3 Architectures

MASs need an organization for specifying agent interactions through the definition of roles, behavior expectations and authority relations [33]. For example, a simple kind of architecture is the hierarchical one, in which an agent at a given hierarchy level is responsible for making decisions for the resolution of a problem, and uses subordinate agents so that communication is only vertical.

For handling distribution and heterogeneity and making more effective use of the agent paradigm, more sophisticated architectures have been elaborated.

a) Federations

In a federation [36], agents belong to groups that are typically distributed. Each group has a particular program (possibly an agent) called a facilitator. Agents register their interests and capabilities and sent requests and notifications to their facilitator. Facilitators are then in charge of sending these messages to the right agents, possibly by the intermediary of other facilitators. Thus facilitators have to perform some intelligent routing of messages, to select the right agents to accomplish some tasks, to process messages for semantic translation, to manage the communications across the network [42]. This organization is powerful in that it allows transparent access to services provided by agents, it is scalable and it is open since it permits dynamic addition and retraction of such services [35].

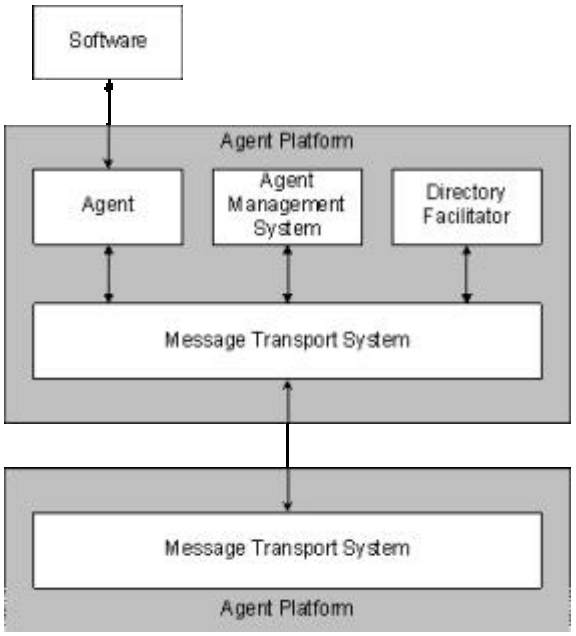
Depending on the system implemented, facilitators tend to have all or some of these capabilities. In [43], a Broker Agent is defined as a simple facilitator that provides "yellow pages" services. Agents register services offered and requested so that the broker dynamically connects services to requests. In addition, an Agent Name Server (ANS) allows for the actual inter-agent communication by providing a "white pages" service. In other words, the ANS is in charge of matching the symbolic names of agents with their addresses.

b) FIPA architecture

The architecture specified by the FIPA goes in the sense of the approach described above in that it tends to separate the services described in [42]. It distinguishes between the white pages service, the yellow pages service and the transport service.

FIPA specifies that agents reside on an Agent Platform (AP) that consists of some machine(s), the operating system and the agent support software. In addition, an AP must have 3 agent management components:

- An Agent Management Service (AMS) offers white pages services to agents. Every agent on a given Agent Platform has to register to the AMS. The AMS maintains a directory of agent identifiers containing transport addresses.
- A Directory Facilitator (DF) plays the role of a yellow pages service provider. Agents may register their services with the DF or query the DF to find out what services are offered by other agents.
- A Message Transport System (MTS) is in charge of handling inter-Agent Platform communication.



FIPA Agent Management Reference Model. Source: www.fipa.org

2.3 Examples of agents and MASs

2.3.1 Personal assistants

P. Maes' "interface agents" are built on the metaphor of the personal assistant [44]. They help users and collaborate with them in order to perform some tasks. They are initially not very good at it but they learn the user's habits and become progressively useful. Concretely, interface agents monitor the user's activity, remembering the actions and learning from them, then perform actions on their own to "reduce work and information overload". Thus such personal assistants are customized and personalized for specific users.

Such an assistance can take different forms: perform tasks on the user's behalf, train or teach the user, facilitate collaboration between users… [44]. The last point means that the agent can assist in exchanging know-how and efficient habits between the different users of a community. In all the cases, the agent should not restrain the freedom of the user in the sense that the user is able to behave just the way he would without the agent.

The learning phase appears to be fundamental. It can be achieved by different means:
- The agent "watches over the shoulder", i.e. observes the user's behavior for imitation.
- It adapts thanks to the feedback he receives from the user.
- It can be trained by the user on the basis of examples.
- It can ask for advice from other agents assisting other users.

Yet two assumptions determine if such an approach is suitable:
1. "The application should involve a significant amount of repetitive behavior" since the agent needs to detect patterns.
2. Such repetitive behavior should not be the same for all users, otherwise it is better to hard-code the procedures and there is not need for learning agents.

However, the usefulness of personal assistants has been questioned since then [45]. On one hand, if the task is simple there may be no use for a personal assistant; on the other hand, a complex task is likely to require that a deep knowledge about the user and about the task is provided to the agent.

As an example of interface agent, Lieberman et al. discuss in [46] Reconnaissance Agents. This kind of agents help the user browse the Web. By observing the user navigating with his usual browser, they generate profiles that they use to infer the user's interests and preferences. Then when the user has reached a web page, they propose relevant links for him to navigate next. The interface agent is independent from the browser so it does not limit the user in his manual browsing activity.

Lieberman et al. have developed two such reconnaissance agents. The first, Letizia, transparently explores the links on the current page to eliminate those that are bad or irrelevant and proposes the best one. The second one, PowerScout, uses search engines to perform *concept browsing*: it extracts keywords from the current pages and combines them or not with the user profiles to query search engines. It then displays the results, grouped by concept.

2.3.2 Other information agents

Information agents are more general than personal assistants. They can be defined as "computational software systems that have access to multiple, heterogeneous and geographically distributed information sources as in the Internet or corporate Intranets" [47]. They are generally in charge of looking for relevant information among scattered data (information gathering) or filtering and organizing such data coherently. The need for such agents has become critical with the explosion of Internet.

An example of agents for managing information is the Zuno Digital Library. "Digital libraries are a set of well-organised technologies and, above all, a very important source of structured, well-organised and well-stored information" [48]. In the case of ZDL, the system consists in a multi-agent system that provides a coherent view of heterogeneous, disorganized data sources like the Web [49]. In [48], P. Isaias proposes an architecture for a virtual digital library that is composed of 8 kinds of collaborative agents, each of them being specialized in a well-defined role: for example, user interface agents for consumers and providers, broker agent, information retrieval agent.

HuskySearch/Grouper [10] is a descendent of MetaCrawler, a meta-search engine that helps users find information on the Web without maintaining any database. The whole system is a Softbot ("software robot"), i.e. an "intelligent agent that uses software tools and services on a person's behalf". It is called intelligent agent in the sense that it uses the same tools as users do and determines dynamically how to satisfy the user's request. HuskySearch/Grouper queries several popular web search engines, then organizes the results using a clustering algorithm. In other words it tries to group documents in several topics, based on their similarities, in order to help users locate the interesting ones and get an overview of the retrieved document set.

2.3.3 Electronic commerce

Electronic commerce is a field in which agents can typically prove useful by reducing information overload and saving time for users. Autonomy and personalization allow agents to act efficiently at the stages of product brokering, merchant brokering and negotiation [50]. Agents can contact other agents or explore the Web to select relevant products within the scope of interest of the user, then select interesting offers for a given product and possibly determine the terms of the transaction.

However agents are not limited to the role of the buyer: they can be the seller or an intermediary (mediator, facilitator that maps consumers and producers, information provider) [51]. Thus many business tasks involved in e-commerce can be automated. Among others, examples of agents dedicated to e-commerce are MIT Media Lab's Firefly and Kasbah [52].

2.3.4 Industrial systems and logistics

Centralized, hierarchically-organized manufacturing planning and control is a model that is often considered as being too rigid for dealing with today's dynamically changing environments [53]. Instead, more sophisticated systems are needed for more flexibility and fault tolerance. MASs are thus a useful approach thanks to their ability to handle complex, distributed systems. A big number of such systems are referenced in [53].

For similar reasons MASs have been applied to logistics, for example air traffic management or military operations (MokSAF [54]).

2.3.5 Games

A big number of different kinds of games involve computer AI. A category that really involves agents is 3D action "Quake-like" games, in which agents are virtual characters. Such autonomous characters have well-defined goals, typically seek for enemies to destroy. They sense their environment through their "range of sight" or by "hearing" noises. Then they react to such signals, for example they protect the leader if he is in danger. They also take initiatives to reach their objective, for instance deciding which path to take to reach the enemy base.

2.4 Existing tools

There exists a big number of tools that aim at facilitating the development of agent-based systems. They range from specific programming languages ([55]) and component libraries to agent development frameworks ([56]). Frameworks for building MASs are interesting in the context of this project because they do not provide only facilities for building agents but also a generic design for agents and a basic implementation, easing rapid prototyping.

Besides, this project can involve a big number of agents, which may require to distribute them on several machines for correct performance. Agent platforms provide the infrastructure for allowing agents to interoperate and they sometimes handle distribution. Such platforms are thus interesting. In addition to handling distribution and inter-machine agent communication,

they often provide implementation of middleware agents like FIPA's DF and AMS for agent management. Furthermore, most of agent platforms also provide agent frameworks.

Another interesting feature is the compliance of platforms with FIPA specifications. FIPA has defined experimental specifications that are to evolve to become a standard. FIPA has released three sets of experimental specifications until now: FIPA 97, FIPA 98 and FIPA 2000, the last one making the others obsolete. Specification domains include the abstract architecture, agent management, agent message transport system, communication language and content language.

The following software is non-commercial platforms for developing MASs.

2.4.1 JADE

JADE, the Java Agent DEvelopment Framework [57], was developed by Telecom Italia and the University of Parma. It requires a Java 2 Runtime Environment and has been tested on many platforms. The last version, 2.5, was released in February 2002. It complies with FIPA 2000 specifications.

JADE implements an agent platform and a development framework. The platform can be distributed over several hosts regardless of the OS. It also supports agent migration and cloning. Agents lifecycle can be controlled via a GUI that also supports debugging.

It is a complete, rather mature tool that has been successful. It can be extended with many add-ons. For example, JESS allows for the development of rule-based behaviours. It also supports the Protégé ontology editor. Furthermore, many research prototypes have been based on JADE.

Lastly, JADE has been integrated with the LEAP project (Lightweight Extensible Agent Platform) [58]. This project targets mobile enterprises and ensures compatibility with mobile Java environments down to J2ME-CLDC. Thus it aims at providing FIPA-compliant agents on PDAs and mobile phones.

2.4.2 FIPA-OS

FIPA-OS (FIPA Open Source) is an Open Source implementation of the FIPA standard originating from a research lab of Nortel Networks [Networks, #41][Forge, #42]. It provides implementation for agent platforms and a component-based toolkit for developing domain-dependent agents. It is thus intended to enable rapid prototyping. FIPA-OS supports most of the recent FIPA experimental including the agent management and communication systems.

Since its first release in 1999, it has been continuously improved as a managed Open Source community project, leading to more than 10 formal new releases. Upgrades, bug fixes and extensions have been provided. In particular, a version of FIPA-OS aimed at PDAs and mobile phones, µFIPA-OS, has been developed by the University of Helsinki. In addition, useful tutorials are proposed as well as an active newsgroup.

FIPA-OS is totally implemented in Java 2. However, releases converted to JDK 1.1 are proposed as well.

2.4.3 ZEUS

ZEUS [59] is another sophisticated Open Source agent system developed by British Telecom Labs. It provides support for generic agent functionality and advanced settings for the planning and scheduling of the behaviours of agents. It also includes facilities for building agents in a visual environment. For specifying the agents' behaviour, goals are represented using a chain of actions that have to be fulfilled before the goal can be met. Agent communications comply to FIPA specifications for the message transport.

ZEUS uses Swing GUI components so it requires a JDK1.2 virtual machine. It has been successfully tested on Windows 95/98/NT4 and Solaris platforms.

2.4.4 Comtec Agent Platform

Comtec Agent Platform [60] is an implementation of FIPA 97 Agent Management, Agent Communication Language and Agent/Software Integration first released in 1998. The FIPA 98 Ontology Service was added later. Nevertheless, it is a limited platform with little documentation.

2.4.5 April Agent Platform

The April Agent Platform (AAP) [61] is a free agent platform provided by the Fujitsu Labs. It is FIPA 2000-compliant. It provides the basic environment in which FIPA agents can be launched and can operate. It is written in a language called April, the Agent PRocess Interaction Language.
However, it is available on Linux and Solaris only.

2.4.6 Grasshopper

Grasshopper is a Java-based mobile agent platform developed by IKV++ Technologies AG.
An add-on allows Grasshopper to be compliant with the FIPA98 specifications. Similarly, an add-on enables Grasshopper to comply with the OMG Mobile Agents Facility (MASIF) that provides a framework for agent mobility. Support is provided through an active forum on the internet. Grasshopper requires Java 1.2.2.

2.4.7 Conclusion

There exists sophisticated agent platforms and frameworks with advanced features like visual agent building, visual agent management and debugging, mobility or experimental support for mobile devices. However, the huge majority of these platforms require a Java environment.

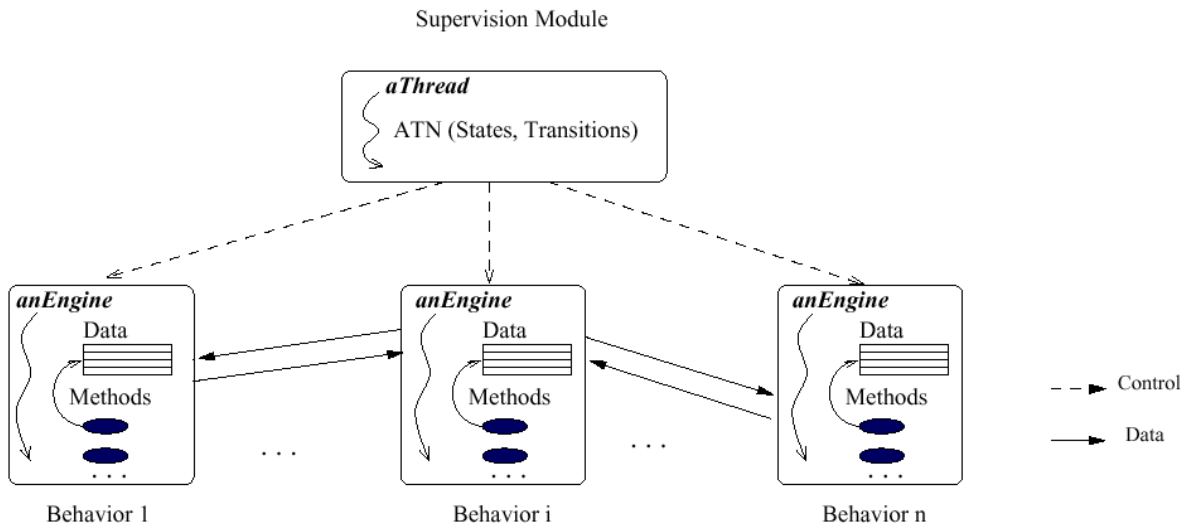
2.5 Design

2.5.1 The DIMA design

In [30], a generic agent structure for implementation in OO languages is proposed, based on active objects. The definition of an agent is based on the one proposed by Wooldridge (e.g. [23]]). Characteristics are derived from that definition, including:

- An agent's behavior can be decomposed into several behaviors like perceiving, reasoning, communicating. Each of them can be procedural or knowledge-based.
- An autonomous agent must have autonomous therefore concurrent behaviors. Besides, the communication behavior must incorporate a message interpreter (opposed to direct method invocation) so that the agent keeps control over its internal state and behavior.
- An agent is proactive so it must incorporate a meta-behavior that manages its set of behaviors, depending on its internal state and the external state of the world.
- An agent is sociable so it has to understand a communication language.

The proposed structure is this one:



From [30]

It is made of a first layer of concurrent behaviors that are managed by a Supervision Module at the meta-level (meta-behavior). This module is implemented as an Augmented Transition Network where states represent decision points. These decisions are about activating or suspending a behavior.

The behaviors are modules that have their own data, methods and engine. The data can be updated by the methods or any asynchronous event. The engine is a thread that controls the activation of the methods. It can be interrupted by the Supervision Module between two method/rule firings. Examples of behaviors are deliberation, perception, communication. A behavior is called reactive when it is purely procedural or cognitive when it is knowledge-based, e.g. when the engine is an inference engine. Thus the structure was implemented in Smalltalk-80 augmented with NéOpus for allowing rule-based programming.

2.5.2 The Brainstorm design

In the Brainstorm approach, an object becomes an agent thanks to its associations to agent capabilities that are provided by the meta-level [62]. The meta-level is composed of meta-objects and each of them provides a single capability. So an agent is composed of an object and a set of meta-objects. Agent capabilities include communication, perception, knowledge. For example, the communication meta-object intercepts the messages received by the object and treats them. In addition, a second meta-level is composed of reaction and deliberation meta-objects that manage the behavior of the first meta-level.

The second meta-level has a similar role as the Supervision Module in the DIMA approach. Yet here behaviors are not aggregated by the agentified object but defined at the meta-level. The drawback is that it requires that meta-objects are supported.

2.5.3 The FIPA-OS design

In FIPA-OS the root class for agents is the abstract class FIPAOSAgent that essentially specifies behavior for:
- registering/unregistering to the agent management services defined by FIPA specifications;
- receiving messages;
- handling Tasks.

A Task defines an activity of an agent. It has explicitly defined states and it can have subtasks. It registers to certain types of messages that it is in charge of handling. Thus a Task is automatically activated when some kinds of messages are received. A Task may create a new thread, enabling agents to perform parallel processing.

This approach provides a good modularity for handling messages by delegating this job to Tasks automatically. On the opposite, other approaches centralize message treatment in a communication module. However, there is no global supervision of the agent's behavior for activating or interrupting activities.

3 Metadata

During the earlier investigations in the start of the project, metadata was examined as a base for inferring virtual semantic links between resources. However, the use of metadata would imply to abandon the generic nature of our solution. The issue of interoperability presented in the last section about metadata illustrates this problem. It was therefore decided not make any use of metadata in our solution.
Note: we consider the word "metadata" as a singular as many authors do.

3.1 Definition

Metadata is literally "data about data". More precisely, Dempsey and Heery define it as:

> "Data associated with objects which relieves their potential users of having to have full advance knowledge of their existence or characteristics." [63]

The term 'user' can refer to a program or service (e.g. a software agent) as well as a person. The object described by the metadata can be a resource of any type: web page, text, image… It can also be an aggregation of several resources, as long as it can be manipulated as a single one [64].

Metadata can reflect the following features of the resources [64]: content (intrinsic), context (about the creation of the resource), intrinsic structure (associations between resources contained in the resource) and extrinsic structure (associations between the resource and other resources).

There exist different sorts of metadata that can be characterized on many different points [64, 65] including:

- Source: internal/external, creation at resource creation time or later.
- Creator: human (resource creator/user) or program (portal, resource creation tool).
- Nature: created by non-specialists or by experts.
- Status: static/dynamic, long-term/short term.
- Granularity level: relates to a single resource or a collection of resources.

3.2 Roles and use

Different actors need metadata for different purposes [66-68]. First, users need resource descriptions to search across the range of available resources in order to find, identify and interpret them. They must be able to combine and compare descriptions in order to select the resources that fit to their needs then obtain them.

Then metadata increases the accessibility of resources to users. In particular, metadata enables repository administrators to create catalogs or indexes that ease searches and resource discovery. The creation of such indexes can be automatically performed by a program if all the resources of the target repository or repositories have consistent metadata. It also allows for the use of search engines, particularly when the resources are not text-based.

Thirdly, repository administrators and content providers need to administer resources through time, classify them, preserve them, promote them. They also want to enable and control both access and use, for example for commerce, privacy, property rights, authenticity…

Finally, third party services like portals or brokers performing queries for users have the same needs as providers, and in addition they may need to annotate or re-contextualise resources.

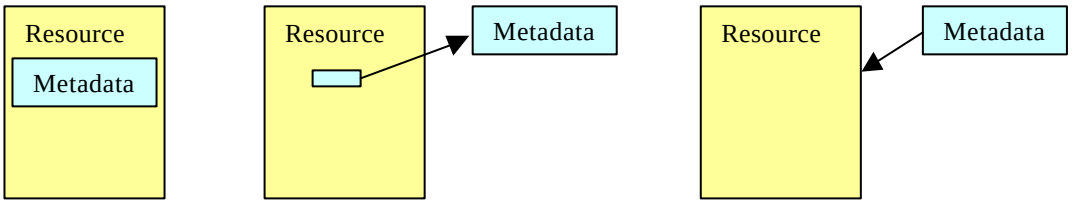
3.3 Storage

Metadata is always linked to a resource. This link can be implemented in different ways: metadata can be [65]:

- (1) Embedded in resource. It requires that the resource supports such an embedding of metadata, that the metadata creator/modifier has write access to the resource and that services support extraction of embedded metadata. An example of embedded metadata is the META tags in HTML documents.

- (2) Linked from resource, for example web links. This allows metadata to be remote from the resource. It requires that the resource supports embedding of link and that services are able to follow the link. Write access to the resource is also still required at metadata creation time.

- (3) Pointing to resource, which is the most common case. It allows services to get metadata independently of the resource: metadata can be a remote database entry for example. Services just need to be able to find and read the metadata records. It does not put any constraint on the resource and does not require resource editing.



(from [65])

3.4 Interoperability issue

As it has been seen, metadata is all the more useful as it is used and exchanged by different parties inside a community or even across communities.

> "Metadata can […] make it possible to search across multiple collections or to create virtual collections from materials that are distributed across several repositories" [64]

However metadata is static and can be defined by many different actors. Thus the problem of interoperability arises: metadata cannot be exchanged across repositories unless it is defined the same way in both repositories or it can be mapped from a repository to the other. For allowing for the full power of metadata, automated processing through software robots should be possible across the repositories.

This requires some agreement and standardization efforts between the parties on the following points [67]:

- Syntax of metadata: the rules of expression.
- Structure: the grammar or significance of the arrangement of terms.
- Semantics: the vocabulary of terms and what they mean.

3.4.1 Syntax

There is now a consensus on the use of XML, the Extensible Markup Language [69]. XML defines means of describing tree-structured data in text-based format. It gets now widely adopted for transferring any type of data, including metadata, between programs or systems (for example, it is the base of web services).

XML only provides syntax but that syntax is suitable for the definition of metadata standards. First, defining a markup vocabulary and associating it with a documentation allows one to provide semantics. Then XML supports the validation of structural models like DTDs (Document Type Definition) or XML Schemas. In other words, it is possible to specify a structural model and check that the structure of a given XML document conforms to it.

3.4.2 Structure

Communities use different structural conventions for expressing semantic relationships, which reduces interoperability. Thus the Resource Description Framework (RDF), which is a recommendation from the W3C (1999), has been elaborated. Its goal is "to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines (a priori) the semantics of any application domain." [70] Thus it should be domain neutral but at the same time be suitable for describing information about any domain.

RDF provides constraints on structure so that a document cannot be misinterpreted. It exclusively relies on URIs (Uniform Resource Identifiers) for identification of resources and their properties (descriptive attributes). However, despite its power RDF has not yet been widely adopted on the Web: it requires a hard-coding conversion effort.

3.4.3 Semantics

Several initiatives for creating semantics standards, sometimes providing also a structure, have been carried out like MARC 21 for bibliographic information [71], EAD [72], ISAD [73]. In the field of Learning Objects, the IEEE LTSC (Learning Technology Standards Committee) has a Learning Object Metadata working group currently working on the elaboration of a standard [74].

Such a need for the development of a standard also exists for audio, video and audiovisual resources. The Fraunhofer Institute is working on the definition of a Multimedia Content Description Interface, also called MPEG-7 [75].

One of the best known standards comes from the Dublin Core Initiative [Initiative, #89]. Developed by a group of librarians, information professionals and subject specialists, it aims at proposing a smallest common denominator for generically describing resources, thus easing cross-disciplinary resource discovery. However its generic aspect comes at a cost: it cannot be a replacement for richer, community-specific vocabulary.

While standards such as Dublin Core or MPEG-7 aim at describing single-medium atomic digital resources, people have claimed that the full potential of digital libraries is reached when they provide multimedia resources combining text, image, audio and video components. This is why the Harmony Project was initiated [76]. It aims at supporting the development of metadata standards for multimedia components.

4 Perspectives

Recommender systems make use of interesting techniques for helping users in resource discovery, thus in repository navigation. However, most are developed ad hoc in the sense that they target on specific repositories and specific resources in order to use targeted recommendation algorithms and be more effective. Moreover, providing navigation assistance only solves one part of the problem of the effective use of repositories. Assistance for repository administration and resource building is indeed necessary too.

Metadata is a mean for actors to make effective use of resources: resource discovery for users, resource management for repository administrators, provision of complementary information for content providers. However, it is static data that must be maintained at a significant cost. In addition, its static nature raises interoperability problems in terms of syntax, semantics and structure. Big standardization efforts are being carried out to overcome the problem. Before this objective is achieved, an application that makes use of metadata must target on a specific metadata specification and cannot be generic. Furthermore, generality goes against power: the more generic the metadata, the less relevant the information it holds.

Another issue is that metadata is generally created by an actor for being used by another actor, for example by a resource creator for a user. Thus it is limited by the creator's knowledge and point of view and designed for an intended usage that may not be the usage the user is interested in. This is particularly crucial in the case of learning objects as their power lies in reuse. For example, if a document about bushfires in Australia is provided by a biologist, it may contain metadata related to biological issues. But it could also be used by an economist building a course on agricultural opportunities in Australia. For him, the metadata will not be suitable and he will probably miss the resource. Another approach would therefore be needed.

Agents and multiagent systems are powerful concepts for handling assistance issues. The notion of autonomous problem-solving entities encapsulating some "intelligence" and able to collaborate when they do not have sufficient data is an elegant approach that has proven to be effective in many cases.

Chapter 2: Contribution

A team was created for working at the CSSE (School of Computer Science and Software Engineering) – Monash University on the problem of effective use of large repositories. The team includes Prof. Christine Mingins, Dr. Annya Réquilé, Honours student Brian Yap and myself. In the future, a PhD student and a MSc student will join the team too. The project was called LEOPARD for "Learning EnvirOnment Platform for Agent-based Repository Discovery". As the project was in its early beginning, the approach that would be followed had to be elaborated. This was carried out as a team work through weekly meetings.

1 Our approach

Opposite to the approaches followed by most recommender systems and strategies based on "hard" data like metadata, we chose to investigate how to assist all actors involved in the use of repositories – users, administrators and content providers – on a minimal base. What is meant by minimal base is the restriction of the data that is collected and computed in order to allow for the design of an assistance application that:

(1) **Works dynamically**, ignoring all hardwired metadata. Only information collected by the observation of the users is used.
(2) **Is generic** in the sense that no assumption is made at all about the nature of the repository neither the nature or content of the resources.

Because of (1), the application does not require any tedious maintenance from administrators. The application is based on actual usage only so that its effectiveness does not depend on non-users' knowledge or points of view. The navigation assistance provided to users relies strictly on the observation of the actions of other users. In addition, the application is then able to provide feedback about actual usage to both administrators and content providers.

At the same time, (2) makes it possible to handle different kinds of repositories like a web site or a relational database. But the main advantage is that it makes it possible to handle highly heterogeneous resources. This is fundamental for learning objects that can be in the form of text, video, image, sound as well as composite multimedia documents. Whatever the nature of the resources, the assistance application should be able to handle them the same way.

2 Expected outcomes

Outcomes are expected for all categories of actors – users, administrators and content providers.

2.1 User navigation assistance.

Users need to be assisted in resource discovery (or repository navigation). The application must thus be able to provide dynamic recommendations to the users about resources to access.

Not only are the resources important but also possibly the order in which they are suggested to be accessed. This is true for learning objects because they can have prerequisites. The same

way Lego™ blocks should be assembled in a specific, ordered manner to build something, learning objects should be used in a specific order in the context of a course. The prerequisites thus depend on the course involving the learning objects or on the context in which the learning object is used.

2.2 Feedback for administrators and content providers.

The observation of the actual usage of repositories enables the application to provide assistance to administrators and content providers as feedback.

For example it may be possible for administrators, as inspired by the Prototype Category theories of E. Rosch [77] and G. Lakoff [78], to discover communities of users and categories of content that administrators were not aware of, and that they had thus not made explicit in static access facilities like indexes. At a more basic level, if administrators introduce a new interesting resource and see it is not accessed, they will deduce that they should provide more straightforward access to it.

In this sense the application does not substitute itself to static access facilities but it helps administrators maintain and improve them. The dynamic nature of the application plays an important role here. As it keeps "up-to-date", it follows the evolution of the users' interests as well as the evolution of the content of the repository (addition/removal of resources) without any human intervention.

Concerning content providers, they may discover that their resource is used in different contexts from the one they expected. Again, the example of the document about bushfires that can be handled from an economical and a biological perspective illustrates this point. A biologist who receives such feedback can then redesign his resource, for example extending its metadata or adding web links to resources related to economical consequences of storms.

Obviously extracting feedback from data about usage is not a trivial task. Elaborating adapted algorithms is a promising field for investigation.

3 Principles of our solution

The main principles of our solution can be derived from the statement of the approach. The generic and dynamic nature of the application restrains the collected data to the most simple and general observable actions performed by users: accesses to resources. An access is basically a piece of information containing:

- the identity of the user performing the access;
- the resource accessed;
- the timestamp of the access.

By performing sequences of accesses, users define navigation pathways. Hence all the assistance provided by the system relies on the dynamic collection of pathways into user profiles and the computation of these profiles. Although it can seem to be rough, low-level information, its classification and computation can allow for the generation of a higher-level business intelligence layer (BIL) providing useful assistance. For the elaboration of this BIL, pathways can be handled from 2 different perspectives: user and resource. It is through the

combination of the information acquired from these 2 perspectives that the BIL can be generated.

3.1 User perspective

Pathways originate from specific users. Considering a given user, the application has knowledge of:

- his current pathway;
- his previous pathways in history;
- other users' pathways.

The user's current pathway comes from the domain or topic that the user is currently investigating. Using a notion introduced in recommender systems, it is then possible to distinguish **experts** among the other users. Experts are, in this case, users who have knowledge of (1) the domain or topic investigated by the user, and (2) the resources related to this domain/topic within the repository. It can thus be useful to let experts define reference pathways into "**standard user profiles**" (SUPs) that are related to specific domains. Such information can be used for refining user navigation assistance.

For example, a specialist in the lifestyle of kangaroos can define in a standard user profile one or several pathway(s) of accesses to resources in the repository dealing with that topic. A user specifying explicitly, or implicitly through his current pathway, that he is interested in kangaroos can then be better assisted in his navigation by the recommendation of such a predefined pathway.

SUPs are therefore a mean for the explicit definition of communities of users. SUPs can be explicitly created by experts, but administrators may as well "discover" implicit SUPs through the feedback provided by the application. Administrators can then make these SUPs explicit by actually creating them and associating users with them.

3.2 Resource perspective

Pathways virtually define links between resources (origin, destination). Using passive profiling techniques from recommender systems, it is possible to maintain relevancy indicators about links. This can be achieved by monitoring the tendency of users to traverse the links, i.e. to access the destination of the link after its origin. The more a link is traversed, the more we can assume that its origin and destination are somehow semantically related. This leads to the inference of **virtual semantic links** between resources. These virtual semantic links are higher-level than the raw links from pathways because they indicate that 2 resources are related by themselves, not made artificially related by pathways of users. Although the semantics itself cannot be really known by the application from simple pathways, the presence of a virtual semantic link can be assumed when the relevancy indicators on the link have high values.

The particularity of these links is that they are not static, hardwired links but instead dynamically-generated ones in the specific context of a business intelligence generation. When assistance is required for a particular actor in a particular situation (e.g. navigation assistance for user U who is accessing resource R after having navigated pathway P), specific

business intelligence is generated. Links, which are part of the BIL, are thus generated depending on that context.

3.3 Business intelligence generation

Dynamically generating specific business intelligence on demand requires to make numerous decisions in order to combine and compute information coming from both perspectives – resource and user – intelligently. For example, in the case of navigation assistance, choosing which resources are the most relevant for recommendation implies to consider at the same time the profile of the user, the SUPs that may suit to him and resources semantically linked to the current one in the context.

A large amount of data is therefore involved in the computation. For keeping good modularity, it seems natural to decentralize the data into individual pieces like user profiles. Thus the business intelligence generation relies on the decentralized computation then on the "intelligent" combination of the pieces of data. Multiagent systems appear as a natural approach for handling such a problem. Agents are suitable for computing their own encapsulated piece of information, making micro-decisions according to the context, then cooperating intelligently depending on these decisions for reaching the overall goal.

4 Consequences

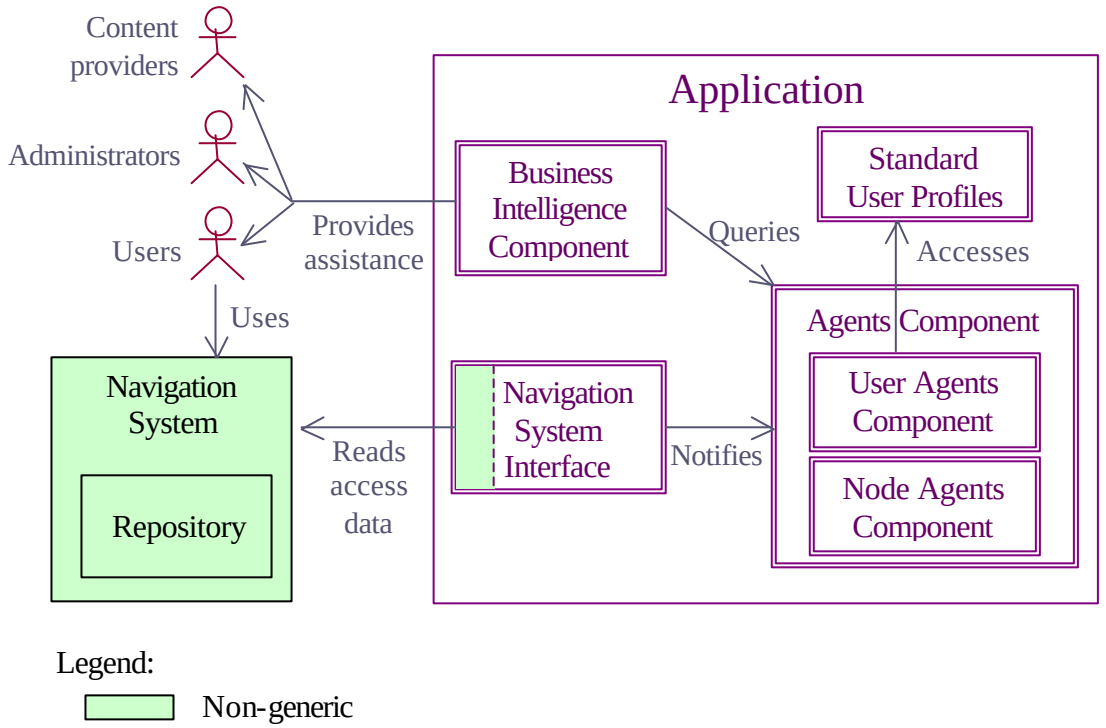
LEOPARD is an ambitious project that has to be carried out in the long term. The first phase is to test the approach by designing and developing a first application as an experimental platform. In addition, by validating the concepts the experimental platform should be a mean for obtaining support from the industry. As a consequence, it was decided to carry out the development on and for Microsoft's new development platform, .Net.

Chapter 3: Application Design

I was in charge of the design of the application. During that phase, my choices were exposed and discussed during our team meetings.

1 Structure overview

In its generic and complete form, the application is structured as shown by the figure below.



Overview of the structure of the application

The application is aimed at being layered over existing repositories of any type. The Navigation System comes with the repository and is external to the application. It allows users to access the resources. It provides the input point of the application as data describing the accesses. Inside the application, the Navigation System Interface is in charge of reading that Access Data. The Navigation System Interface is therefore dependent, at least in part, on the nature of the access data thus on the Navigation System. Also, assistance may be provided through the Navigation System and thus be dependent on it. All the rest of the application is generic.

For managing pathways, 2 groups of agents correspond to the 2 different perspectives described in Chapter 2: User Agents and Node (resource) Agents. On the overall, the same information is maintained by both groups of agents but it is organized differently. At this stage of the project, clarity in the design and in the way the application works is more

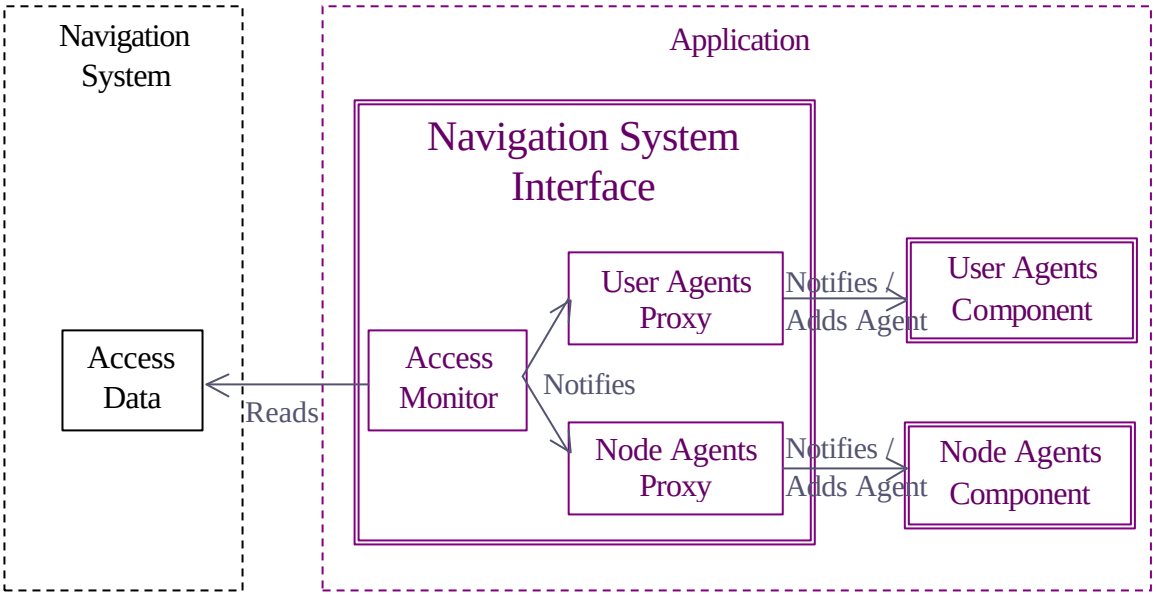
important than memory space optimization.

A description of the generic elements of the structure is provided below, then the non-generic elements are described in the case of the particular context of use that was chosen for the prototype application.

2 Navigation System Interface

An Access Monitor is an agent in charge of collecting the access data logged by the Navigation System. It notifies the User Agent Proxy and the Node Agent Proxy of every access. As explained in Chapter 2, access information includes a user identifier, a resource identifier and a timestamp. The proxies are then in charge of forwarding that piece of information to the concerned agents within the User Agents Component and the Node Agents Component. The concerned agents depend on the content of the access information as each resource is handled by a Node Agent and each user is handled by a User Agent.

Both proxies must therefore know all the agents of the component they are proxies for. Whenever a resource or user is not handled, they should require for the creation of the missing agent. This is done through an auxiliary User/Node Agent Creator.



The Navigation System Interface

Thanks to this principle, every user and accessed resource is handled by an agent that is kept informed of accesses involving its user/resource.

3 Agents Component

3.1 User Agents Component

User agents keep track their dedicated users identified by user identifiers. Access information helps retrieve pathways for building user profiles. Every User Profile is encapsulated by a User Agent. Links in a user profile are associated with data about the user's behaviour: the number of traversals performed, the timestamp of the last traversal, the time spent on the destination resource. In addition, the profile can be associated with Standard User Profiles that are known to suit the user.

When a user does not navigate the repository for a while, his dedicated User Agent can save its profile and terminate. It is regenerated whenever the user starts navigating again.

3.2 Node Agents Component

A Node Agent is in charge of tracking the usage of a resource. This is done by keeping trace of the traversals of links originating from the resource, independently from the users. Like in user profiles, links are associated with relevance indicators: last timestamp, number of traversals. A node Agent thus has information about the popularity of a resource in the context of different pathways.

4 Standard User Profiles Component

A Standard User Profile (SUP) defines interesting pathways for people interested in a certain domain. SUPs can be created by experts. Alternatively, administrators can create SUPs as well if the application enables them to discover categories of content.

5 Business Intelligence Component

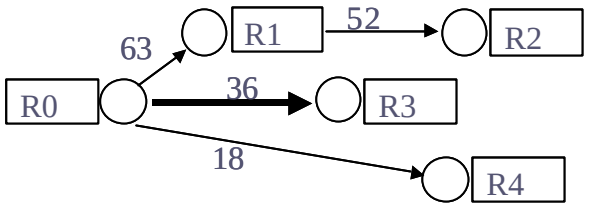
This component is in charge of elaborating assistance dynamically. Because designing an algorithm for providing assistance to administrators or content providers would require a lot more investigation, only navigation assistance for users has been considered for this application.

Navigation assistance is elaborated through the generation of a directed graph representing recommended navigational pathways. The vertices represent resources while the edges symbolize links. The graph originates from the resource the user is currently on. The edges hold relevancy indicators representing the relevancy of the traversal of the link they represent.

Relevancy indicators come from the computation of data provided by 3 sources:

(1) the Node Agents that give an indication of the general popularity of the link;
(2) the SUPs associated with the user, representing some typical interest for some categories of users; and
(3) the personal profile of the user.

The graph is then formatted to be visualized and sent for being displayed to the user. A possible format is naturally a visual directed graph. As an example, the graph could look like this:



Example of a graph as navigation assistance

In this example, "RX" is the identifier of a resource.

- The length of the edges and the weights indicate the general popularity of the link: the bigger the weight, the shorter the arrow, the more popular the link.
- The thickness of the arrows indicates the relevancy of the traversal of the link for the user based on his former pathways and his domain interests (SUPs).

This is just an example since elaborating a relevant visualization is an issue that would require a specific study.

6 Non-generic part

Although most of the application can be designed in a generic way, it is necessary to decide which sort of Navigation System the application layers over for a complete design.

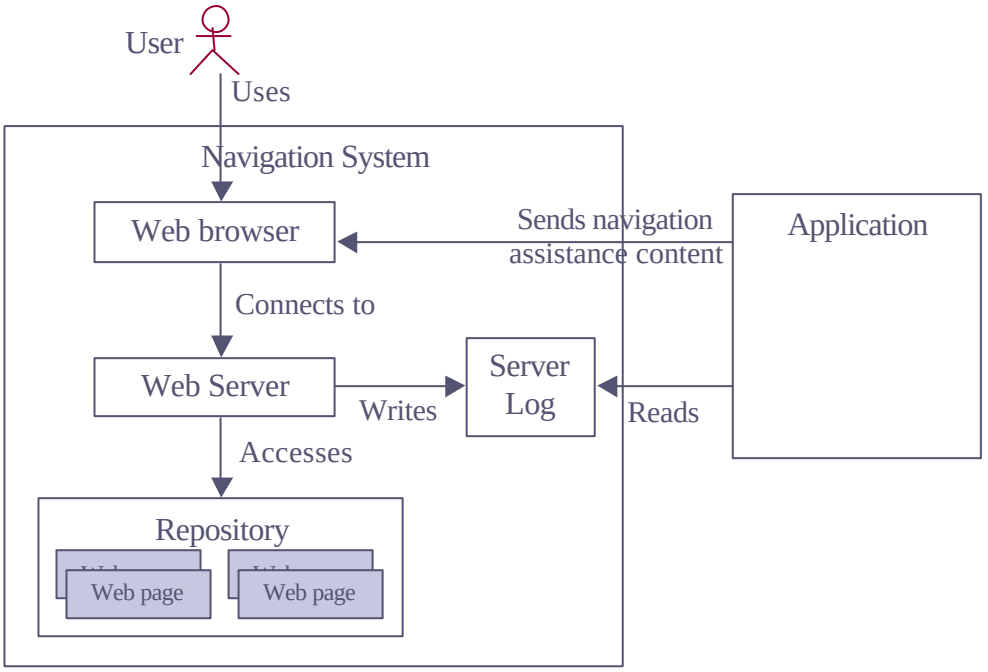
6.1 Choice of Navigation System

It has been chosen to operate on a repository of web pages (e.g. a web site). The core of the Navigation System is then a web server. Users simply navigate the repository with a web browser connecting to the web server. This configuration has been chosen for several reasons. First it is the most common configuration in which users experience navigation. Then it is simple: the technology is very familiar and the resources are clearly identified. Lastly, it is easily suitable for carrying out tests in a real context of use: for example the application could be layered over the actual website of our school.

The Access Data is then in the form of a server log. Server logs classically keep record, for each access to a web page, of the IP address of the user, the URL of the web page and the timestamp of the access. User identifiers are then IP addresses while resource identifiers are URLs.

However, some constraints must be fulfilled for enabling the application to work correctly. First, users must have static IP addresses and connect from one same computer. Then, the web pages should not be dynamically generated as each should have a distinct URL. Although these constraints may not be acceptable in real use, they are suitable for an experimental platform. In different contexts, users may be asked to log onto the application for being

identified for example.



The Navigation System

6.2 Consequences on the application

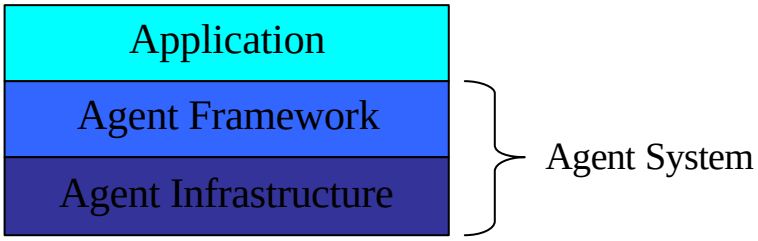
The Access Monitor agent is a "web log monitor". It senses its environment by reading the lines in the web log.

Another consequence is the way navigation assistance is provided. As the user's access program is a web browser, a possibility is to dynamically modify the web page viewed by the user in order to incorporate the assistance data in the form of additional web content. For example, there could be an additional frame on top of the original web page. This should be done at the level of the web server. An alternative is to use a dedicated plug-in for the web browser. Assistance is then displayed in a specific part of the browser window. In all the cases, the nodes of the assistance graph can be actual web anchors that can be clicked for immediate navigation. However, this issue has not been fully investigated so no decision has been made yet. In a first development, the application can just display assistance in a text format on the application's machine.

Chapter 4: Implementation

1 Implementation strategy

The application has been designed as a multiagent system. Agents need an infrastructure that enables them to live and interact. On top of that layer, an agent framework provides an agent design and tools for implementing agents. Both layers together can be called the Agent System.



Implementation layers

Therefore, the Agent System must be obtained first. Generic, mature agent systems already exist and are freely available. However, a constraint on the application is that it runs under the Microsoft .Net environment.

A good solution for rapid prototyping is to use an already existing platform and tools. However, as .Net is very recent, no agent platform has been found that has been created for .Net. Almost all the existing agent platforms are implemented in Java. A strategy has thus to be chosen for solving the problem.

- A first strategy consists in keeping the Java code and try to run it under .Net. The J# plugin for VisualStudio makes it possible to execute J# code under .Net. Microsoft J# is very close to Sun's Java language, however it is not compatible with features later than JDK 1.1.4. Among the agent platforms, only FIPA-OS is provided as a JDK1.1.x-compliant version. Although its source code is compatible with J# .Net, it makes use of 7 external Java libraries (e.g. Xerces, Swing, Java2 Collections) that have to be incorporated as source code as well, which sometimes requires to decompile bytecode (e.g. with JAD [79]). Finally, in addition to decompiling problems, it appears that such a .Net project cannot work because of the absence of RMI in J#.

- Another strategy is to convert all the Java code to C# using Microsoft's JUMP pack. However, the current version of JUMP is Beta 2 at this time and it is not complete enough to provide a satisfactory solution.

- A third possibility is to make .Net cooperate with a Java Virtual Machine at run-time. This can be done by using web services and exchanging XML or by wrapping Java objects into COM components by the mean of tools like J-Integra [80]. Nonetheless, in addition to being complicated, this solution allows only for the exchange of data or objects and not the reuse of classes.

As none of the strategies was satisfactory, it was decided to develop a simple agent platform and agent framework from scratch for .Net. C# was chosen as the implementation language. It is indeed a modern OO language that has been designed especially for .Net. At the same time, it is close enough to Java so that learning it is a rather short phase.

I designed and implemented the whole Agent System, then a big part of the application.

2 Agent infrastructure level

The infrastructure enables agents to find each other and communicate by exchanging messages. The application can involve a number of agents that grows with the number of users and resources used. There is thus no threshold about the number of agents, which is not acceptable if the application is located on one single machine. For avoiding machine overload, an important characteristics of the infrastructure is that it supports the distribution of agents among several machines or hosts.

2.1 Agent communication

All inter-agent communication is achieved through exchanges of messages. Every agent has a unique identifier that is specified in message "addresses" for proper delivery to the recipient.

In the ideal case, agents communicate by the mean of dedicated communication language and content language. Information is text-based and agents use an interpreter and an ontology for interpreting it. More heavy-weight data structures like objects are confined inside the agents. Thus agent encapsulation is preserved, only "high-level" information is transmitted and interoperation between heterogeneous agents is eased.

In our case however, developing such a system would conflict with the objective of rapid prototyping. Our agents are homogeneous and are designed specifically for cooperating. Thus a simple communication system has been designed and implemented based on messages holding a subject and a content object.

More precisely, messages are composed of the following data:

- The ID of the intended recipient agent for allowing the posting service to deliver the message;
- The ID of the sender agent in order to enable the recipient to send a reply;
- A subject for allowing the recipient to identify what the message is about. In other words, the subject defines a message category;
- The nature of the message: namely notification or request;
- Possibly some additional data as a content that the recipient should process. Such content can be any object, thus it is important that it is not part of the sender's internal state otherwise it would break the sender's encapsulation. Alternatively, the content object can be passed by value or serialized.

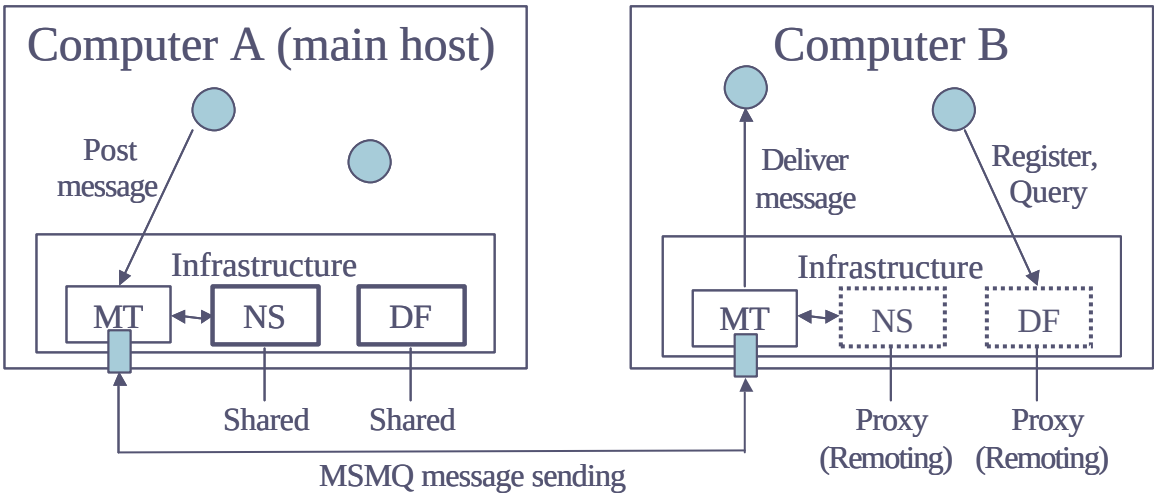
2.2 Agent management

Following the principles of the experimental FIPA agent management specification, the infrastructure is composed of 3 services.

(1) A Name Server (NS) maps every agent identifier (ID) with the address of the host on which the agent runs. Every new agent gets a unique identifier and must register to the NS at initialization time. This allows for the proper delivery of messages to agents. Handling agent IDs and hosts independently is better in terms of design, and it is also necessary for enabling agent migration in the future.

(2) A Directory Facilitator (DF) enables agents needing a certain service to get the ID of agents providing the service. In other words, it enables service consumers to find service producers. A service is any task that an agent can do. The DF brings flexibility (even dynamic) in agent organizations. In contrast, agents could hold simple references to the producer agents that they need, but this would not be very suitable in a distributed configuration. For example, if several agents on different hosts are able to provide the same service, it is better that a consumer queries the local producer or a producer on the least loaded host.

(3) Message Transporters (MTs) are in charge of delivering messages. There is one MT per machine. In the general case, an agent willing to communicate with another agent creates a message, fills the "recipient" field with the ID of the second agent then posts the message to the local MT. The MT knows all the local agents so if it finds that the recipient is local, it delivers the message to it. Otherwise, it gets the address of the recipient's host thanks to the NS and forwards the message to the MT of that host.



The agent infrastructure

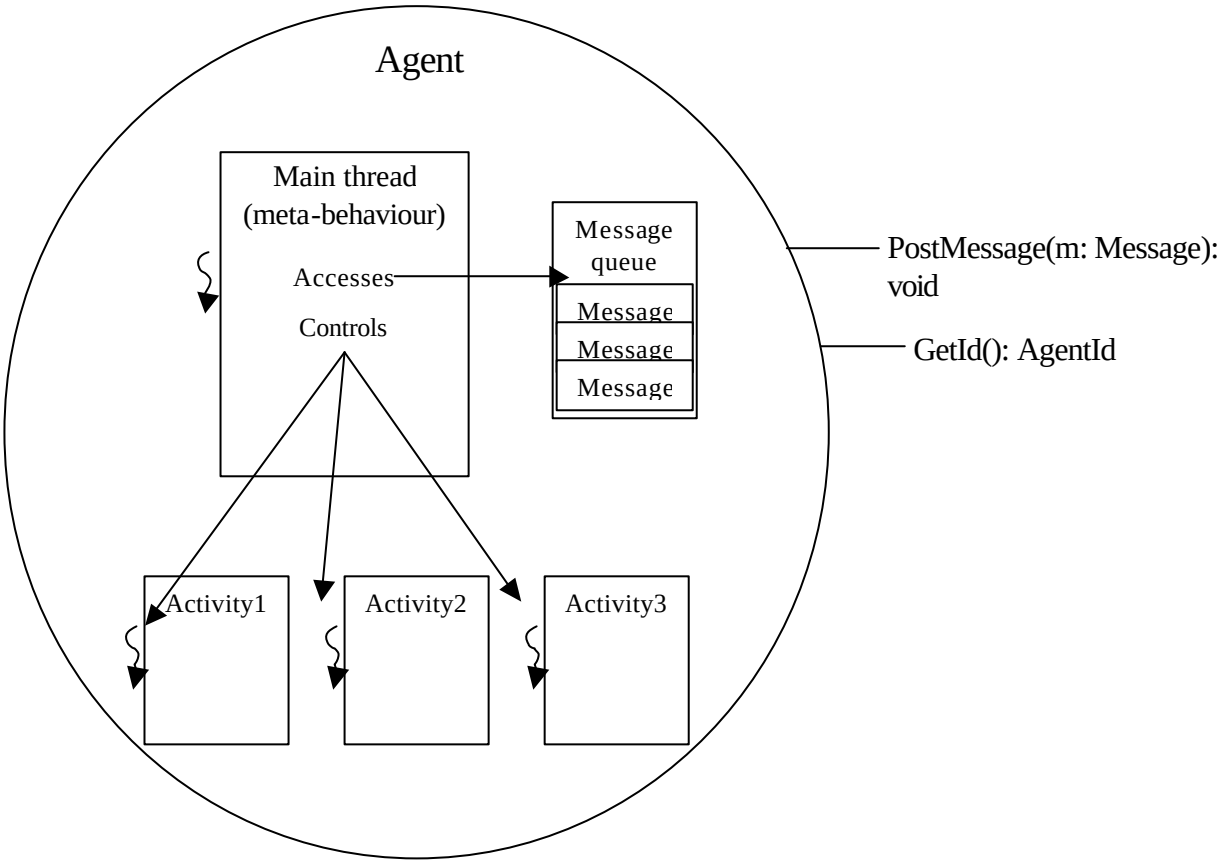
The NS and DF must be unique within an agent infrastructure. Therefore, they must be instantiated on one machine only, which is called the Main Host. The NS and DF can then be accessed from other hosts using synchronized remote method invocation. This has been implemented using the facilities provided in the .Net's Remoting namespace.

In contrast, each host has its own MT. Implementation for remote inter-MT communication is based on the MSMQ (Microsoft Message Queuing) service. It allows for asynchronous message sending with high-level administration facilities. Each agent message is simply embedded in a MSMQ message.

3 Agent framework level

3.1 Agent design

For its external world, an agent is simply an entity that has the ability to receive agent messages. An agent must also have a unique identifier, typically a string. Thus the interface that agents comply with simply specifies a method for posting messages and a method for getting the agent's ID. The inner structure is presented in the following figure and explained below.



Agent structure

3.1.1 Main thread and activities

Inspired by the DIMA design, the agents are active objects in order to allow for their pro-activity. Thus they own a main thread that defines the meta-behaviour of the agent. The behaviour level is composed of different activities that are threads having their own data and methods. Activities allow for the modularity of the agent's behaviour. The main thread defines the meta-behaviour as it controls the activities. For example, it decides when to initiate an activity and when to suspend or stop it if necessary.

For consistency, all the activities must be dependent on the existence of the agent. When an agent terminates, all its activities and threads must terminate as well. This is achieved by keeping weak references to all the internal threads of the agent. If a thread is still alive when

the agent terminates, the thread is terminated as well. The .Net framework supports concurrency facilities for ensuring that no inconsistent state is reached.

When the framework is extended, it can be appropriate to model the agent's meta-behaviour through a statechart diagram.

3.1.2 Message queue and strong encapsulation

Because of its strong encapsulation and unpredictable autonomy, an agent should not react synchronously to external events or the arrival of messages. Thus external threads should not be allowed to invoke the agent's methods directly. Such a thread would possibly modify the agent's internal state independently from the agent's will.

Instead, agents react to incoming messages asynchronously. When a thread accesses an agent by posting a message, the message is just stored in the agent's message queue. Then the message is processed whenever the agent decides so. The message queue has the role of an interface between the agent and the external world.

3.2 Synchronized inter-activity communication

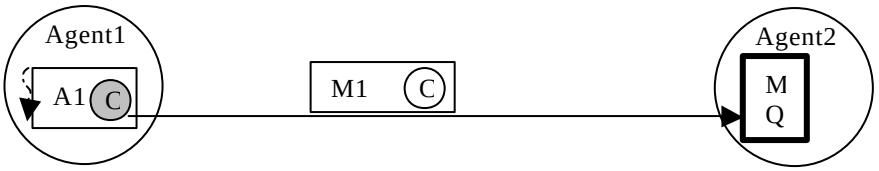
Because agents work asynchronously with their external world, it is necessary to provide some synchronization mechanisms for allowing them to interoperate easily. For that purpose, agent activities can use conversation objects.

A conversation defines a specific context in which messages can be exchanged. This allows for synchronized and asynchronous requests/replies between activities from different agents. The synchronized case is similar to method invocation: the thread of the activity is blocked until a reply is obtained. The asynchronous case allows the activity to do some work while expecting the reply.

The agent whose activity creates a conversation is the initiator. The conversation is transmitted to another agent as an embedment in a standard message. The message is processed normally by the recipient agent through its message queue. However, when the message is handled by an activity, the conversation can be obtained from the message for sending new messages in the context of the conversation. The conversation is then considered as handled by the agent.

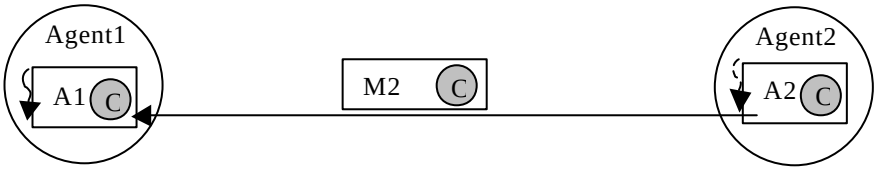
The implementation is based on C# events. The principle is that when a conversation is handled by an agent, incoming messages that have been sent in the context of the conversation bypass the agent' message queues and are obtained directly by the handling activity through an event handler.

The figure below summarized the steps of the use of conversations.



Step 1: Activity A1 of Agent1 creates a conversation C which is immediately considered as handled by Agent1 (grey color). A message M1 is sent by A1 to agent Agent2 in the context of C. Thus A1 expects a reply in the context of C (dotted thread arrow) and a clone of C is embedded in M1. The clone is not considered as handled by Agent2 (white color), hence it is normally stored in Agent2's message queue MQ.

Step 2: An activity A2 handles M1 in Agent2 and gets the non-handled conversation.



Step 3: A2 sends a reply M2 to Agent1 in the context of C. C is thus considered as handled by Agent2. M2 is obtained directly by A1, bypassing Agent1's message queue, since C is handled by Agent1. As C is considered handled by both agents, A1 and A2 can do synchronized and asynchronous exchanges of messages.

Principle of conversations

A conversation finishes in two cases. First, it is possible to send a message and do not expect any reply. All replies will be ignored since the messages are not stored in the agent's message queue. Secondly, a conversation can be simply closed, which has the same effect.

4 Application level

4.1 Overview

The figure at the end of this section shows an overview of the agents that have to be implemented and the status of the implementation. I did part of the implementation, another part has been affected to another team member and a last part has not been implemented yet. Below is the description of the part I have implemented.

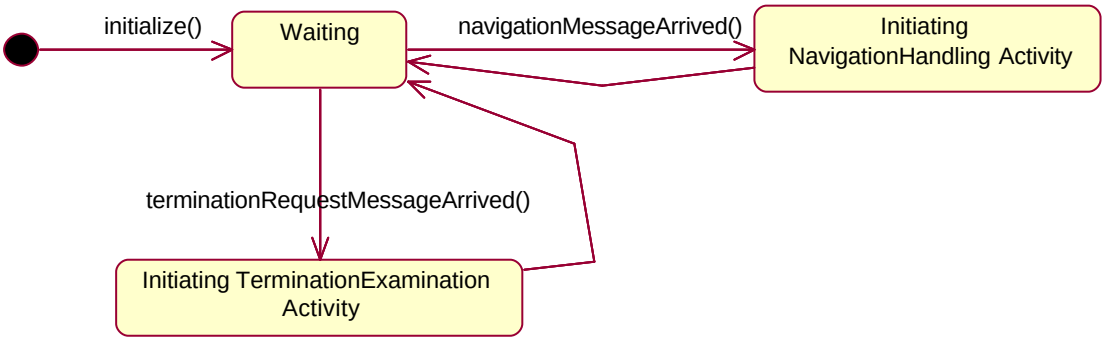
4.2 User Agent Proxy

The singleton User Agent Proxy receives 2 kinds of messages:

(1) navigation messages from the Web Log Monitor that are notifications of accesses. This is for notifying User Agents;

(2) termination request messages from User Agents. This happens when a User Agent considers that the user it handles has stopped using the Navigation System after a certain time. The User Agent Proxy decides whether to give termination authorization or not.

Therefore the behaviour of the main thread of the User Agent Proxy is described as follows:



Statechart diagram of the main thread of the User Agent Proxy

Each time a new message arrives, a new activity is created for handling it. This is because a lot of messages are likely to be sent to the User Agent Proxy. It was thus chosen to perform concurrent computation of the messages.

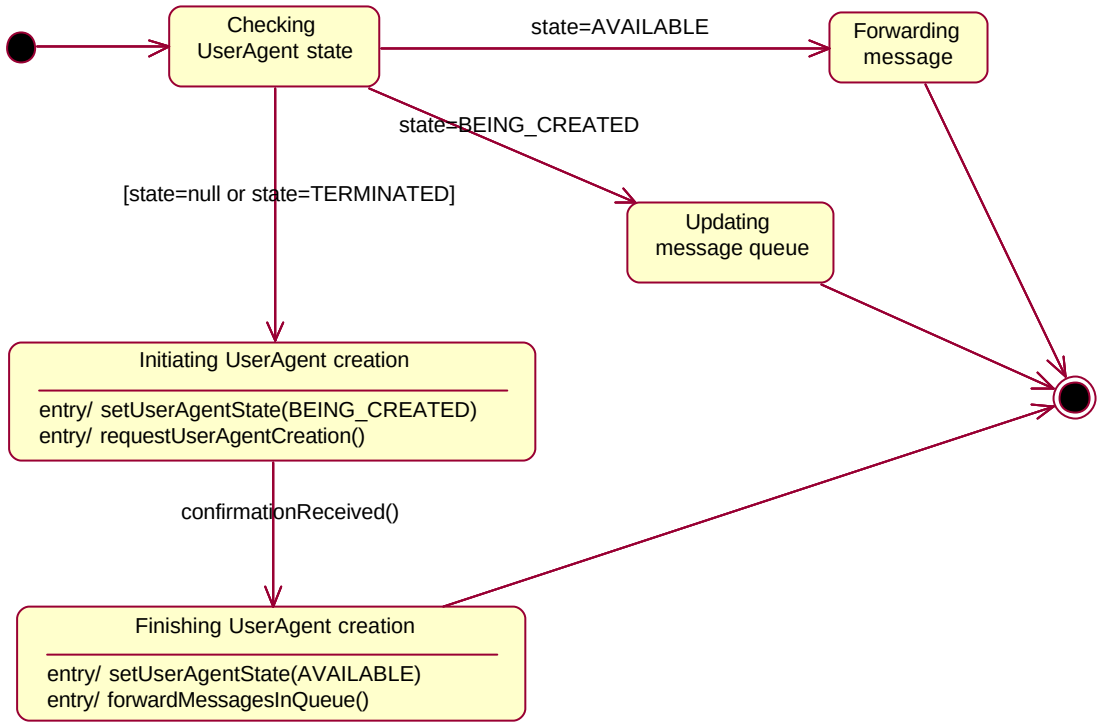
4.2.1 Navigation message handling

When a notification about an access is received as a navigation message, 3 configurations are possible depending on the state of the User Agent that handles the user of the access:

- The User Agent is currently available: the message is forwarded to it.
- No User Agent handles the user: its creation is requested to the User Agent Maker agent. An acknowledgement is received in the context of a conversation when the User Agent is ready to handle messages. In the meantime, a temporary queue is created for keeping all the messages that should be handled by the User Agent. When

the new User Agent is ready, all the messages in the temporary queue are forwarded to it.

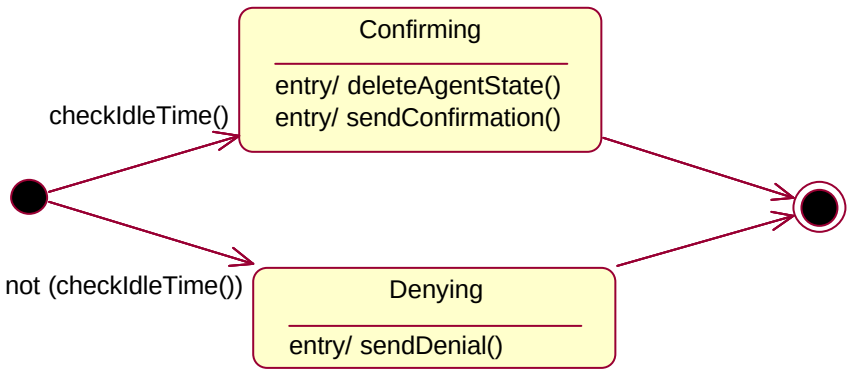
- The User Agent is being currently created: the message is pushed into the temporary message queue.



Statechart diagram of the NavigationHandling activity

4.2.2 Termination examination message handling

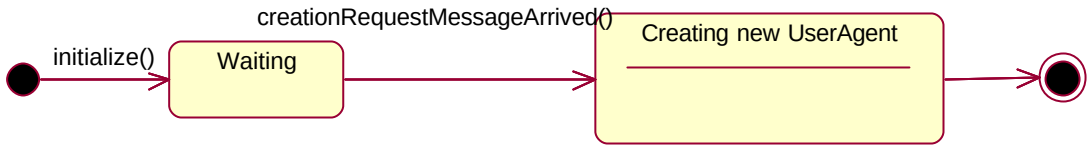
For being sure that no message is lost when a User Agent wishes to terminate, the User Agent Proxy keeps the timestamp of the last message sent to the User Agent. If the time elapsed is sufficient for being sure that no message has arrived since the User Agent's decision to terminate, the User Agent Proxy gives authorization for termination.



Statechart diagram of the TerminationExamination activity

4.3 User Agent Maker

This agent has the ability to act as a factory for building User Agents. Although the current version is very simple, creating new empty User Agents, further versions will have to check if a user profile already exists and provide the possibility to create a User Agent with an initial user profile associated with SUPs.



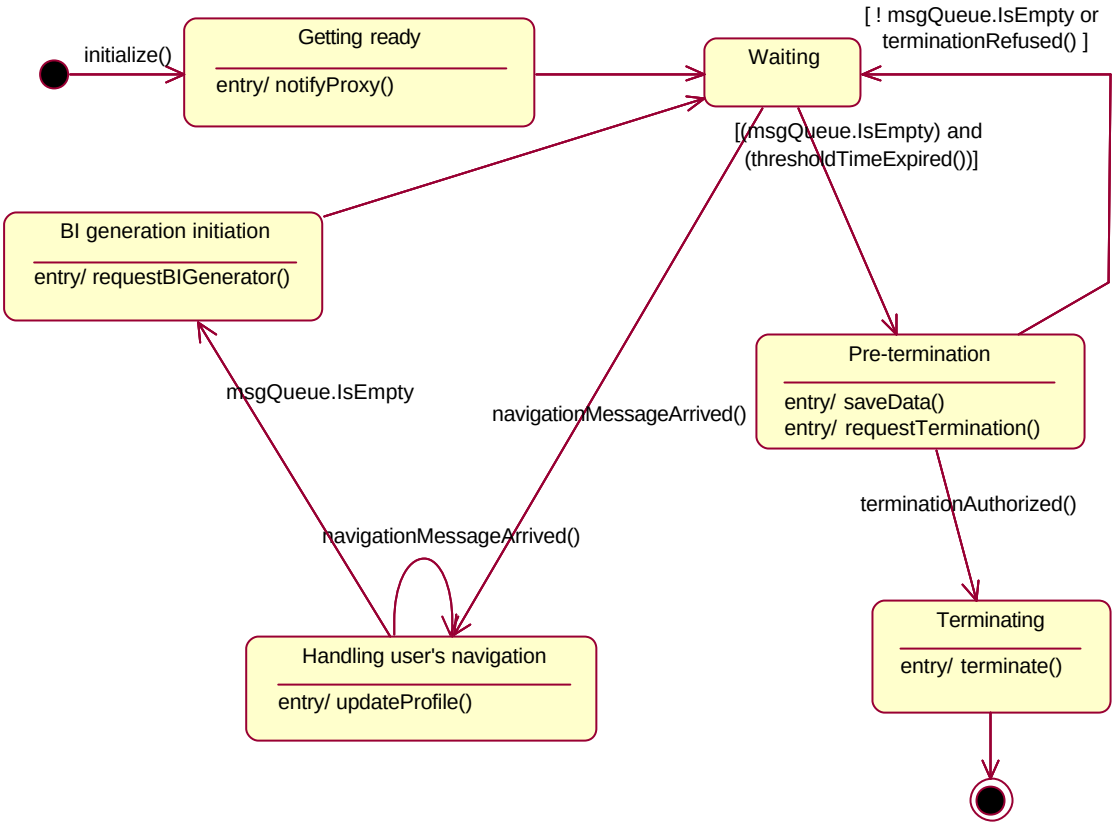
Statechart diagram of the User Agent Maker

4.4 User Agent

4.4.1 General behaviour

A User Agent is notified by the User Agent Proxy of accesses by the user it handles. This allows for the maintenance of a user profile. Each time a notification has arrived, the User Agent updates its user profile. Then if a new notification has arrived, it means that the user keeps navigating thus there is no point in generating navigation assistance (called BI for Business Intelligence). The user profile is just updated again. Otherwise, the user is examining a web page so a BIGenerator agent is requested by the User Agent to generate navigation assistance.

Another point is the decision that the User Agent can make to terminate if the user does not navigate any more. This is for situations where the user has gone to sleep or logged off his computer for example. The issue here is to determine the threshold idle time after which the agent decides to terminate. It requires to study statistics about users' behaviour and elaborate a specific algorithm. The current implementation just uses an arbitrary threshold of 30 minutes.



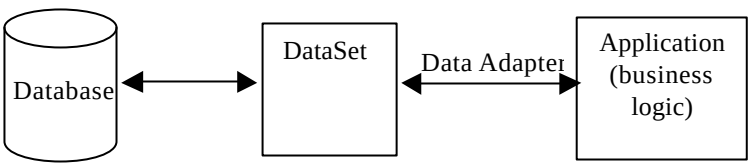
Statechart diagram of a User Agent

4.4.2 User profile management

User profiles hold relevancy indicators as described before. A timestamp of the last access of the user is also kept for knowing how long the user has not accessed the repository.

As all the profile data has to be persistent on the long term, is potentially big and is independent of the existence of User Agents, it is maintained in a database. The database is a SQL relational database handled by MS SQL Server. The detail of the design of the tables is presented in Appendix B.

For managing databases, .Net provides facilities through ADO (Active Data Object) .Net. The strong point of this technology lies in its intermediate data layer, the DataSet, between the database and the business logic.



Instead of managing connections to the database and handling database-dependent commands, the business logic only manipulates the data in the DataSet. The DataSet is initially filled with data from the database but it is disconnected. Only on demand does it connect to execute commands like update on the database.

Thus the profile, node and link classes for user profiles and SUPs encapsulate ADO code. User Agents just keep a reference to their user profiles whose data belongs to the DataSet. As the DataSet is separated from the business logic, the User Agents are virtually stateless. This point can become significant for further evolutions like agent mobility.

4.5 Business Intelligence Component

Simple pathways are not high-level information. However the obtention of series of [user, resource, timestamp] makes it possible to compute relevancy indicators allowing for the inference of semantic links between resources. Simple relevancy indicators are for example the number of traversals between two resources and the "age" of the last traversals that can act as a "moderator" since an old last access may mean that the destination resource has been removed from the repository.

Also, the time spent by the user "using" (reading / watching / listening to) a resource that is the destination of a link reveals the interest of the resource in a context defined by the origin resource of the link. However the calculation of this "use time" is not trivial. A user can interrupt his navigating activity because some event has interfered like a phone call, and start navigating again a moment later. It is impossible to distinguish between this case and a long period of use of the resource indicating a strong interest. There is also a time when the user simply stops navigating and starts a different activity. Detecting this situation may appear as more feasible than the previous one: a threshold period can be set for deciding that the user has gone or that he has kept using the resource for a long time. Nevertheless, such a threshold period is arbitrary. An approach like the profiling technique described in [15] is suitable for homogeneous resources but not in the general case. The intelligent determination of a threshold period requires investigation on statistics about usage. Such work is being carried out at Monash in parallel to our project. Typically, our application is a tool that can be used for easing this work.

As a first approach, a simple algorithm is proposed for the elaboration of navigation assistance although it has not been implemented. The data the algorithm can use comes from 3 perspectives: the user's profile, the associated SUPs and general popular links from the current resource. The simplest strategy consists in computing a relevancy indicator for each of these perspectives. This leads to edges holding 3 different relevancy indicators in the assistance graph, that can then be graphically represented through parameters like the length and thickness of arrows, or even explicitly through textual weights. As for the depth of the graph, it can be limited to 1. More sophisticated algorithms will handle a bigger depth. The problem is thus reduced to the computation of a relevancy indicator or weight for each perspective.

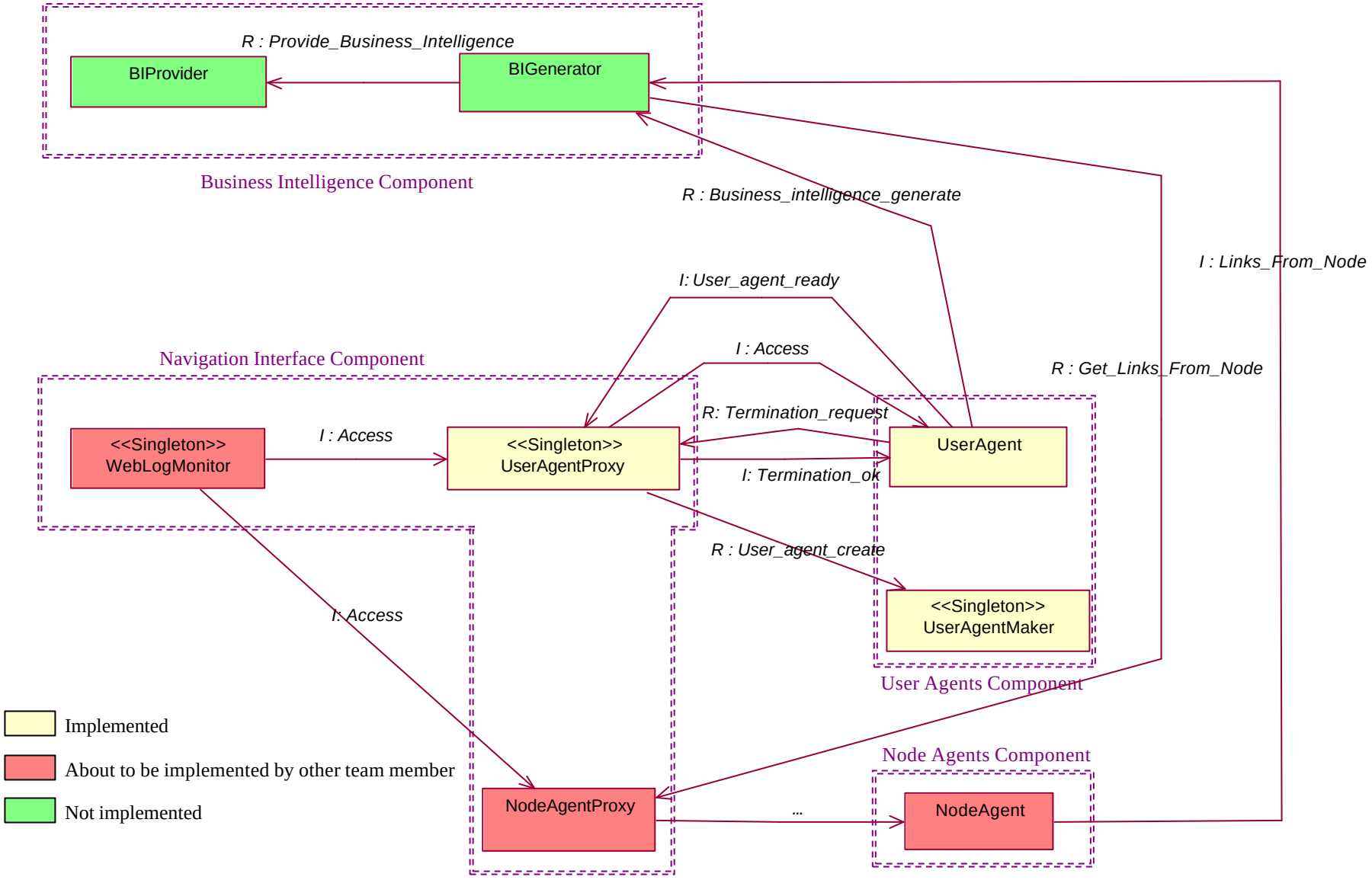
(1) In the case of SUPs, the current implementation already defines a weight that is an integer. Thus if a link, originating from the current resource, is in several SUPs associated with the user, the final weight of the link in the graph can just be the sum of the weights.

(2) General popular links provided by the Node Agents are links whose origin is the current resource. The links hold a number of occurrences of traversals and a timestamp of the last access. Given an arbitrary integer N and threshold date D, the

algorithm can be: select among the links those whose last access is more recent than D (for filtering out out-of-date links). Then among the remaining ones select the N ones that have the more occurrences. The weights are the number of occurrences. Again, an issue is: how are those arbitrary values intelligently chosen? Number N is used to ensure a maximum number of links when formatting the graph, but many other strategies can be followed, for example: select all the links that have more occurrences than number O, or select those that have a bigger number of occurrences than the average.

(3) The user profile provides the same kind of data as general links, except that it is targeted at the user and includes use time data. Thus the same algorithm can be used as in (2) except for the computation of weights as they now depend on both the occurrences and the use time spent on the destination resource. Since the use time is error-prone, priority can be given to the occurrences. The fact that the same user traversed a link several times clearly indicates that the origin and destination resources are related so a semantic link can be assumed. Then among links with the same number of occurrences, those with the biggest use time can be selected. The computation of a weight may arbitrarily be the number of occurrences plus the use time in minutes.

The algorithm described is simple but rough. Elaborating a good algorithm is a wide field of investigation since there are numerous possibilities. Getting feedback through the application about repository usage and working on statistics on this data is probably necessary before thinking about developing a satisfying algorithm.



Agents of the application and nature of the messages exchanged

Chapter 5: Future Work

This is the beginning of a vast project, thus numerous points are still to investigate.

1 Application outputs

First, the algorithm for navigation assistance must be implemented for enabling testing. But more importantly, a good algorithm should be developed which is far from being trivial. The pathways collected by the application are raw data that can be studied from a statistical point of view. Such a work is probably necessary since the development of the algorithm requires some knowledge about how the repository is used. The parameters of the algorithm are thus bound to depend on the repository and the kind of resources and users involved.

Then formatting the navigation assistance and actually delivering it to the user is another matter. A type of graph representation has been proposed but more work is required for a precise design. Furthermore, the formatting depends closely on the output of the navigation assistance algorithm. The delivery of the assistance can occur through a plugin of the user's navigator or by direct alteration of the web pages. In that case, a separate HTML frame may be added at the top of the web page. The graph displayed should allow for direct navigation by clicking on the vertices like on anchors.

While navigation assistance is for users, administrators and content providers also need facilities for getting feedback from the application on the usage of the repository. Tools for discovering communities of users and categories of content, creating SUPs based on them and refining the SUPs over time are needed.

2 Explicit user inputs

Although the input system of the application has been designed to be mostly transparent to users, the effective use of SUPs may require explicit inputs in certain cases, first from experts and secondly from some user.

SUPs might be created by administrators as well as simple users who declare to be experts in a domain. The application should therefore allow experts to identify themselves as such and create new SUPs. In that case SUPs could just be generated by monitoring experts' pathways. Which control would experts have on their SUPs? How would they indicate different relevance weights on the links of the SUPs? Experts should be allowed to refine and improve their SUPs over time. It may also make sense to allow communities of experts to improve SUPs as a team work. In that case, an identification system would be required. For example, an expert could create a SUP and associate a password with it so that his colleagues can later modify it using the password.

As for "simple" users, it is certainly a good point that they do not have to worry about SUPs if they do not want to. When they use the Navigation System for the first time, an empty user profile is created for them. SUPs can possibly be associated to their profile later if their pathways indicate an interest in the domain of a SUP. However, it may as well be very useful

to enable users to explicitly specify some SUPs that should be associated with their profile. A new user may wish to get assistance navigation about kangaroos although his profile is still empty. Thus the application may have to display currently existing SUPs and let users choose the ones they think are suitable for them. This is equivalent, to some extent, to allowing users to edit their user profiles.

3 Validation of the approach

The application is an experimental platform for testing if the "minimalist" approach that has been chosen can prove effective. Therefore it is necessary to carry out tests in a real situation. This can be done on any web site like the one of the CSSE. Thanks to such tests, it should be possible to determine if simple pathways are enough for providing assistance, or if it is essential to observe other activities of users. For example, in the case of web pages other activities could be downloading or bookmarking: existing recommender systems already handle such activities. However, the generic nature of the approach would then be lost, since navigation is the only activity that is generic to every kind of repository.

4 Genericity checking

The application has been designed in the case of a web site as a repository. Furthermore, some constraints have been assumed like the use of static IP addresses and the fact that every user uses his own computer. This makes it possible for the application to be entirely transparent for the users. However for a generic, real-life use it will probably be necessary to identify users another way, through a login system for example. Similarly, there can be problems for identifying resources in the case of dynamically-generated web pages for example.

Furthermore, the generic nature of the approach must be tested on other kinds of repositories. For example, a relational database raises some other issues. In particular, it is necessary to specify explicitly what a resource is.

5 Application implementation

For allowing for the actual distribution of User Agents, a problem due to the use of ADO .Net must be addressed. DataSets, that are the intermediate data layer, are serializable. It means that they cannot be marshaled by reference: when a DataSet must be used on a second host, a copy of it is always sent to the host. Therefore, if there are User Agents on different machines, they will alter different instances of the DataSets. So the modified DataSets have to be merged before updating the database in order not to lose changes.

As an alternative implementation, it could be interesting in the future to consider distributing some agents to the users' machines. As the User Agents are virtually stateless (as long as they can communicate with the profiles) they could reside on the users' machines, which would solve the identification problem. In addition, they could be accompanied by a navigation assistance generator that would therefore do all the computation on the user's machine.

6 Improvement of the agent system

The agent system is rather simple so it can be extended in many ways.

First, a real text-based communication language and a content language can be used for a cleaner communication.

Then some performance issues could be considered. A multiagent system involves a lot of multithreading. Sometimes it is better to create new threads, sometimes not. Furthermore, distribution raises the problem of load balancing. The directory facilitator could be improved in order to select providers based on criteria like the current load on the agents' host. It can even be thought about allowing for mobile agents for dynamic load rebalancing. The agent system should not be too hard to extend in that direction. Real tests should make load problems clearer.

Lastly, our current agents are "lightweight" in the sense that their autonomy and "intelligence" are limited. Nevertheless they will certainly evolve in the future and get more and more "intelligence" as the application and its algorithms get more sophisticated. It could then be useful to provide higher-level layers in the agent design. For example, explicit support for agent states and transitions could be added for a direct implementation of state diagrams, or a Prolog-like inference engine for defining a rule-based behaviour.

For the time being, these issues, particularly those from subsections 1 to 3, are being actively researched in the LEOPARD project.

Conclusion

A “minimalist” approach has been chosen for the elaboration of assistance to repository users, administrators and content providers, since it is exclusively based on the observation of users’ pathways. This approach allows for the design of an agent-based application that is generic, dynamic and transparent for the users. An agent system has been developed for the .Net platform, then an experimental platform was partly implemented on top of it. This platform will make it possible to validate the approach and will be a base for research about algorithms for the generation of assistance and the extraction of feedback. This work is the starting point of a vast project that is at its early stage and currently keeps moving forward.

Index of terms used

Access: access of a resource by a user at a certain timestamp for use. Use can consist in reading, viewing, listening depending on the nature of the resource, through a specific program on the user's machine.

Access facility: mean for resource discovery by a user. Examples include indexes, search engines, web links.

Actor: human person concerned with the use of a repository. Can be a user, an administrator or a content provider.

Administrator: person in charge of maintaining a repository, which includes providing access facilities to the users.

Content provider: person who is the author of a resource.

Expert: user who has expert knowledge in (1) a domain or topic, and (2) the resources related to this domain/topic in a repository.

Link (if not qualified): virtual relation between the resources of two consecutive accesses in a pathway. The resource of the first access is the origin and the resource of the second access is the destination.

Navigation: activity of accessing resources performed by a user.

Pathway: chronologically ordered sequence of accesses from the same user.

Repository: set of resources that can be accessed. For example: a database, a web site.

Resource: piece of data or content that can be accessed as a whole by a user.

Traversal: action of accessing the destination resource of a link after its origin resource.

User: person who navigates one or several repositories for satisfying an interest in a topic.

Appendix A: Addendum on Industrial Issues

Learning objects

1. The promises of e-learning

Traditional educational material, typically books, is expensive for users like school pupils or students. It is all the worse as users have to pay for material that they generally do not fully use. On the other side, it is a profitable industry for publishers and sellers. For example, the American National Association of College Stores estimated the combined American and Canadian college store sales to be $8.959 billion for the 1998-99 academic year [81].

This situation facilitates the development of online teaching and learning as an alternative. In addition to obviously removing location constraints, online courses holds many promises among which a cost that is lower than that of traditional material.

A study published in February 2002 by Apex Learning called “Online Courses and Other Types of Online Learning for High School Students” [82] reveals that cost effectiveness is one of the top reasons for the adoption of online courses.

Not surprisingly, this study also shows that e-learning becomes more and more widespread. 40 percent of U.S. high schools are currently using online courses or are planning to start using them during the 2001-2002 school year. In addition, “another 17 percent are interested in offering online courses in the future”. At the level of public school districts, 32 percent will adopt and use an online learning platform for the first time in 2002. Therefore Apex Learning’s CEO Keith Oelrich concludes that "in just a few years, online courses are quickly becoming an integral part of the high school experience".

In addition, online learning is much more developed in higher education. The Institute for Higher Education Policy estimated in 1999 that 85 percent of American four-year colleges would offer courses online by 2002 [83].

However, Downes indicates in [2] that developing courses from scratch leads to a cost varying from $4,000 to $100,000 (Canadian dollars). When delivered to a small number of students, it may result in course fees that are comparable with fees for traditional courses. This is because the possibilities of online material are not fully exploited.

2. The necessary emergence of learning objects

A strength of online material is that is can be shared. Sharing material between universities should allow for sharing the costs and getting a big number of students involved, therefore reducing the fees. However, Downes points out that, although the cost saving should be

tremendous, this principle does not work because what is shared is courses [2]. He argues that courses are generally too specific to satisfy the needs of different universities and teachers.

Instead, small chunks of educational content that are designed for being reused as components of courses – learning objects – appear as a solution.

This new approach for online material and course building still has to prove suitable. However, Downes claims that the economics of sharing learning objects are relentless:

> "It makes no financial sense to spend millions of dollars producing multiple versions of similar learning objects when single versions of the same objects could be shared at a much lower cost per institution. There will be sharing, because no institution producing its own materials on its own could compete with institutions sharing learning materials."

Learning objects are thus a natural paradigm according to economics.

Learning objects are indeed being adopted with gusto, as proven by the Australian Le@rning Federation. Other Australian initiatives to establish learning objects repositories include the following:

- The Building the Internet Workforce Project [84]. It is funded by organisations such as SUN Microsystems [85], Telstra [86], Compuware [87] and DSTC [88]. It aims at creating a set of learning objects for education in IT (Information Technology).

- Learning Resource Exchange [89]. It aims at developing and maintaining a national database of metadata about learning objects to support discovery and re-use. This project was originally funded by the DEST (Commonwealth Department of Education Science & Training) [90].

- Peer Review of ICT (Information and Communications Technology) Resources [91]. It is funded by DEST and aims at developing "conceptual and procedural bases for a national scheme for independent and expert peer-review of ICT-based teaching resources."

Despite these proofs of success, some issues about learning objects still need to be researched. Are they really a viable approach? How should they actually be created, managed, used? The study of the effective use of complex repositories fits into this context. Learning objects can indeed be highly heterogeneous resources and exist in big quantities, considering for example that SchoolNet already holds over 5000 learning resources. Tackling this issue is quite urgent as industry is ready to adopt and invest on learning objects.

Appendix B: SQL Tables Design

1 Table UP (User Profile)

Name	Key	Data type	Size	Allows nulls
userId	yes	varchar	50	no
lastUse	no	dateTime	8	no

2 Table UPNode

Name	Key	Data type	Size	Allows nulls
identifier	yes	uniqueidentifier	16	no
resourceId	no	varchar	50	no
UP	no	varchar	50	no

3 Table UPLink

Name	Key	Data type	Size	Allows nulls
identifier	yes	uniqueidentifier	16	no
originNode	no	uniqueidentifier	16	no
destinationNode	no	uniqueidentifier	16	no
lastUse	no	datetime	8	no
occurrences	no	int	4	no
useTime	no	bigint	8	no

(4) Table UP_SUP (matches User Profiles with SUPs)

Name	Key	Data type	Size	Allows nulls
UP	yes	varchar	50	no
SUP	yes	varchar	50	no

(5) Table SUP

Name	Key	Data type	Size	Allows nulls
SUPName	yes	varchar	50	no
description	no	varchar	50	yes

4 Table SUPNode

Name	Key	Data type	Size	Allows nulls
identifier	yes	uniqueidentifier	16	no
resourceId	no	varchar	50	no
SUPName	no	varchar	50	no

5 Table SUPLink

Name	Key	Data type	Size	Allows nulls
identifier	yes	uniqueidentifier	16	no
originNode	no	uniqueidentifier	16	no
destinationNode	no	uniqueidentifier	16	no
weight	no	int	4	no

Appendix C: Conference Paper

A paper has been submitted to ICITA 2002, the First International Conference on Information Technology & Applications, Bathurst, Australia, 25-29 November 2002 (http://odysseus.mit.csu.edu.au/icita2002.html).

At this time, a short version of the paper has been accepted and the following full version is in the acceptation process.

Deriving Action-based Semantics from Learning Repositories

Olivier Constant*º, Christine Mingins*, Annya Réquilé-Romanczuk*º and Brian Yap*,

http://www.csse.monash.edu.au/projects/LEOPARD

**** School of Computer Science and Software Engineering,***
Monash University, PO Box 197 Caulfield East, Victoria 3145, Australia
º Ecole des Mines de Nantes, La Chantrerie, 4 rue Alfred Kastler, BP 20722,
44307 Nantes Cedex 3, France

***Abstract*-- Motivated by recent developments in Category theory, we have designed a generic virtual layer that overlays repositories of learning objects. Agents embedded in this layer observe traversals from both the repository and the user perspective, and support the inference of dynamic semantics based on actual usage. We will experiment with the dynamically generated metadata with the goal of enhancing users' navigation and discovery experiences.**

***Index Terms*--Multi-Agent, Learning Objects, Intelligent environment, User pathway, Recommender Systems.**

I. Introduction and Motivation

The learning community has adopted the idea of repositories of learning objects with gusto. In Australia, for example, "… all States, Territories and the Commonwealth of Australia are collaborating in this Initiative-The Le@rning Federation-to generate, over time, online curriculum content for Australian schools. " Our concern is that coded metadata, hard indexes and search mechanisms will provide insufficient support for content users to explore and discover useful materials in very rich and complex repositories. We have designed a generic, virtual layer that sits over repositories and collects information about users' traversals. Inspired by the Prototype Category theories of Elinor Rosch [11] and George Lakoff [12], we intend to experiment with the information derived from these traversals to attempt to infer action-based semantics about the repository. For example, we may 'discover' communities of users and categories of content that are not explicit in the indexes; We will use this derived metadata to inform user profiles and more generally attempt to enhance the experience of the users, content providers and site managers in navigating, discovering and managing the material in the repository.

This paper describes the architecture of this 'business intelligence layer'.

II. A Case Study with Learning Objects

The multi-agent architecture presented in this paper is designed to address the problem of making effective use of

Olivier Constant is a student in the EMOOSE Master of Science at Ecole des Mine de Nantes in France, currently completing his Masters Thesis at Monash University. (e-mail: Olivier.Constant@eleve.emn.fr).

Christine Mingins is Associate Professor in the School of Computer Science at Monash University (e-mail: cmingins@csse.monash.edu.au).

Annya Réquilé-Romanczuk, Ecole des Mines de Nantes, France, is currently on sabbatical in the School of Computer Science and Software Engineering at Monash University (e-mail: arequile@csse.monash.edu.au).

Brian Yap is a research student at Monash University. (e-mail: brian.yap@csse.monash.edu.au)

ICITA2002 ISBN: 1-86467-114-9

very large repositories of learning objects. In this section we describe the case study in which this problem appears.

Technological advances in the past few years, particularly in the area of online delivery and e-learning, have inspired changes in the way educational materials are designed, developed, and delivered to teachers and students. A major shift in educational materials development has occurred where there is a move away from the traditional method of developing courses in an integrated way to accomplish a learning objective to one that is based on the use of individual building blocks or bite-sized "learning objects" [8]. This approach resembles what Wayne Hodgins, Director of Worldwide Learning Solutions, has called the Legos™ approach. In the same way that Legos™ building blocks can be used to build a variety of structures, so too learning objects can be used by lecturers, teachers and others in creative ways to build courses which meet different learning outcomes. The theoretical underpinning is that instead of thousands of people wasting time "re-inventing the wheel", a learning object once constructed can be re-used and shared. Steven Downs [9] has argued that the economics of sharing learning objects are relentless.

This new approach has resulted in the establishment of a large number of learning object repositories both within Australia and overseas.

While theoretical underpinnings of learning object repositories are difficult to challenge, a number of issues relating to their establishment still need to be researched. Our project will address major problems/issues, such as the lack of consultation/analysis of user needs in the creation of repositories and inadequate resource discovery tools. It has been reported that many of the learning repositories are difficult to use.

We will try to address these problems:

- by undertaking user needs and usages analysis, and by using the data collected by the multi-agent component (see fig.1);
- by facilitating access to the plethora of content repositories and to address the problems of locating, exploring and manipulating learning resources expertly and creatively;
- by using the Learning Object Exploration System, cross-domain searching software and profiling systems to automatically match the needs of users with the appropriate learning objects;
- by using "intelligent" agents to "remember" frequently used and relevant resources and to inform users through presenting more intelligently guided pathways within that virtual environment.

The project will deliver an intelligent Learning Object Exploration System capable of identifying the needs of teachers, lecturers and course builders. This "intelligence" will be based on "profiling", extensive user analysis and resource assessment, and the construction of an "intelligent" agents to provide appropriate feedback.

The development of an intelligent learning architecture incorporates:

- the ability to actively collect and access a wide range of content repositories
- substantial improvements in the usefulness of learning objects within any given repository leading to enhanced exploitation of learning materials .
- the provision of enhanced feedback for the better management of content repositories.

III. Multi-Agent Architecture

In this section we describe this architecture. An overall view of the architecture is depicted in Fig.1.

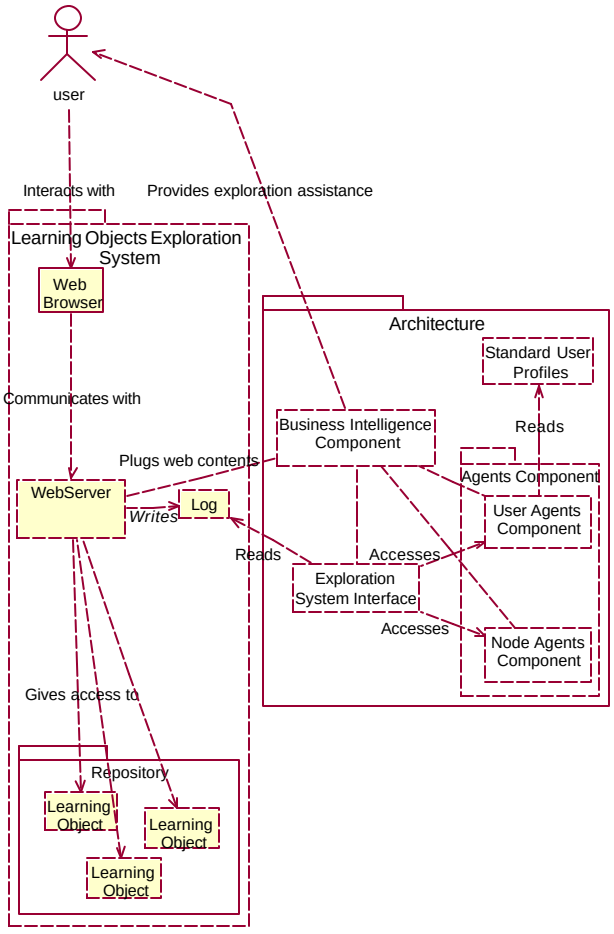


Fig. 1. Overview of the whole system

The prototype is based on an agent infrastructure and agent framework developed in C#. The infrastructure supports multi-hosts distribution via .Net's remoting and MSMQ. Persistent

data like profiles is stored in a SQL database on demand and accessed through ADO .Net.

A. Learning Object Exploration System

The architecture is designed to be plug into layered over existing repositories of Learning Objects. The system provides access to repositories through a web server. Users simply explore the repository with a web browser and view the Learning Objects as web pages.

The system is linked to the architecture through output and input points on the web server. The output point is a simple server log. For each access to a Learning Object, the log is classically required to record a user identifier (a static IP address for instance), an identifier of the Learning Object and the date and time.

The input point consists in adding web content generated by the architecture to the web page that is viewed by the user.

B. The Agent Components

User Agents Component

A User Agent tracks each user of the system. This agent may have references to Standard User Profiles (SUP) that suit its user. In addition, the User Agent is in charge of maintaining a personal user profile. This profile contains information about the user's pathways. The profile is refined over time as the User Agent Proxy informs the User Agent of the user's activities. When a user logs off for a while, his dedicated User Agent terminates. The profile persists independently from the User Agent and it can be stored into a SQL database on demand. It is reactivated when the user navigates again.

Node Agents Component

Node Agents form a virtual layer supplying information about traversals made between nodes. This information can then be exploited by others elements, such as the Business Intelligence Component. Node linkage information is formulated by the traversal of users between nodes. The term "node" is a general terminology used to represent a particular repository artifact. The artifact could be a web page or a relation in a relational database or it could even be more fine-grained such as a field in a relation. In the case study in this paper, a node is synonymous with a unique web page, thus a Node Agent will be associated with a unique node. In the virtual network.

Each Node Agent has the responsibility of:

- Capturing a user's page traversals (destinations). For instance a node agent (A) observes a web page (P). A user (U) accessed page P and then from there accessed a new page (Q). The node agent A must capture the next web page traverse by the user P, which is page Q in this instance.
- Storing the most recent timestamp for each destination navigated.
- Keeping a count on each destination page accessed from the observed page.

The Log Monitor gathers the information above is gathered by analyzing web server log files. Each Node Agent registers its associated node identity with the Log. The Node is then notified whenever a relevant log line appeared in the web server log and 'memorizes' the information.

C. Business Intelligence Component

This component is in charge of elaborating exploration assistance dynamically for every given user involved in exploring a Learning Repository. This is achieved by generating a directed graph representing navigation pathways. The vertices represent Learning Objects while the edges symbolize navigational links. All the pathways have the current Learning Object as origin. The edges hold information about the relevance of the traversal to the link they represent.

Such information comes from 3 sources:

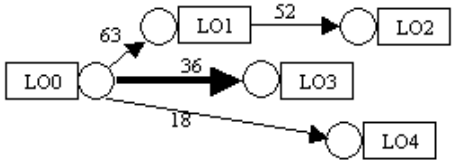
(1) the Node Agents that give an indication of the general popularity of the link,

(2) the SUPs associated with the user, representing users' interest in some categories and

(3) the personal profile of a user maintained by his User Agent.

This information is computed in order to obtain relevancy indicators.

The graph is then formatted to be visualized and sent to the web server for being displayed to the user. A possibility is that the user sees an additional frame on top of the web page by the mean of a plug-in for his web browser. The frame shows a graph whose nodes are actual web links that can be clicked. The graph provides navigation assistance in that the user is proposed relevant pathways. As an example, the graph could be something like this:



In this example, "LOX" is the identifier of a Learning Object; the weight and the length of the edges indicate the general

popularity of the link (the bigger the weight, the shorter the arrow) and the thickness of the arrows represents the relevancy of the traversal of the link for the user. This last information comes from the computation of the user profile and the Standard User Profiles (SUP) associated with the user.

D. *Standard User Profiles Component*

Every Standard User Profile (SUP) defines a category of users. A SUP contains pathway information for exploring Learning Objects that are interesting for the category of users. Administrators of the repository initially define SUPs, either directly or by computing similar user profiles maintained by the User Agents.

When a new user enters the system, he has the option of explicitly selecting SUPs that suit him via a special web page. His User Agent then references the SUPs. Otherwise, a User Agent is automatically created for him and initialized with no SUP. Later, the User Agent may infer a SUP after some time, thanks to the knowledge of the exploratory pathways of the user.

E. *Exploration System Interface*

A Log Monitor is in charge of reading the server log periodically. For every line in the log, the Log Monitor notifies the User Agent Proxy and the Node Agent Proxy and transmits the logged information (Fig.2).

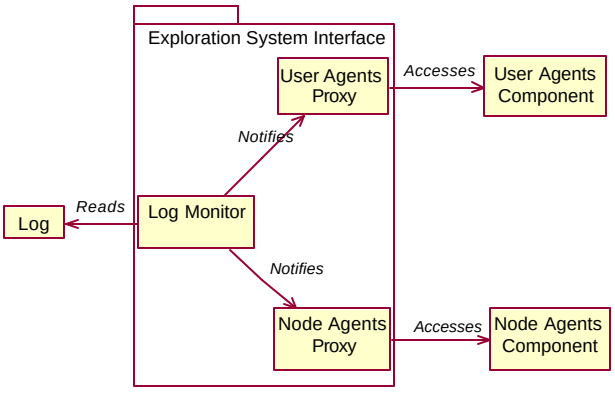


Fig. 2. The Exploration System Interface component

The proxies act as name servers for User Agents and Node Agents. In other words, they know all the agents and their corresponding identifiers, permitting for example access to the Node Agent that manages a given Learning Object. Besides, they transmit the log notifications from the Log Monitor to the agents that are concerned. Thus every User Agent is kept informed of the exploration of the user it manages, and similarly Node Agents keep aware of the navigation of all the users.

When the User Agent Proxy does not recognize a user that is referenced in the log, it means that it is a new user. Hence the User Agent Proxy creates a new User Agent with no SUP.

IV. Related Work

Recommender systems [1] learn about the preferences of users in order to assist them in finding items they are interested in, like books or movies. These systems aim at addressing the problem of information overload, particularly on the Web and in e-commerce. They make use of user profiles that are built either from explicit or implicit feedback from the user. The recommendation mechanism is based on comparing items (content-based), user profiles (collaborative filtering) or both [2].

Our system is comparable to recommender systems in that recommends pathways based on user profiles. Like many recommender systems it facilitates access to information without requiring the user to formulate explicit queries. Additionally, it uses similar techniques such as passive user profiling [3, 4] based on server logs [5] and relies on the notion of categories of users in the same manner as collaborative systems.

However, in addition our system provides assistance based on 3 different dimensions:

- the general community of users,
- categories specific to the user
- and individual user profiles.

Reconnaissance agents like those of the MIT Media Lab [6] help users browse the Web. These interface agents are on the client side to observe the user navigating and generate profiles. When the user reaches a web page, they propose links for further navigation. For example, the well-known Letizia explores all the links on the page viewed by the user in order to eliminate irrelevant links, and then recommends the links that fit best with the user profile.

Likewise, our system provides navigation assistance without interfering with the normal browsing behavior of the user. Also, this assistance changes according to the position of the user in the navigational space. However, our recommendations are not only based on the profile of the user but also on the experience of others.

Also, our attempt to build semantic links can be compared to systems with ontology-based semantics. The idea of the Semantic Web as proposed by Tim Berners-Lee and Jim Hendler is based on coded ontologies permitting software agents to “understand” the relationships between web pages [7].

While we are not seeking to replace ontologies, indexes and other repository metadata, we are taking a diametrically opposite approach to metadata tagging – that is, constructing

an architecture that will allow us to experiment with 'discovering' rather than coding categories, communities and other interesting semantics based on actual usage.

Our system does not require any ontology since it builds semantic links (infers semantics from links) pragmatically, based on the actual navigation of users. The advantage is that the generation and the maintenance of links are dynamic, hence our system adapts dynamically to any change within the repository.

Although our system is obviously not suitable for navigating the whole Web, it is based on generic principles that make it adaptable to any sort of repository, from a relational database of learning objects or other resources to a local area network of web pages.

V. Conclusion and Future Work

Coded metadata or indexing mechanisms fix the content semantics of information repositories. We believe that we can enhance users' experiences in discovering information in rich repositories by using the mechanism described in this paper to derive semantics based on users' navigations. Of particular interest is the discovery of communities or categories of both information and users, and uncovering untagged aspects of complex objects relevant to the user community. The business intelligence layer described in this paper has been designed as a test bed for such experiments.

References

[1] P. Resnick, H.R. Varian, "Recommender systems", Communications of the ACM, vol. 40 issue 3, p. 56-58, 1997.

[2] M. Balabanovic, Y. Shoham, "Fab: Content-Based, Collaborative Recommendation", Communications of the ACM, vol. 40 issue 3, March 1997.

[3] D.M. Nichols, "Implicit Rating and Filtering", in Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering, 10-12 November 1997, Budapest, Hungary.

[4] M. Claypool, D. Brown, P. Le, M. Waseda, "Inferring User Interest", in IEEE Internet Computing, November/December 2001, p. 32-39.

[5] K. Bradley, R. Rafter, B. Smyth, "Inferring Relevance Feedback from Server Logs: A Case Study in Online Recruitment", 2000, in Proceedings of the 11th Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2000), Galway, Ireland.

[6] H. Lieberman, C. Fry, and L. Weitzman, "Exploring the Web with Reconnaissance Agents", in Communications of the ACM, Volume 44 Issue 8, August 2001.

[7] Web Ontology Working Group: http://www.w3.org/2001/sw/WebOnt/

[8] Carnevale, Dan. (2001). "Some online educators turn to bite-sized instruction", *Chronicle of Higher Education*, May 3, 2001. [Online] http://chronicle.com/free/2001/05/2001050301u.htm

[9] Downs, Stephen (May 2000). "Learning objects". [Online] Accessed: 25 March 2002. (http://www.atl.ualberta.ca/downes/Learning_Objects.doc)

[10] http://socci.edna.edu.au/content/index.asp

[11] Rosch, E. (1978): Principles of categorization. In E. Rosch & B. B. Lloyd (eds.): *Cognition and categorization* (pp. 27-48). Hillsdale, NJ: Erlbaum

[12] Lakoff, G (1990): Women, Fire and Dangerous Things What Categories Reveal about the Mind, University of Chicago Press, Chicago

Appendix D: Source Code

1 Agent infrastructure

1. Class MAP.Middleware

```
using System;
using System.Windows.Forms;
using System.Diagnostics;

namespace MAP
{
      /// <summary>
      /// Utility class for the infrastructure management.
      /// Two configurations are possible: (1) this process on this
computer
      /// is the main host, i.e. the centralized parts of the
infrastructure
      /// reside in this process, or (2) this process is not the main host:
      /// it has to communicate with the remote main host for accessing the
      /// centralized parts of the infrastructure.
      /// These configurations depend on the way the infrastructure is
installed
      /// by method Install.
      /// </summary>
      public sealed class Middleware
      {
            // For Http Remoting operations
            internal static readonly int SERVER_PORT = 1989;
            internal static readonly int CLIENT_PORT = 1990;

            // Maximum latency time of network for Messages to be delivered
            public static readonly TimeSpan MAX_LATENCY = new TimeSpan(0,
0, 5);

            /// <summary>
            /// The address of the current process on this machine.
            /// </summary>
            public static readonly AgentAddress ThisAddress = new
AgentAddress(
                  GetProcessId(), GetComputerName());

            /// <summary>
            /// The name of the main host machine, i.e. the one on which
the
            /// NS and the DF should reside. If this process is the main
host,
            /// then the value is "localhost".
            /// </summary>
            private static string MAIN_HOST;

            /// <summary>
            /// Returns the name of this computer.
            /// </summary>
            /// <returns></returns>
            public static string GetComputerName()
            {
```

```
                return SystemInformation.ComputerName.ToLower();
            }

            /// <summary>
            /// Returns the unique process ID of this process.
            /// </summary>
            /// <returns></returns>
            public static string GetProcessId()
            {
                return Process.GetCurrentProcess().Id.ToString();
            }

            /// <summary>
            /// Returns the name of the computer on which the main host
resides.
            /// </summary>
            /// <returns></returns>
            public static string GetMainHostName()
            {
                return MAIN_HOST;
            }

            /// <summary>
            /// Setups the agent infrastructure in this process as main
host.
            /// </summary>
            public static void Install()
            {
                Install("localhost");
            }

            /// <summary>
            /// Setups the agent infrastructure in this process. Parameter
is the machine
            /// name of the main host. If its value is "localhost" then
this process
            /// becomes the main host.
            /// </summary>
            /// <param name="mainHostName"></param>
            public static void Install(string mainHostName)
            {
                MAIN_HOST = mainHostName;
                NameServer.Install();
                DirectoryFacilitator.Install();
                // MessageTransporter does not need any install because
of MSMQ
            }

            /// <summary>
            /// Returns whether the specified AgentAddress is this host.
            /// </summary>
            /// <returns></returns>
            public static bool IsThisHost(AgentAddress address)
            {
                return ThisAddress.Equals(address);
            }

            /// <summary>
            /// Returns whether this process is the main host.
            /// </summary>
            /// <returns></returns>
```

```
            public static bool ProcessIsMainHost()
            {
                  return GetMainHostName() == "localhost";
            }

            /// <summary>
            /// Uninstalls the agent infrastructure by cleaning up
resources
            /// (for example MSMQ queues).
            /// </summary>
            public static void Uninstall()
            {
                  MessageTransporter.Uninstall();
            }

            // Class cannot be instantiated
            private Middleware() {}
      }
}
```

2. Class MAP.MessageTransporter

```
#define MULTI_PROCESSES // For testing several platforms on the same
machine
using System;
using System.Diagnostics;
using System.Collections;
using System.Messaging;

namespace MAP
{
      /// <summary>
      /// The MessageTransporter provides facilities for delivering
Messages to their
      /// recipient. Agents have to register to their MT in order to
receive an ID that
      /// allows them to receive Messages by the mean of the 'Recipient'
property.
      /// There is one instance of MT per machine. Each instance has a MSMQ
queue for
      /// receiving messages from other machines.
      /// </summary>
      public class MessageTransporter
      {
            // The singleton instance
            private static MessageTransporter instance;

            /// <summary>
            /// Defines the path of a MSMQ MessageQueue for IAgent
Messages.
            /// </summary>
            /// <param name="computer"></param>
            /// <param name="process"></param>
            /// <returns></returns>
            private static string MakePath(string computer, string process)
            {
                  string path = computer + "\\private$\\agentmt";
                  #if MULTI_PROCESSES
                        path += process;
                  #endif
```

```
                  return path;
            }

            /// <summary>
            /// Returns the singleton instance.
            /// </summary>
            /// <returns></returns>
            public static MessageTransporter GetMT()
            {
                  if (instance == null) instance = new
MessageTransporter();
                  return instance;
            }

            /// <summary>
            /// Cleans up resources like MSMQ queues and kills all local
Agents.
            /// </summary>
            public static void Uninstall()
            {
                  if (instance != null)
                  {
                        // Kill all local Agents
                        lock(instance.table)
                        {
                              IList agents = new
ArrayList(instance.table.Values);
                              foreach (IAgent ia in agents)
                              {
                                    if (ia is Agent)
                                    {
                                          Agent agent = ia as Agent;
                                          agent.Terminate();
                                    }
                              }
                        }
                        // Delete MSMQ queue

      System.Messaging.MessageQueue.Delete(instance.mQueue.Path);
                  }
            }

            // A table of (agentID: string, agent: IAgent) referencing the
local agents
            private Hashtable table;
            // A table of (agentAddress, queue: MessageQueue) for
remembering
            // the MessageQueues of remote MessageTransporters.
            private Hashtable queueTable;
            // The own MessageQueue of this for receiving Messages.
            private System.Messaging.MessageQueue mQueue;

            /// <summary>
            /// Private constructor because this is singleton.
            /// </summary>
            private MessageTransporter()
            {
                  // Initializes tables
                  table = new Hashtable();
                  queueTable = new Hashtable();
                  // Creates own MessageQueue for receiving remote Messages
```

```
                  string path = MakePath(".", Middleware.GetProcessId());
                  if (! System.Messaging.MessageQueue.Exists(path))
                mQueue = System.Messaging.MessageQueue.Create(path);
                  else
                        mQueue = new System.Messaging.MessageQueue(path);
                  // Setups asynchronous receiving
                  mQueue.ReceiveCompleted +=
                        new
ReceiveCompletedEventHandler(RemoteMessageArrived);
                  mQueue.BeginReceive ();
            }

            /// <summary>
            /// Allows for the retrieval of a local IAgent from its ID.
            /// Returns null if not found.
            /// </summary>
            /// <param name="id"></param>
            /// <returns></returns>
            private IAgent FindAgent(AgentId id)
            {
                  Debug.Assert(id != null);
                  IAgent agent = null;
                  lock (table)
                  {
                        if (table.ContainsKey(id))
                              agent = (IAgent) table[id];
                        else
                              Console.WriteLine("Cannot find agent with id
{0} in process {1}",
                                    id, Middleware.ThisAddress);
                  }
                  return agent;
            }

            /// <summary>
            /// Returns the MessageQueue of a MessageTransporter on a
remote computer.
            /// </summary>
            /// <param name="computerName"></param>
            /// <returns></returns>
            private System.Messaging.MessageQueue
GetMessageQueue(AgentAddress address)
            {
                  System.Messaging.MessageQueue queue = null;
                  lock (queueTable)
                  {
                        if (queueTable.ContainsKey(address))
                              // Queue already known
                              queue = (System.Messaging.MessageQueue)
queueTable[address];
                        else
                        {
                              // Get queue object
                              System.Messaging.MessageQueue[] queues =
null;
                              try
                              {
                                    queues = System.Messaging.MessageQueue.

      GetPrivateQueuesByMachine(address.Computer);
                              }
```

```
                              catch
                              {
                                    // Cannot get remote queues
                                    return null;
                              }
                              string path = MakePath(address.Computer,
address.Process);
                              // Obtaining queue directly fails if it is
remote (.Net bug or
                              // insufficient documentation?) so instead we
get all the private
                              // queues on the remote machine and select
the right one. Not the
                              // most efficient way, but it works.
                              foreach (System.Messaging.MessageQueue q in
queues)
                              {
                                    if (q.Path.EndsWith(path))
                                    {
                                          queue = q;
                                          break;
                                    }
                              }
                              if (queue != null) queueTable.Add(address,
queue);
                        } // End queue already known
                  } // End lock
                  return queue;
            }

            /// <summary>
            /// Transmits a Message to the (possibly remote) recipient
IAgent.
            /// </summary>
            /// <param name="m"></param>
            public void PostMessage(Message m)
            {
                  Debug.Assert(m != null);
                  AgentId recipientId = m.Recipient;
                  AgentAddress address =
NameServer.GetNS().FindAddressOf(recipientId);
                  Debug.Assert(address != null);
                  if (Middleware.IsThisHost(address))
                  {
                        // The specified computer name in the agent ID is
this computer
                        // so the agent should be local
                        IAgent recipient = FindAgent(recipientId);
                        if (recipient != null)
                              recipient.PostMessage(m);
                        else
                              Console.WriteLine("MT could not find
recipient "
                                    + " for {0} in host {1}", m,
Middleware.ThisAddress);
                  }
                  else
                  {
                        // Forward the Message to a remote
MessageTransporter via MSMQ
```

```
                        System.Messaging.MessageQueue queue =
GetMessageQueue(address);
                        if (queue == null)
                        {
                              // Failed to obtain queue
                              Console.Write("Cannot find MSMQ queue on host
{0}",
                                    address);
                              Console.WriteLine(" so cannot send message
{0}.", m);
                        }
                        else
                        {
                              // Queue successfully obtained
                              System.Messaging.Message msg = new
System.Messaging.Message(m);
                              msg.Formatter = new BinaryMessageFormatter();
                              queue.Send(msg);
                        }
                  }
            }

            /// <summary>
            /// Allows IAgents to register and get an ID.
            /// </summary>
            /// <param name="agent"></param>
            /// <returns></returns>
            public AgentId Register(IAgent agent)
            {
                  Debug.Assert(agent != null);
                  AgentId id = null;
                  lock (table)
                  {
                        if (!table.ContainsValue(agent))
                        {
                              do {id = AgentId.NewId();} while
(table.Contains(id));
                              table.Add(id, agent);
                              Console.WriteLine("MT - Added: {0} for {1}",
                                    id, agent);
                        }// Else agent is already registered
                  }
                  // Register AgentAddress
                  NameServer.GetNS().Register(id, Middleware.ThisAddress);
                  return id;
            }

            /// <summary>
            /// Handles incoming Messages asynchronously.
            /// </summary>
            /// <param name="source"></param>
            /// <param name="asyncReceive"></param>
            private void RemoteMessageArrived(Object source,
                  ReceiveCompletedEventArgs asyncReceive)
            {
                  System.Messaging.MessageQueue queue =
(System.Messaging.MessageQueue)
                        source;
                  // Get the System.Messaging.Message
                  System.Messaging.Message msqmMessage = queue.EndReceive(
                        asyncReceive.AsyncResult);
```

```
                  // Set up formatter for unserialization
                  BinaryMessageFormatter reader = new
BinaryMessageFormatter();
                  msqmMessage.Formatter = reader;
                  // Get the Message and post it on this
                  Message agentMsg = (Message) msqmMessage.Body;
                  PostMessage(agentMsg);
                  // Try to receive other messages
                  queue.BeginReceive ();
            }

            /// <summary>
            /// Allows IAgents to unregister during their finalization.
            /// </summary>
            /// <param name="agent"></param>
            /// <returns></returns>
            public void Unregister(AgentId id)
            {
                  NameServer.GetNS().Unregister(id);
                  Debug.Assert(id != null);
                  lock (table)
                  {
                        table.Remove(id);
                  }
            }
      }
}
```

3. Class MAP.NameServer

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;

namespace MAP
{
      /// <summary>
      /// Name Server: maps AgentIds with AgentAddresses.
      /// The singleton NS resides on the main host only. Other hosts use a
proxy and
      /// synchronized remote calls to the instance on the main host.
      /// </summary>
      public class NameServer : MarshalByRefObject // For remote access
      {

            //****** STATIC PART ******

            // The unique instance.
            private static NameServer Instance;

            // Remoting parameters
            private static readonly string FUNCTION = "NS";

            /// <summary>
            /// Returns the singleton instance.
            /// </summary>
            /// <returns></returns>
```

```
            public static NameServer GetNS()
            {
                  if (Instance == null) Install();
                  return Instance;
            }

            /// <summary>
            /// Makes a (possibly remote) Name Server available on this
computer.
            /// This method is required because C# does not support static
initialization
            /// blocks.
            /// There is no need to call this method explicitly if the
platform on the
            /// main host is started first.
            /// </summary>
            public static void Install()
            {
                  if (Instance != null) return; // Already installed
                  if (Middleware.ProcessIsMainHost())
                        InstallServer();
                  else
                        InstallClient();
            }

            /// <summary>
            /// Current process is not main host: get proxy to DF on main
host.
            /// </summary>
            private static void InstallClient()
            {
                  string uri = "http://" + Middleware.GetMainHostName() +
":"
                        + Middleware.SERVER_PORT + "/" + FUNCTION;
                  // e.g.: http://samson.csse.monash.edu.au:1979/NS
                  HttpChannel chan = new
HttpChannel(Middleware.CLIENT_PORT);
                  ChannelServices.RegisterChannel(chan);
                  try
                  {
                        Instance = (NameServer) Activator.GetObject(
                              typeof (NameServer),
                              uri);
                        Console.WriteLine("NS - Available remotely");
                  }
                  catch(System.Net.WebException e)
                  {
                        // Cannot get proxy
                        string msg = "Cannot find Name Server on main host:
"
                              + uri + "\\n" + e.Message;
                        throw new Exception(msg);
                  }
            }

            /// <summary>
            /// Current process is main host: create DF and make it
accessible remotely.
            /// </summary>
            private static void InstallServer()
            {
```

```
                HttpChannel chan = new
HttpChannel(Middleware.SERVER_PORT);
                ChannelServices.RegisterChannel(chan);
                Instance = new NameServer();
                RemotingServices.Marshal(Instance, FUNCTION);
                Console.WriteLine("NS - Available locally");
          }


          //****** NON-STATIC PART ******

          // Table of (agentId, agentAddress)
          private Hashtable table;

          /// <summary>
          /// Private constructor because this is Singleton.
          /// </summary>
          private NameServer()
          {
                table = new Hashtable();
          }

          /// <summary>
          /// Allows for the retrieval of an AgentAddress. Returns null
if none found.
          /// </summary>
          /// <param name="agentId"></param>
          /// <returns></returns>
          public AgentAddress FindAddressOf(AgentId agentId)
          {
                Debug.Assert(agentId != null);
                AgentAddress address = null;
                lock (table)
                {
                      if (table.ContainsKey(agentId))
                      {
                            // IAgent is registered
                            address = (AgentAddress)table[agentId];
                      }
                }
                return address;
          }

          /// <summary>
          /// Registers an IAgent. If already registered, updates its
address.
          /// </summary>
          /// <param name="agentId"></param>
          /// <param name="address"></param>
          public void Register(AgentId agentId, AgentAddress address)
          {
                Debug.Assert(agentId != null);
                Debug.Assert(address != null);
                lock (table)
                {
                      if (!table.Contains(agentId))
                      {
                            table.Add(agentId, address);
                      }
                      else
                      {
```

```
                        table[agentId] = address;
                    }
                }
            }

            /// <summary>
            /// Unregisters an IAgent. Does nothing is it is not registered
            /// </summary>
            /// <param name="agentId"></param>
            public void Unregister(AgentId agentId)
            {
                Debug.Assert(agentId != null);
                lock(table)
                {
                    if (table.Contains(agentId))
                    {
                        table.Remove(agentId);
                    }
                }
            }
        }
}
```

4. Class MAP.DirectoryFacilitator

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;

namespace MAP
{
      /// <summary>
      /// Directory Facilitator: helps service consumers find service
providers.
      /// The singleton DF resides on the main host only. Other hosts use a
proxy and
      /// synchronized remote calls to the instance on the main host.
      /// </summary>
      public class DirectoryFacilitator : MarshalByRefObject // For remote
access
      {

            //****** STATIC PART ******

            // The unique instance.
            private static DirectoryFacilitator Instance;

            // Remoting parameters
            private static readonly string FUNCTION = "DF";

            /// <summary>
            /// Returns the singleton instance.
            /// </summary>
            /// <returns></returns>
            public static DirectoryFacilitator GetDF()
            {
                if (Instance == null) Install();
```

```
                  return Instance;
            }

            /// <summary>
            /// Makes a (possibly remote) Directory Facilitator available
on this computer.
            /// This method is required because C# does not support static
initialization
            /// blocks.
            /// There is no need to call this method explicitly if the
platform on the
            /// main host is started first.
            /// </summary>
            public static void Install()
            {
                  if (Instance != null) return; // Already installed
                  if (Middleware.ProcessIsMainHost())
                        InstallServer();
                  else
                        InstallClient();
            }

            /// <summary>
            /// Current process is not main host: get proxy to DF on main
host.
            /// </summary>
            private static void InstallClient()
            {
                  string uri = "http://" + Middleware.GetMainHostName() +
":"
                        + Middleware.SERVER_PORT + "/" + FUNCTION;
                  // e.g.: http://samson.csse.monash.edu.au:1989/DF
                  try
                  {
                        Instance = (DirectoryFacilitator)
Activator.GetObject(
                              typeof (DirectoryFacilitator),
                              uri);
                        Console.WriteLine("DF - Available remotely");
                  }
                  catch(System.Net.WebException e)
                  {
                        // Cannot get proxy
                        string msg = "Cannot find Directory Facilitator on
main host: "
                              + uri + "\\n" + e.Message;
                        throw new Exception(msg);
                  }
            }

            /// <summary>
            /// Current process is main host: create DF and make it
accessible remotely.
            /// </summary>
            private static void InstallServer()
            {
                  Instance = new DirectoryFacilitator();
                  RemotingServices.Marshal(Instance, FUNCTION);
                  Console.WriteLine("DF - Available locally");
            }
```

```
            //****** NON-STATIC PART ******

            // Table of (service name, IProviderSet of agentIds)
            private Hashtable providersTable;

            /// <summary>
            /// Private constructor because this is Singleton.
            /// </summary>
            private DirectoryFacilitator()
            {
                  providersTable = new Hashtable();
            }

            /// <summary>
            /// Allows for the retrieval of a service provider. Returns
null if none found.
            /// </summary>
            /// <param name="service"></param>
            /// <returns></returns>
            public AgentId FindProviderOf(string service)
            {
                  Debug.Assert(service != null);
                  AgentId providerId = null;
                  lock (providersTable)
                  {
                        if (providersTable.ContainsKey(service))
                        {
                              // Service is registered
                              IProviderSet aSet =
(IProviderSet)providersTable[service];
                              providerId = aSet.SelectProvider();
                        }
                  }
                  return providerId;
            }

            /// <summary>
            /// Registers a service provider.
            /// </summary>
            /// <param name="service"></param>
            /// <param name="agentId"></param>
            public void RegisterProvider(string service, AgentId agentId)
            {
                  Debug.Assert(service != null);
                  Debug.Assert(agentId != null);
                  lock (providersTable)
                  {
                        if (providersTable.Contains(service))
                        {
                              // Service already registered
                              IProviderSet providers =
(IProviderSet)providersTable[service];
                              if (!providers.Contains(agentId))
                              {
                                    providers.Add(agentId);
                              }
                        }
                        else
                        {
                              // Register new service
```

```
                              IProviderSet providers = new
ProviderSet(service, providersTable);
                              providers.Add(agentId);
                              providersTable.Add(service, providers);
                        }
                  }
            }

            /// <summary>
            /// Unregisters a service provider.
            /// Does nothing if the service provider is not registered.
            /// </summary>
            /// <param name="service"></param>
            /// <param name="agentId"></param>
            /// <returns></returns>
            public void Unregister(string service, AgentId agentId)
            {
                  Debug.Assert(service != null);
                  lock (providersTable)
                  {
                        if (providersTable.ContainsKey(service))
                        {
                              IProviderSet providers =
(IProviderSet)providersTable[service];
                              if (providers.Contains(agentId))
                              {
                                    providers.Remove(agentId);
                                    if (providers.Count == 0)
                                    {
                                          providersTable.Remove(service);
                                    }
                              }
                        }
                  }
            }

            /// <summary>
            /// Unregisters an IAgent from all its registered services.
            /// </summary>
            /// <param name="agentId"></param>
            public void Unregister(AgentId agentId)
            {
                  lock(providersTable)
                  {
                        IList list = new ArrayList(providersTable.Values);
                        foreach (IProviderSet providers in list)
                        {
                              providers.Remove(agentId);
                        }
                  }
            }
      }

      /// <summary>
      /// Defines a set of providers for the same service.
      /// Allows, in particular, for the selection of a provider.
      /// </summary>
      interface IProviderSet
      {
            void Add(AgentId agentId);
            bool Contains(AgentId agentId);
```

```
            int Count {get;}
        void Remove(AgentId agentId);

            /// <summary>
            /// Selects one provider among the possible providers for the
service.
            /// </summary>
            AgentId SelectProvider();
      }

      /// <summary>
      /// Default implementation for IProviderSet.
      /// </summary>
      class ProviderSet : IProviderSet
      {
            private IList list;
            private readonly string _service;
            private readonly Hashtable _table;

            public ProviderSet(string service, Hashtable table)
            {
                  list = new ArrayList();
                  _service = service;
                  _table = table;
            }
            public void Add(AgentId agentId)
            {
                  if (!list.Contains(agentId))
                        list.Add(agentId);
            }
            public bool Contains(AgentId agentId)
            {
                  return list.Contains(agentId);
            }
            public int Count
            {
                  get
                  {
                        return list.Count;
                  }
            }
            public void Remove(AgentId agentId)
            {
                  list.Remove(agentId);
                  if (list.Count == 0)
                  {
                        lock(_table)
                        {
                              _table.Remove(_service);
                        }
                  }
            }
            public AgentId SelectProvider()
            {
                  // Selects the first provider in the list and puts it at
the end
                  // so that the selection is circular.
                  Debug.Assert(Count > 0);
                  object first = list[0];
                  list.Remove(first);
                  list.Add(first);
```

```
                  return (AgentId) first;
            }
      }
}
```

5. Class MAP.Message

```
using System;

namespace MAP
{
      /// <summary>
      /// Defines messages for inter-agent communication.
      /// </summary>
      [Serializable]public class Message
      {
            // Message natures.
            public enum Natures {Request, Inform};

            private Natures _nature;
            private AgentId _sender, _recipient;
            private string _subject;
            private object _content;
            private Conversation _conversation;

            /// <summary>
            /// Message creation with a content object.
            /// The object must be serializable for remote communication.
            /// Warning: be sure not to break the encapsulation of the
agent's mental
            /// state in the case of local communication. If the object
belongs to the
            /// agent's state, be sure it is Marshal-By-Value and not
Marshal-By-Ref,
            /// or pass a deep copy of it.
            /// </summary>
            /// <param name="sender"></param>
            /// <param name="recipient"></param>
            /// <param name="nature"></param>
            /// <param name="subject"></param>
            /// <param name="content"></param>
            public Message(AgentId sender, AgentId recipient, Natures
nature,
                  string subject, object content)
            {
                  _sender = sender;
                  _recipient = recipient;
                  _nature = nature;
                  _subject = subject;
                  _content = content;
                  _conversation = null;
            }

            /// <summary>
            /// Simple message creation.
            /// </summary>
            /// <param name="sender"></param>
            /// <param name="recipient"></param>
            /// <param name="nature"></param>
            /// <param name="subject"></param>
```

```
            public Message(AgentId sender, AgentId recipient, Natures
nature,
                  string subject) : this(sender, recipient, nature,
subject, null) {}

            public object Content
            {
                  get
                  {
                        return _content;
                  }
            }
            public Conversation Conversation
            {
                  get
                  {
                        return _conversation;
                  }
            }
            public Natures Nature
            {
                  get
                  {
                        return _nature;
                  }
            }
            public AgentId Recipient
            {
                  get
                  {
                        return _recipient;
                  }
                  set // Used for Message forwarding
                  {
                        _recipient = value;
                  }
            }
            public AgentId Sender
            {
                  get
                  {
                        return _sender;
                  }
                  set // Used for Message forwarding
                  {
                        _sender = value;
                  }
            }
            internal void SetConversation(Conversation conversation)
            {
                  _conversation = conversation;
            }
            public string Subject
            {
                  get
                  {
                        return _subject;
                  }
            }

            public override String ToString()
```

```
            {
                  string contentStr = Content == null ? "" : (" content:" +
Content);
                  return "[Message"
                        //+ " from:" + Sender
                        //+ " to:" + Recipient
                        + " nature:" + Nature
                        + " subject:" + Subject
                        + contentStr + "]";
            }
      }
}
```

6. Class MAP. MessageCategory

```
using System;

namespace MAP
{
      /// <summary>
      /// Defines a category of Message through a nature and subject.
      /// </summary>
      public class MessageCategory
      {
            private Message.Natures _nature;
            private string _subject;

            public MessageCategory(Message.Natures nature, string subject)
            {
                  _nature = nature;
                  _subject = subject;
            }

            // For use with Hashtable
            public override bool Equals(object obj)
            {
                  if (!(obj is MessageCategory)) return false;
                  MessageCategory peer = (MessageCategory) obj;
                  return
                        peer.Subject == this.Subject
                        &&
                        peer.Nature == this.Nature;
            }

            /// <summary>
            /// Returns whether this is the category of the Message.
            /// </summary>
            /// <param name="m"></param>
            /// <returns></returns>
            public virtual bool IsCategoryOf(Message m)
            {
                  if (m == null) return false;
                  return
                        m.Subject == this.Subject
                        &&
                        m.Nature == this.Nature;
            }

            // For use with Hashtable
            public override int GetHashCode()
```

```
            {
                  return Subject.GetHashCode() + Nature.GetHashCode();
            }

            public Message.Natures Nature
            {
                  get
                  {
                        return _nature;
                  }
            }

            public string Subject
            {
                  get
                  {
                        return _subject;
                  }
            }

            public override string ToString()
            {
                  return "[Nature:" + Nature + ", Subject:" + Subject +
"]";
            }
      }
}
```

2 Agent framework

7. Class MAP.AgentId

```
using System;

namespace MAP
{
      /// <summary>
      /// Defines a unique identifier for an IAgent.
      /// </summary>
      [Serializable]public class AgentId
      {
            private static readonly Object aLock = new Object();

            /// <summary>
            /// Generates a new unique ID.
            /// Current implementation is based on current time. Class Guid
could be used
            /// too.
            /// </summary>
            /// <returns></returns>
            public static AgentId NewId()
            {
                  lock (aLock) // For being sure there is no duplicate with
current time
                  {
                        DateTime now = System.DateTime.Now;
                        string name = "ag" + now.Year + now.Month + now.Day
+ now.Hour
```

```
                        + now.Minute + now.Second + now.Millisecond;
                  return new AgentId(name);
            }
      }

      private string _name;

      /// <summary>
      /// Constructor.
      /// </summary>
      /// <param name="name"></param>
      private AgentId(string name)
      {
            _name = name;
      }

      public string Name
      {
            get
            {
                  return _name;
            }
      }

      public override bool Equals(object o)
      {
            if (!(o is AgentId)) return false;
            AgentId peer = (AgentId) o;
            return this.Name == peer.Name;
      }

      public override int GetHashCode()
      {
            return Name.GetHashCode();
      }

      public override string ToString()
      {
            return Name;
      }
   }
}
```

8. Class MAP.AgentAddress

```
using System;

namespace MAP
{
   /// <summary>
   /// Defines locations at which IAgents reside.
   /// </summary>
   [Serializable]public class AgentAddress
   {
      private string _process;
      private string _computer;

      /// <summary>
      /// Constructor. The address is composed by a process ID and a
computer
```

```
            /// name in order to allow for the retrieval of the process in
which the
            /// agent resides. Note: Process is for testing several
processes on the same
            /// machine: process identification can be removed eventually.
            /// </summary>
            /// <param name="process"></param>
            /// <param name="computer"></param>
            public AgentAddress(string process, string computer)
            {
                  _process = process;
                  _computer = computer;
            }

            public string Computer
            {
                  get
                  {
                        return _computer;
                  }
            }

            public string Process
            {
                  get
                  {
                        return _process;
                  }
            }

            public override bool Equals(object o)
            {
                  if (!(o is AgentAddress)) return false;
                  AgentAddress peer = (AgentAddress) o;
                  return
                        this.Process == peer.Process &&
                        this.Computer == peer.Computer;
            }

            public override int GetHashCode()
            {
                  return Process.GetHashCode() + Computer.GetHashCode();
            }

            public override string ToString()
            {
                  return Process + "@" + Computer;
            }
      }
}
```

9. Class MAP.IAgent

```
namespace MAP
{
      /// <summary>
      /// Specifies a very generic definition of an agent as an object that
can
      /// be sent agent messages.
      /// </summary>
```

```
	public interface IAgent
	{
		/// <summary>
		/// Sends a Message to the agent.
		/// </summary>
		/// <param name="m"></param>
		void PostMessage(Message m);

		/// <summary>
		/// Returns the agent identifier.
		/// </summary>
		AgentId GetId();
	}
}
```

10. Class MAP.Agent

```
using System;
using System.Threading;
using System.Collections;

namespace MAP
{
	/// <summary>
	/// Defines a delegate that can be used for handling the
MessageArrived event for
	/// Conversations.
	/// </summary>
	public delegate void MessageArrivedEventHandler(Message m);

	/// <summary>
	/// Defines a generic design for agents. Instantiable subclasses must
provide an
	/// implementation for method Execute.
	/// </summary>
	public abstract class Agent : MarshalByRefObject, IAgent
	{
		// The Agent ID
		private readonly AgentId id;
		// Queue of Messages
		protected MessageQueue msgQueue;
		// Event raised when Messages arrive.
		internal event MessageArrivedEventHandler MessageArrived;
		// List of WeakReferences to the Threads depending on this and
its Activities
		private IList threads;
		// Main Thread
		private readonly Thread mainThread;

		/// <summary>
		/// To be invoked by subclasses.
		/// </summary>
		/// <param name="withMessageQueue"></param>
		protected Agent()
		{
			threads = new ArrayList();
			// Initialization of the queue of Messages.
			msgQueue = new MessageQueue(this);
			// Registration to the Message Transporter.
			id = MessageTransporter.GetMT().Register(this);
```

```
                // Start agent's main Thread.
                ThreadStart starter = new ThreadStart(Execute);
                mainThread = new Thread(starter);
                mainThread.Start();
        }

        /// <summary>
        /// Defines the meta-behaviour of the agent. The meta-behaviour
manipulates
        /// (starts, suspends, stops) all the different behaviours of
the Agent.
        /// </summary>
        protected abstract void Execute();

        /// <summary>
        /// See IAgent.GetId
        /// </summary>
        /// <returns></returns>
        public AgentId GetId()
        {
                return id;
        }

        /// <summary>
        /// Defines the default behaviour when a not-understood Message
is received.
        /// Called by MessageQueue.
        /// </summary>
        /// <param name="m"></param>
        protected internal virtual void MessageNotUnderstood(Message m)
        {
                Console.WriteLine("Not understood: {0} received by {1}",
m, this);
        }

        /// <summary>
        /// Returns a new Thread that is referenced by this so that
this has control
        /// over the Thread. Should only be called by subclasses or
class Activity.
        /// </summary>
        /// <param name="starter"></param>
        /// <returns></returns>
        internal protected Thread NewThread(ThreadStart starter)
        {
                Thread t = new Thread(starter);
                // A WeakReference references the Thread until its
finalization starts
                // Look for already existing free WeakReferences in the
List
                for (int i = 0; i < threads.Count; i++)
                {
                        WeakReference wr = (WeakReference) threads[i];
                        if (!wr.IsAlive)
                        {
                                // This WeakReference is free: use it
                                wr.Target = t;
                                return t;
                        }
                }
                // No free WeakReference found: create and add a new one
```

```
                  WeakReference weak = new WeakReference(t);
                  threads.Add(weak);
                  return t;
            }

            /// <summary>
            /// See IAgent. Enqueues all Messages in the MessageQueue
except if they
            /// belong to an already started Conversation.
            /// </summary>
            /// <param name="m"></param>
            public void PostMessage(Message m)
            {
                  Conversation conv = m.Conversation;
                  if (conv == null || !conv.IsHandled)
                        // Message is not handled by a Conversation so push
it in the
                        // MessageQueue
                        msgQueue.Enqueue(m);
                  else
                        // Notify Conversation (asynchronous because it is
an event)
                        if (MessageArrived != null) MessageArrived(m);
            }

            /// <summary>
            /// A shortcut for posting a Message to the local
MessageTransporter.
            /// </summary>
            /// <param name="m"></param>
            protected void SendMessage(Message m)
            {
                  MessageTransporter.GetMT().PostMessage(m);
            }

            /// <summary>
            /// Should be called when method Execute terminates or when
Middleware is
            /// uninstalled.
            /// Unregisters this from the system and aborts all own
Threads.
            /// Can be overriden for making some work before terminating
but base method
            /// must always be invoked.
            /// </summary>
            protected internal virtual void Terminate()
            {
                  // If method is not called by main Thread, abort main
Thread, i.e.
                  // kill Agent
                  if (!Thread.CurrentThread.Equals(mainThread))
                  {
                        try {mainThread.Abort();}
                        catch(ThreadStateException)
                        {
                              Console.WriteLine("Cannot abort main Thread
in {0}", this);
                        }
                  }
                  // Unregister this
                  MessageTransporter.GetMT().Unregister(GetId());
```

```
                DirectoryFacilitator.GetDF().Unregister(GetId());
                // Abort all dependent threads
                for (int i = 0; i < threads.Count; i++)
                {
                      WeakReference wr = (WeakReference) threads[i];
                      Thread t = (Thread) wr.Target;
                      if (t != null)
                            t.Abort();
                }
          }

          public override String ToString()
          {
                return "[Agent:" + GetType().Name + "]";
                //return "[Agent:" + GetId() + "]";
          }
      }
}
```

11. Class MAP.Activity

```
using System;
using System.Threading;
using System.Collections;

namespace MAP
{
      /// <summary>
      /// Defines a piece of behaviour for an Agent.
      /// It is aimed at providing modularity in the implementation of an
Agent's
      /// behaviour. An Activity can have its own data, allowing for a
clear
      /// separation between activity-dependent data and the Agent's mental
state.
      /// If work involves multithreading, Threads should be created via
the NewThread
      /// method for allowing the Agent to keep control over its behaviour.
      /// </summary>
      public abstract class Activity
      {
            // The Agent this belongs to.
            protected readonly Agent agent;

            // The Thread that executes this
            protected Thread mainThread;

            protected Activity(Agent ag)
            {
                  agent = ag;
                  mainThread = null;
            }

            /// <summary>
            /// Defines the work performed by this when its main thread is
started.
            /// Default implementation does nothing (thread finishes
immediately).
            /// </summary>
            public virtual void Execute() {}
```

```
            /// <summary>
            /// Returns the main Thread owned by this. Returns null if
thread has never
            /// been started.
            /// Allows for the control by the agent over its behaviour.
            /// </summary>
            public Thread MainThread
            {
                  get
                  {
                        return mainThread;
                  }
            }

            /// <summary>
            /// Returns a new thread that is referenced by the agent for
control.
            /// </summary>
            /// <param name="starter"></param>
            /// <returns></returns>
            protected Thread NewThread(ThreadStart starter)
            {
                  return agent.NewThread(starter);
            }

            /// <summary>
            /// A shortcut for posting a Message to the local
MessageTransporter.
            /// </summary>
            /// <param name="m"></param>
            protected void SendMessage(Message m)
            {
                  MessageTransporter.GetMT().PostMessage(m);
            }

            /// <summary>
            /// Starts the main thread of this. Main thread executes method
Execute.
            /// </summary>
            public void Start()
            {
                  ThreadStart starter = new ThreadStart(Execute);
                  mainThread = NewThread(starter);
                  mainThread.Start();
            }

            public override string ToString()
            {
                  return "[Activity:" + base.ToString() + "]";
            }
      }
}
```

12. Class MAP.Conversation

```
using System;
using System.Threading;

namespace MAP
```

```
{
      /// <summary>
      /// Allows for specific conversations between Agents with
synchronized and
      /// asynchronized Message sending.
      /// Conversation messages are not stored into the Agent's
MessageQueue after the
      /// Conversation has been initiated, i.e. the Agent has sent at least
one Message
      /// as part of the Conversation. Instead, Messages are obtained
directly from the
      /// Conversation object by the mean of the MessageArrived event of
the Agent.
      /// </summary>
      [Serializable]public class Conversation : ICloneable
      {
            // Last received Message in the synchronized and async cases
            private Message syncReply, asyncReply;
            // To get the agent's MessageArrived event
            private Agent agent;
            // The main lock
            private Object aLock;
            // Whether a Message is expected after an async Message sending
            private bool expectingMessage;
            // MessageArrived event handlers
            private MessageArrivedEventHandler syncHandler, asyncHandler;
            // Unique ID of this
            private Guid identifier;
            // States and current state
            private enum States {JUST_CREATED, NOT_INITIATED, INITIATED,
CLOSED};
            private States currentState;

            public Conversation() : this(Guid.NewGuid(),
States.JUST_CREATED) {}

            private Conversation(Guid id, States state)
            {
                  identifier = id;
                  syncReply = null;
                  asyncReply = null;
                  expectingMessage = false;
                  aLock = new Object();
                  agent = null;
                  currentState = state;
                  syncHandler = new
MessageArrivedEventHandler(HandleSyncNotification);
                  asyncHandler = new
MessageArrivedEventHandler(HandleAsyncNotification);
            }

            /// <summary>
            /// Returns whether a Message belongs to the context of this.
            /// </summary>
            /// <param name="m"></param>
            /// <returns></returns>
            public bool AppliesTo(Message m)
            {
                  return this.Equals(m.Conversation);
            }
```

```
            /// <summary>
            /// Sends a Message asynchronously in the context of this.
            /// Reply can be detected through the ExpectingReply and Reply
properties.
            /// Raises an exception if a reply to an AsyncSend is already
expected.
            /// </summary>
            /// <param name="m"></param>
            /// <param name="ag"></param>
            /// <returns></returns>
            public void AsyncSend(Message m, Agent ag)
            {
                  Monitor.Enter(aLock);
                  CheckState();
                  asyncReply = null;
                  expectingMessage = true;
                  agent = ag;
                  agent.MessageArrived += asyncHandler;
                  m.SetConversation(GetUpdatedClone());
                  MessageTransporter.GetMT().PostMessage(m);
                  Monitor.Exit(aLock);
            }

            /// <summary>
            /// Checks that a reply to an AsyncSend is not expected.
            /// Called by SendAndWait and AsyncSend.
            /// </summary>
            private void CheckState()
            {
                  // Supposedly holding lock already
                  if (expectingMessage || currentState == States.CLOSED)
                  {
                        Monitor.Exit(aLock);
                        throw new Exception("Cannot send message because a
"
                              + "reply is expected or Conversation is
closed.");
                  }
            }

            public object Clone()
            {
                  return new Conversation(this.identifier,
this.currentState);
            }

            /// <summary>
            /// Closes this, i.e. it becomes impossible to send Messages in
the context of
            /// this, and if a Message is expected after an AsyncSend then
it is ignored.
            /// </summary>
            public void Close()
            {
                  Monitor.Enter(aLock);
                  if (expectingMessage)
                  {
                        agent.MessageArrived -= asyncHandler;
                        expectingMessage = false;
                  }
                  currentState = States.CLOSED;
```

```
            Monitor.Exit(aLock);
        }

        public override bool Equals(object obj)
        {
            if (obj == null || !(obj is Conversation)) return false;
            Conversation peer = (Conversation) obj;
            return this.identifier.Equals(peer.identifier);
        }

        /// <summary>
        /// Returns whether a reply is expected in the context of this.
        /// True only after an AsyncSend until Reply becomes not null.
        /// </summary>
        public bool IsExpectingReply
        {
            get
            {
                lock (aLock) {return expectingMessage;}
            }
        }

        public override int GetHashCode()
        {
            return identifier.GetHashCode();
        }

        /// <summary>
        /// Gets a clone to send and updates states of clone and this.
        /// </summary>
        /// <returns></returns>
        private Conversation GetUpdatedClone()
        {
            // Clone because can be used by local peer Agent
            Conversation clone = (Conversation)this.Clone();
            if (currentState == States.JUST_CREATED)
            {
                clone.currentState = States.NOT_INITIATED;
                this.currentState = States.INITIATED;
            }
            else if (currentState == States.NOT_INITIATED)
            {
                clone.currentState = States.INITIATED;
                this.currentState = States.INITIATED;
            } // If state = INITIATED or CLOSED than keep state
            return clone;
        }

        /// <summary>
        /// Called by event MessageArrived in MessageQueue after an
AsyncSend.
        /// </summary>
        /// <param name="m"></param>
        private void HandleAsyncNotification(Message m)
        {
            Monitor.Enter(aLock);
            if (this.AppliesTo(m))
            {
                asyncReply = m;
                agent.MessageArrived -= asyncHandler;
                agent = null;
```

```
                        expectingMessage = false;
                  }
                  Monitor.Exit(aLock);
            }

            /// <summary>
            /// Called by event MessageArrived in MessageQueue after a
SendWaitForReply.
            /// </summary>
            /// <param name="m"></param>
            private void HandleSyncNotification(Message m)
            {
                  Monitor.Enter(aLock);
                  if (this.AppliesTo(m))
                  {
                        syncReply = m;
                        Monitor.Pulse(aLock); // Awakens thread in
SendWaitForReply
                  }
                  Monitor.Exit(aLock);
            }

            /// <summary>
            /// Returns whether Messages in the context of this do not need
to be pushed
            /// into the Agent's MessageQueue since they are handled
already.
            /// </summary>
            internal bool IsHandled
            {
                  get
                  {
                        return currentState != States.NOT_INITIATED;
                  }
            }

            /// <summary>
            /// Returns the Message received in the context of this after
an AsyncSend.
            /// </summary>
            public Message Reply
            {
                  get
                  {
                        lock (aLock) {return asyncReply;}
                  }
            }

            /// <summary>
            /// Sends a Message asynchronously in the context of this and
closes this.
            /// </summary>
            /// <param name="m"></param>
            /// <returns></returns>
            public void SendAndClose(Message m)
            {
                  CheckState();
                  currentState = States.CLOSED;
                  m.SetConversation(GetUpdatedClone());
                  MessageTransporter.GetMT().PostMessage(m);
            }
```

```
            /// <summary>
            /// Allows for synchronized Message exchanges between Agents.
            /// Raises an exception if a reply to an AsyncSend is expected.
            /// </summary>
            /// <param name="m"></param>
            /// <param name="ag"></param>
            /// <returns></returns>
            public Message SendWaitForReply(Message m, Agent ag)
            {
                  Monitor.Enter(aLock);
                  CheckState();
                  syncReply = null;
                  agent = ag;
                  agent.MessageArrived += syncHandler;
                  m.SetConversation(GetUpdatedClone());
                  MessageTransporter.GetMT().PostMessage(m);
                  Monitor.Wait(aLock); // Wait for notification
                  agent.MessageArrived -= syncHandler;
                  Monitor.Exit(aLock);
                  agent = null;
                  return syncReply;
            }

      }
}
```

3 Application

13. Class Architecture.UserId

```
using System;

namespace Architecture
{
      /// <summary>
      /// Defines a unique user identifier.
      /// </summary>
      [Serializable]public class UserId
      {
            // The static IP address of the user
            private string _ip;

            public UserId(string ipAddress)
            {
                  _ip = ipAddress;
            }

            public string IPAddress
            {
                  get
                  {
                        return _ip;
                  }
            }

            public override bool Equals(object obj)
            {
```

```
                  if (!(obj is UserId)) return false;
                  UserId peer = (UserId) obj;
                  return peer.IPAddress == this.IPAddress;
            }

            public override int GetHashCode()
            {
                  return _ip.GetHashCode();
            }

            public override string ToString()
            {
                  return "[User IP:" + IPAddress + "]";
            }

      }
}
```

14. Class Architecture.ResourceId

```
using System;

namespace Architecture
{
      /// <summary>
      /// Defines an identifier for a resource in a repository.
      /// </summary>
      [Serializable]public class ResourceId
      {
            // Simply the full unique name of the resource
            private string _resourceName;

            public ResourceId(string resourceName)
            {
                  _resourceName = resourceName;
            }

            public string Name
            {
                  get
                  {
                        return _resourceName;
                  }
            }

            public override bool Equals(object obj)
            {
                  if (!(obj is ResourceId)) return false;
                  ResourceId peer = (ResourceId) obj;
                  return peer.Name == this.Name;
            }

            public override int GetHashCode()
            {
                  return _resourceName.GetHashCode();
            }

            /// <summary>
            /// Returns the name of the service consisting in managing the
corresponding
```

```
            /// resource.
            /// </summary>
            /// <returns></returns>
            public string GetManagingService()
            {
                  return "Resource_management_" + Name;
            }

            public override string ToString()
            {
                  return "[Resource:" + Name + "]";
            }

      }
}
```

15. Structs for message contents

```
using System;
using Architecture.Profiles;

namespace Architecture
{
      public struct Subjects
      {
            public static readonly string BIGeneration =
"Business_intelligence_generate";
            public static readonly string Navig = "Navigation_act";
            public static readonly string UACreation = "User_agent_create";
            public static readonly string UAReady = "User_agent_ready";
            public static readonly string SaveUserProfile =
"Save_user_profile";
            public static readonly string TerminationAccepted =
"Termination_ok";
            public static readonly string TerminationRequest =
"Termination_request";
      }

      [Serializable]public struct NavigationActContent
      {
            public UserId userId;
            public ResourceId resourceId;
            public DateTime timestamp;

            public NavigationActContent(UserId user, ResourceId resource,
                  DateTime occurrenceTimestamp)
            {
                  userId = user;
                  resourceId = resource;
                  timestamp = occurrenceTimestamp;
            }
      }
}
```

16. Class Architecture.UAProxyAg – User Agent Proxy

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Threading;
using MAP;

namespace Architecture
{
	/// <summary>
	/// User Agents Proxy agent.
	/// </summary>
	public class UAProxyAg : Agent
	{
		public static readonly string SERVICE = "User_agents_proxy";

		// Message filters
		private static readonly MessageCategory Navigation =
			new MessageCategory(Message.Natures.Inform,
Subjects.Navig);
		private static readonly MessageCategory ReadyNotification =
			new MessageCategory(Message.Natures.Inform,
Subjects.UAReady);

		// Table of (userId : UserId, UserAgentData : UAData)
		// For knowing all the UserAgents' states.
		private Hashtable uaTable;
		// UserAgent maker
		internal AgentId uaMaker;

		public UAProxyAg() : base()
		{
			DirectoryFacilitator.GetDF().RegisterProvider(SERVICE,
GetId());
			uaTable = Hashtable.Synchronized(new Hashtable());
			msgQueue.AddFilters(new MessageCategory[]
				{Navigation, ReadyNotification});
			uaMaker = DirectoryFacilitator.GetDF().FindProviderOf(
				UAMakerAg.SERVICE);
			Debug.Assert(uaMaker != null);
		}


		protected override void Execute()
		{
			Step1: // Waiting
				while (msgQueue.IsEmpty);
				Message m = msgQueue.Dequeue();
				if (Navigation.IsCategoryOf(m)) goto Step2;
				else goto Step3;

			Step2: // Initiating NavigationHandling Activity
				RunNavigationHandling(m);
				goto Step1;
			Step3: // Initiating TerminationExamination Activity
				RunTerminationExamination(m);
				goto Step1;
		}

		private void RunNavigationHandling(Message m)
		{
			new NavigationHandlingActivity(this, uaTable, m).Start();
```

```
            }

            private void RunTerminationExamination(Message m)
            {
                  new TerminationExaminationActivity(this, uaTable, m,
                        m.Conversation).Start();
            }
      }

      /// <summary>
      /// Data about a User Agent in uaTable in UAProxyAg
      /// </summary>
      class UAData
      {
            public enum States {AVAILABLE, BEING_CREATED, TERMINATED};

            public AgentId agent;
            public States state;
            public DateTime lastMessageTimestamp;
            public Queue messages;

            public UAData()
            {
                  agent = null;
                  state = States.BEING_CREATED;
                  lastMessageTimestamp = DateTime.Now;
                  messages = null;
            }
      }

      class NavigationHandlingActivity : Activity
      {
            private Message message;
            private Hashtable uaTable;

            public NavigationHandlingActivity(Agent agent, Hashtable
_uaTable,
                  Message _message) : base(agent)
            {
                  message = _message;
                  uaTable = _uaTable;
            }

            public override void Execute() {
                  NavigationActContent navig =
(NavigationActContent)message.Content;
                  UserId userId = navig.userId;

                  // Checking UserAgent state
                  UAData data = (UAData) uaTable[userId]; // uaTable is
synchronized
                  if (data == null)
                  {
                        // Create new entry in table
                        data = new UAData();
                        Monitor.Enter(data);                    // LOCK
ON on UAData
                        uaTable.Add(userId, data);
                        InitiateUACreation(data, userId);
                  }
                  else
```

```
                {
                      Monitor.Enter(data);                        // LOCK
ON on UAData
                      UAData.States state = data.state;
                      if (state == UAData.States.AVAILABLE)
                      {
                            ForwardMessage(data);
                      }
                      else if (state == UAData.States.BEING_CREATED)
                      {
                            UpdateMessageQueue(data);
                      }
                      else // state == TERMINATED
                      {
                            InitiateUACreation(data, userId);
                      }
                }
          }

          private void FinishUACreation(UAData data, UserId userId)
          {
                // Forward all Messages in waiting queue
                foreach (Message m in data.messages)
                {
                      m.Sender = agent.GetId();
                      m.Recipient = data.agent;
                      SendMessage(m);
                }
                data.messages = null;
                data.state = UAData.States.AVAILABLE;
                Monitor.Exit(data);                                   //
LOCK OFF on UAData
          }

          private void ForwardMessage(UAData data)
          {
                data.lastMessageTimestamp = DateTime.Now;
                message.Sender = agent.GetId();
                message.Recipient = data.agent;
                Monitor.Exit(data);                                   //
LOCK OFF on UAData
                SendMessage(message);
          }

          private void InitiateUACreation(UAData data, UserId userId)
          {
                data.state = UAData.States.BEING_CREATED;
                // Create queue of waiting Messages
                data.messages = new Queue();
                data.messages.Enqueue(message);
                Monitor.Exit(data);                                   //
LOCK OFF on UAData
                // data can be accessed again: threads that access it
will be in the
                // BEING_CREATED case
                Message reply = RequestUACreation(userId);

                Monitor.Enter(data);                          // LOCK
ON on UAData
                data.agent = reply.Sender;
                FinishUACreation(data, userId);
```

```
            }

            private Message RequestUACreation(UserId userId)
            {
                  Message m = new Message(agent.GetId(),
((UAProxyAg)agent).uaMaker,
                        Message.Natures.Request, Subjects.UACreation,
userId);
                  Conversation conv = new Conversation();
                  Message reply = conv.SendWaitForReply(m, agent);
                  return reply;
            }

            private void UpdateMessageQueue(UAData data)
            {
                  Debug.Assert(data.messages != null);
                  data.messages.Enqueue(message);
                  data.lastMessageTimestamp = DateTime.Now;
                  Monitor.Exit(data);                                    //
LOCK OFF on UAData
            }
      }

      class TerminationExaminationActivity : Activity
      {
            // Security time gap for confirming agent termination
            private static TimeSpan SecurityGap =
Middleware.MAX_LATENCY.Add(
                  Middleware.MAX_LATENCY.Add(Middleware.MAX_LATENCY));

            // The Message sent by the UserAgent as a termination request
            private Message message;
            private Hashtable uaTable;
            private Conversation conversation;

            public TerminationExaminationActivity(Agent agent, Hashtable
_uaTable,
                  Message _message, Conversation _conversation) :
base(agent)
            {
                  message = _message;
                  uaTable = _uaTable;
                  conversation = _conversation;
            }

            public override void Execute()
            {
                  UserId userId = (UserId)message.Content;
                  UAData data = (UAData) uaTable[userId]; // uaTable is
synchronized
                  Debug.Assert(data != null);
                  Monitor.Enter(data);                // LOCK ON on UAData
                  if (CheckIdleTime(data.lastMessageTimestamp))
                  {
                        data.state = UAData.States.TERMINATED;
                        Monitor.Exit(data);                 // LOCK OFF on
UAData
                        SendConfirmation();
                  }
                  else
                  {
```

```
                        Monitor.Exit(data);                // LOCK OFF on
UAData
                        SendDenial();
                  }
            }

            // Checks that the last Message sent to the UserAgent was sent
long enough
            // ago to be sure that it has arrived
            private bool CheckIdleTime(DateTime timestamp)
            {
                  TimeSpan gap = DateTime.Now.Subtract(timestamp);
                  // Sent since a time that is greater than the security
gap
                  return gap.CompareTo(SecurityGap) > 0;
            }

            private void SendConfirmation()
            {
                  Message m = new Message(agent.GetId(), message.Sender,
                        Message.Natures.Inform,
Subjects.TerminationAccepted);
                  conversation.SendAndClose(m);
            }

            private void SendDenial()
            {
                  Message m = new Message(agent.GetId(), message.Sender,
                        Message.Natures.Inform, "Deny termination");
                  conversation.SendAndClose(m);
            }
      }
}
```

17. Class Architecture.UAMakerAg – User Agent Maker

```
using System;
using System.Threading;
using MAP;

namespace Architecture
{
      /// <summary>
      /// Plays the role of a factory for UserAgents.
      /// </summary>
      public class UAMakerAg : Agent
      {
            public static readonly string SERVICE = "User_agent_factory";

            // Message filter
            public static readonly MessageCategory UACreation = new
MessageCategory(
                  Message.Natures.Request, Subjects.UACreation);

            public UAMakerAg() : base()
            {
                  DirectoryFacilitator.GetDF().RegisterProvider(SERVICE,
GetId());
```

```
                  msgQueue.AddFilters(new MessageCategory[] {UACreation});
            }

            protected override void Execute()
            {
                  while (true)
                  {
                        while (msgQueue.IsEmpty);
                        Message m = msgQueue.Dequeue();
                        new AgentMakingActivity(this, m).Start();
                  }
            }
      }

      class AgentMakingActivity : Activity
      {
            private Message requestMessage;

            public AgentMakingActivity(Agent agent, Message request) :
base(agent)
            {
                  requestMessage = request;
            }

            public override void Execute()
            {
                  UserId userId = (UserId) requestMessage.Content;
                  Conversation conversation = requestMessage.Conversation;
                  // CALL TO DBMANAGER HERE
                  new UserAg(userId, conversation);
            }
      }
}
```

18. Class Architecture.UserAg – User Agent

```
using System;
using MAP;
using Architecture.Profiles;
using System.Diagnostics;

namespace Architecture
{
      /// <summary>
      /// User Agent class.
      /// </summary>
      public class UserAg : Agent
      {
            // Threshold timespan after which this terminates if not
notified of any
            // user navigation act
            public static TimeSpan STOP_THRESHOLD = new TimeSpan(0, 30, 0);

            // ID of user managed and UP
            private UserId user;
            private UP userProfile;
            // UserAgent and NodeAgent Proxy IDs
            private AgentId proxy;
            // Message filter
            private MessageCategory Navigation = new MessageCategory(
```

```
                  Message.Natures.Inform, Subjects.Navig);
            // Last resource explored
            private ResourceId lastResource;
            // Previous Link traversed
            private UPLink prevLink;
            // Timestamp of arrival on previous resource
            private DateTime prevArrival;
            // Conversation between UAProxy and UAMaker for the creation of
this
            private Conversation initConversation;

            public UserAg(UserId userId, Conversation conversation)
            {
                  user = userId;
                  userProfile = UP.CreateUP(userId);
                  initConversation = conversation;
                  UserAgInit();
            }

            public UserAg(UserId userId, string[] supNames, Conversation
conversation)
            {
                  user = userId;
                  userProfile = UP.CreateUP(userId, supNames);
                  initConversation = conversation;
                  UserAgInit();
            }

            private void UserAgInit()
            {
                  lastResource = null; prevLink = null;
                  msgQueue.AddFilters(new MessageCategory[] {Navigation});
                  proxy = DirectoryFacilitator.GetDF().FindProviderOf(
                        UAProxyAg.SERVICE);
                  Debug.Assert(proxy != null);
            }

            protected override void Execute()
            {
                  DateTime idleStart = DateTime.Now;
                  NotifyProxy();
                  Message m;
                  NavigationActContent navig;

                  Step1: // Wait
                        while (msgQueue.IsEmpty)
                              if (ThresholdTimeExpired(idleStart)) goto
Step2;
                        goto Step4;

                  Step2: // Pre-termination
                        SaveData(); // Before termination request otherwise
a new UA could be
                                          // created and initialized before
data is saved
                        Conversation termConv = RequestTermination();
                        while (termConv.IsExpectingReply &&
msgQueue.IsEmpty);
                        if (!msgQueue.IsEmpty)
                        {
```

```
                              // Message received: interrupt pre-
termination phase
                              termConv.Close();
                              idleStart = DateTime.Now;
                              goto Step1;
                        }
                        // Answer from proxy received
                        Message answer = termConv.Reply;
                        bool authorized = (answer.Subject ==
Subjects.TerminationAccepted);
                        if (authorized) goto Step3;
                        else
                        {
                              // Add network latency time for waiting a bit
more
                              idleStart.Add(Middleware.MAX_LATENCY);
                              goto Step1;
                        }

                  Step3: // Terminating
                        Terminate();
                        return;

                  Step4: // Handling user's navigation
                        m = msgQueue.Dequeue();
                        navig = (NavigationActContent) m.Content;
                        UpdateProfile(navig);
                        if (!msgQueue.IsEmpty) goto Step4;
                        else goto Step5;

                  Step5: // BI generation initiation
                        RequestBIGeneration();
                        idleStart = DateTime.Now;
                        goto Step1;
            }

            // ****** Agent activities and transitions ******

            // Inform UserAgentProxy that this is ready
            private void NotifyProxy()
            {
                  Message m = new Message(GetId(), proxy,
Message.Natures.Inform,
                        Subjects.UAReady);
                  initConversation.SendAndClose(m);
                  initConversation = null;
            }

            private void RequestBIGeneration()
            {
                  AgentId generator =
DirectoryFacilitator.GetDF().FindProviderOf(
                        BIGeneratorAg.SERVICE);
                  Debug.Assert(generator != null);
                  Message m = new Message(GetId(), generator,
Message.Natures.Request,
                        Subjects.BIGeneration, userProfile);
                  SendMessage(m);
            }

            private Conversation RequestTermination()
```

```
            {
                  Message m = new Message(GetId(), proxy,
Message.Natures.Request,
                        Subjects.TerminationRequest);
                  Conversation termConv = new Conversation();
                  termConv.AsyncSend(m, this);
                  return termConv;
            }

            private void SaveData()
            {
                  // Useless for the moment since all User Profile data is
static
                  /*
                  AgentId dbManager =
DirectoryFacilitator.GetDF().FindProviderOf(
                        DBManagerAg.SERVICE);
                  Message m = new Message(GetId(), dbManager,
Message.Natures.Request,
                        Subjects.SaveUP, userProfile);
                  // Wait until operation is finished
                  Conversation c = new Conversation();
                  c.SendWaitForReply(m, this);*/
            }

            private bool ThresholdTimeExpired(DateTime idleStart)
            {
                  return DateTime.Now.Subtract(idleStart) > STOP_THRESHOLD;
            }

            private void UpdateProfile(NavigationActContent navig)
            {
                  ResourceId newResource = navig.resourceId;
                  if (lastResource == null)
                        // No previous resource navigated
                        userProfile.AddNodeOn(newResource);
                  else
                  {
                        // User comes from another resource
                        if (prevLink != null)
                        {
                              // User comes from another Link
                              // Update read time on previous Link
                              TimeSpan elapsed =
navig.timestamp.Subtract(prevArrival);

      prevLink.AddReadTime((long)elapsed.TotalMilliseconds);
                        }
                        // Add/get new Link
                        prevLink = userProfile.AddLinkBetween(lastResource,
newResource)
                              as UPLink;
                  }
                  userProfile.LastTimestamp = navig.timestamp;
                  prevArrival = navig.timestamp;
                  lastResource = newResource;
                  Console.WriteLine(userProfile);
            }

      }
}
```

19. Class Architecture.DBManagerAg – Database manager

```
using System;
using System.Data;
using System.Data.SqlClient;
using MAP;
using Architecture.Profiles;

namespace Architecture
{
	/// <summary>
	/// Database Manager.
	/// </summary>
	public class DBManagerAg : Agent
	{
		public static readonly string SERVICE = "DB_management";

		// Database name
		private static readonly string DB = "agentdb";

		private SqlConnection connection;
		private DataSet upDataSet;

		public DBManagerAg() {}

		protected override void Execute()
		{
			Initialize();
			CleanupDB();

			/*
			// Nodes
			ResourceId a = new ResourceId("a");
			ResourceId b = new ResourceId("b");
			ResourceId c = new ResourceId("c");
			ResourceId d = new ResourceId("d");
			ResourceId e = new ResourceId("e");

			// Create a SUP
			string supName = "My SUP";
			SUP sup = SUP.GetSUP(supName);
			if (sup == null) sup = SUP.CreateSUP(supName, "A dummy
description");
			sup.AddLinkBetween(a, b);
			sup.AddLinkBetween(b, c);
			sup.AddLinkBetween(d, b);
			sup.AddLinkBetween(b, d);

			// Create an UP
			UP up = UP.CreateUP(new UserId("1.1.1.1"), new string[]
{supName});
			up.AddLinkBetween(a, b);
			up.AddLinkBetween(b, c);
			up.AddLinkBetween(c, b);
			UPLink link = up.GetLinkBetween(b, c) as UPLink;
			link.Traverse();
			link.AddReadTime((long)1500);
```

```
                  Console.WriteLine("Done");*/
            }

            /// <summary>
            /// Deletes all data in DB.
            /// </summary>
            private void CleanupDB()
            {
                  try
                  {
                        connection.Open();
                        UP.Cleanup(connection);
                        Console.WriteLine("{0} - DB cleaned up", this);
                  }
                  catch (Exception e)
                  {
                        Console.WriteLine("{0} - Cannot clean up DB: {1}",
this, e);
                  }
                  finally
                  {
                        connection.Close();
                  }
            }

            /// <summary>
            /// Agent initialization.
            /// </summary>
            private void Initialize()
            {
                  Console.WriteLine("{0} - Initializing data from local SQL
DB '{1}'...",
                        this, DB);
                  InitializeData();
                  Console.WriteLine("{0} - Ready", this);
                  DirectoryFacilitator.GetDF().RegisterProvider(SERVICE,
GetId());
            }

            /// <summary>
            /// Fills DataSet with DB data.
            /// </summary>
            private void InitializeData()
            {
                  try
                  {
                        connection = new SqlConnection(

      "server=(local)\\;Trusted_Connection=yes;database=" + DB);
                        upDataSet = new DataSet();
                        UP.Initialize(upDataSet, connection);
                  }
                  catch (Exception e)
                  {
                        Console.WriteLine("{0} - Data initialization
aborted: {1}", this, e);
                        Console.WriteLine(e.StackTrace);
                  }
                  finally
                  {
```

```
                        connection.Close();
                  }
            }

            /// <summary>
            /// For saving changes in DB before terminating.
            /// </summary>
            protected override void Terminate()
            {
                  UpdateDB();
                  base.Terminate();
            }

            /// <summary>
            /// Saves all changes in the DB.
            /// </summary>
            private void UpdateDB()
            {
                  try
                  {
                        Console.WriteLine("{0} - Updating DB...", this);
                        connection.Open();
                        UP.Update();
                        Console.WriteLine("{0} - DB updated", this);
                  }
                  catch (Exception e)
                  {
                        Console.WriteLine("{0} - Cannot update data: {1}",
this, e);
                  }
                  finally
                  {
                        connection.Close();
                  }
            }
      }
}
```

4 Application – Profiles

20. Class Architecture.Profiles.IProfile

```
using System;
using System.Collections;
using System.Text;

namespace Architecture.Profiles
{
      /// <summary>
      /// Defines a generic Profile made of INodes and ILinks.
      /// </summary>
      public interface IProfile
      {
            /// <summary>
            /// Adds a ILink to this, doing the linkage to already existing
INodes or
            /// creating new ones if needed. Returns the new ILink, or the
existing one
```

```
            /// if it is already present.
            /// </summary>
            /// <param name="origin"></param>
            /// <param name="destination"></param>
            /// <returns></returns>
            ILink AddLinkBetween(ResourceId origin, ResourceId
destination);

            /// <summary>
            /// Adds a new INode for the specified resource. If the INode
already exists,
            /// returns it otherwise returns a new one.
            /// </summary>
            /// <param name="resource"></param>
            /// <returns></returns>
            INode AddNodeOn(ResourceId resource);

            /// <summary>
            /// Returns a ILink between the specified resources, or null if
none exists.
            /// </summary>
            /// <param name="origin"></param>
            /// <param name="destination"></param>
            /// <returns></returns>
            ILink GetLinkBetween(ResourceId origin, ResourceId
destination);

            /// <summary>
            /// Returns a INode on the specified resource, or null if none
exists.
            /// </summary>
            /// <param name="resource"></param>
            /// <returns></returns>
            INode GetNodeOn(ResourceId resource);

            /// <summary>
            /// Returns all the INodes in this.
            /// </summary>
            INode[] GetNodes();

            /// <summary>
            /// Returns whether the specified ILink exists in this.
            /// </summary>
            /// <param name="origin"></param>
            /// <param name="destination"></param>
            /// <returns></returns>
            bool HasLinkBetween(ResourceId origin, ResourceId destination);

            /// <summary>
            /// Returns whether the specified INode exists in this.
            /// </summary>
            /// <param name="resource"></param>
            /// <returns></returns>
            bool HasNodeOn(ResourceId resource);
      }
}
```

21. Class Architecture.Profiles.INode

```
using System;
```

```
using System.Collections;

namespace Architecture.Profiles
{
	/// <summary>
	/// Defines a Node on a given resource in a Profile.
	/// </summary>
	public interface INode
	{
		/// <summary>
		/// The resource of this.
		/// </summary>
		ResourceId Resource {get;}

		/// <summary>
		/// Returns all the links whose origin is this.
		/// </summary>
		/// <returns></returns>
		ILink[] GetLinks();

		/// <summary>
		/// Returns a Link whose origin is this and whose destination
is the specified
		/// resource. If no such Link exists, returns null.
		/// </summary>
		/// <param name="resource"></param>
		/// <returns></returns>
		ILink GetLinkTo(ResourceId resource);

		/// <summary>
		/// Returns whether this contains a Link to the specified
resource.
		/// </summary>
		/// <param name="resource"></param>
		/// <returns></returns>
		bool HasLinkTo(ResourceId resource);
	}
}
```

22. Class Architecture.Profiles.ILink

```
using System;

namespace Architecture.Profiles
{
	/// <summary>
	/// Defines a Link between two resources (Nodes) in a Profile.
	/// </summary>
	public interface ILink
	{
		INode Destination {get;}
	}
}
```

23. Class Architecture.Profiles.DataUtilities

```
using System;
using System.Data;
```

```
using System.Data.SqlClient;

namespace Architecture.Profiles
{
	/// <summary>
	/// Utility class for data management by ADO.
	/// </summary>
	internal class DataUtilities
	{

		/// <summary>
		/// Deletes all data from the specified table in the database.
		/// </summary>
		/// <param name="connection"></param>
		/// <param name="tableName"></param>
		/// <returns></returns>
		public static void CleanupTable(SqlConnection connection,
			string tableName)
		{
			SqlCommand command = new SqlCommand("DELETE FROM " +
tableName,
				connection);
			command.ExecuteNonQuery();
		}

		/// <summary>
		/// Fills a DataSet with content of a table through a
Connection and creates
		/// and returns a DataAdapter.
		/// </summary>
		/// <param name="dataSet"></param>
		/// <param name="connection"></param>
		/// <param name="tableName"></param>
		/// <returns></returns>
		public static SqlDataAdapter InitializeTable(DataSet dataSet,
			SqlConnection connection, string tableName)
		{
			SqlDataAdapter adapter = new SqlDataAdapter();
			adapter.MissingSchemaAction =
MissingSchemaAction.AddWithKey;

			// Create commands
			string commandText = "SELECT * FROM " + tableName;
			adapter.SelectCommand = new SqlCommand(commandText,
connection);
			SqlCommandBuilder builder = new
SqlCommandBuilder(adapter);
			// Fill dataset
			adapter.Fill(dataSet, tableName);

			return adapter;
		}

	}
}
```

24. Class Architecture.Profiles.UP – User Profile

```
using System;
using System.Collections;
```

```
namespace Architecture.Profiles
{
	using System;
	using System.Data;
	using System.Data.SqlClient;
	using System.Diagnostics;

	/// <summary>
	/// Defines a User Profile. Such a Profile stores information on the
Links
	/// that provides indicators of the relevancy of the traversal of the
Links
	/// based on the user's navigation history. In addition, a User
Profile can
	/// have references to SUPs that suit to the user.
	/// </summary>
	[Serializable]public class UP : IProfile
	{
		// Data tables
		private static readonly string TABLE_NAME = "UP";
		private static readonly string UP_SUP_TABLE_NAME = "UP_SUP";
		private static DataTable table, upSupTable;
		// SqlDataAdapters
		private static SqlDataAdapter adapter, upSupAdapter;
		// Column names
		private static readonly string USER_ID = "userId";
		private static readonly string LAST_USE = "lastUse";
		// In UP_SUP
		private static readonly string UP_COL = "UP";
		private static readonly string SUP_COL = "SUP";
		// Relations
		private static DataRelation ToNodes, ToSUPs;

		public static void Cleanup(SqlConnection connection)
		{
			SUP.Cleanup(connection);
			UPNode.Cleanup(connection);
			DataUtilities.CleanupTable(connection, TABLE_NAME);
			// Clear all Tables in DataSet
			DataSet dataSet = Data.DataSet;
			foreach (DataTable dt in dataSet.Tables)
				dt.Rows.Clear();
		}

		/// <summary>
		/// Creates new UP data.
		/// </summary>
		/// <param name="resource"></param>
		internal static UP CreateUP(UserId user) {
			return CreateUP(user, new string[] {});
		}
		internal static UP CreateUP(UserId user, string[] supNames)
		{
			Debug.Assert(user != null && supNames != null);
			if (Data.Rows.Find(user.IPAddress) != null)
				throw new Exception("UP for user " + user + "
exists already");
			DataRow row = Data.NewRow();
			row[USER_ID] = user.IPAddress;
			row[LAST_USE] = DateTime.Now;
```

```
                lock(Data) {Data.Rows.Add(row);}
                UP up = new UP(user);
                // Add SUPs
                foreach (string supName in supNames)
                        up.AddSUP(supName);
                return up;
        }

        internal static DataTable Data
        {
                get
                {
                        return table;
                }
        }

        public static UP GetUP(UserId user)
        {
                DataRow row = Data.Rows.Find(user.IPAddress);
                if (row == null) return null;
                return new UP(row);
        }

        public static void Initialize(DataSet dataSet, SqlConnection
connection)
        {
                SUP.Initialize(dataSet, connection);
                UPNode.Initialize(dataSet, connection);
                adapter = DataUtilities.InitializeTable(dataSet,
connection, TABLE_NAME);
                table = dataSet.Tables[TABLE_NAME];
                // Create relation
                ToNodes = table.DataSet.Relations.Add("UP_To_UPNodes",
                        table.Columns[USER_ID],
                        UPNode.Data.Columns[UPNode.UP],
                        false);
                // Linkage between UP and SUP
                upSupAdapter = DataUtilities.InitializeTable(dataSet,
connection,
                        UP_SUP_TABLE_NAME);
                upSupTable = dataSet.Tables[UP_SUP_TABLE_NAME];
                ToSUPs = upSupTable.DataSet.Relations.Add("UP_To_SUPs",
                        table.Columns[USER_ID],
                        upSupTable.Columns[UP_COL],
                        false);
        }

        public static void Update()
        {
                SUP.Update();
                UPNode.Update();
                adapter.Update(Data);
        }


        //************ NON-STATIC PART *************

        private UserId _user;

        /// <summary>
        /// Wraps already existing data as a SUP object.
```

```
            /// </summary>
            /// <param name="name"></param>
            private UP(DataRow row) : this(new
UserId((string)row[USER_ID])) {}
            private UP(UserId user)
            {
                  _user = user;
            }

            public UserId User
            {
                  get
                  {
                        return _user;
                  }
            }

            private DataRow Row
            {
                  get
                  {
                        return Data.Rows.Find(_user.IPAddress);
                  }
            }

            private void AddSUP(string supName)
            {
                  SUP sup = SUP.GetSUP(supName);
                  if (sup == null)
                        throw new Exception("Cannot find SUP " + supName);
                  DataRow row = upSupTable.NewRow();
                  row[UP_COL] = User.IPAddress;
                  row[SUP_COL] = supName;
                  // Should not be present already
                  lock (upSupTable)
                  {
                        upSupTable.Rows.Add(row);
                  }
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public ILink AddLinkBetween(ResourceId origin, ResourceId
destination)
            {
                  ILink link = GetLinkBetween(origin, destination);
                  if (link != null) return link;
                  UPNode origNode = AddNodeOn(origin) as UPNode;
                  UPNode destinNode = AddNodeOn(destination) as UPNode;
                  return UPLink.CreateLink(origNode, destinNode);
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public INode AddNodeOn(ResourceId resource)
            {
                  INode node = GetNodeOn(resource);
                  if (node != null) return node;
                  return UPNode.CreateNode(this, resource);
```

```
            }

            public override bool Equals(object obj)
            {
                  if (!(obj is UP)) return false;
                  UP peer = (UP) obj;
                  return this.User.Equals(peer.User);
            }

            public override int GetHashCode()
            {
                  return _user.GetHashCode();
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public ILink GetLinkBetween(ResourceId origin, ResourceId
destination)
            {
                  INode node = GetNodeOn(origin);
                  if (node == null) return null;
                  return node.GetLinkTo(destination);
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public INode GetNodeOn(ResourceId resource)
            {
                  DataRow[] rows = Row.GetChildRows(ToNodes);
                  foreach (DataRow row in rows)
                  {
                        ResourceId peer = new
ResourceId((string)row[UPNode.RESOURCE]);
                        if (resource.Equals(peer))
                              return new UPNode(row);
                  }
                  return null;
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public INode[] GetNodes()
            {
                  DataRow[] rows = Row.GetChildRows(ToNodes);
                  System.Collections.ArrayList list = new
System.Collections.ArrayList(
                        rows.Length);
                  foreach (DataRow row in rows)
                        list.Add(new UPNode(row));
                  return (INode[])list.ToArray(typeof (INode));
            }

            /// <summary>
            /// Returns all the SUPs associated with this.
            /// </summary>
            /// <returns></returns>
            public SUP[] GetSUPs()
            {
```

```
                DataRow[] rows = Row.GetChildRows(ToSUPs);
                ArrayList list = new ArrayList(rows.Length);
                foreach (DataRow row in rows)
                {
                        string supName = (string)row[SUP_COL];
                        list.Add(SUP.GetSUP(supName));
                }
                return (SUP[]) list.ToArray(typeof (SUP));
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public bool HasLinkBetween(ResourceId origin, ResourceId
destination)
        {
                return GetLinkBetween(origin, destination) != null;
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public bool HasNodeOn(ResourceId resource)
        {
                return GetNodeOn(resource) != null;
        }

        public DateTime LastTimestamp
        {
                get
                {
                        return (DateTime) Row[LAST_USE];
                }
                set
                {
                        lock (Row)
                        {
                                Row[LAST_USE] = value;
                        }
                }
        }
    }
}
```

25. Class Architecture.Profiles.UPNode

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Collections;

namespace Architecture.Profiles
{
    /// <summary>
    /// Defines a Node on a given resource in a User Profile.
    /// </summary>
    [Serializable]public class UPNode : INode
    {
        // Data table
```

```
            public static readonly string TABLE_NAME = "UPNode";
            private static DataTable table;
            // SqlDataAdapter
            private static SqlDataAdapter adapter;
            // Column names
            internal static readonly string ID = "identifier";
            internal static readonly string RESOURCE = "resourceId";
            internal static readonly string UP = "UP";
            // Relation
            private static DataRelation ToLinks;

            public static void Cleanup(SqlConnection connection)
            {
                  UPLink.Cleanup(connection);
                  DataUtilities.CleanupTable(connection, TABLE_NAME);
            }

            /// <summary>
            /// Creates new Node data.
            /// </summary>
            /// <param name="resource"></param>
            internal static UPNode CreateNode(UP profile, ResourceId
resource)
            {
                  Debug.Assert(profile != null && resource != null);
                  // Node should not exist already
                  Guid _id = Guid.NewGuid();
                  DataRow row = Data.NewRow();
                  row[ID] = _id;
                  row[RESOURCE] = resource.Name;
                  row[UP] = profile.User.IPAddress;
                  lock(Data) {Data.Rows.Add(row);}
                  return new UPNode(_id);
            }

            internal static DataTable Data
            {
                  get
                  {
                        return table;
                  }
            }

            public static void Initialize(DataSet dataSet, SqlConnection
connection)
            {
                  UPLink.Initialize(dataSet, connection);
                  adapter = DataUtilities.InitializeTable(dataSet,
connection, TABLE_NAME);
                  table = dataSet.Tables[TABLE_NAME];
                  // Create relation
                  ToLinks =
table.DataSet.Relations.Add("UpNode_To_UpLinks",
                        table.Columns[ID],
                        UPLink.Data.Columns[UPLink.ORIGIN],
                        false);
            }

            public static void Update()
            {
                  UPLink.Update();
```

```
      adapter.Update(Data);
}


//************ NON-STATIC PART *************

private Guid _identifier;

/// <summary>
/// Wraps already existing data as a Node object.
/// </summary>
/// <param name="row"></param>
internal UPNode(DataRow row) : this((Guid)row[ID]) {}
internal UPNode(Guid id)
{
      _identifier = id;
}

internal Guid Identifier
{
      get
      {
            return _identifier;
      }
}

public ResourceId Resource
{
      get
      {
            return new ResourceId((string)Row[RESOURCE]);
      }
}

private DataRow Row
{
      get
      {
            return Data.Rows.Find(_identifier);
      }
}

/// <summary>
/// See INode.
/// </summary>
/// <returns></returns>
public ILink[] GetLinks()
{
      DataRow[] rows = Row.GetChildRows(ToLinks);
      ArrayList list = new ArrayList(rows.Length);
      foreach (DataRow row in rows)
      {
            list.Add(new UPLink(row));
      }
      return (ILink[])list.ToArray(typeof (ILink));
}

/// <summary>
/// See INode.
/// </summary>
/// <param name="resource"></param>
```

```
            /// <returns></returns>
            public ILink GetLinkTo(ResourceId resource)
            {
                  ILink[] links = GetLinks();
                  foreach (ILink link in links)
                  {
                        if (link.Destination.Resource.Equals(resource))
                              return link;
                  }
                  return null;
            }

            /// <summary>
            /// See INode.
            /// </summary>
            /// <param name="resource"></param>
            /// <returns></returns>
            public bool HasLinkTo(ResourceId resource)
            {
                  return GetLinkTo(resource) != null;
            }

            public override bool Equals(object obj)
            {
                  if (!(obj is UPNode)) return false;
                  UPNode peer = (UPNode) obj;
                  return this._identifier.Equals(peer._identifier);
            }

            public override int GetHashCode()
            {
                  return _identifier.GetHashCode();
            }
      }
}
```

26. Class Architecture.Profiles.UPLink

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

namespace Architecture.Profiles
{
      /// <summary>
      /// Link for User Profile.
      /// </summary>
      public class UPLink : ILink
      {
            // Table
            public static readonly string TABLE_NAME = "UPLink";
            private static DataTable table;
            // SqlDataAdapter
            private static SqlDataAdapter adapter;
            // Column names
            internal static readonly string ID = "identifier";
            internal static readonly string ORIGIN = "originNode";
            internal static readonly string DESTINATION =
"destinationNode";
```

```
            internal static readonly string LAST_USE = "lastUse";
            internal static readonly string OCCURRENCES = "occurrences";
            internal static readonly string READ_TIME = "readTime";

            public static void Cleanup(SqlConnection connection)
            {
                  DataUtilities.CleanupTable(connection, TABLE_NAME);
            }

            /// <summary>
            /// Creates new UPLink data.
            /// </summary>
            /// <param name="origin"></param>
            /// <param name="destination"></param>
            internal static UPLink CreateLink(UPNode origin, UPNode
destination)
            {
                  Debug.Assert(origin != null && destination != null);
                  // Should not exist already
                  Guid _id = Guid.NewGuid();
                  DataRow row = Data.NewRow();
                  row[ID] = _id;
                  row[ORIGIN] = origin.Identifier;
                  row[DESTINATION] = destination.Identifier;
                  row[LAST_USE] = DateTime.Now;
                  row[OCCURRENCES] = 1;
                  row[READ_TIME] = 0;
                  lock(Data) {Data.Rows.Add(row);}
                  return new UPLink(_id);
            }

            internal static DataTable Data
            {
                  get
                  {
                        return table;
                  }
            }

            public static void Initialize(DataSet dataSet, SqlConnection
connection)
            {
                  adapter = DataUtilities.InitializeTable(dataSet,
connection, TABLE_NAME);
                  table = dataSet.Tables[TABLE_NAME];
            }

            public static void Update()
            {
                  adapter.Update(Data);
            }


            //************ NON-STATIC PART *************

            private Guid _identifier;

            internal UPLink(DataRow row) : this((Guid)row[ID]) {}
            internal UPLink(Guid _id)
            {
                  _identifier = _id;
```

```
}

public void AddReadTime(long readTime)
{
      Row[READ_TIME] = ReadTime + readTime;
}

public INode Destination
{
      get
      {
            return new UPNode((Guid)Row[DESTINATION]);
      }
}

private DataRow Row
{
      get
      {
            return Data.Rows.Find(_identifier);
      }
}

public override bool Equals(object obj)
{
      if (!(obj is UPLink)) return false;
      UPLink peer = (UPLink) obj;
      return this._identifier.Equals(peer._identifier);
}

public override int GetHashCode()
{
      return _identifier.GetHashCode();
}

public DateTime LastTimestamp
{
      get
      {
            return (DateTime) Row[LAST_USE];
      }
}

public int Occurrences
{
      get
      {
            return (int) Row[OCCURRENCES];
      }
}

public long ReadTime
{
      get
      {
            return (long) Row[READ_TIME];
      }
}

public void Traverse()
{
```

```
                Row[OCCURRENCES] = Occurrences + 1;
                Row[LAST_USE] = DateTime.Now;
            }
        }

}
```

27. Class Architecture.Profiles.SUP – Standard User Profile

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

namespace Architecture.Profiles
{
      /// <summary>
      /// Standard User Profile.
      /// </summary>
      [Serializable]public class SUP : IProfile
      {
            // Data table
            private static readonly string TABLE_NAME = "SUP";
            private static DataTable table;
            // SqlDataAdapter
            private static SqlDataAdapter adapter;
            // Column names
            private static readonly string SUP_NAME = "SUPName";
            private static readonly string DESCRIPTION = "description";
            // Relation
            private static DataRelation ToNodes;

            public static void Cleanup(SqlConnection connection)
            {
                  SUPNode.Cleanup(connection);
                  DataUtilities.CleanupTable(connection, TABLE_NAME);
            }

            /// <summary>
            /// Creates new SUP data.
            /// </summary>
            /// <param name="resource"></param>
            internal static SUP CreateSUP(string name)
            {
                  return CreateSUP(name, "");
            }
            internal static SUP CreateSUP(string name, string description)
            {
                  Debug.Assert(name != null);
                  if (Data.Rows.Find(name) != null)
                        throw new Exception("SUP named " + name + " exists
already");
                  DataRow row = Data.NewRow();
                  row[SUP_NAME] = name;
                  row[DESCRIPTION] = description;
                  lock(Data) {Data.Rows.Add(row);}
                  return new SUP(name);
            }
```

```
            internal static DataTable Data
            {
                  get
                  {
                        return table;
                  }
            }

            public static SUP GetSUP(string name)
            {
                  DataRow row = Data.Rows.Find(name);
                  if (row == null) return null;
                  return new SUP(row);
            }

            public static void Initialize(DataSet dataSet, SqlConnection
connection)
            {
                  SUPNode.Initialize(dataSet, connection);
                  adapter = DataUtilities.InitializeTable(dataSet,
connection, TABLE_NAME);
                  table = dataSet.Tables[TABLE_NAME];
                  // Create relation
                  ToNodes = table.DataSet.Relations.Add("SUP_To_SUPNodes",
                        table.Columns[SUP_NAME],
                        SUPNode.Data.Columns[SUPNode.SUP],
                        false);
            }

            public static void Update()
            {
                  SUPNode.Update();
                  adapter.Update(Data);
            }


            //************ NON-STATIC PART *************

            private string _name;

            /// <summary>
            /// Wraps already existing data as a SUP object.
            /// </summary>
            /// <param name="name"></param>
            private SUP(DataRow row) : this((string)row[SUP_NAME]) {}
            private SUP(string name)
            {
                  _name = name;
            }

            public string Name
            {
                  get
                  {
                        return _name;
                  }
            }

            private DataRow Row
            {
```

```
                get
                {
                        return Data.Rows.Find(_name);
                }
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public ILink AddLinkBetween(ResourceId origin, ResourceId
destination)
        {
                ILink link = GetLinkBetween(origin, destination);
                if (link != null) return link;
                SUPNode origNode = AddNodeOn(origin) as SUPNode;
                SUPNode destinNode = AddNodeOn(destination) as SUPNode;
                return SUPLink.CreateLink(origNode, destinNode);
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public INode AddNodeOn(ResourceId resource)
        {
                INode node = GetNodeOn(resource);
                if (node != null) return node;
                return SUPNode.CreateNode(this, resource);
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public ILink GetLinkBetween(ResourceId origin, ResourceId
destination)
        {
                INode node = GetNodeOn(origin);
                if (node == null) return null;
                return node.GetLinkTo(destination);
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public INode GetNodeOn(ResourceId resource)
        {
                // Get all Node rows
                DataRow[] rows = Row.GetChildRows(ToNodes);
                foreach (DataRow row in rows)
                {
                        ResourceId peer = new
ResourceId((string)row[SUPNode.RESOURCE]);
                        if (resource.Equals(peer))
                                return new SUPNode(row);
                }
                return null;
        }

        /// <summary>
        /// See IProfile.
        /// </summary>
        public INode[] GetNodes()
```

```
            {
                  DataRow[] rows = Row.GetChildRows(ToNodes);
                  System.Collections.ArrayList list = new
System.Collections.ArrayList(
                        rows.Length);
                  foreach (DataRow row in rows)
                        list.Add(new SUPNode(row));
                  return (INode[])list.ToArray(typeof (INode));
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public bool HasLinkBetween(ResourceId origin, ResourceId
destination)
            {
                  return GetLinkBetween(origin, destination) != null;
            }

            /// <summary>
            /// See IProfile.
            /// </summary>
            public bool HasNodeOn(ResourceId resource)
            {
                  return GetNodeOn(resource) != null;
            }

            public override bool Equals(object obj)
            {
                  if (!(obj is SUP)) return false;
                  SUP peer = (SUP) obj;
                  return this.Name.Equals(peer.Name);
            }

            public override int GetHashCode()
            {
                  return _name.GetHashCode();
            }
      }
}
```

28. Class Architecture.Profiles.SUPNode

```
using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

namespace Architecture.Profiles
{
      /// <summary>
      /// Defines a Node on a given resource for a SUP.
      /// </summary>
      [Serializable]public class SUPNode : INode
      {
            // Data table
            public static readonly string TABLE_NAME = "SUPNode";
            private static DataTable table;
            // SqlDataAdapter
```

```
            private static SqlDataAdapter adapter;
            // Column names
            internal static readonly string ID = "identifier";
            internal static readonly string RESOURCE = "resourceId";
            internal static readonly string SUP = "SUPName";
            // Relation
            private static DataRelation ToLinks;

            public static void Cleanup(SqlConnection connection)
            {
                  SUPLink.Cleanup(connection);
                  DataUtilities.CleanupTable(connection, TABLE_NAME);
            }

            /// <summary>
            /// Creates new Node data.
            /// </summary>
            /// <param name="resource"></param>
            internal static SUPNode CreateNode(SUP profile, ResourceId
resource)
            {
                  Debug.Assert(profile != null && resource != null);
                  // Node should not exist already
                  Guid _id = Guid.NewGuid();
                  DataRow row = Data.NewRow();
                  row[ID] = _id;
                  row[RESOURCE] = resource.Name;
                  row[SUP] = profile.Name;
                  lock(Data) {Data.Rows.Add(row);}
                  return new SUPNode(_id);
            }

            internal static DataTable Data
            {
                  get
                  {
                        return table;
                  }
            }

            public static void Initialize(DataSet dataSet, SqlConnection
connection)
            {
                  SUPLink.Initialize(dataSet, connection);
                  adapter = DataUtilities.InitializeTable(dataSet,
connection, TABLE_NAME);
                  table = dataSet.Tables[TABLE_NAME];
                  // Create relation
                  ToLinks =
table.DataSet.Relations.Add("SupNode_To_SupLinks",
                        table.Columns[ID],
                        SUPLink.Data.Columns[SUPLink.ORIGIN],
                        false);
            }

            public static void Update()
            {
                  SUPLink.Update();
                  adapter.Update(Data);
            }
```

```
//************* NON-STATIC PART *************

private Guid _identifier;

/// <summary>
/// Wraps already existing data as a Node object.
/// </summary>
/// <param name="row"></param>
internal SUPNode(DataRow row) : this((Guid)row[ID]) {}
internal SUPNode(Guid id)
{
      _identifier = id;
}

internal Guid Identifier
{
      get
      {
            return _identifier;
      }
}

public ResourceId Resource
{
      get
      {
            return new ResourceId((string)Row[RESOURCE]);
      }
}

private DataRow Row
{
      get
      {
            return Data.Rows.Find(_identifier);
      }
}

/// <summary>
/// See INode.
/// </summary>
/// <returns></returns>
public ILink[] GetLinks()
{
      DataRow[] rows = Row.GetChildRows(ToLinks);
      ArrayList list = new ArrayList(rows.Length);
      foreach (DataRow row in rows) {
            list.Add(new SUPLink(row));
      }
      return (ILink[])list.ToArray(typeof (ILink));
}

/// <summary>
/// See INode.
/// </summary>
/// <param name="resource"></param>
/// <returns></returns>
public ILink GetLinkTo(ResourceId resource)
{
      ILink[] links = GetLinks();
```

```
                  foreach (ILink link in links)
                  {
                        if (link.Destination.Resource.Equals(resource))
                              return link;
                  }
                  return null;
            }

            /// <summary>
            /// See INode.
            /// </summary>
            /// <param name="resource"></param>
            /// <returns></returns>
            public bool HasLinkTo(ResourceId resource)
            {
                  return GetLinkTo(resource) != null;
            }

            public override bool Equals(object obj)
            {
                  if (!(obj is SUPNode)) return false;
                  SUPNode peer = (SUPNode) obj;
                  return this._identifier.Equals(peer._identifier);
            }

            public override int GetHashCode()
            {
                  return _identifier.GetHashCode();
            }
      }
}
```

29. Class Architecture.Profiles.SUPLink

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Diagnostics;

namespace Architecture.Profiles
{
      /// <summary>
      /// SUPLink.
      /// </summary>
      [Serializable]public class SUPLink : ILink
      {
            // Table
            public static readonly string TABLE_NAME = "SUPLink";
            private static DataTable table;
            // SqlDataAdapter
            private static SqlDataAdapter adapter;
            // Column names
            internal static readonly string ID = "identifier";
            internal static readonly string ORIGIN = "originNode";
            internal static readonly string DESTINATION =
"destinationNode";
            internal static readonly string WEIGHT = "weight";

            public static void Cleanup(SqlConnection connection)
            {
```

```
                DataUtilities.CleanupTable(connection, TABLE_NAME);
            }

            /// <summary>
            /// Creates new SUPLink data.
            /// </summary>
            /// <param name="origin"></param>
            /// <param name="destination"></param>
            internal static SUPLink CreateLink(SUPNode origin, SUPNode
destination)
            {
                Debug.Assert(origin != null && destination != null);
                // Should not exist already
                Guid _id = Guid.NewGuid();
                DataRow row = Data.NewRow();
                row[ID] = _id;
                row[ORIGIN] = origin.Identifier;
                row[DESTINATION] = destination.Identifier;
                row[WEIGHT] = 1;
                lock(Data) {Data.Rows.Add(row);}
                return new SUPLink(_id);
            }

            internal static DataTable Data
            {
                get
                {
                    return table;
                }
            }

            public static void Initialize(DataSet dataSet, SqlConnection
connection)
            {
                adapter = DataUtilities.InitializeTable(dataSet,
connection, TABLE_NAME);
                table = dataSet.Tables[TABLE_NAME];
            }

            public static void Update()
            {
                adapter.Update(Data);
            }


            //************ NON-STATIC PART *************

            private Guid _identifier;

            internal SUPLink(DataRow row) : this((Guid)row[ID]) {}
            internal SUPLink(Guid _id)
            {
                _identifier = _id;
            }

            public INode Destination
            {
                get
                {
                    return new SUPNode((Guid)Row[DESTINATION]);
                }
```

```
		}

		private DataRow Row
		{
			get
			{
				return Data.Rows.Find(_identifier);
			}
		}

		public override bool Equals(object obj)
		{
			if (!(obj is SUPLink)) return false;
			SUPLink peer = (SUPLink) obj;
			return this._identifier.Equals(peer._identifier);
		}

		public override int GetHashCode()
		{
			return _identifier.GetHashCode();
		}
	}
}
```

References

1. Carnevale, D., *Some online educators turn to bite-sized instruction*, in *Chronicle of Higher Education*. 2001. http://chronicle.com/free/2001/05/2001050301u.htm

2. Downes, S., Learning objects. http://www.atl.ualberta.ca/downes/naweb/Learning_Objects.doc

3. Wiley, D., *Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy*, in *The Instructional Use of Learning Objects*, D. Wiley, Editor. 2000, Association for Instructional Technology, Association for Educational Communications and Technology.

4. Apple Learning Interchange, Exhibit. http://ali.apple.com/ali/resources.shtml

5. California State University Center for Distributed Learning, Merlot - Multimedia Education Resource for Learning and Online Teaching. http://www.merlot.org/Home.po

6. IEEE - LTSC, Draft Standard for Learning Object Metadata v. 6.4. http://ltsc.ieee.org/wg12/

7. Australian Capital Territory, Le@rning Federation. http://www.decs.act.gov.au/schools/lfindex.htm

8. Canada's SchoolNet. http://www.schoolnet.ca/pagemasters/e

9. MERLOT. http://www.merlot.org/Home.po

10. O. Zamir and O. Etzioni, *Grouper: A Dynamic Clustering Interface to Web Search Results*, A. Mendelzon, Editor. 1999, Elsevier Science: Toronto, Canada.

11. CSSE, The LEOPARD project. http://www.csse.monash.edu.au/projects/LEOPARD

12. P. Resnick and H.R. Varian, *Recommender systems*, in *Communications of the ACM*. 1997. p. 56-58.

13. D. Goldberg, et al., *Using collaborative filtering to weave an information tapestry*, in *Communications of the ACM*. 1992. p. 61-70.

14. S.E. Middleton, et al. *Exploiting Synergy Between Ontologies and Recommender Systems*. in *Semantic Web Workshop 2002*. 2002. Hawaii, USA.

15. R. Rafter, B. Smyth, and K. Bradley. *Inferring Relevance Feedback from Server Logs: A Case Study in Online Recruitment*. in *11th Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2000)*. 2000. Galway, Ireland.

16. M. Claypool, et al., *Inferring User Interest*, in *IEEE Internet Computing*. 2001. p. 32-39.

17. Nichols, D.M. *Implicit Rating and Filtering*. in *5th DELOS Workshop on Filtering and Collaborative Filtering*. 1997. Budapest, Hungary.

18. N. Good, et al. *Combining collaborative filtering with personal agents for better recommendations*. in *Sixteenth National Conference on Artificial Intelligence*. 1999.

19. M. Balabanovic and Y. Shoham, *Fab: Content-Based, Collaborative Recommendation*, in *Communications of the ACM*. 1997.

20. S. El-Beltagy, D. DeRoure, and W. Hall. *The Evolution of a Practical Agent-based Recommender System*. in *Workshop on Agent-based Recommender Systems, Autonomous Agents 2000*. 2000.

21. R. Rafter, B. Smyth, and K. Bradley. *Case-Based User Profiling for Content Personalisation*. in *International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH2000)*. 2000. Trento, Italy.

22. S. Franklin and A. Graesser, *Is it an agent, or just a program?*, in *Intelligent Agents III (Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages 1996)*. 1996, Springer: Budapest, Hungary.

23. N.R. Jennings and M. Wooldridge. *Agent-Oriented Software Engineering*. in *9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99)*. 2000. Valencia, Spain: Springer.

24. Shoham, Y., *Agent Oriented Programming*. Journal of Artificial Intelligence, 1993. **60**: p. 51-92.

25. Bradshaw, J.M., *An Introduction to Software Agents*, in *Software Agents*, J.M. Bradshaw, Editor. 1997, The AAAI Press.

26. M. Wooldridge and P. Ciancarini, *Agent-Oriented Software Engineering: The State of the Art*, in *First International Workshop on Agent-Oriented Software Engineering (AOSE'2000)*. 2000, Springer: Limerick, Ireland.

27. Odell, J., *Objects and Agents Compared*. Journal of Object Technology, 2002. **1**(1): p. 41-53.

28. A.S. Rao and M.P. Georgeff. *BDI Agents: From Theory to Practice*. in *First International Conference on Multiagent Systems (ICMAS'95)*. 1995. San Francisco.

29. M.N. Huhns and L.M. Stephens, *Multiagent Systems and Societies of Agents*, in *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, Editor. 1999, The MIT Press. p. 79-120.

30. Z. Guessoum and J.-P. Briot, *From Active Objects to Autonomous Agents*, in *IEEE Concurrency*. 1999. p. 68-76.

31. G. Armano and E. Vargiu, *Implementing Autonomous Reactive Agents by Using Active Objects*, in *WOA 2000 -- Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, A. Corradi, A. Omicini, and A. Poggi, Editors. 2000, Pitagora Editrice Bologna. p. 35-40.

32. Lesser, V.R., *Multiagent Systems: An Emerging Subdiscipline of AI*. ACM Computing Surveys, 1995. **27**(3).

33. Sycara, K.P., *Multiagent Systems*, in *AI Magazine*. 1998.

34. N.R. Jennings and M. Wooldridge, *Applications of Intelligent Agents*, in *Agent Technology: Foundations, Applications, and Markets*, N.R. Jennings and M. Wooldridge, Editors. 1998, Springer Verlag.

35. S. El-Beltagy, D. De Route, and W. Hall. *A Multiagent system for Navigation Assistance and Information Finding*. in *Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'99)*. 1999. London, UK.

36. M.R. Genesereth and S.P. Ketchpel, *Software agents*, in *Communications of the ACM*. 1994.

37. T. Finin, J. Werber, and e. al., Draft Specification of the KQML Agent Communication Language. http://www.cs.umbc.edu/kqml/kqmlspec/spec.html

38. T. Finin, Y. Labrou, and J. Mayfield, *KQML as an Agent Communication Language*, in *Software Agents*, J.M. Bradshaw, Editor. 1997, The AAAI Press. p. 291-316.

39. FIPA, The FIPA web pages. http://www.fipa.org/about/index.html

40. M.R. Genesereth and R.E. Fikes, Knowledge Interchange Format Version 3.0 Reference Manual. http://logic.stanford.edu/kif/Hypertext/kif-manual.html

41. W3C, The Semantic Web project. http://www.w3.org/2001/sw/

42. T. Khedro and M.R. Genesereth, *Facilitators: A Networked Computing Infrastructure for Distributed Software Interoperation*, in *The 1995 International Joint Conference on AI: Workshop on AI in Distributed Information Networks*. 1995: Montreal, Canada.

43. Y. Peng, et al., *An Agent-Based Approach for Manufacturing Integration - The CIIMPLEX Experience*. International Journal of Applied Artificial Intelligence, 1999. **13**(1-2).

44. Maes, P., *Agents that Reduce Work and Information Overload*, in *Communications of the ACM*. 1994. p. 30-40.

45. H.S. Nwana and D.T. Nduma, *A Perspective on Software Agent Research*. Applied Artificial Intelligence, 1999. **13**(Special issue).

46. H. Lieberman, C. Fry, and L. Weitzman, *Exploring the Web with Reconnaissance Agents*, in *Communications of the ACM*. 2001. p. 69-75.

47. Oliveira, E., *Applications of Intelligent Agent-Based Systems*, in *Proceedings of SBAI - Simpósium Brasileiro de Automação Inteligente*. 1999: São Paulo, Brazil. p. 51-58.

48. Isaias, P. *An Agent Architecture for a Virtual Research Digital Library*. in *TERENA (Trans-European Research and Education Networking Association) Networking Conference 2000: Pioneering Tomottow's Internet*. 2000. Lisbon, Portugal.

49. D. Derbyshire, et al., *Agent-Based Digital Libraries: Driving the Information Economy,* in *Proceedings of the Sixth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. 1997, IEEE: Cambridge, MA, USA. p. 82-86.

50. R.H. Guttman, A.G. Moukas, and P. Maes, *Agent-mediated Electronic Commerce: A Survey*. Knowledge Engineering Review, 1998. **13**(2): p. 147-159.

51. A.R. Lomuscio, M. Wooldridge, and N.R. Jennings, *A Classification Scheme for Negotiation in Electronic Commerce*, in *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, F. Dignum and C. Sierra, Editors. 2000, Springer. p. 19-33.

52. MIT Media Lab, Past projects. http://agents.media.mit.edu/projects/past.html

53. W. Shen and D.H. Norrie, *Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey*. Knowledge and Information Systems (KAIS), 1999. **1**(2): p. 129-156.

54. Carnegie Mellon University - Software Agents Group and Robotics Institute and U.o.P.-D.o.I.S. Telecommunications, MokSAF: Software Environments for Route Planning and Team Coordination. http://www-2.cs.cmu.edu/~softagents/moksaf/index.html

55. Agent Oriented Software Pty. Ltd., Jack Intelligent Agents. http://www.agent-software.com.au/shared/home/index.html

56. A. Zunino and A. Amandi. *Brainstorm/J: a framework for intelligent agents.* in *Second Argentinian Symposium on Artificial Intelligence (ASAI 2000)*. 2000. Buenos Aires, Argentina.

57. Telecom Italia Lab, JADE web pages. http://sharon.cselt.it/projects/jade/

58. LEAP, Lightweight Extensible Agent Platform. http://leap.crm-paris.com/

59. H.S. Nwana, D.T. Ndumu, and L.C. Lee, *ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems,* in *Proceedings of the Third International Conference and Exhibition on the Practical*

Application of Intelligent Agents and Multi-Agent Technology (PAAM 98). 1998: London, U.K. p. 377-391.

60. Communication Technologies, Comtec Agent Platform. http://ias.comtec.co.jp/ap/

61. Fujitsu Labs, April Agent Platform. http://www.nar.fujitsulabs.com/aap/about.html

62. A. Amandi and A. Price. *Towards Object-Oriented Agent Programming: The Brainstorm Meta-Level Architectu.* in *First International Conference on Autonomous Agents.* 1997. Marina del Rey, California, USA.

63. L. Dempsey and R. Heery, *Metadata: a current view of practice and issues.* Journal of Documentation, 1998. **54**(2): p. 145-172.

64. Gilliland-Swetland, A.J., *Setting the stage*, in *Introduction to metadata: pathways to digital information v.2.0*. 2000, Getty Information Institute.

65. Johnston, P., *XML and "meta-tagging"*, in *presentation in Technical seminar for Pathfinder LEAs, BECTa*. 2002: Coventry, UK. http://www.ukoln.ac.uk/interop-focus/presentations/bectapf/tsld001.htm

66. Johnston, P., *Metadata : an overview*, in *presentation in XML and Educational Metadata Workshop*. 2001: London, UK. http://www.ukoln.ac.uk/interop-focus/presentations/sbu/ppt/overview.ppt

67. Johnston, P., *Metadata sharing and XML*. 2002, NOF Technical Advisory Service. http://www.ukoln.ac.uk/nof/support/help/papers/metaxml.htm

68. Johnston, P., *An Introduction to Metadata*, in *Presentation to the "Metadata : from soup to nuts" seminar for NOF-digitise projects*. 2002: London, UK. http://www.ukoln.ac.uk/nof/support/workshops/metadata-2002/presentation2/presentation2.ppt

69. W3C, XML. http://www.w3.org/XML/

70. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax/

71. MARC. http://www.loc.gov/marc/

72. Encoded Archival Description. http://www.loc.gov/ead/

73. International Standard Archival Description from the International Council of Archives. http://www.ica.org/

74. IEEE LTSC Learning Object Metadata Working Group. http://ltsc.ieee.org/wg12/index.html

75. MPEG-7. http://ipsi.fhg.de/delite/Projects/MPEG7/

76. The Harmony project. http://metadata.net/harmony/

77. Rosch, E., *Principles of categorization*, in *Cognition and categorization*, E. Rosch and B.B. Lloyd, Editors. 1978, Erlbaum: Hillsdale, USA. p. 27-48.

78. Larkoff, G., *Women, Fire and Dangerous Things: What Categories Reveal about the Mind*. 1990, Chicago: University of Chicago Press.

79. Kouznetsov, P., Jad - the fast JAva Decompiler. http://kpdus.tripod.com/jad.html

80. Linar, J-Integra. http://www.linar.com/

81. National Association of College Stores. http://www.nacs.org

82. Interactive Educational Systems Design Inc. (IESD), *Online Courses and Other Types of Online Learning for High School Students*. 2002. http://www.apexlearning.com/results/results_schools_dist.asp

83. Council for Higher Education Accreditation, Distance Learning in Higher Education, Update Number 3 (June 1999). http://www.chea.org/Commentary/distance-learning.html

84. The Building the Internet Workforce Project. http://www.itee.uq.edu.au/~seminar/archive/sem-0382.html

85. Sun website. www.sun.com

86. Telstra website. www.telstra.com

87. Compuware website. www.compuware.com

88. DSTC website. www.dstc.com

89. The Learning Resource Exchange project (LRX). http://www.admin.utas.edu.au/academic/acservices/meetings/talc/Appendix/2_01C3.doc

90. Commonwealth Department of Education Science & Training (DEST). http://www.dest.gov.au

91. The Peer Review of ICT Resources project. http://www.detya.gov.au/highered/eippubs/eip01_3/default.htm