How Many Small Libraries Can Be a Large Library

Michael Christoffel, Guido Wojke, Max Gensthaler

Institute for Program Structures and Data Organization
University of Karlsuhe, Germany
{christof, wojke, gensth}@ipd.uka.de

Abstract

The success of the Internet has caused a significant change in the way of literature supply. Not only traditional libraries, booksellers and publishing houses have begun to enter the new digital world and offer their services world-wide, completely new information providers have been developed such as digital libraries, delivery services, citation services, and bibliographic databases. The amount of knowledge and information available world-wide grows every day. However, very few people will be able to access all the information that is available, since the information is distributed among a large number of computers and databases. People need assistance to gain the whole utility from the new situation. The idea presented in this paper is to integrate library and information services worldwide, so that they build a large virtual library with different services, offering the user a uniform access to the system.

1 Introduction

In our modern society, information has become one of the most important and valuable goods. Many professions depend on the steady supply with actual information. At the same time, the success of the Internet has caused a significant change in the way of literature supply. Now computers all over the world can be linked together, and information can be distributed and dealt over the Internet.

Many traditional libraries, booksellers and publishing houses took the change to "go online" and now offer their services world-wide over the Internet. But also completely new kinds of information providers have been developed in the recent years, such as digital libraries, delivery services, citation services, and bibliographic databases. The amount of knowledge and information available world-wide grows every day.

However, very few people will be able to access all the information that is available all over the world, because this information is distributed among a large number of computers and databases. The sheer number of different libraries and information services is that large that it is nearly not possible to survey. In general, a person looking for information will not be able to find the most appropriate information sources for his/her demand.

Even if a person knows some information sources by chance, there is no easy way to estimate and compare them. They may differ in content, quality of services, accessibility, and, not to forget, prices.

And also concerning a single information source, it may be difficult to use this source for the advantage of the user. Not only the services itself, also the user interfaces differ, requiring a learning process for the user, before he/she can use the source completely. Different conditions, media formats, and language barriers have an additional effect.

Moreover, in a typical scenario a person looking for some information needs to access several different information sources sequentially. This way, information search will at least use a large amount of time. Together with the commercialization of the Internet, information search may also use a large amount of money.

People need assistance to gain the whole utility from the new situation. However, every approach dealing with this problem must face the distribution and heterogeneity of the existing libraries and information services. Each plan carrying together the knowledge of the world will fail because of technical, economical, legal, and political reasons. Also the creation of new information services is not an alternative, as long as these new services can not be used together with the existing services.

Search and meta-search engines provide a way to find services and send queries to several information sources in parallel. However, they do not give assistance in the evaluation and combination of services and in the interpretation of results. Most activities in the process of literature search and delivery still have to be done by the user manually.

The idea presented in this paper is to integrate different services for literature services worldwide, so that they build a large virtual library with a manifold of different, value-added services. This system will provide a uniform interface to the user, which can be adapted to his/her needs. Literature search world-wide will be not more difficult than entering the local library.

We continue as follows: In section 2, we want to introduce some approaches related to our own idea of an integration environment. In section 3, we will point out the challenges of our approach. In section 4, we will introduce the realization of our system and show how we have solved the challenges. In section 5, we will describe the architecture of our system. In section 6, we will describe the main ideas for the implementation of the integration system and our previous work. We conclude the paper in section 7.

2 Related Work

There are a number of projects concerning the problem of library integration, coming to different solutions than we do. In this section, we want to introduce some of these works.

The Stanford Digital Library Project aims in the integration of autonomous distributed collections with a central architecture. Core of the architecture is the InfoBus where all collections are linked together and which is implemented using CORBA. Search is based on complete metadata catalogues and full text glossaries of all participating collections [1]. Communication is based on the SDLIP protocol [11]. A large set of tools have been developed for this architecture.

The University of Michigan Digital Library aims in the integration of collections by an infrastructure of software agents [8]. The agent infrastructure has been realized using CORBA. For communication, a set of protocols have been developed which are oriented on KQML. The main paradigm for the agent interactions are negotiations [9]. In addition to task-specific and independent agents such as user interface agents, task planning agents, mediator agents, and collection interface agents, there are also central and unique architecture elements.

The focus of the Daffodil project is the development of high-level search possibilities on distributed, heterogeneous collections and information services [10]. The Daffodil system consists of a (not distributed) set of software agents. Inter-agent communication is based on KQML. Additional to the user interface and wrappers which form the interface to the collections, there are three types of agents: tactics which perform simple searches such as metadata and full text search, stratagems for complex searches such as author search, and strategies which assist in the choice of the appropriate stratagems.

The aim of the MeDoc project was the creation of a distributed electronic library in the field of computer science [2]. The project underlies a layered architecture which consists of user interfaces, brokers, and provider interfaces. The communications betweens the layers is done by an extension of HTTP. All documents are supposed to be transferred to special document servers. MeDoc supports electronic commerce features, so the use of the system and the access to the documents can be charged.

3 Challenges

Our aim is to build a system capable for the integration of autonomous and heterogeneous libraries and information sources world-wide. This aim is accompanied with a number of challenges:

Challenge 1
   The system must be scalable and extensible.

Challenge 2
   Extensions of the systems must be possible for any organization. In the long run, it will not be possible that the system is built and maintained by one organization only.

Challenge 3
   The system must give the user a uniform access that is adapted to his/her needs. The system has to support users coming from all parts of the world.

Challenge 4
   The independence of the information sources must be preserved. The integration system must be adapted to the existing information sources, not vice versa.

Challenge 5
   The system must be able operate in a distributed environment, crossing borders between operating systems and hardware platforms.

Challenge 6
   The system must be robust. A dropout of one part of the system must not stun the rest of the system.

Challenge 7
   The system must be prepared for electronic commerce transactions.

Challenge 8
   The system must defend itself against any misuse.

In the following section, we will describe how we have solved these challengers. Due to the limited length of this paper, we cannot describe our solution in detail. In this paper, we will describe our main ideas and give a sketch on the architecture and the implementation.

4 Realization

In this section, we give an overview on the realization of our system and the way we have solved the challenges.

The main paradigm of our work is the idea of an open environment. For the participants – the information providers and the potential users of the system – the open environment appears as a black box: There are adapted interfaces that allow interaction with the environment, but they do not need to have insight in the internals of the environment.

The environment is inhabited by a society of independent but communicative software agents. More

concretely, the environment consists of these agents only; there is no underlying middleware and no elements other than the agents.

Each agent speaks and understands a certain set of messages. For simplification, each agent is assigned to an agent type, which describes the behavior of the agent. If an agent knows the type of its communication partner, it is able to communicate in the correct way. The knowledge about the concrete implementation of an agent is not necessary.

There is no rule that prescribes which agents are inside the environment at one point of time. In fact, agents are free to enter and leave the environment on their own decision. As a consequence of this, it is not possible to define the relations among the agents in advance. An agent that enters the environment does not know which other agents it will find. Hence, it must learn about the environment and find communication partners in order to fulfill its task. The behavior of the agents in the environment can be described under the metaphor of an open market [4]. Information providers and users act as providers and customers in this market.

It is obvious that the open environment is scalable and extensible in a high grade. The learning behavior of the agents allows them to adapt to changes in the environment. Moreover, it is possible to add new agents at runtime. It is even possible to create completely new agent types after the system has been launched; the functionality of the existing agents will not be touched. Agents designed by different organizations will work together. This way, the open environment meets challenge 1 and 2. Instead of adding new agents, it is possible to upgrade existing agents. Since the knowledge of the agent can be preserved during the upgrade, the agent does not have to repeat the learning process of its processor.

There are special agents that act as interfaces to customers and providers. These interfaces can be adapted to the needs of the market participants. For each customer (the user of the virtual library), the information system provides the view of a personal workplace. Not only user profiles and presets can be stored for each customer, but also documents, references, and annotations. This clearly meets challenge 3. For the provider side interface, we can even go one step further. It is possible to install a separate agent for each provider. This agent can be tailored for this provider (challenge 4).

The communication among the agents in the environment is based on Web services. This way, we do not need any additional middleware for communication, and we can cross the borders of operating systems and hardware platforms, and, equally important, firewalls. This meets challenge 5.

Due to the open system, it is possible (and likely) to have several agents of the same type which can replace each other. This creates a high level of redundancy and robustness and meets challenge 6. It is also possible to create identical copies of the same agents that share the same databases. This way, there is a good change that there is a perfect replacement if an agent failed

temporarily or permanently. This strategy does not only increase the robustness of the system, it also can be used for load balancing, because queries can be distributed among identical agents according to the actual load.

An important issue of the integration platform is security. An agent may decide to send a message encrypted using SSL. This way it is possible to transmit private data such as credit card numbers or passwords. In addition to encryption, the system also uses digital signatures in order to prove the identity of a customer and uncover manipulations. The same procedure can be used to certificate the identity and the quality of the agents itself. Every agent can decide to decline cooperation with other agents or customers it does not trust. This way, our system is able to meet challenges 7 and 8.

5   Architecture

In this section, we will discuss the architecture of the integration environment we developed and which bases on the ideas described above. We will first discuss a simplified approach which shows the most important agent types. Then we will discuss these agent types in a little bit more details Afterwards we will present the extensions necessary for an environment capable for electronic commerce transactions.

5.1   Overview

For the integration system, we use an instance of an open agent environment which we refer to as the UniCats[1] environment [7].

Figure 1 contains a simplified view on the UniCats environment, showing the most important agent types. Customer Interface Agents (CIA) provide the customer-side interfaces of the system. They have connections to customer agents (CA) which act as the representatives of the customers in the system and provide for each customer a personal workplace. In order to perform queries in behalf of the customers, they can ask a Provider Selection Agents (PSA) for recommendations about those providers suitable as sources for the performance of the queries. The Customer Agents sends a query together with a list of the selected providers to an Integration Agent (IA). The Integration Agent sends the query in parallel to the Provider Agents (PA) that act as the representatives of the providers. The Provider Agents translate the incoming query into the native protocol of the provider and re-translates the delivered results into the common protocol. The Integration Agent collects the incoming results from the different information sources and integrates them to one result list. The final result list is sent back to the Customer Agent, which can present the results to the customer with the help of the Customer Interface Agent.

It is important to consider that this is only an example of possible interactions. A more complex

---

[1] a Universal Integration of Catalogues based on an Agent-supported Trading and Wrapping System
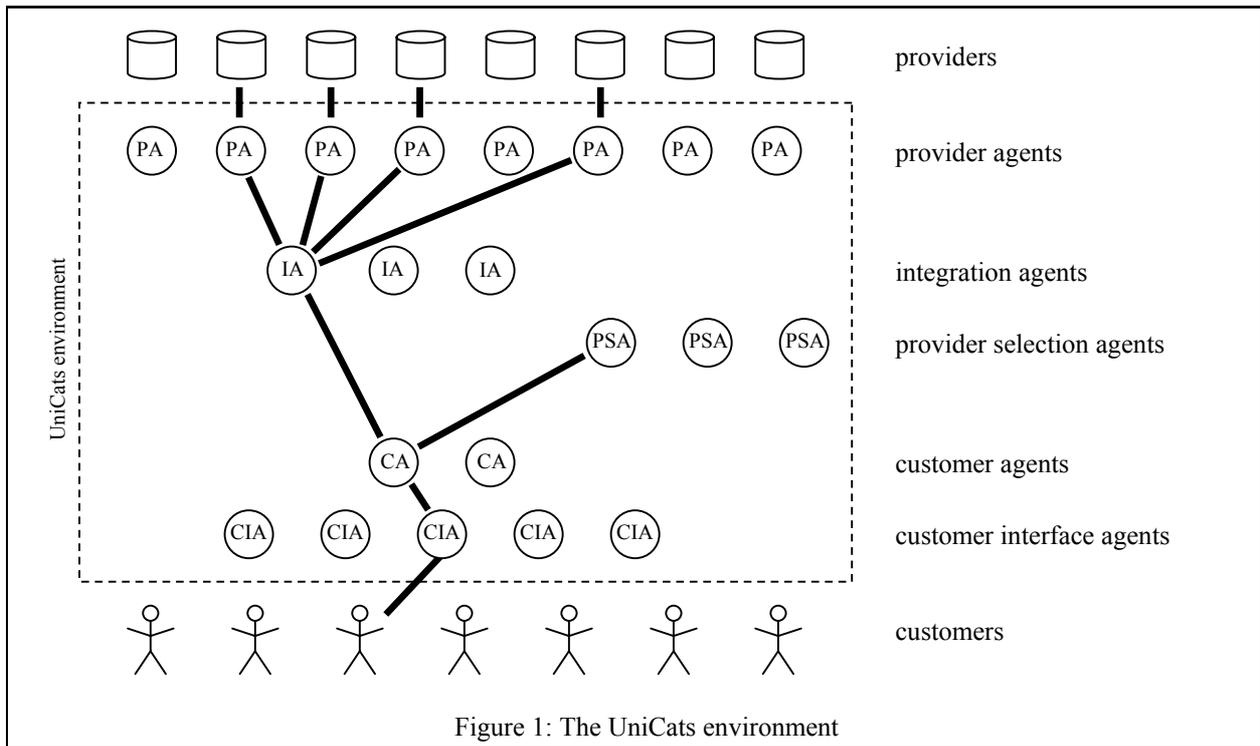
Figure 1: The UniCats environment

scenario may contain many customers who operate with the system at the same time, the combination of several queries (including order and delivery) and involve more agents of different agent types. In fact, the agent is not able make any concrete plans for task execution in advance, because it is not known which agents will be present in the environment at a given point of time and which communication partners the agents will have.

We now want to take a closer look on the five agent types shown in Figure 1.

5.2 Customer Interface Agents

Customer Interface Agents (CIA) are the interfaces to the customers. They are the customers' only connection to the integration system, hiding the internal structure of the UniCats environment. Following the Model View Controller Paradigm, Customer Interface Agents have to designed independent from the further layers of data processing; they are restricted in user interaction and data presentation. They have to make use of the standardized interfaces to the other agents.

There may be different customer interface agents, and they may differ in the look and feel they provide to the customers. This way, the customers can choose their favorite interfaces. Most likely, customers will choose different customer interface agents in different situation. E.g., they may use a powerful Customer Interface Agent which is fully adapted to their needs at their workplace, but choose a more lightweight one when they are on business travel and have to access the system with a mobile device.

Customer Interface Agents can be used to integrate the existing library systems of the tradition libraries with the UniCats environment, so that the integration platform (and the connected sources) can be used from the library system. Doing this, it is only necessary to implement a new Customer Interface Agent that is connected with the library system.

Until now, we have developed customer interfaces using different technologies such as graphical Java application, HTML, WAP, and virtual reality [5].

In order to adapt the interface to the flavor of the customer, Customer Interface Agents store the setup of the customers and the data needed to determine the individual look and feel. These data encompass colors and other details for the design of the customer interface and also the favorite natural language of the customer. As soon as the Customer Interface Agent knows the identity of a customer who logged in, the agent can restore the look and feel of this customer.

5.3 Customer Agents

Customer Agents (CA) are the representatives of the customers in the system. Customers can use any Customer Interface Agent to access their favorite Customer Agent. Even if the change the interface, they still can work with the same Customer Agent. But it is also possible to change the Customer Agent and still use the same Customer Interface Agent.

Customer interface agents provide a personal workplace for the customers. In their personal workplace, customers can hold queries, result lists, bibliographic data, and also (electronic) documents itself. It is also possible to make annotations and share data with other customers in the system. When customers leave the system and come back later, they will find the personal workplace in the same state as they have left and can immediately continue working.

Customer Agents assist the customer in the formulization of its demands. Customer Agents hold customer profiles with interests and preferences of the customers gained from questionnaires and observations of the customer behavior, and can use these date to enrich or auto-formulate queries. Choosing the most appropriate sources, they can make complex query plans, where results from different kinds of queries can be combined or mixed together. As soon as results arrive, the Customer Agent presents them to the customer so that the customer can work with the results, while the Customer Agent finds more results or suggest additional services.

Customer Agents have to assist the customer in different states of the process of information supply, beginning with overview searches to get an overview on the area and reaching to order (and delivery) of complete documents.

5.4   Provider Selection Agents

Since the dynamism of the environment, Customer Agents will not be able to keep a permanent overview on the information services and sources which are available in the system Provider Selection Agents (PSA) assist Customer Agents finding the most appropriate services and sources. They supply Customer Agents with ranked lists of recommendations of the most appropriate providers for a customer's demand, together with additional information about the providers [3].

In order to able to perform this task, they have to permanently collect information about the providers which are currently available in the system. This is not a trivial task, since providers are free to appear and vanish without further notice.

Provider Selection Agents may use different ways to gain the needed market overview. The easy way is that the providers themselves support the Provider Selection Agent with the information. It is not irrational to assume a friendly behavior of the providers, since the providers have a self-interest in being recommended. However, Provider Selection Agents should not rely on the friendly behavior of the providers and not trust the received information completely. At least they have to check whether the provider profiles are complete, consistent, and up-to-date.

Alternative ways to gain information about the providers are test-query and the analysis of the feedback of the Customer Agent.

5.5   Integration Agents

While it is possible that the Customer Agent will address some services or sources in a sequence (e.g., if the output of the first provider is used as input for the second), in most cases a set of providers can be addresses in parallel. Integration Agents (IA) assist Customer Agents in parallel queries. They forward each query to several providers (or provider agents, respectively) in parallel, collect the incoming results and integrate them to one result list [12]. Result

integration includes the detection and elimination and the combination and completion of attributes.

It is not the best strategy for the Integration Agent to wait until all providers give a response, because this could take some time, and the customer had to wait long for an answer. A better strategy is incremental result integration. Incremental Result Integration means that results are sent back to the customer agent as soon as they arrive. When new results arrive, the Integration Agent merges these with the result list already sent to the Customer Agent. The customer receives the answer to a query as soon as possible and can work with the results. With the time, new results and attributes will be added, until all sources have answered. If there is a failure (e.g., one provider does not answer at all), the customer will not notice this, as long as there are alternative sources. If alterative sources are not present, the Customer Agent could ask the Provider Selection Agent.

5.6   Provider Agents

Provider agents are the representatives of the providers to the system. They are responsible for the syntactical and semantical homogenization of the different kinds of providers. In general, they translate incoming queries into the native protocol of the provider, and re-translate the answer. In many cases, however, a purely syntactical translation of the query is not enough. Then the Provider Agent must take part in the execution of a query and apply operations such as result filtering. E.g., while it is possible in many library catalogues to receive the publication year of a document, it is not possible to query a range of year, but this can be done quite easy be a filter operation. This way, Provider Agents can enlarge the capacity of the query language of the providers. Another task is the planning and the optimization of the query execution. Since Provider Agents are quite closely coupled to 'their' providers, they should be tailored to individual providers so that there is at least one Provider Agent for each provider. This increases the load balance and the performance of the system.

It is important that the independence of the providers is guaranteed. Therefore, we can neither demand a direct access to the provider's database nor any requirements for the provider's computer system, but use an existing interface. While it would be possible to use any other standardized or native interface, we concentrate our work on the Web interface of the provider. This way, we use the information provider the same way a human user would do. Since not only each provider has an individual Web interface, but also this web interface tends to be modified from time to time, we have developed a tool for the generation and modification of the converter, which establishes the connection to the Web interface of the provider [6].

5.7   Extensions

Figure 1 shows a simplified view on the integration platform. Extensions are necessary in order to maintain

a level of security in the system which is mandatory for the practical use of the system, especially under the consideration of an e-commerce scenario with commercial sources.

A first step is to use encryption for the data transfer among the agents, or – what we did – to offer different security levels and leave the agents the choice which security level to use for a transmission However, since the system is completely open for providers, customers, and agents, there is a problem of trust.

We have declared that all providers are free to enter and leave the system and offer their services to the public. There is a danger that this could lead to an anarchical situation and open the door to misuse. However, we have implemented two mechanisms to prevent a situation without control. First, for each provider must be at least one Provider Agent. So the problem of trust against the provider is reduced to the problem of trust against an agent. A Provider Agent should never allow a provider to harm the system or a customer. E.g., it would stop any query processing, when cost arise higher than a given limit. Second, the Provider Selection Agent would never recommend a provider that has been shown as 'bad' in some sense.

For customers, the situation is not that easy, because they can use any Customer Interface Agent to access the system. So there must be a login mechanism in order to prove their identity and authorization. This login procedure can only be done at the Customer Interface Agent, but the authorization should be done in a separate agent type. Customer Authorization Agents (CAA) store the accounts and passwords of the customers together with information about the privileges and rights of the customer. It is also possible to hold digital signature of the registered customers. There is also an anonymous guest account with limited privileges. During login process, Customer Interface Agents contact the Customer Authorization Agent in order to validate the identity of the customer; but it is also possible that other agents want to confirm the identity of a customer. E.g., a Provider Agent might double-check the identity of a customer before it performs any transactions liable to charges.

While privileges and rights can be assigned to individual customers, in real life they are often assigned to customer organizations. A customer organization may be a university or a company which offers some special conditions ton their member, e.g., the free access to a commercial database. Hence, we need an extension of our system in order to consider customer organization. Customer Organization Agents (COA) are the representatives of the customer organization. Customer organization agent can interact in the system in behalf of their members. Whenever a Customer Agent realizes that a customer who is member of a customer organization could use the advantages of the membership, it forwards the query to the Customer Organization Agent. The Customer Organization Agent proves the rights of the member, then performs the query in the name of the organization.

Billing and payment is another situation where the authorization of the customer is necessary. Billing Agents (BA) act as mediators in commercial situations and play a role as trusted third party in payment procedures, while Payment Agents (PMA) are the representatives of banks and financial institutions.

We still have the problem how to provide trust against the agents themselves. Since the UniCats environment is open and extensible, everyone is free to add an agent capable to interact with other agents. Again, there is a danger of misuse. One step is to provide agent authorization. Therefore, agents and groups of agents can register themselves at Agent Name Agents (ANA) and Group Name Agents (GNA). This way, an agent can contact the corresponding registry in order to prove the identity of an agent. It is also possible to use digital certificates. Of course, the idea of the registries only works as long as an agent trusts the agents that hold the registries.

Proving the identity of an agent is not enough. It is also important to know the quality of an agent. This is a difficult task, because there is no objective measure for the quality of an agent. Agent Certificate Agents (ACA) give certificates to agents for a limited period of time. Certificated agents have a self-interest to keep a standard of quality in order not to loose the certificate. Agent Rating Agents (ARA) follow another idea. They provide a measure of the quality of an agent which is generated by the ratings of other agents, usually business partners or previous business partners of the rated agent. A third effect to hold a level of quality and trust in the environment comes from the market itself. Agents will avoid any further contact to agents they are disappointed. This way, an agent must show a good quality in order to be successful.

The UniCats environment is designed to work without having any central instance. However, for reasons of testing and error handling (and also scientific reasons), it makes sense to artificially introduce a central instance in the form of System Administration Agents (SAA). System Administration Agents can survey the environment or at least a part of the environment. Therefore, the agents sent reports about their work to the System Administration Agent. It is clear that an agent will only report to a System Administration Agent it trusts completely.

6 Implementation

In this section we describe our own implementation of the architecture described in section 5.

6.1 Overview

We implemented the system using Java programming language. This brings the advantage that our agents can run platform-independent on most computers. There is also a large set of free tools and packages available. However, the development of agents does not depend on the chosen programming language. It is also possible to implement agents using other platforms and languages, and it is also possible to use a mixed
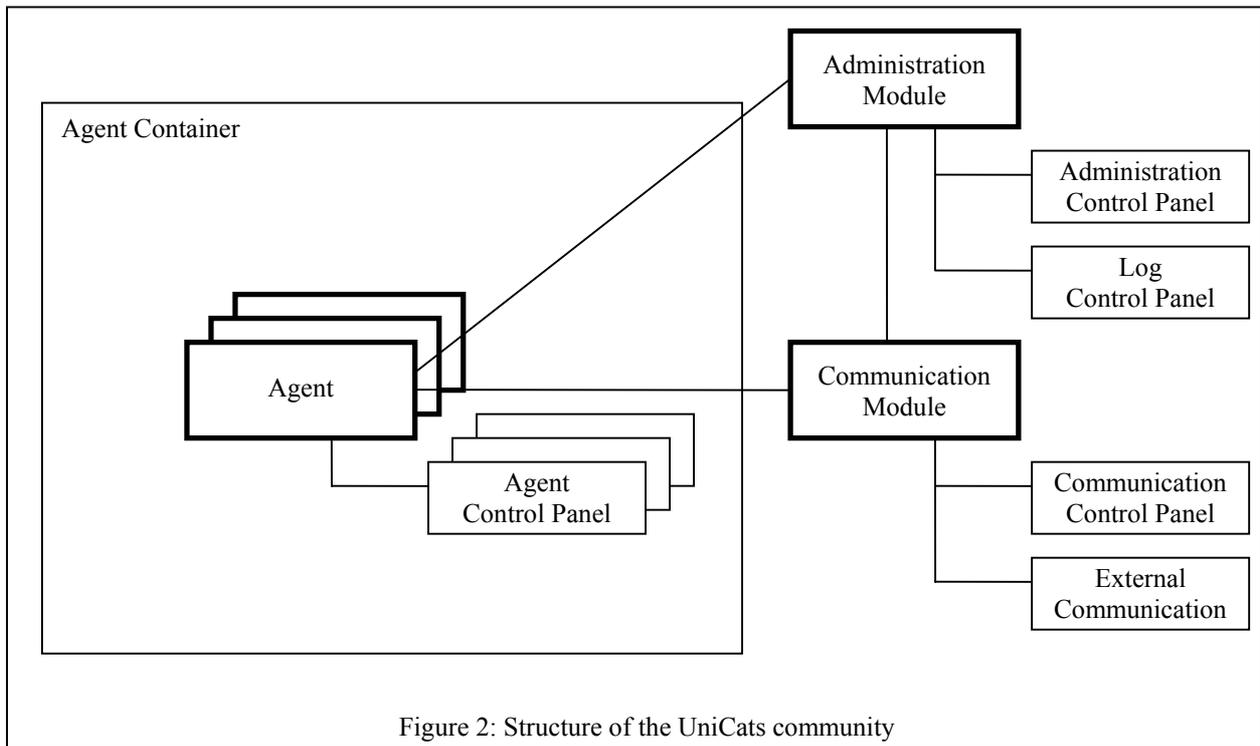
Figure 2: Structure of the UniCats community

environment. In fact, we already have implemented some sample agents using other platforms and programming languages, and these sample agents have been able to interact with our Java-agents without problems.

The communication among the agents is based on Web services. This guarantees a very high grade of independence of computer platforms and programming language. As a side effect, we can bypass the firewall problematic, where computer systems are being shut more and more from the outside for security reasons. Using Web services, we can use the standard TCP/IP ports that are used for HTTP, or we can tunnel our communication in secure channels using SSH. The disadvantage of this method in contrast to direct implementation in TCP/IP (which we also tried, for testing reasons) is the need of a Web server where the agents run.

In order to minimize this disadvantage, we have developed the concept of agent communities. Agent communities allow several agents to be executed in one virtual machine, sharing the same Web server and other resources. Of course, it is also possible to have more than one community on one computer.

An additional advantage of the concept of agent communities is that the communication among the agents in the same community is very fast, so it makes sense to put those agents with strong interrelations together in one community. However, the concept of the community is more a technical detail introduced to increase efficiency. The logical interrelations among the agents do not depend from their location. In fact, agents can be located anywhere in the Internet, and it is even possible for them to migrate from one community to another. However, it is possible that the location of an agent is restricted because of another reason, e.g., if the agent uses a database which is accessible only from one computer or from a computer in a special network.

There are three different ways of message interchange among the agents:

- Agent communication works directly between two agents.
- Group communications works an agent and a group of agents.
- Community communication works between an agent and a community of agents.

Communication is always synchronous, which means that for every message a short answer must be given instantly.

However, there is a need to establish longer dialogs between two agents, and some messages can only be understood in the context of older messages. Therefore, agents can save messages which they have received or sent under a context. Whenever a new message arrives, the agent can prove whether the new message belongs to a context that already exists. Additionally, the contexts allow the agent to become aware of those contexts where an answer of another agent is missing.

Figure 2 shows the structure of the agent community. The community consists of the administration module, the communication module, and the agent container which can hold several autonomous agents. In the following, we will discuss these components of the agent community.

6.2 Administration Module

The administration module is the central module of the community. When the community starts, first an instance of the administration module is created, and this module creates the other components of the community and those agents that should be present at startup. Similarly when the community terminates, the communication module surveys shutdown process and fists ends execution, when all components and agents are ready for this. The administration module – and the community, respectively – can be parameterized by command line or by a configuration file.

By default, the administration module creates an administration control panel which can be used by a human administration to overview the whole environment. From the administration control panel, it is possible to access the other control panels in the environment, including the control panels of the agents. From the administration control panel, it is possible to change the configuration of the environment, especially it is possible to start agents.

There is also a log control panel, which can be used to display status and error messages of the community or the agents. It is also possible to write the log to file.

When the community terminates, the actual state of the community can be saved to file, and also the state of the agents that are currently in the community, can be saved. This gives the chance to rebuild the community in the last state after a temporary shutdown. The configuration can also be saved automatically in short periods, which increases the chance for a restart without looses, if the community is terminated uncontrolled (e.g., because of a hardware error).

6.3 Communication Module

The communication module is responsible for the communication among the agents of the community, and also for the communication of the agents of the community with agents outside the community. Communication can be encrypted or unencrypted, and it can be agent, group, or community communication.

Agent communication is the easiest way of communication. When the communication module receives an agent message directed to an agent of the own community, it simply forwards this message to the agent by a local method call, and the answer of this message is determined by the return value of the method. Only in the case of an error, the communication module has to answer the message. There is no difference whether the sender of the message is an agent of the own or a external community. However, when an agent of the own community sends an agent message (again by local message call), the communication module has to differ whether the receiver of the message is an agent of the own or another community. In the first case, the communication module calls the local receive method; in the second case, the communication module calls the send method of the external communication module.

The communication module does not have to know how the message is transmitted to the receiver.

When the communication module receives a community message, this message is forwarded to each agent in the community. The communications module collects the answers of the members of he community, and then gives a common answer for the system. When an agent of the community sends a community message, the communication module has to differ whether the addressed community is the own community or a foreign community. In the first case the local receive method is invoked; in the second case the corresponding send method of the external communication module.

While communities are software programs which physically exist on one or more computers, groups only logically exist. The members of a group can be distributed among different communities. All members of the group know the other members. It is possible to register groups so that they become public. However, only members of a group are allowed to send messages to this group.

Agents are free to create new groups; in this case, they are the only member of the group. They can ask other agents to join the group, and also other agents can apply to membership.

Group messages are treated as agent messages to each member of the group. When the communication module receives a group message, then this message is addressed to only one agent in the community, and the communication module can simply forward the message to the agent. However, when an agent of the community sends a group message, then this message can have several receivers. The communication module forwards the group message to each member of the group (and has to differ whether the member is an agent of the own community or an external community). The communication module collects the answers of the group members and creates a common answer.

There is a communication control panel which can be used to survey the entire communication of the community (internal and external).

6.4 Agent Container

The agent container holds the agents of the community. It is also responsible for the creation and termination of agents.

The creation of a new agent is not trivial, because the agents can have different types. The type of the new agent is given as a parameter, and the agent container must load the correct class file for the agent.

The advantage of this dynamic behavior is that the implementation of a new agent must be known at the moment this agent is creates, and not earlier. This means that it is possible to add new agent types while the community is running. There is no need to shutdown the system.

Each agent runs in an own thread, which gives the agent the chance of pro-active behavior.

6.5   Agents

The basic class of an agent defines the behavior of an agent that is common to all agent types. This encompasses the ability to some social behavior, e.g. contacts with other agents and dealing with groups, and error handling. A concrete agent type can inherit this basic class and add additional behavior. Of course, it is also possible to overwrite the behavior of the basic class. Instead of the basic class, it is also possible to inherit the class of another agent type, e.g., in order to create an enhanced version of this agent type. For a new agent type, only the new behavior has to be added, which make the creation of new agent types relatively easy.

The basic class of the agent hierarchy defines four repositories: The agent repository, the group repository, the community repository, and the context repository. The agent repository, the group repository, and the community repository hold data about the agents, groups, and communities known to the agent. The context repository holds the active contexts of the agent.

The content of the agent, group, and community repository are part of the agent configuration, together with the personal data of the agent (name, address, and type of the agent) and the name and address of the registry, if the agent is registered. In fact, agents can store any data in the agent configuration. The agent configuration is saved on disk, so the agent can re-load the configuration after a temporary shutdown or a computer failure. The configuration also allows the agent to move from one community to another. Short time data such as the messages stored in the context repository are not part of the configuration and not saved on disk, because most likely, they would be out of date after the restart of the agent. In the shutdown sequence, the agent usually closes all active contexts and informs the conversation partner about the situation. However, the agent can be offline without having the chance to perform the shutdown sequence. Hence, the agent must be able to notice a dropout of an agent and find alternative ways to perform their tasks.

Each agent has an own agent control panel that can be accessed by the administration control panel. The control panel shows the personal data of the agent (so that an agent can be identified) and the status of the agent. In the status, the agent can show with different status flags whether it is working normally or a problem has arose. It is also possible to display confirmations of messages sent or received and to inform about errors.

The agent control panel is the only way for the administrator of the agent (or the community, respectively) to maintain control over the agent. The basic class of the agent control panel allows the administrator to have insight in the four repositories of the agent and to change the contents. It is also possible to cause the agent to send messages. The status of the agent can be changed, the agent can be halted temporarily, or the shutdown sequence can be activated. Each agent type has an own agent control panel which should inherit the basic class or another class of an agent control panel.

There is also the possibility to remote control an agent by a System Administration Agent. However, the agent can refuse the commands of the System Administration Agent (e.g., if the agent does not trust the System Administration Agent). However, the agent must follow the commands of the agent control panel. Even in case of an error, the administrator still has influence on the agent through the agent control panel. The technical reason for this is that the agent control panel invokes methods of the agent directly, while the System Administration Agent can only send messages.

Additional to the agent control panel, which is designed for the administrator and always shown within the administration control, an agent may also have a graphical user interface that is designed for a customer or another human user of the system.

6.6   Web Services

While the internal communication of an agent is based on local method calls, the external communication is done by Web services. This approach can lead to a really open system, because any application can be a UniCats agent, as long as a Web service interface is provided. The underlying system, especially the programming language is not important.

In our Java-based implementation of the agent community, it was possible to create the necessary classes for the Web service interface automatically using the free AXIS toolkit. This does not only reduce the programming effort, it also increases the robustness of the system, since difficult to detect implementation errors could be avoided. Another positive effect is that after changes in the communication module, a functional community can be created very fast. However, instead of using AXIS, it is also possible to use any other tool for the Web service creation or write the Web service manually.

The external communication in the community is based on the external communication module. The external communication module contains the same send and receive methods as the communication module. The receive methods – which are called themselves by the Web service – simply call the corresponding receive messages of the communication module, while the send messages – which are called themselves by the corresponding methods of the communication module – call the Web service of the distant community.

The AXIS toolkit creates both the necessary classes for the Web service, which simply link the send methods of one community to the receive messages of the other community, and a WSDL document of the Web service interface, which can be used as a unified description of the interface.

In order to use the community as a Web services, it is necessary to couple the Web services with a Web server. Again, we did not write a Web server on our own – although this would be possible – but used the free Tomcat Web server, which is able to handle Java Web services. For simplification reasons, we created a second Web services in order to be able to start the community remotely.

7 Conclusion

In this paper, we have introduced the idea of an integration platform for libraries and information services which bases on an extensible society of software agents. We have pointed out the importance of the integration problem in the modern society and the challenges which accompany the development of such an integration platform. Due to the commercialization of the Internet, aspects of security and electronic commerce will have to be considered also in the scientific library scenario. We also have introduced our solution to the problem with the open architecture of the UniCats environment and our described our implementation of this system.

It is important to illustrate that the major design objective is extensibility. In a real open system, it is necessary to add new libraries and information providers, and there is also a need for new agents and agent types. It is also necessary that agents work together which are developed by different organization. The advantage of the design in this paper is that new agent or even new agent types can be added while the system is already running.

There are three different ways how to add new services to the integration system: (1) implement a UniCats agent (by inheriting the agent basic class or any other agent class); (2) create a Web service compatible with the UniCats environment; (3) do nothing and use a Provider Agent to join the integration system.

The practical usage of an open integration platform in the scientific literature area is obvious. The integration platform allows to bring together services and sources without touching the independence of the individual information providers. This gives the chance that many small libraries can be a large library.

The requirements for the practical application of the UniCats environments are few. Any computer with Internet connection can host one or more agents, and the use of communities even reduces the requirements in the computer system.

The scientific usage of the agent system lies in the study of the behavior of the agents in the market environment and in the study of open markets themselves. In the UniCats environment, agents decide about their way on task execution according to their programming, their knowledge about the structure of the environment, and their experiences. Of special interest is the fact that there is no central instance in the environment that can survey or manage the entire system. It is a scientific challenge to demonstrate that a complex system with only distributed knowledge can work.

It is necessary to extend the system continuously. The continued development and extension of the system may be a joined task for people from all over the world. The implementation of the system started at the University of Karlsruhe, but the development is now transferred to an open source project.

References

[1] M. Baldonado, C.-C. Chang, L. Gravano, A. Paepcke. The Stanford Digital Library Metadata Architecture. In *Inernational. Journal of Digital Libraries*, Vol. 1, Nr. 2, 1997.

[2] D. Boles, M. Dregger, et. al. The MeDoc System – a Digital Publication and Reference Service for Computer Science. In *Digital Libraries in Computer Science: The MeDoc Approach*, A. Barth, M. Breu, et. al. (eds), Springer LNCS, 1998.

[3] M. Christoffel. Cooperation and Federation of Traders in an Information Market. In *Proceedings of the AISB Symposium Intelligent Agents in Electronic Commerce*, York, 2001

[4] M. Christoffel. Information Integration as a Matter of Market Agents. In *Proceedings of the 4th International Conference on Electronic Commerce Research*, Montreal, 2002.

[5] M. Christoffel, B. Schmitt. Accessing Libraries as Easy as a Game. In *Proceedings of the 2nd International Workshop on Visual Interfaces to Digital Libraries*, Portland (Oregon), 2002.

[6] M. Christoffel, B. Schmitt, J. Schneider. Semi-Automatic Wrapper Generation and Adaptation: Living with Heterogeneity in a Market Environment. In *Proceedings of the 4th International Conference on Enterprise Information Systems*, Ciudad Real, 2002.

[7] M. Christoffel, S. Pulkowski, B. Schmitt, P. Lockemann. Electronic Market: The Roadmap for University Libraries and their Members to Survive in the Information Jungle. In *ACM Sigmod Record* Vol. 27 Nr. 4, 1999.

[8] E. Durfee, D. Kiskis, W. Birmingham. The Agent Architecture of the University of Michigan Digital Library. In *Readings in Agents*, M. Huhns, M. Singh (Eds.), Morgan Kaufman, 1998.

[9] E. Durfee, T. Mullen, S. Park, J. Vidal, P. Weinstein. The Dynamics of the UMDL Market Society. In *Proceedings of the 2nd Workshop on Cooperative Information Agents*, Paris, 1998.

[10] N. Fuhr, N. Gövert, C.-P. Klas. An Agent-Based Architecture for Supporting High-Level Serach Activities in Federated Digital Libraries. In *Proceedings of the International. Conference Asian Digital Libraries*, Seoul, 2000.

[11] A. Paepcke, R. Brandriff, et. al. Search Middleware and the Simple Digital Library Middleware Protocol. In *DLib Magazine*, Vol. 6, Nr. 3, 2000.

[12] B. Schmitt, A. Schmidt. METALICA: An Enhanced Meta Search Engine for Literature Catalogs. In *Proceedings of the 2nd International Conference Asian Digital Libraries*, Taipei, 1999.