Journal of Web Engineering, Vol. 4, No.3 (2005) 224-243
© Rinton Press

AN AGENT-ORIENTED APPROACH TO THE INTEGRATION OF INFORMATION SOURCES

MICHAEL CHRISTOFFEL, GUIDO WOJKE, SEBASTIAN WERNER, RINA REZEK, SU XU

Institute for Program Structures and Data Organization,
University of Karlsruhe, Germany
{christof,wojke,werners,rezekr,xu}@ipd.uni-karlsruhe.de

Received November 1, 2004
Revised March 9, 2005

The success of the Internet and the World Wide Web opened new ways of information supply. While more and more information sources become available, people are faced with the problem of information overload. New kinds of information systems are needed. They give people searching for information the opportunity to participate in the new development and profit from the new information sources that become available through the Web. A special challenge for Web information system modelling arises from the openness of the system: Everything is liable to change, and information sources come and go without further notice. In this paper, we present an approach to a flexible information system that is able to adapt to a dynamic environment. This approach is based on the idea of an open and dynamic information market with independent and autonomous providers and customers. We present the agent-oriented architecture of the information system and its realization in the application domain of scientific literature.

Key words: Information Integration, Electronic Markets, Multi-Agent Systems

1 Introduction

In modern society, information has become a very important commodity. Many professions depend on the steady supply of actual information; consider the information needs of scientists, journalists, managers, and even politicians. At the same time the demand for information began to increase, the success of the Internet and the World Wide Web opened new ways of information supply. Very often, people searching for information can do so without leaving their work desk, just using a standard Web browser. Information may either be contained directly in some static Web pages or stored in databases but accessible through a Web interface. Even information that is not available online can usually be searched and ordered electronically through the World Wide Web.

These statements are especially true in the field of scientific literature. Here, the Internet also opened new ways of search and delivery of scientific literature. The monopoly of the local library and the local book seller in scientific literature supply is broken. Not only is it possible to reach libraries, book sellers, and publishing houses world-wide through the Internet, but completely new services have also appeared such as bibliographic databases, technical report servers, and delivery services. More and more publications are published electronically, and the number of electronic books and journals is rapidly growing.

However, people searching for information are often faced with the problem of information overload. Most people are not able to survey the huge amount of information services and sources available, nor are they able to compare and estimate them in order to find out which sources are best for their special need. Moreover, finding and selecting services and sources is not the only problem. Although services are accessible through the Internet using standard protocols such as HTTP, the user has to learn how to use each service separately. Not only contents, but also site structure, query language, result format and media types may change from one service to another. A special complication arises from the fact that the Internet is dynamic and open. So all conditions can change from one moment to another, and information services can vanish and new services appear without any notice.

In a typical situation, a user has to access more than one information service in sequence to perform a search for information. In addition to the problem of knowing the most appropriate services, he/she has to learn how to use each single service. The combination of results received from different sources has to be done manually, often done by some ‘copy and paste’ actions.

However, searching information can be expensive not only in time but also in money. Together with the commercialization of the Internet, we can observe that new services and information sources tend to become commercial. We are witnesses of the development of world-wide information markets, where the value of information is determined by the law of supply and demand.

In this paper, we present an approach for the integration of information contained in different information sources on the Web. This integration system is based on the model of an open information market. In order to ensure scalability and robustness, the integration system is distributed itself, consisting of a large number of autonomous agents. The agents communicate using only standard Web protocols and implementing Web services.

The work presented in this paper is supported by German research association (DFG) as a part of the national German research offensive “Distributed Processing and Delivery of Digital Documents (V3D2)”.

We continue as follows. In section 2, we give more details on the open market model and the architecture of the integration system. In section 3, we introduce the most important agent types needed to perform the information integration task. This architecture is extended in the following sections: In section 4, we discuss the main aspects of security; in section 5, we introduce our extensions for billing and payment; and in section 6, we discuss the organisation of the agent system. Thereafter, we present the realization of the agent infrastructure in section 7. In section 8, we demonstrate the implementation of the before mentioned agent types. Section 9 contains a summary of practical experiences with the integration system. In section 10, we give a short overview on related work. We finish this paper with a conclusion in section 11.

2 Architecture Overview

The integration system described in this paper is based on the model of an open market. Consequently, information services and users of the system are treated as providers and customers in this market. Providers and customers are free to leave the market at their own decision; in the same way, new providers and customers may enter the market. The market participants may be distributed anywhere in the world, as long as they are connected to the Internet. The integration system acts as a market infrastructure between customers and providers. We call this infrastructure the UniCats environment, where UniCats stands for “a Universal Integration of Catalogues based on an Agent-supported Trading and Wrapping System”. A more detailed description on this market-based model can be found in [1].

The UniCats environment consists of a society of autonomous and communicative UniCats agents. Same as the market participants, UniCats agents can be distributed on any computers, but must have a connection to the Internet. Agents are free to enter and leave the market on their own. It is even possible that UniCats agents are created and controlled by different organizations, and this will be necessary when the environment becomes very large. The concept of UniCats agents is not limited to a special operating system or programming language. As long as they follow the same protocols, they will be able to interact. It is possible to have several distinct UniCats environments. In this case, there is no relation between the agents in different environments.

It is important that there is no need for a central authority that supervises and controls the agents. There is also no common plan to perform tasks. All agents follow their self interests, driven by incentives. These incentives can be monetary – agents maximize their profits by charging money for performed tasks. Since their income is correlated by the number of tasks, they are interested in doing as many tasks on behalf of the market participants as possible, and they want to show good quality in order to gain continuous relations with their clients and outdo their competitors. However, in most cases an agent is not able to perform tasks alone but has to rely on other agents. Because of this, they are interested in cooperation with other agents, building supply chains for complex tasks. Agents can only receive income if the task is fully completed and the market participants are satisfied.

This market-based approach has several benefits. First, the market is robust against the dropout of individual agents. Since there are no fixed relations between the agents, other agents can replace the agent that has dropped out. Second, the market-based approach supports the survival of the fittest. The better the quality of an agent's work, the higher is the chance of being selected by market participants and other agents. Third, and maybe most important, the market-based approach forces agents to be prepared for an unknown environment. Agents have to advertise their own capabilities and find other agents for business relations. Market agents are able to find their way in a dynamic environment.

The behaviour of an agent is determined by its agent type. However, the agent type only determines the behaviour of the agent towards other agents; it does not restrict the implementation of the agent, e.g., the algorithms applied. At the moment, 17 different agent types have been created, and this will not be the end of the line. It is possible for every user to create new agent types and add them to a new or existing environment, even at runtime.

The flexibility of our approach lies in the fact that the necessary capabilities are divided among different agent types and agents of different types may (and must) collaborate in order to perform their tasks. However, the type of an agent only determines the other agent types an agent can interact with, not the particular agents. So an agent will adapt itself to the current environment and react to changes. It will never depend on the presence of an individual communication partner, but will be able to find alternatives in the case that an agent drops out. Moreover, an agent will change its business partners on its own if it finds other agents that are more suited to their task. Prices that agents charge for their work can be decisive, too. In general, an agent has to find a compromise between the capabilities and the price of its business partners.

Agents working together can form groups of agents. There is no restriction on the location of the members of a group. An agent can be a member of any number of groups (including zero). Agents are free to enter and leave groups, or to create new groups.

While groups support the logical collaboration of agents, communities serve the physical collaboration. A community is the conjunction of the agents located at one computer node. An agent is member of exactly one community. A community may host any number of agents. Communities are a mere technical construct, primarily introduced for resource sharing. It is not necessary that agents of a community build groups or even know each other. Agents may also move from one community to another.

There are four different ways of communications among the agent of one UniCats environment:

- Agent communication works between two agents.
- Group communication works between an agent and a group of agents.
- Community communication works between an agent and a community of agents.
- System communication works between communities and is outside the control of the agents.

Communication may be secure. For each message, an agent can decide whether it is necessary to encrypt the message or not.

In the next section, we give an example of a possible application of the UniCats environment and introduce the agent types needed for the integration of information from Web sources in the field of scientific literature.

3 Agents for Information Integration

Although the architecture of the UniCats environment is more general and could be used in different applications, initial development focussed on information integration. For information integration, at least five different agent types are needed:

Provider Agents. These agents are the interface of the provider to the system. Provider agents are tailored to one provider; however, it is possible that several provider agents serve one and the same provider. When a provider agent receives incoming queries from other agents, these queries are transformed into the native query language of the provider. This can be done by filling out a Web form or by invoking a Web service, depending on the available interfaces of the provider. The results received from the providers are transformed into the uniform language of the environment, which is a superset of Dublin Core, the metadata standard in the digital library field. The Provider Agents also performs additional services such as query validation, monitoring, and optimization.

Provider Selection Agents. These agents assist in finding the most appropriate providers for a customer's demand. They hold profiles of the providers actually available in the market. Typical criteria used for the matching function are the location of the provider, the offered services, languages, and estimated cost.

Integration Agents. These agents send a query to several provider agents in parallel and integrate the received results to a single result list. The post-processing operations include duplicate elimination, result fusion, and thematic grouping.

Customer Agents. These agents are the representatives of the customers in the system. They provide each registered customer a personal workspace, where the customer can perform queries and receive results, examine past queries and results, and make annotations. The customer agent assists the customer in query formulization, selects appropriate providers with the help of a Provider Selection Agent, plans query execution, sends queries with the help of the Integration Agent, and presents the results to the customer. There is also a forum function so that customers working with the same Customer Agent can exchange experiences and found results.

Customer Interface Agents. These agents are the interface of the customer to the system. Same as a Customer Agent, a Customer Interface Agent can be used by several customers at the same time. The Customer Interface Agent holds information of the customers regarding their preferred 'look and feel' for the customer interface. Customer Interface Agents control the login procedure and provide a view on the personal workspace. A customer may use different Customer Interface Agents for the connection to the same Customer Agent. For example, he/she may have a favourite Customer Interface Agent for his/her desktop computer, but use another Customer Interface Agent when he/she is on a business travel and has to contact the UniCats system with a mobile device. The same way, it is also possible to apply different Customer Agents using the same Customer Interface Agent.

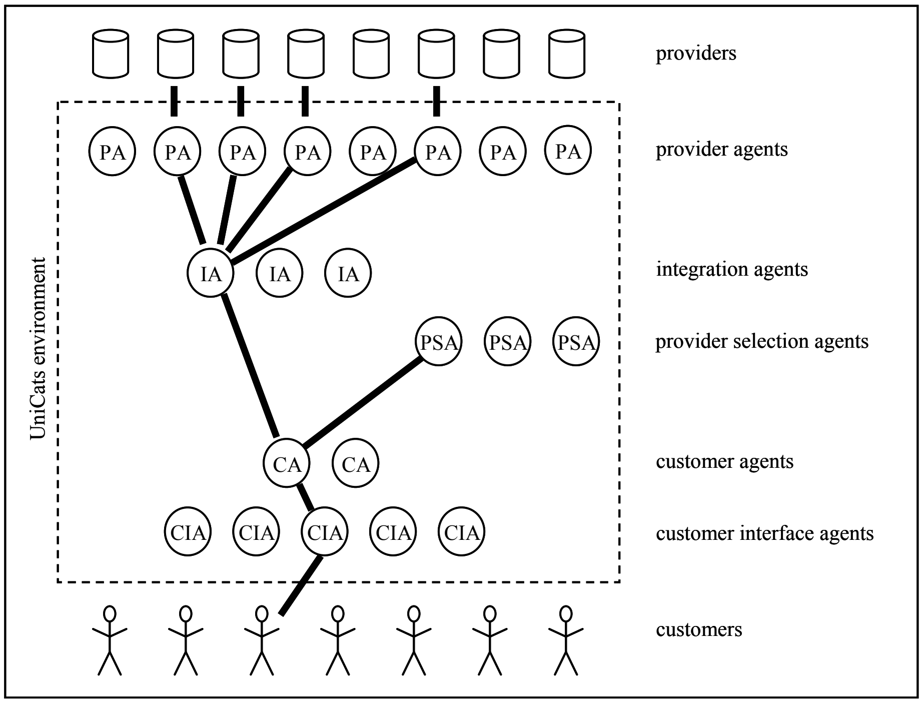


Figure 1 The UniCats environment as an integration system.

Figure 1 contains a UniCats environment with the five agent types described. In a typical scenario, a customer logs in at a Customer Interface Agents (CIA) and contacts a Customer Agent (CA) which opens the personal workspace for the customer. When the Customer Agent performs queries on behalf of the customers, it contacts a Provider Selection Agents (PSA) for recommendations about those providers suitable for the given query of the customer. Then the Customer Agent sends the 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), which 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 presents the results to the customer with the help of the Customer Interface Agent.

It is important to consider that this is only one possible interaction. A more complex 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 the following sections, we introduce some extensions of the information integration systems, regarding security, billing and payment, and the organisation of the multi-agent system.

4 Security

A platform capable of realizing an electronic market must consider some kind of security. Under the term security, we understand four properties:

- Authentication: Both sender and receiver of a message must be able to clearly identify themselves.
- Authorization: Each person allowed to access the system can do this only within the range of given rights or privileges.
- Privacy: Private data should not be available to any other person without the permission of the owner. If a person's private data fall in the hands of a third person, the third person must not be able to use these data.
- Integrity: Messages transmitted between sender and receiver should not be changed during transmission. If a message has been changed during transmission, the receiver must be able to notice the change. Consequently, he/she will be able to throw the message away and ask the sender for a new transmission.

Transferring these properties to the information integration scenario, we can find four concrete requirements (compare [2]):

- Authentication of customers: The identity of a customer must be assured.
- Authorization of customers: Customers using the system may have different rights. E.g., in a university library system, faculty members may have more rights than students, and these have more rights than external users.
- Authentication of agents: The identity of an agent must be assured.
- Secure communication channels: It must be possible to encrypt messages transmitted between agents so that only the intended receiver of the message is able to understand the message and manipulations of the message can be detected.

While the latter requirement can be met by the agent infrastructure, the other requirement needs the presence of two new agent types:

Customer Authentication Agents. These agents hold databases with the registered customers including customer id, passwords, and access rights. Customer Authentication Agents are invoked by Customer Interface Agents during the login processing and issue an electronic customer passport for the customer who wants to enter the system. The customer passport describes the identity and the access level of the customer.

Agent Authentication Agents. These agents work similar to the Customer Authentication Agents, but issue agent passports that certify the identity of an agent. Agent passports can be used to sign messages and create trust between agents which do not know each other.

It is an important aspect of the flexibility of the UniCats system that passports are not needed in all cases. E.g., if a customer only wants to perform a free search query, it is not necessary that he/she has been registered. However, although it is possible for a customer to access the system with a guest account, it is advantageous for a customer to be registered. Registered customers can access more services than guests and have a personal workspace. The customer passport is attached to each message originated by a customer query, and all agents can identify the initiator of the query. They can reject queries if the customer does not have a valid passport or the authorization level is too low. In the example mentioned before, a Provider Agent representing an online database of a university library could refuse access if the customer is not a campus user.

In the same way, agents do not necessarily have to own an agent passport. When an agent enters a community, it does not have a passport but can interact with other agents. However, other agents may refuse cooperation if the agent can not identify itself with a valid passport. For interactions involving costs, agent passports are needed.

5 Billing and Payment

Since the UniCats system has been designed to support commercial services and agents are intended to charge for their work, it must handle billing and payment. For financial transactions, it is important to consider security requirements, which have been discussed in the previous section. Moreover, it is mandatory that all payments are comprehensible. Each payment is based on an invoice, and accounting assures the assignment of each money transfer. In order to guarantee the regularity of the claims, billing and payment is not left to payer and payee, but assigned to two new agent types:

Billing Agents. These agents hold internal accounts for registered agents and market participants, where money transfers can be booked. They perform accounting and protocol invoices and payments.

Payment Agents. These agents perform payment transactions between two internal accounts. The accounts can be held by one and the same billing agent or by different billing agents. Payment agents can also be in communications with external financial facilities.

In the case that payer and payee are agents, both agents must have an internal account with a billing agent (it can be the same billing agent or two different billing agents). The payee issues an invoice, which must be registered at the payee's billing agent, and sends it to the payer. If the payer accepts the invoice, it registers the invoice at its own billing agent. There are two different payment ways: credit transfer and direct debit. For credit transfer, the payer engages a payment agent to settle and open invoice. The payment agent contacts the two billing agents where the invoice is registered and leads them to debit the amount of the invoice to the account of the payer and credit the same amount to the account of the payee. The payment agent guarantees the regular execution of the credit transfer; both billing agents mark the invoice as paid. For direct debit, the payee engages a payment agent to settle an open invoice. The payment agent will refuse the order unless the payee can show a direct debit instruction issued by the payer.

While the internal accounts are only a virtual construct inside the UniCats environment, there must be at least one real bank account for each billing agent. If the billing agent supports different currencies, it must have one real account for each currency. In order to pay money into an internal account of an agent, the organisation that owns the agents must transfer the money to the real account of the billing agent. The credit on each internal account of a billing agent is correlated to the credit on the real account. If the credit is used up, the billing agent refuses further paying, except the holder of the account can use an overdraft provision.

As long as payer and payee use the same billing agent, transactions are only shifts from one internal account to another and do not affect the real account. If two billing agents are involved, transactions lead to the transfer of money between the billing agents. For performance reasons, these transfers are not updated to the real accounts instantly but only periodically. Credit transfers between the real accounts of the billing agents can be done automatically by the agents themselves (if they have electronic access) or manually by a human administrator.

The introduction of billing and payment allows agents to charge for their services, e.g., the Customer Agent can charge the customer a rent for the personal workspace, or the Provider Selection Agent can charge the Customer Agent a fee for finding the best providers. As a consequence, the monetary cost of services enlisted becomes an additional parameter for the selection of an agent. Very often, the agent – similar to a human person – has to find a compromise between quality and cost. In the example mentioned before, a customer may use a cheap Customer Agent, which does not save queries and does not offer any advanced features, instead of a more sophisticated but expensive one. Similarly, the Customer Agent can decide on a provider without obtaining the recommendation of a Provider Selection Agent, using its own experiences in previous queries. Until now, prices are fixed and set by the administrator of the agent. A new version of the agent framework should allow agents to determine their prices on their own, trying out their own market value.

From a theoretical point of view, billing and payment between agents can be used as incentives. A piece of software can do its job because it has been programmed to do so. However, this is not typical for a software agent, which is supposed to decide about its actions on its own. This decision can be determined by monetary incentives. Agents act so that they maximize their profits. They have a reason to permanently improve their quality and execute tasks given to them by their clients as well as possible, because they want to be preferred to their competitors and get more tasks in the future. Clients – who are the source of income – can be customers and customer organisation, and even providers who pay for being mediated by the integration system. When the payment mechanism is used as an incentive, it is not necessary to have real currencies; the mechanism works fine with toy money. However, it is reasonable to have an organisation that employs an agent with the clear objective to earn money. In this case, an underlying bank account and real currencies can be used.

Not only agents but also market participants (providers and customers) can have an internal account. The billing and payment mechanism works the same. However, especially for customers this mechanism can be complicated. Moreover, it would require too many system resources if all customers have their own monetary account. For simplification reasons, we introduced a new agent type:

Customer Organisation Agents. These agents are the representatives of customer organisations to the systems, e.g., universities, research associations, or companies. Customer Organisation Agents can perform queries on behalf of their members in order to make use of special conditions granted by the providers for the customer organisations. They also can hold an internal account for all payments of the members of the organisation.

The alternative to the internal payment transfer between two internal accounts is external payment. For external payment, a payment agent transfers money between a billing agent and an external financial institution. The payment agent can use different kinds of means of payment, including forms of electronic cash. In the same way, transactions between two external financial institutions are possible, e.g., payment transactions between customers and providers. Customers can settle a bill to the UniCats system using his/her preferred payment method, and then the UniCats system pays the bill to the provider using another payment method.

6 Organisation of the Multi-agent system

In the previous sections, we assumed that agents know each other and each agent can select other agents for interaction. However, the market and with it the UniCats environment is supposed to be dynamic and open. Agents can appear and vanish without further notice. There is no authority in the market that can survey all agents within the environment. As a consequence, agents must be able to learn about other agents. They must find other agents that can be possible partners for interactions.

The easiest way to find other agents is to send messages to the own and other agent communities. These messages are forwarded to each agent in the communities, so these agents learn about the agent that sends the community message. If other agents are interested in the initiating agent and want to learn more about it, they will answer the message.

There are two more sophisticated ways for agents to advertise themselves and learn about other agents in the environments. We have introduced two more agent types:

Agent Naming Agents. These agents hold an agent registry. All agents should register themselves to an Agent Naming Agent. When an agent moves to a new community, the agent should notify the registry and provide the new address. So the agent registry can be used to find out the address of a specific agent. It is also possible to search for agents of a given type.

Billboard Agents. These agents are used to publish messages to unknown receivers. Other agents can search the message boards and read the messages. They can answer the messages if they want to contact the agent that wrote the message.

Agents are not forced to find other agents or to advertise for themselves. They can also stay in the background and simply observe the market.

Groups are another important way for agents to collaborate. Agents joined in a group can use group messages to address the group members. An agent can be a member of any number of groups. Similar to the environment in general, groups have a decentralized structure and are organized by the group members themselves. Each group has a group policy which specifies who can be a member of this group and which procedure has to be applied in order to admit or remove a group member.

Group Naming Agents. These agents hold a group registry. Each group should be registered at a Group Naming Agent. Agents can search for groups through the Group Naming Agent.

While there is no central control over the environment, it is possible to survey the environment or a part of the environment.

System Administration Agents. These agents can be used to monitor the environment or a part of the environment and react in the case of a failure. Agents can send messages to System Administration Agents, inform about their current state and report irregularities. Incoming messages are logged and can be analyzed automatically or manually by a human administrator. The System Administration Agent can also be used for remote diagnostics. Agents receive messages from the System Administration Agent even if they are stopped. However, no agent can be forced to answer messages or follow orders given by the System Administration Agent. Because of that, the value of System

Administration Agents is that they provide the ability for a person to supervise the distributed agent system at a glance. This is important for system development and test, and also to perform experiments on the system. In the practical application, agents are supposed to run autonomous without any supervision and guidance by human persons.

7 Agent Infrastructure

We implemented the UniCats environment using the Java programming language. This brings the advantage that our agents can run platform-independent on most computers. There is also a large set of open source tools and class libraries 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 these agents will work together in one environment. We tested this with a sample environment encompassing different hardware platforms and agents written in seven different programming languages in order to prove that our approach is feasible to cross-language applications, which is necessary especially for the connection to legacy systems.

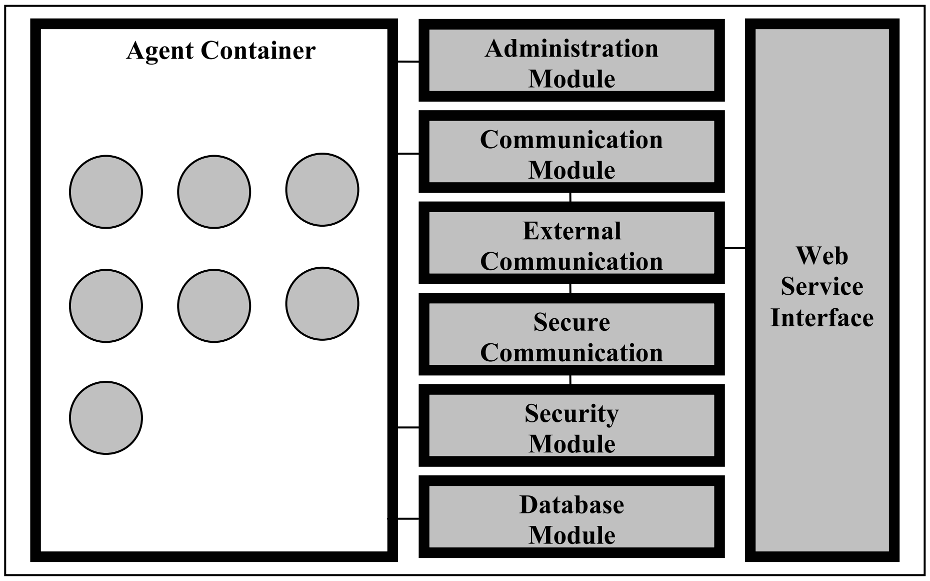


Figure 2 The UniCats community.

Figure 2 shows the basic structure of a UniCats community. The community consists of an agent container and six modules. The agent container can hold any number of UniCats agents, sharing resources. Agents can be added and deleted at runtime. It is also possible for agents to migrate from one agent community to another.

Administration Module. The administration module is the main module of the community. It is responsible for the initialization of the community and controls startup and shutdown of the agents.

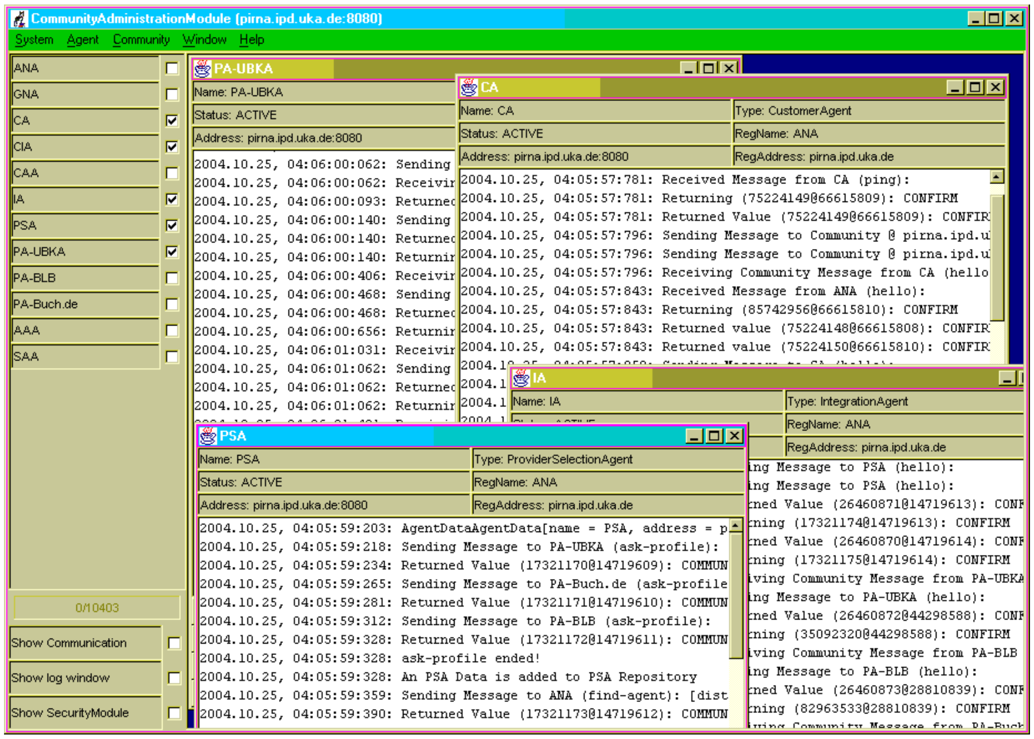
Communication Module. The communication module is responsible for the communication of all agents in the community and manages outgoing and incoming messages.

External Communication Module. While messages directed to an agent inside their own community are forwarded to the agent directly, the Communication Module delegates messages that are to be delivered outside the community to the External Communication Module. Similarly, the External Communication Module receives all messages coming from outside the community and forwards them to the Communication Module. The sending agent can decide whether secure transmission channels and encryption of the message are necessary or not.

Secure Communication Module. When secure communication is necessary, the secure communication module invokes a certificate exchange with the community of the receiver and generates a common key by means of the Diffie Hellman Protocol. The Secure Communication Module encrypts all messages sent to this community using the common key.

Security Module. The security module validates certificates of the local or external community. It is also able to issue new certificates which are signed with the digital signature of the community.

Database Module. The database module stores any data used by the agents in a relational database system. Alternatively, the data can be stored in XML files on the computer's file system. If the agent moves from one community to another, the data can be sent as XML documents through the Web services interface.



ANA	☐
GNA	☐
CA	✓
CIA	✓
CAA	☐
IA	✓
PSA	✓
PA-UBKA	✓
PA-BLB	☐
PA-Buch.de	☐
AAA	☐
SAA	☐

0/10403

Show Communication	☐
Show log window	☐
Show SecurityModule	☐

Name: PA-UBKA
Status: ACTIVE
Address: pirna.ipd.uka.de:8080

```
2004.10.25, 04:06:00:062: Sending
2004.10.25, 04:06:00:062: Receivir
2004.10.25, 04:06:00:093: Returnec
2004.10.25, 04:06:00:140: Sending
2004.10.25, 04:06:00:140: Returnec
2004.10.25, 04:06:00:140: Returnir
2004.10.25, 04:06:00:406: Receivir
2004.10.25, 04:06:00:468: Sending
2004.10.25, 04:06:00:468: Returnec
2004.10.25, 04:06:00:656: Returnir
2004.10.25, 04:06:01:031: Receivir
2004.10.25, 04:06:01:062: Sending
2004.10.25, 04:06:01:062: Returnec
2004.10.25, 04:06:01:062: Returnir
```

Name: CA | Type: CustomerAgent
Status: ACTIVE | RegName: ANA
Address: pirna.ipd.uka.de:8080 | RegAddress: pirna.ipd.uka.de

```
2004.10.25, 04:05:57:781: Received Message from CA (ping):
2004.10.25, 04:05:57:781: Returning (75224149@66615809): CONFIRM
2004.10.25, 04:05:57:781: Returned Value (75224149@66615809): CONFIR
2004.10.25, 04:05:57:796: Sending Message to Community @ pirna.ipd.u
2004.10.25, 04:05:57:796: Sending Message to Community @ pirna.ipd.u
2004.10.25, 04:05:57:796: Receiving Community Message from CA (hello
2004.10.25, 04:05:57:843: Received Message from ANA (hello):
2004.10.25, 04:05:57:843: Returning (85742956@66615810): CONFIRM
2004.10.25, 04:05:57:843: Returned value (75224148@66615808): CONFIR
2004.10.25, 04:05:57:843: Returned value (75224150@66615810): CONFIR
2004.1
2004.1
2004.1
```

Name: IA | Type: IntegrationAgent
RegName: ANA
RegAddress: pirna.ipd.uka.de

```
ing Message to PSA (hello):
ing Message to PSA (hello):
rned Value (26460871@14719613): CONF
rning (17321174@14719613): CONFIRM
rned Value (26460870@14719614): CONF
rning (17321175@14719614): CONFIRM
iving Community Message from PA-UBKA
ing Message to PA-UBKA (hello):
rned Value (26460872@44298588): CONF
rning (35092320@44298588): CONFIRM
iving Community Message from PA-BLB
ing Message to PA-BLB (hello):
rned Value (26460873@28810839): CONF
rning (82963533@28810839): CONFIRM
```

Name: PSA | Type: ProviderSelectionAgent
Status: ACTIVE | RegName: ANA
Address: pirna.ipd.uka.de:8080 | RegAddress: pirna.ipd.uka.de

```
2004.10.25, 04:05:59:203: AgentDataAgentData[name = PSA, address = p
2004.10.25, 04:05:59:218: Sending Message to PA-UBKA (ask-profile):
2004.10.25, 04:05:59:234: Returned Value (17321170@14719609): COMMUN
2004.10.25, 04:05:59:265: Sending Message to PA-Buch.de (ask-profile
2004.10.25, 04:05:59:281: Returned Value (17321171@14719610): COMMUN
2004.10.25, 04:05:59:312: Sending Message to PA-BLB (ask-profile):
2004.10.25, 04:05:59:328: Returned Value (17321172@14719611): COMMUN
2004.10.25, 04:05:59:328: ask-profile ended!
2004.10.25, 04:05:59:328: An PSA Data is added to PSA Repository
2004.10.25, 04:05:59:359: Sending Message to ANA (find-agent): [dist
2004.10.25, 04:05:59:390: Returned Value (17321173@14719612): COMMUN
```

.

Figure 3 The control panel.

Each module and each agent has a graphical interface, the control panel. The control panels are used to survey and administer one community. They are hierarchically structured with the administration control panel as the parent frame (Figure 3). In addition to the direct control through the control panel, it is also possible to configure the agents and the community with human-readable configuration files. Most commands can also be applied without the need of the graphical interface through a text-based command prompt

The communication between different UniCats communities is operated by web services. Each community has a Web Service Interface, which is controlled by the External Communication Module. In this way, every message transmitted includes all parameters and is automatically converted to an XML document which is delivered to the receiver Web service using standard Internet protocols. The use of Web Services as transport layer is the main reason for the ability of the UniCats system to build cross-platform applications. Another advantage is that we can overcome the firewall problem. While many network administrators close Internet ports for security reason, UniCats is not touched by this, because the Web service communication uses only those standard ports accessible at every system.

While messages sent through Web services are usually stateless, it is necessary in many cases to have longer sequences of messages exchange between two or more agents that belong together. All these messages can be assigned to a context designated by a context id. For each message sent or received, the agent can decide whether this message is to be saved and assigned to a context. Contexts can be arranged in a work queue. The agent can access the contexts in the work queue (more precisely, the work objects which contain a reference to a context together with some additional information about the execution) either in a sequence or directly by the context id. The agent can also remove a context from the queue. It is also possible to set triggers on the work queue that watch time passing or any event. For example, a work object could automatically alarm the agent when a timeout occurred.

For a more detailed description on the UniCats community, compare [3].

8 Implementation

The agent types described in this paper have been implemented as extensions of an abstract agent class. This base class of the agent hierarchy provides the functionality that allows an agent to interact with the community and the agent container, send and receive messages, communicate with other agents, manage groups, and use the work queue. There is also a base class for the agent control panel.

Provider Agent. The major design goal for a Provider Agent was to have a uniform access to a provider without restricting the autonomy of this provider. Provider agents are tailored to a special provider. However, writing a Provider Agent manually for each provider is a hard and expensive undertaking. Moreover, for each change at the provider or its interface, a new provider agent has to be created. Because of that, we implemented a general Provider Agent class that can be adapted to a special provider by configuration.

The Provider Agent consists of several modules (Figure 4). The Coordinator oversees each query in the work queue and checks whether the agent (or the customer) is allowed to access the provider. The query is handed over to the Query Processor, which checks whether the query can be executed and which attributes are to be extracted.

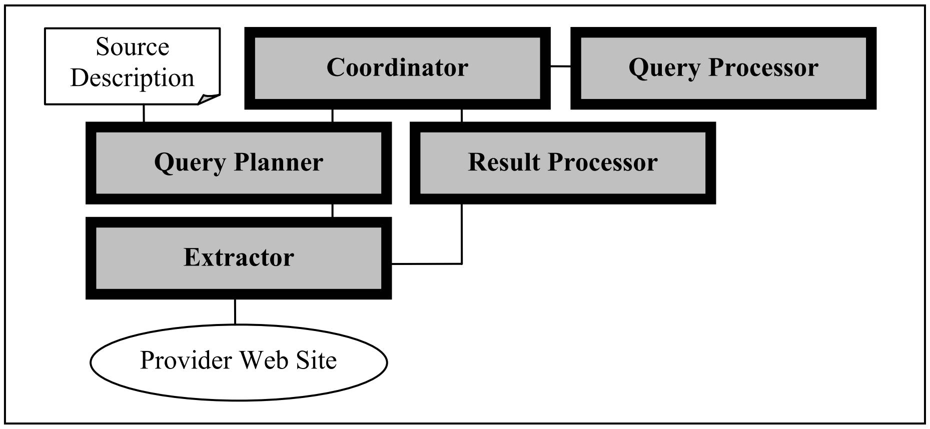


Figure 4 Provider Agent.

In general, the Web site of a provider consists of several Web pages, and the information needed is distributed among these pages. It is also possible that a piece of information can be found in more than one Web page. For each query, the Plan Processor generates a plan that describes how the needed attributes can be extracted with minimal cost. The query plan contains the sequence of the Web pages that have to be loaded in order to extract all attributes. Very often, the query plan contains iteration, as in literature search, where detailed information is to be extracted for each document found. The extraction of the attributes out of the Web pages is described by extraction rules, using hierarchical path expressions and regular expressions. The query plan is based on an abstract graph of the result pages, which is obtained from the source description that describes the structure of the provider's Web site. This source description can be created automatically or semi-automatically (compare [4]).

The query together with the query plan is given to the Extractor which sends the query to the provider, loads the result pages, and extracts the attributes out of the HTML code, according to the query plan. The Result Processor creates a result list from the received results and gives this result list to the Coordinator, which sends the results back to the agent that sent the query. If the result set is very extensive, the result list is not sent back completely but in smaller portions.

If the provider offers a Web service interface, the work of the Provider Agent is much easier, because the planning and extraction process can be skipped.

Provider Selection Agent. The selection of appropriate providers for a given query is done on the base of descriptions of the providers (profiles). Provider Selection Agents hold profiles for available providers in the Metadata Repository. The collection of profiles is done autonomously by the Metadata Management module. In general, profiles are transmitted by the Provider Agents. The profiles are validated for completeness and correctness.

Each query in the work queue is accepted by the Coordinator which hands each valid query over to Query Handling. The query is examined against each available profile using a matching function, and a value of correspondence is calculated, depending on the attributes and the weights. The results are ranked according to the value of correspondence and handed back to the Coordinator, which sends the recommendations back to the agent that sent the query. Provider Selection Agents joined in a group can help each other and share queries and profiles. The principle modules of the Provider Selection Agents are shown in Figure 5.

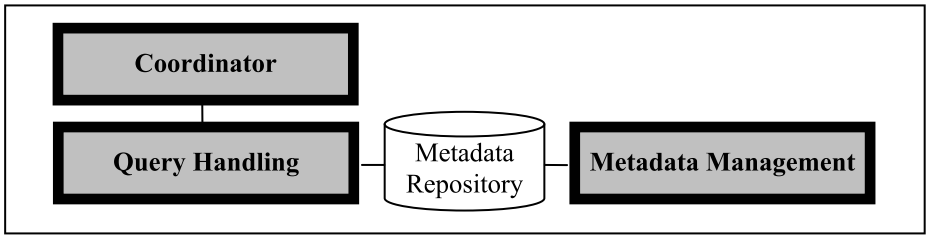


Figure 5 Provider Selection Agent.

Integration Agent. The Integration Agent forwards each query contained in the work queue to the Provider Agents listed in the query. As soon as the first results come in, the Integration Agent starts duplicate elimination and grouping and sends the processed results to the requesting agent. When new results are received, they are merged with the previously received results. This way, the customer never has to wait long for the first results and can work with the first results while result integration still continues, but can be sure to get the complete result set in the end.

Duplicate elimination is based on k-way sorting (compare [5]). This way, the decision whether two documents can be treated as equal can be based on the comparison of different attribute sets. Documents marked as duplicates are merged together so that the information about the document contained in different places can be preserved. After duplicate elimination, the documents contained in the result list are grouped into different result groups (e.g., the results of a query for 'Java' could be divided in groups dealing with South Asia, coffee, or programming languages). Grouping is based on keyword lists that are generated from the first results that have been received.

Integration Agents can cooperate in the execution of queries in order to share work.

Customer Agent. Customer Agents hold the personal workspace for each customer. The personal workspace contains previous queries and results. It is possible to make notes and annotations to documents. There is also an email and a forum function available. Customers can exchange messages and share queries and results. This exchange can be initiated by the customers themselves or by the Customer Agent which can give recommendations about other users with similar interests.

Customer Agents can learn about a customer observing his/her behaviour. This observation is based on search terms used in previous queries. They also collect information about the customer provided by questionnaires, such as areas of interest and language skills. They can use the background knowledge about the customers to assist in query formulization, e.g., they can give advice or automatically fill out fields. However, the customer has the full control about the content over his/her profile and can overwrite all settings in an expert mode.

For each query, the Customer Agent can ask a Provider Selection Agent for recommendations about appropriate providers or base its decision on their own knowledge about the providers contacted previously. The Customer Agent either contacts a Provider Agent directly or uses an Integration Agent to contact several providers in parallel. The results which come in incrementally are either forwarded to the Customer Interface Agent to be presented to the customer or combined with other results in order to process complex queries.

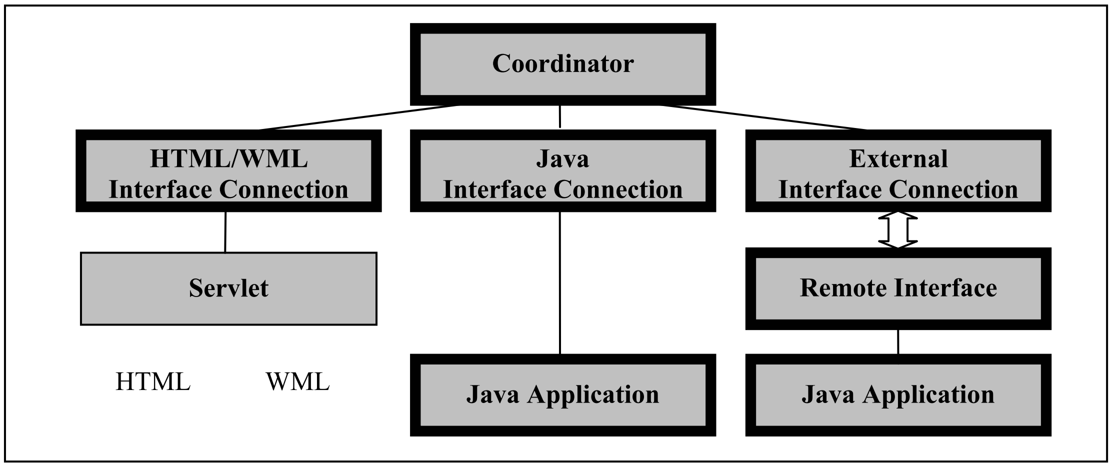

Coordinator
HTML/WML Interface Connection
Java Interface Connection
External Interface Connection
Servlet
Remote Interface
HTML
WML
Java Application
Java Application

Figure 6 Customer Interface Agent.

Customer Interface Agent. The Customer Interface Agent is responsible for the connection at the interface to the customers. This connection is done by the Customer Interface Connection module. We have implemented different Interface Connections that can be used with different interfaces (see Figure 6). However, each Customer Interface Agent is supposed to provide only one interface.

The HTML/WML Interface Connection connects to a servlet that is executed in the servlet engine of a Web server. Both HTML/HTTP and WMP/WAP are supported. The Java Interface Connection connects to a Java application. We implemented Java-based customer interfaces using Swing and Java 3D. Additionally, there is a text-only interface that can come in handy when a graphical interface is not available. However, the Java application must run on the same computer as the Customer Interface Agent. Since this is not a practical solution, there is the Remote Interface Connection, which connects to a second Web service interface. With this interface, a Java application (or an application written in any other programming language) can be connected from any computer that has access to the Internet.

The Customer Interface Agent saves the setup of the customer. This setup contains the preferred dialog language of the customer and the name of the Customer Agent, but also the preferred 'look and feel' of the interface, especially colours. The Customer Interface Connection also provides a translation mechanism for natural language expressions that appear on the customer interfaces. Text elements used in the interfaces are represented by tokens independent from the concrete dialog language. When a user is logged on, the tokens are translated to the preferred language of the customer. It is also possible to change between different languages. With the translation mechanism, it is not necessary to have different language versions of a Web page (or the output of a Java application). In order to add a new language, it is only necessary to add the new words and expressions to the translation file. There is no need to change the interfaces.

Customer Authentication Agent. The Customer Authentication Agent holds a list of the registered customers. This list can be maintained by the administrator of the agent. The Customer Authorization Agent only issues customer passports to registered customers. Customers are allowed to self-register during login process. Customer passports are only valid for a given period of time. There is also a revocation list that contains passports that have been revoked, e.g., if the customer refused to pay a bill.

Agent Authentication Agent. The Agent Authentication Agent holds a list of the registered agents, which can also be maintained by the administrator of the agent. In contrast to customers, agents are not allowed to self-register. If an agent is not registered but requests an agent passport, the request stays pending. The administrator can decide whether to register this agent or reject the request. The Agent Authentication Agent also holds a revocation list.

Billing Agent. The Billing Agent holds the internal accounts for its customers and also a repository of invoices. Transfers to and from the internal accounts can be done manually by the administrator or automatically initiated by a Payment Agent on the basis of an existing invoice. All transfers are documented.

Payment Agent. Payment Agents also contain a repository of all transactions performed. Until now, only internal payment is supported.

Customer Organisation Agent. The Customer Organisation Agent holds a repository of the members of the organisation. The Customer Agent forwards queries of organisation members to the corresponding Customer Organisation Agent. The Customer Organisation Agent then executes the query in the identity of the customer organisation using a customer passport issued for the organisation. It sends all results back to the Customer Agent. The Customer Organisation Agent can agree to pay invoices for their members, too.

Agent Naming Agents and **Group Naming Agent**. These agents hold the agent and group repository, respectively. Within a repository, the names of agents and groups are unique.

Billboard Agent. Messages sent to a Billboard Agents are stored in message boards. Other agents can search or browse the billboards for specific messages. Alternately, they can subscribe to boards so that the Billboard Agent will forward incoming messages to the subscribing agents. If there are several Billboard Agents in an environment, they can share messages and billboards.

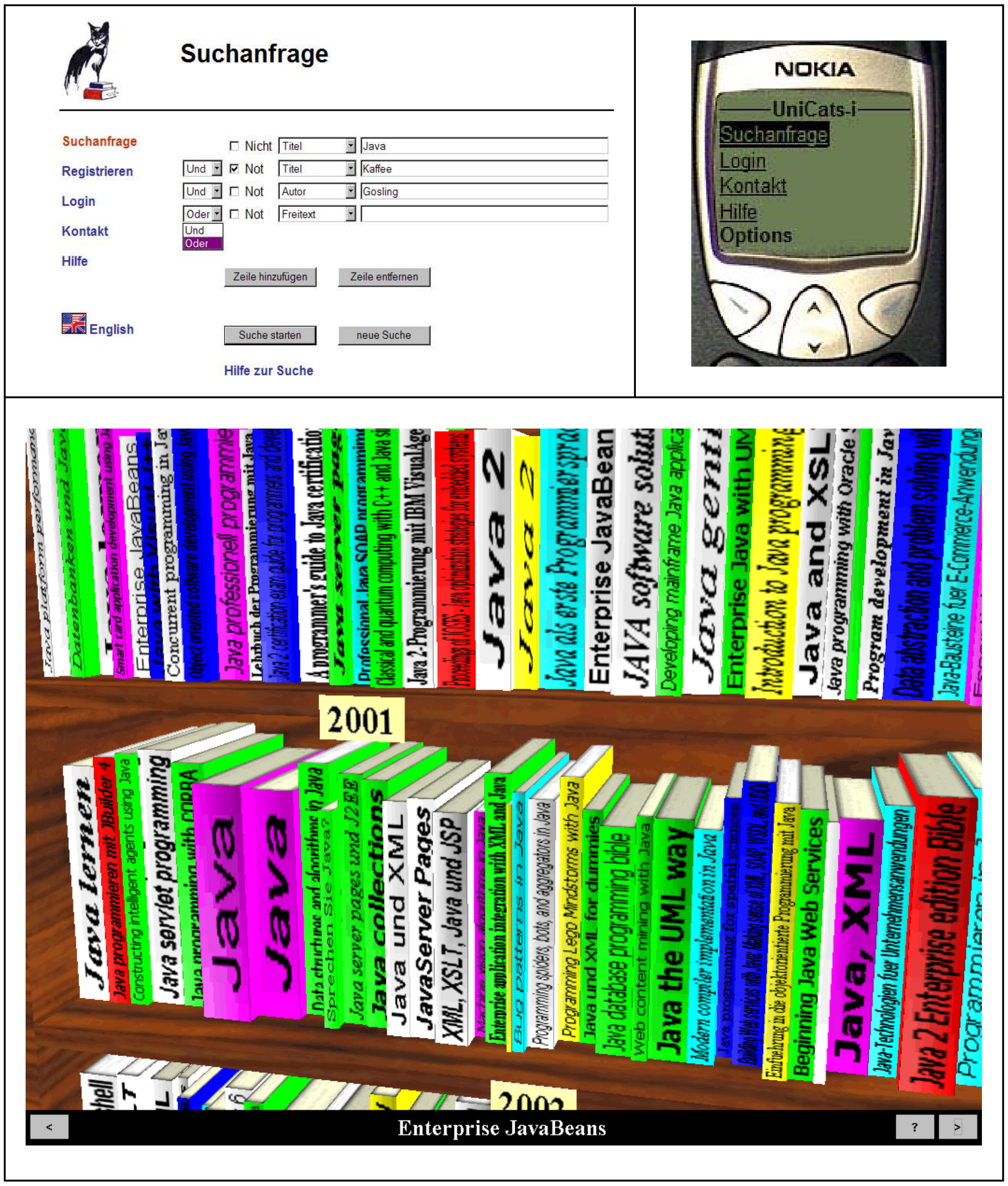
System Administration Agent. An Agent can (but do not have to) send messages to a System Administration Agent and inform it about their position, their current status and load, and failures. This information can be used by a human administrator to survey the actual state of the environment (or a part of the environment), which is important for error management and the evaluation of the system. The System Administration Agent can order other agents to stop, so that a human administrator can control an agent more easily. In the current implementation, all agents support the System Administration Agent.

9 Practical Experiences

The UniCats has been tested with more than 50 different providers in the field of scientific literature supply: traditional and electronic libraries, booksellers, and publishing houses. As long as the providers offer a HTML interface (or a Web services interface), it is possible to create a Provider Agent to this provider. In many cases, the creation of a Provider Agent can be automated using a generated source description.

Different customer interfaces have been created for the UniCats environment: HTML, WML, Java Swing, and Java 3D (compare Figure 7). Surveys about the acceptance of the system are not yet finished.

An early result of the field studies is that the performance of the system is low. The choice of Java programming language, the extensive use of XML as data format, and the communication through Web services cause a relatively high utilization of resources. The performance can be increased, if agents collaborating together are on the same community, so that external communication is not needed. However, in general, it cannot be guaranteed that agents working together are on the same computer node, especially when they are controlled by different organizations.



Suchanfrage

	☐ Nicht	Titel	Java
Und	☑ Not	Titel	Kaffee
Und	☐ Not	Autor	Gosling
Oder	☐ Not	Freitext	

Figure 7 Different customer interfaces: HTML, WML, and Java 3D

We see four strategies that can increase the performance of the system. The first strategy is the use of groups of agents having the same type. Agents joined in groups can share work and better distribute incoming queries. The second strategy is a restricted application of secure communication. The third strategy is a better distribution of agent among the available communities. We implemented this strategy in the agent framework. Agents can call up the actual load of the own and other known communities and decide to move to another community which is less occupied. However, the movement procedure makes the agent unavailable for a period of time and causes additional load, because the agent's database must be transferred, too.

We also have tested a fourth strategy, which allows the multi-agent system to create new agents of a given agent type at runtime. These agents could have a default configuration. If an agent is overwhelmed with work, it can create a new agent of its own type and forward tasks to the new agent. It is also possible to start a new community on a computer. For this purpose, we implemented a Web service interface for the remote control of a community, including startup and shutdown. However, the access to the remote control gives agents much power and can be misused.

The performance of the UniCats system is not really slow in comparison to a manual search or the answer time of some information sources. Each agent can handle several tasks in parallel, but there is a configurable limit which prevents the agent from slowing down. If the number of open tasks in the work queue of an agent reaches the limit, the agent switches to busy state and refused new tasks until a task are finished.

10 Related Work

There are a number of projects working with information integration of information sources in the field of scientific literature. In this section, we want to introduce some of these approaches.

The Stanford Digital Library Project attempts 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 [6]. Communication is based on the SDLIP protocol [7]. A large set of tools have been developed for this architecture. Although there is a general protocol stack that covers different functions, including billing and payment, the requirements for recourses that can be linked to the system are high. So they must provide online access to their databases or at least provide a complete metadata set of the documents available. Another disadvantage of this centralized approach is the missing robustness that can cripple the entire system when important components fall out.

The University of Michigan Digital Library investigates the integration of collections using 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 agent-oriented approach opens the way to a flexible, extensible, and robust system. However, the direct access to the connected resources is necessary, which makes the agent system not suitable for an electronic commerce environment. Major functions such as result integration and resource selection are still missing.

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 set of software agents that can be distributed. 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. Daffodil succeeds in covering the heterogeneity of the underlying sources. However the problem of selecting the right provider is now transferred to a higher layer, because the user has to choose the right stratagems for his/her search. The user also has no influence in search execution. Since the agents are suited to the resources that are availably in the system, any extensions need a re-implementation of the agents.

The aim of the MeDoc project was the creation of a distributed electronic library in the field of computer science [11]. 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. MeDoc supports a flexible and robust structure for the access to distributed information sources. However, the requirements to providers are very high, since both a direct access to the databases and the presence of a full metadata catalogue is necessary. An integration service which could be used to integrate and combine results from different sources is missing.

11 Conclusions

In this paper, we presented an approach for the integration of information from distributed information sources of scientific literature supply. The basic concepts are the model of an open information market and the capability of increasing flexibility, extensibility, and robustness by distributing the functionality to different software agents. Using only standard protocols, it is possible to create cross-platform applications where individual agents can be implemented by independent organizations.

The system should be extended for new features so that its value for customer and providers rises. Beside the extension of the system, the main focus on future work should be increasing the performance.

The market-based approach allows the creation of large systems with hundreds of providers and thousands of agents. These systems cannot be realized with a centralized approach. Simulation experiments, however, confirm that large systems can be realized with independent agents such as the UniCats systems following a market-oriented approach, keeping scalability and robustness.

Acknowledgements

The authors would like to express their thanks to German Research Foundation (DFG) for their financial support over many years that made this research project possible. We also would like to thank the entire UniCats team for their good work and comradeship.

References

1. Christoffel, M. Information Integration as a Matter of Market Agents. in Proceedings of the 5th International Conference on Electronic Commerce Research (Montreal, Canada, 2002)
2. Christoffel, M., Killat, M. Supporting Security in an Electronic Market System on the Base of Web Services. in Proceedings of the 6th International Baltic Conference on Databases and Information Systems (Riga, Latvia, 2002) 497-509

3. Christoffel M., Wojke G., Gensthaler M. How Many Small Libraries Can Be a Large Library. in Proceedings of the 5th Russian Conference on Digital Libraries. (St. Petersburg, Russia, 2003)
4. Christoffel M., Schmitt, B., Schneider J. 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, Spain, 2002) 60-67
5. Feekin, A., Chen, Z. Duplicate Detection Using k-Way Sorting Method. in Proceedings of the ACM Symposium on Applied Computing (Como, Italy, 2000) 323-327
6. Baldonado, M., Chang, C.-C., Gravano, L. and Paepcke, A. The Stanford Digital Library Metadata Architecture. International Journal of Digital Libraries 1(2), 1997. 108-121
7. Paepcke, A., Brandriff, R, et al. Search Middleware and the Simple Digital Library Interoperability Protocol. Dlib Magazine 6(3), 2000
8. Durfee, E., Kiskis, D., Birmingham, W. The Agent Architecture of the University of Michigan Digital Library. in Huhns M., Singh, M. (eds.) Readings in Agents. Morgan Kaufman, 1998. 98-108
9. Durfee, D. Mullen, T., Park, S., Vidal, J, Weinstein, P. The Dynamics of the UMDL Market Society. In Proceedings of the 2nd Workshop on Cooperative Information Agents (Paris, France, 1998)
10. Fuhr, N., Gövert, N., Klas, C.-P. An Agent-Based Architecture for Supporting High-Level Search Activities in Federated Digital Libraries. in Proceedings of the 3rd International Conference Asian Digital Libraries (Seoul, Korea, 2000) 247-254
11. Boles, D., Dregger, M., et. al. The MeDoc System – a Digital Publication and Reference Service for Computer Science. in Barth, A., Breu, M., et al. (eds.) Digital Libraries in Computer Science: The MeDoc Approach. Springer LNCS 1392, 1998. 12-19