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
Universität Karlsruhe
76128 Karlsruhe, Germany
+49 721 608-{4069, 3911}
{christof,wojke,werners,rezekr,xu}@ipd.uni-karlsruhe.de

Abstract. 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 modeling 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. We present the agent-oriented architecture of the information system and the realization of this system in the application domain of scientific literature.

1 Introduction

In the modern society, information has become a very important good. Many professions depend on the steady supply of actual information; just think of scientists, journalists, managers, and even politicians. In the same time as the demand on information begun 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 this without leaving their work desk, just using a standard Web browser. Information may either be contained directly in some static Web pages or contained 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, completely new services have appeared such as bibliographic databases, technical report services 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 demand. 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 for each service separately how to use it. Not only contents, but also site structure, query language, and 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 a sequence in order 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, too. 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 a system for the integration of information contained in different information sources in 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 (V^3D^2)".

We continue as follows. In the following section, we will give more details on the open market model and the architecture of the integration system. In section 3, we will introduce the most important agent types from a connectional point of view. In section 4, we will present the realization of the agent infrastructure. Thereafter, we will demonstrate the implementation of the selected agent types in section 5. In section 6, we will shortly introduce the remaining agent types needed for the integration system. Due to the limited space, we cannot give a detailed introduction. In section 7, we will give a short overview on related work. We will finish this paper with a conclusion.

2 Architecture

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. Between customers and providers the integration system acts as a market infrastructure. We call this infrastructure the UniCats environment, where UniCats stands for "a Universal Integration of Catalogs based on an Agent-supported Trading and Wrapping System". A more detailed description on this market-based model can be found in [3].

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 organization, 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 environment. In this case, there is no relation between the agents in different environments.

The behavior of an agent is determined by its agent type. However, the agent type only determines the behavior of the agent towards other agents; it does not restrict the implementation of the agent, e.g., the algorithms applied. At the moment, 13 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.

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 will 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, it has been created for information integration. 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.

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 favorite

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.

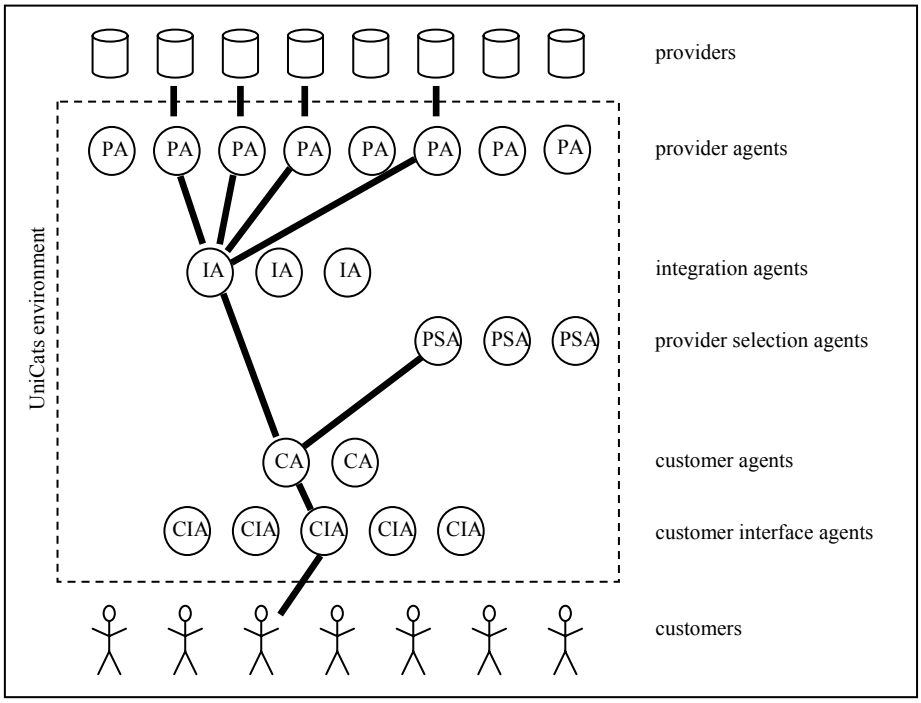

Fig. 1. The UniCats environment as an integration system

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.

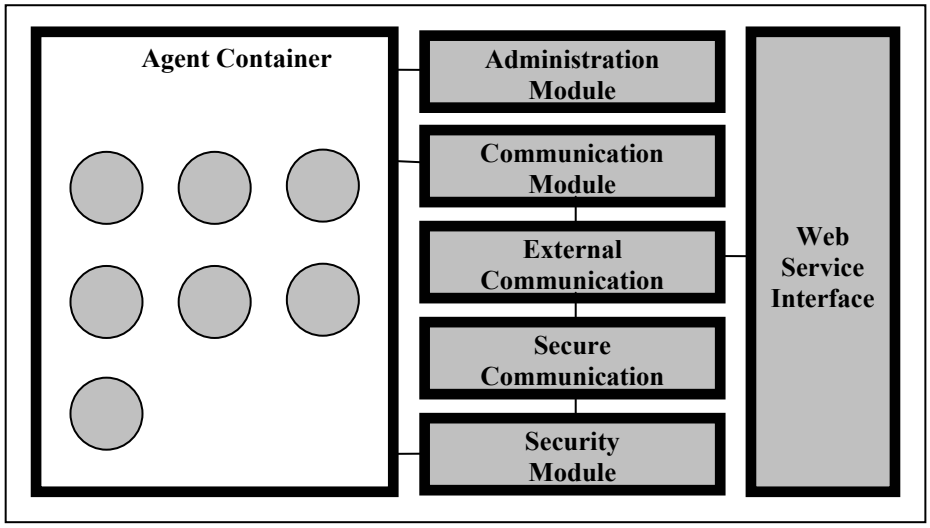

Fig. 2. The UniCats community

4 Agent Infrastructure

We implemented the UniCats environment 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 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 shows the basic structure of a UniCats community. The community consists of an agent container and five 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 the own community are forwarded to this agent on a direct way, the Communication Module delegates messages that are supposed 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 the encryption of the message is 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 validates certificates of the own or other community. It is also able to issue new certificates which are signed with the digital signature of the community.

Any 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. This way, every message transmitted included all parameters is automatically converted to an XML document which is delivered to the receiver Web service using standard Internet protocols. The use of Web Service 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 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.

Each module and each agent has an assigned 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.

For a more detailed description on the UniCats community, compare [6].

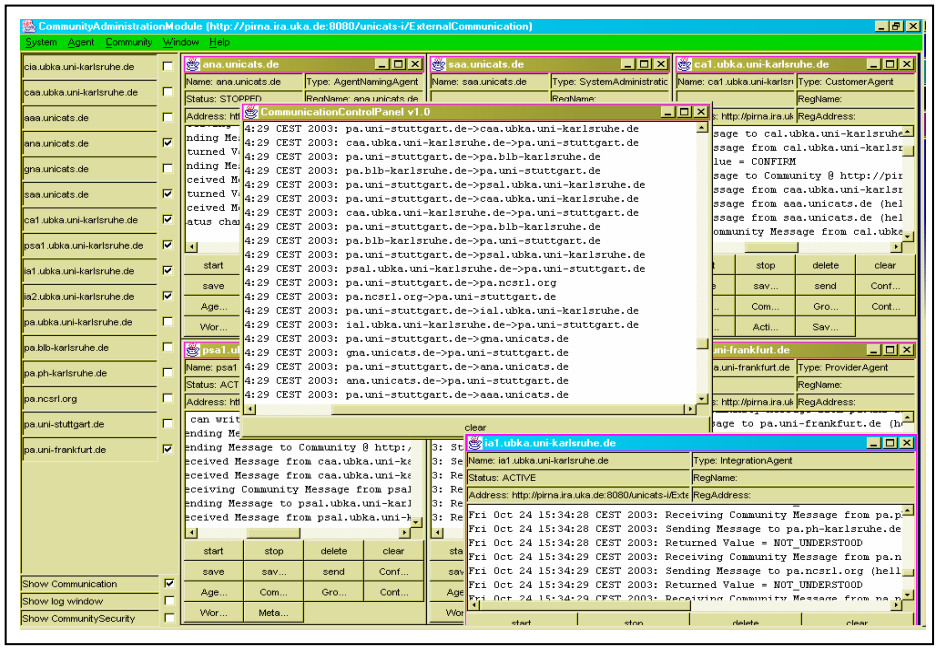

Fig. 3. The control panel

5 Implementation

The agent types described in section 3 have been implemented as extensions of an abstract agent class. This abstract class provides the necessary functionality for the agent in order to interact with the community and the agent container, basic messages such as interactions with other agents and group management, and work queue handling. It also provides a basis class for the agent control panel.

Provider Agent. The major design goal for a Provider Agent is to provide a uniform access to a provider without restricting the autonomy of this provider. Provider agents are tailored for some 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. 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.

In general, these attributes are distributed in several web pages. It is also possible that an attribute can be found in more than one Web page. For each query, the Plan Processor generates a plan how the needed attributes can be extracted with minimal

cost. The plan is based on an abstract graph of the result pages, which is obtained from the source description which describes the structure of the provider's Web site. This source description can be created automatically or semi-automatically (compare [5]).

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 using XPath expressions and regular expressions contained in 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 is done on the base of a provider's profile which is collected by Metadata Management and contained in the Metadata Repository. 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 over to the Coordinator, which sends the recommendations back to the agent that sent the query.

Integration Agent. The Integration Agent forwards each query contained in the work queue to the listed Provider Agents. 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 final.

Duplicate elimination is based on k-way sorting (compare [9]). 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 those results received first.

Customer Agent. Customer Agents hold a profile and the personal workspace for each customer. Observing the behavior of a customer, they can learn the areas of his/her interests. 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.

A complex query may be divided in several single queries. 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.

Customers can exchange notes and share queries and results with other customers. This can be initiated by the customers themselves or by the Customer Agent which can give recommendations about other users with similar interests.

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. This module also holds the setup of the customers for the 'look and feel' of the interface. Until now, three different customer interfaces have been created: A Java application, a HTML site, and a WML site (compare Figure 4). We are also working on an interface based on virtual reality environment using the metaphor of a library building, but this interface is not yet connected to the UniCats system (compare [4]).

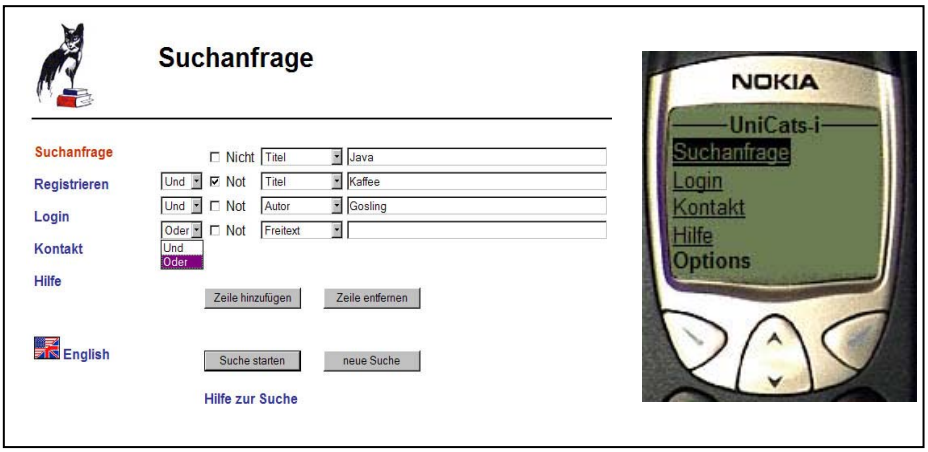

Fig. 4. Different customer interfaces: HTML and WML

6 Other Agent Types

In addition to the agent types which are the focus of this paper, several other agent types have been developed:

Customer Authentication Agents. These agents hold databases with the registered customers including customer id, passwords, and access level. 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 customer can access more services than guests and have a personal workspace. It is also possible to give different access rights to different personal. For example, in a university library systems professors tend to

have some more rights than students, and these tend to have more rights than non-members of the university. Customer Authentication Agents are called by the Customer Interface Agents during the login processing and issue an electronic customer passport for the individual customer that describes the identity and the access level of the customer.

Agent Authentication Agents. These agents work similar to the Customer Authentication Agents, but certify the identity of an agent by issuing an agent passport. These agent passports can be used to assure trust exchanged between agents which do not know each other.

Customer Organization Agents. These agents are the representatives of customer organizations to the systems, e.g., universities, research associations, or companies. Customer Organization Agents can perform queries in behalf of their members in order to make use of special conditions granted by the providers for the customer organizations.

Billing Agents. These agents hold registries of financial transactions performed within the system. The hold accounts for each registered agent, where incomes and expenses are booked. These incomes and expenses can result from internal transactions between the agents or from transactions of the agents with external facilities.

Payment Agents. These agents are the connection of external financial facilities to the system. The most important task of the Payment Agents is to support 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. This way, it is not necessary to have direct contact between customer and provider and the market model can be preserved.

Agent Naming Agents and **Group Naming Agents**. These agents provide a name service for agents and agent groups.

System Administration Agent. These agents can be used to monitor the environment or a part of the environment and react in the case of a failure.

7 Related Work

There are a number of projects working in the field of the 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 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. Although there is a general protocol stack that covers different purposed including billing and payment, the re-

quirements 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 aims in the integration of collections by an infrastructure of software agents [7]. 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 [8]. 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 (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. 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 [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. 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 catalog are necessary. An integration service which could be used to integrate and combine results from different sources is missing.

8 Conclusions

In this paper, we presented an approach for a system 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 flexibil-

ity, 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.

Beside an extension of the system, the main focus on future work should be increasing the performance. 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. In this case, secure communication should only be applied when it is really necessary.

We see a solution for the performance problem in groups of agents of the same type, which share work. The agent joint in this kind of group have to organize this work sharing automatically. In experiments we want to find out, where in the environment bottlenecks can appear and how load balancing could be reached.

References

1. Baldonado, M., Chang C.-C., Gravano L., Paepcke A.: The Stanford Digital Library Metadata Architecture. In: International Journal of Digital Libraries 1(2), pp. 108-121 (1997)
2. 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, Spring LNCS 1392, pp. 12-19 (1998)
3. Christoffel, M.: Information Integration as a Matter of Market Agents. In: Proceedings of the 5th International Conference on Electronic Commerce Research, Montreal (2002)
4. Christoffel, M., Schmitt, B.: Accessing Digital Libraries as Easy as a Game. In: Kathy Börner, Chaomei Chen (eds.): Visual Interfaces to Digital Libraries, Springer LNCS 2539, pp. 25-38 (2002)
5. 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, pp. 60-67 (2002)
6. 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 (2003)
7. 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, pp. 98-108 (1998)
8. 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 (1998)
9. Feekin, A., Chen, Z.: Duplicate Detection Using k-Way Sorting Method. In: Proceedings of the ACM Symposium on Applied Computing, Como, pp. 323-327 (2000)
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, pp. 247-254 (2000)
11. Paepcke, A., Brandriff, R, et al.: Search Middleware and the Simple Digital Library Interoperability Protocol. In: Dlib Magazine 6(3) (2000)