UDC 004.428, DOI: 10.2298/csis0902001V

Extensible Java EE-Based Agent Framework and Its Application on Distributed Library Catalogues

Milan Vidaković[1], Branko Milosavljević[1], Zora Konjović[1], and Goran Sladić[1]

[1]Faculty of Technical Sciences, Trg D. Obradovića 6,
21000 Novi Sad, Serbia
{minja, mbranko, ftn_zora, sladicg}@uns.ac.rs

Abstract. The paper presents the new agent framework XJAF and its application on distributed library catalogues. The framework is based on the Java EE technology and uses the concept of the plug-ins for implementation of the basic framework components. One important plug-in of the agent framework has been introduced into this system: the inter-facilitator connection plug-in, which defines how multiple facilitators form an agent network. The inter-facilitator connection plug-in is particularly important in both design and implementation phases in the field of distributed library catalogues. In order to substantiate the above statement, the framework has been used for implementation of the agent-based central catalogue of the library information system BISIS. Also, the framework has been used to implement the agent-based metadata harvesting system for the networked digital library of theses and dissertations (NDLTD). Both systems have been implemented at the University of Novi Sad.

Keywords: agent framework, Java EE, plug-in, extensible, library catalogue.

1. Introduction

Intelligent agents play an important role in software engineering regarding their intensive deployment in search and processing of information, as well as in complex software products, tools and systems. Agents are entities which are not capable of existing without an agent framework. According to [12], an agent framework consists of a collection of entities, objects, agents and autonomous agents. Agent frameworks in [23] represent a set of services demanded by users or by other frameworks. Agents are providing those services.

Agents need a programming environment which will create and enable agents to execute tasks [12, 19, 8, 7]. Beside controlling the life cycle of an agent, an agent framework also provides messaging and service subsystems to effectively support agents. Messaging allows agents to communicate to each other, and service subsystem gives them the possibility of accessing

various resources or executing complex algorithms that do not need to be implemented in the agent itself. The set of subsystems mentioned above is an agent framework. An agent framework also provides agent mobility and security. Agent mobility allows agents to migrate from one agent framework to another. The security subsystem provides security mechanisms which protect both agents and frameworks.

When analysing an agent framework, one can identify a certain number of requirements it needs to meet. First of all, the agent framework needs to provide working environment for the agents, taking into account security issues. Also, agent mobility is an important feature to be implemented, and this feature automatically raises the question of the implementation technique: to use an existing distributed components technology or make a proprietary solution. In addition to code execution, an agent framework must provide message interchange. Agent and service discovery subsystems are important elements of any agent framework, too.

All of these requirements are subject of the FIPA specification [14]. By implementing an agent framework compliant with the FIPA specification, it is possible for the agent framework to cooperate with other agent frameworks.

The application of agent-based solutions in the field of distributed digital library catalogues contributes easier and more effective usage of such systems. Agent-based systems provide for better connectivity among libraries and can implement various complex algorithms related to search activity more naturally (for example, "parallel" search), which are essential for use of distributed digital library catalogues.

The paper is organised as follows. Section 2 describes related work. Section 3 gives the detailed overview of the new agent framework, which is later used for implementation of the virtual central library catalogue and metadata harvesting. Section 4 represents two agent implementations using the XJAF agent framework in the field of distributed digital catalogues. Finally, the last section gives the conclusion and future work guidelines.

2. Related Work

Early papers on agent frameworks were based on presenting solutions to the particular problems [28, 9]. This led to specialised systems which use agent technology to solve specific problems in various fields [24, 41, 20, 3].

On the other hand, general-purpose agent frameworks appeared [17, 5, 1, 18, 4] due to the need to have systems which support arbitrary agents. These frameworks represent systems which control the life cycle of agents, provide inter-agent communication and agent mobility. Security issues are also taken into consideration.

From the technology point of view, agent frameworks are based on either proprietary solutions or solutions based on the distributed components technology. Agent frameworks like JAF (*Java Agent Framework*) [17] and JAT (*Java Agent Template*) [18] are based on proprietary solutions, while Aglets

[1], JADE (*Java Agent DEvelpment framework*) [4] and SAFT [7] are based on the RMI, CORBA and Java EE technology.

A lot of papers are related to the security issues in agent frameworks [40, 6, 34, 21, 42, 15, 38]. Security issues regarding agent frameworks include: providing message integrity, code protection during agent migration and protecting agent frameworks from malicious agents.

Most of the existing agent frameworks meet all or most of the mentioned requirements. However, the implementation elements of agent frameworks are hard-coded and cannot be changed. If there is a need to use another algorithm in an element of the framework, it is practically impossible to use it without recompiling it (if possible at all).

There is also one concept which is not investigated enough in agent frameworks – inter-facilitator connectivity mechanism. There are papers [13, 22] which deal with the concept of *Multi-Agent Organization* in terms of providing inter-relationship between agents, as well as implementing load-balancing. The paper [13] offers the definition of agent organisation (*Multi-Agent Organisation*) which is a way of providing inter-relationships between agents. Multi-Agent Organisation represents one means of distributing tasks, data and resources. Load-balancing can also be the reason to organise agent frameworks. In [22] it is stated that hierarchical organisation of agent frameworks can give much better results than having a large number of agents in a single agent framework. Also, most agent frameworks use networks, but without awareness of the network environment. In order to communicate, the two agents must know exact IP addresses of each other's host computers. These issues in particular are addressed in this paper as a part of the *ConnectionManager* component (in the section 3.4.).

The agent framework XJAF (eXtensible JavaEE-based Agent Framework) [35] presented in the paper is based on the Java EE technology and is compliant with the FIPA specification. All important elements of the framework are implemented as plug-ins, which provides for flexibility in both design and implementation.

There are several agent-based implementations of digital libraries [25, 32, 29]. The *Daffodil* project is a virtual digital library which enables searching over a federation of heterogeneous digital libraries. Its initial implementation enabled integrated search of more than eighteen digital collections and other resources (including ACM digital library, Springer, Google, GetInfo, HCIBib Human Computer Interaction Resources, the Collection of Computer Science Bibliographies, the Digital Bibliography & Library Project, the Directory of Open Access Journals, the Scirus scientific resources search engine, and Cornell University's ArXIv online database). The system supports collaborative search and provides information of new or changed objects related to previous searches. The system supports both low-level and high-level search functions. Low-level information search is mostly comprised of "moves". These basic moves might include adding a keyword to a search, or following a link. The relevant *Daffodil* tools include a personal library and interactive tools such as the "Did you mean…" feature that checks the search terms in a query and makes suggestions/corrections. The high-level search

includes the following features: reference and citation management, journal and conference proceedings search, author search, the classification tool (presents a topic and domain-based representation scheme), and the thesaurus tool. The backend of the Daffodil architecture is based on the CORBA agent architecture.

The University of Michigan Digital Library (UMDL) [32] project is an agent-based solution that enables users to search through heterogeneous digital libraries. The agents at work in the UMDL process user searches and display the results, filter large quantities of information, monitor usage patterns, and pass information on to other agents for further processing. The agent core consists of three types of agents: user interface agents, mediation agents, and collection agents. User interface agents conduct interviews with users to establish their needs and to specify areas of interest, so that the system can notify the user of items of potential relevance. Mediation agents coordinate search of many distinct but networked collections by taking orders from interface agents. Collection agents are associated with each specific collection and can handle search within specific collections of text, images, graphics, audio and video.

The MALIBU project [29], implemented in the United Kingdom, is also an agent-based digital library solution. It is used to enable search over different kinds of resources and media, making this system a kind of a hybrid library. The search engine is agent-based and it has the following features: keyword, author, and title search, profile management (this feature allowed the user to select only those targets which might be most suited to their research needs), and exportation of results via email, HTML, plain text, RTF or RDF. There are two types of agents: a central communication agent and a query agent. A central communication agent does the interaction with the user and employes the appropriate query agent for the search. A query agent is used to perform the search over a designated target. A special ontology is developed for MALIBU agents which enables proper communication among them.

An exhaustive overview of use of intelligent agents in modern libraries is presented in [33]. According to this paper, agents can be used as flexible infrastructure providing for efficient search within the personalized information environment. The same source also gives notes on challenges related to application of agent technology in libraries.

3. EXtensible Java EE-Based Agent Framework (XJAF)

At the very beginning of this section we shall make reference to some implementation issues of the agent framework XJAF which is presented in this paper. The Java EE technology is used for implementation of the agent framework XJAF [38, 37]. The Java EE technology is particularly useful because it comprises a large set of technologies and provides for scalability, reliability and has the large number of implementations. One element of the Java EE technology is particularly useful – the EJB (*Enterprise JavaBeans*)

technology. This is a technology of distributed software components which are created, executed and destroyed in the application servers. The typical life cycle of an EJB component is: the component is found in the application server container, used and put back into the container. All performance-related issues like load-balancing, distribution-per-server, etc. are left to the implementation of the application server. Beside supporting distributed components, Java EE also has all other technologies for the agent framework implementation: JMS (*Java Message Service*) for message exchange, JNDI (*Java Naming and Directory Interface*) for directory implementation, Java Security, etc.

A XJAF system consists of a Java client, the *FacilitatorProxy* component which hides all implementation details from the client (i.e. JNDI lookup, JMS message composition, etc.), and the *Facilitator* component. The main component of the framework is the *Facilitator* component which is deployed in the application server (Figure 1).

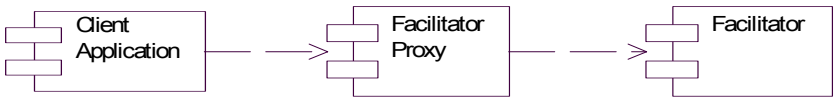

Fig. 1. Component diagram of XJAF system

The *Facilitator* component, which is the main component of the framework is implemented as an EJB component. This component is deployed in the Java EE application server and is not directly used by the client. For this purpose, the *FacilitatorProxy* component is used. It connects the client application and the *Facilitator* component. The *Facilitator* component is represented in the Figure 2.

The facilitator forwards parts of its job to corresponding managers. The managers are instances of classes implementing the corresponding managerial interfaces. The `AgentManager` interface is responsible for allocating and releasing agents. It represents the *Agent Directory Service* component of the FIPA specification. The `TaskManager` interface manages the tasks. It stores tasks, assigns tasks to corresponding agents and provides a way of notifying the client of the results. The `MessageManager` interface is responsible for inter-agent communication and it actually represents the *Transport Message* component of the FIPA specification. The `ConnectionManager` interface manages inter-facilitator connections and relations. The `SecurityManager` interface defines security aspects of the agent framework. The `ServiceManager` interface defines service directory and represents the *Service Directory for Services* component of the FIPA specification.

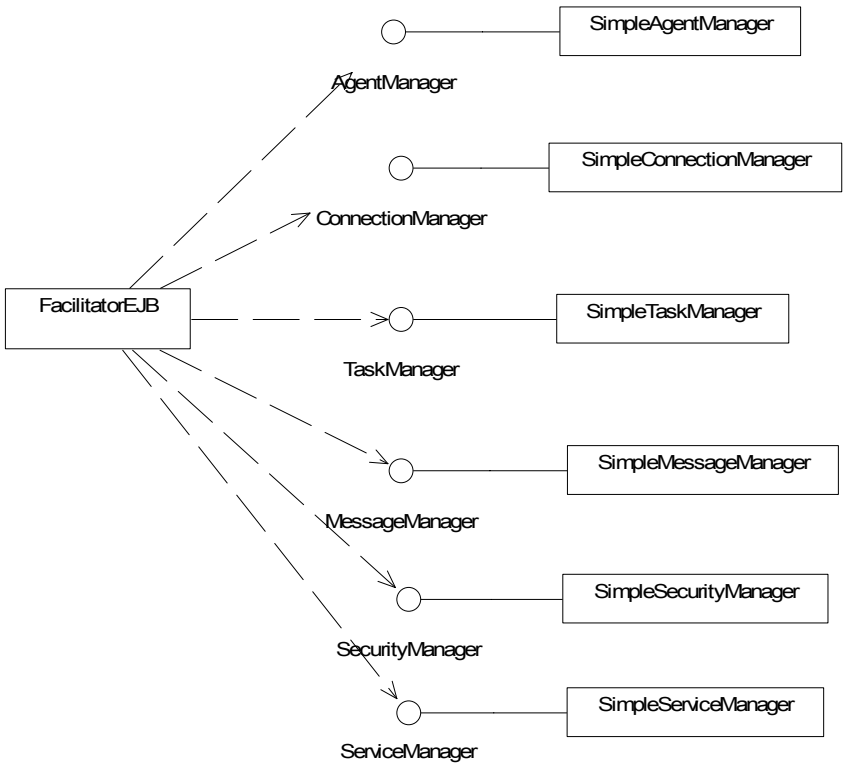

Fig. 2. The Facilitator component

The classes which implement the mentioned interfaces respect the corresponding algorithms for individual functions. The system is designed so that it is possible to choose an arbitrary manager when configuring, provided that it implements the given interface. This enables use of arbitrary managers whose existence is not necessary at compile-time, but is at the time of initialisation (the plug-in concept). This also allows the user to choose the appropriate strategy for implementation of the agent framework. This strategy is implemented by plugging-in the appropriate manager (done merely by configuring) of the agent framework rather than compiling

3.1. Facilitator Component

The *Facilitator* component realises the facilitator functionality. The facilitator forwards parts of its job to the corresponding managers. The managers are instances of classes implementing the corresponding managerial interfaces.

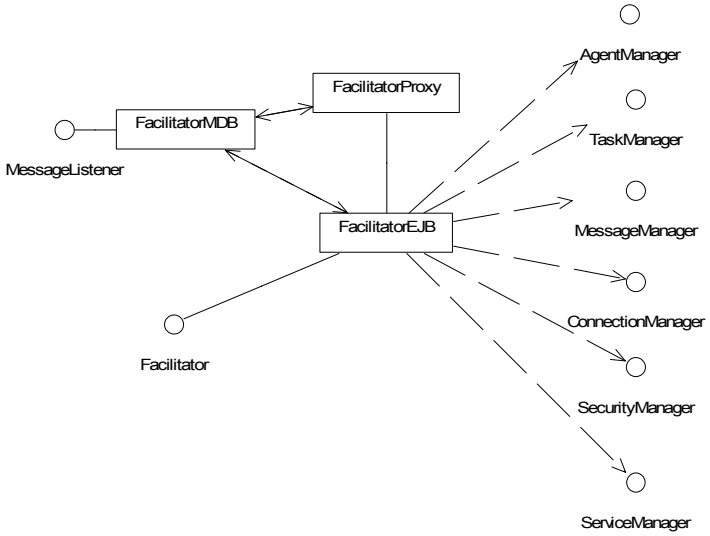

Fig. 3. The *Facilitator* component implemented as an EJB component

The *Facilitator* component is implemented as an EJB component (`FacilitatorEJB` class). This class is actually the *SessionBean* EJB component. The `FacilitatorEJB` class implements the `Facilitator` interface (defines the `Remote` interface, e.g. *business* methods available to users).

`FacilitatorProxy` class is a bridge between the client application and the *Facilitator* component. `FacilitatorMDB` is a *MessageDrivenBean* EJB component and is used for JMS communication between the *Facilitator* component and the rest of the system.

3.2. Agent Management Component

The *AgentManager* component manages agents. It actually represents agent directory and is used for agent allocation and release. It also controls the agent life cycle. Controlling the agent life cycle means creating and destroying an agent. This manager uses the EJB technology to implement agents. However, agents are not EJB components. Rather, they are common Java classes which are embedded into the EJB holder components – components designed just to hold agents. This approach is adopted because it would have restricted agents a great deal if they had been EJB components. One of the reasons for not having agents as EJB components is that agent mobility requires that agents should be able to migrate from one framework to

another. If an agent had been an EJB component, it would have been rather complicated to migrate it from one application server to another.

An example of embedding an agent into an EJB holder (the `AgentHolderBean` class) component is given in the following listing.

```
public Object getAgent(String agentClassName)
            throws AgentNotFoundException {
  // Create one AgentHolder.
  AgentHolder agentHolder = null;
  try {
    InitialContext context = new InitialContext();
    agentHolder = (AgentHolder)context.lookup(
                  "AgentHolderBean/local");
    agentHolder.init(agentClassName);
  } catch(Exception e)
    throw new AgentNotFoundException(
              "AgentManager error: " +
              e.getMessage());
  return agentHolder;
}
```

The listing above displays creation of a customized EJB component which is used to store an agent. When creating an EJB component, the agent class name is forwarded to the `init()` method of the `AgentHolderBean` class, as displayed in the next listing.

```
/** Agent assigned to this agent holder */
private Agent agent;
public void init(String agentClassName)
          throws CreateException {
  // Create an agent and store it.
  try {
    agent = loadClass(agentClassName);
  } catch (Exception ex) {
    ex.printStackTrace();
    throw new CreateException(
              "Could not create agent: "+
              agentClassName + ": " +
              ex.getMessage());
  }
}
```

This approach is useful when an agent moves from one framework to another. In that case, only the agent is moved to the destination and placed into the new agent holder. Figure 4 displays agent mobility scenario.

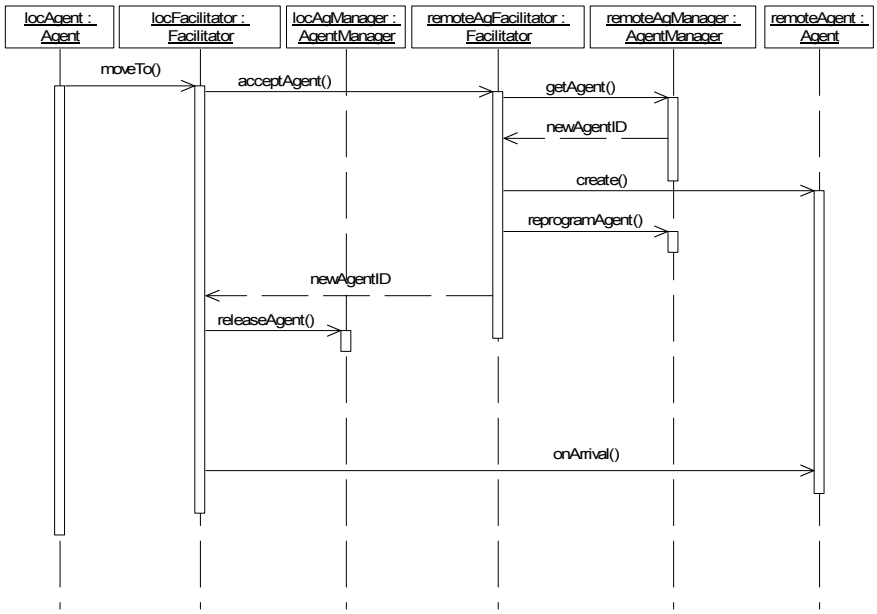

Fig. 4. Moving agents from one framework to another

Agent migration is done by checking availability with the destination facilitator. If the destination facilitator does accept the agent, it will create an agent holder and receive the serialized agent and place it into the holder (the `reprogramAgent()` method). All internal references to an agent are redirected to the new location. The agent as a Java class just needs to implement the `Agent` interface which extends the `java.io.Serializable` interface. This makes it possible that the agent is serialized and moved through the network.

3.3. Task Storage and Reporting Component

The *TaskManager* component manages tasks to be performed by the agent framework. It is realised through the class which implements the `TaskManager` interface. It also provides a way of notifying the client about the task execution progress.

Each task is stored in this component. When completed, it is removed from it. Tasks are instances of classes which implement the `AgentTask` interface. There are two types of task execution: programmatically or by sending a KQML message to the agent.

When executing a task programmatically, an instance of the task is created and forwarded to the *Facilitator* component. This component looks for the

appropriate agent and forwards the task to it by invoking the `execute()` method of the agent. When the agent completes the task, it returns the result as an instance of the `AgentResult` class. This result is sent to the client using the FacilitatorProxy component.

When executing a task by sending a KQML message to the agent, the client application sends the KQML message to the *Facilitator* component. This component looks for the appropriate agent and sends the message to it. When the task is completed, the agent replies to the original message and the message is forwarded to the client using the *FacilitatorProxy* component.

This agent framework uses the concept of *listeners* for communication between the client application and the framework. A listener is a Java class which has specialised methods to be invoked when the appropriate type of event occurs. In this case, there are four types of events:

1. *the job started* (when the agent has received the task),
2. *the job performing* (when the single step of the job is performed),
3. *the job completed* (when the whole task is performed) and
4. *the KQML message received* (the KQML message is sent to the client application).

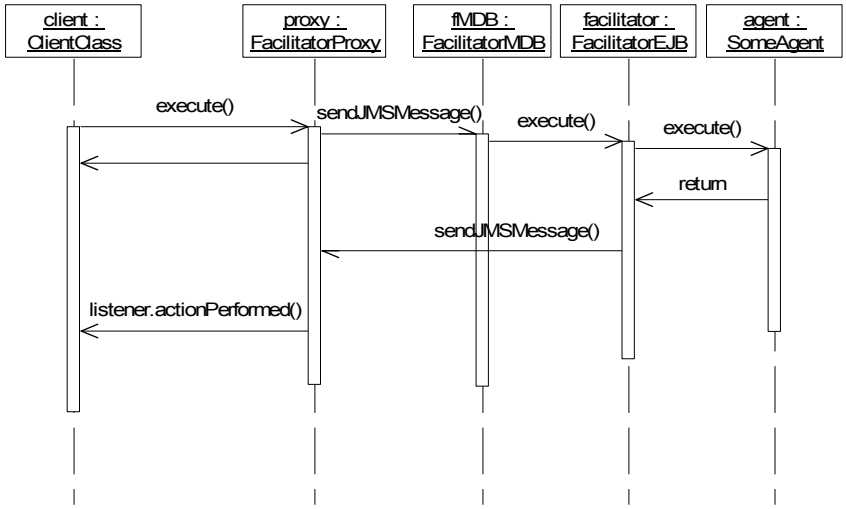

Fig. 5. Programmatically invoked task execution is done asynchronously

The first three types of events occur when the task is realised programmatically. A task can be done in a single step, in which case the first and the third messages appear in the given order. If a task needs more steps to be performed, *the job started* event is generated, then an arbitrary number of *the job performing* events is generated, and at the end, *the job completed* event is generated. The fourth type of event occurs when an agent sends a KQML message to the client. The concept of listeners provides asynchronous

task execution. This is due to the use of JMS as both the transport layer for the task and the result. JMS provides asynchronous message exchange which is in this case used to send tasks and receive results embedded in JMS messages.

Figure 5 displays the sequence diagram of programmatically invoked task execution.

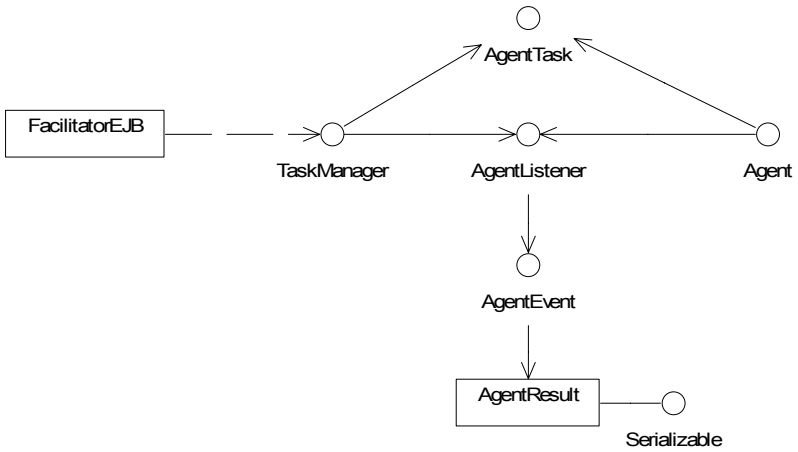

Fig. 6. Classes and interfaces used for the task execution

The client creates an instance of the task and forwards it to the *FacilitatorProxy* component. This component creates a JMS message and sends it to the `FacilitatorMDB` bean. This bean is a *Message-Driven EJB* bean and when it receives the message, it forwards it to the *Facilitator* component. The *Facilitator* component looks for the appropriate agent and sends the task to it. Before, during and after completion, appropriate JMS messages are sent from the *Facilitator* component to the *FacilitatorProxy* component and, as a result, appropriate listeners are invoked in the client application. Listeners hold the appropriate event class which implements the `AgentEvent` interface, and that class holds the result of execution in the `AgentResult` class, as displayed in the Figure 6.

3.4. Inter-facilitator connection component

The *ConnectionManager* component defines an inter-facilitator connectivity mechanism. This mechanism defines how separate facilitators form a network. Each facilitator is a node in this network and is automatically registered on the network at the initialisation time. This means that the programmer does not have to know the exact address of an arbitrary

facilitator and does not have to maintain the list of all available facilitators. Instead, the nodes are registered automatically and the list of all available facilitators is maintained automatically. If each agent framework is connected to a particular system, this automatically makes the network of those systems available (e.g. the library network by the use of the agent network).

One example of an agent network is displayed in the Figure 7. This network is organised as a hierarchical network of agent frameworks. Each node in this network is a single agent framework which is registered on the network at the initialisation time. This organisation provides dynamic network setup since all frameworks register during setup and unregister during shut down. All nodes in this organisation can access other nodes through their *ConnectionManager* components (Figure 8).

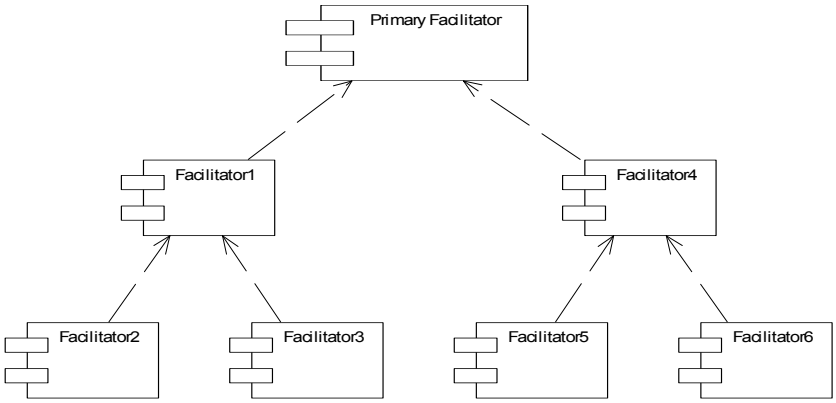

Fig. 7. Example of an agent framework network

When an agent wants to communicate to another agent which is not in the same facilitator, it does not need to know its exact location. The *ConnectionManager* component maintains the map of locations and agents. This means that only the ID of an agent is required for communication between them.

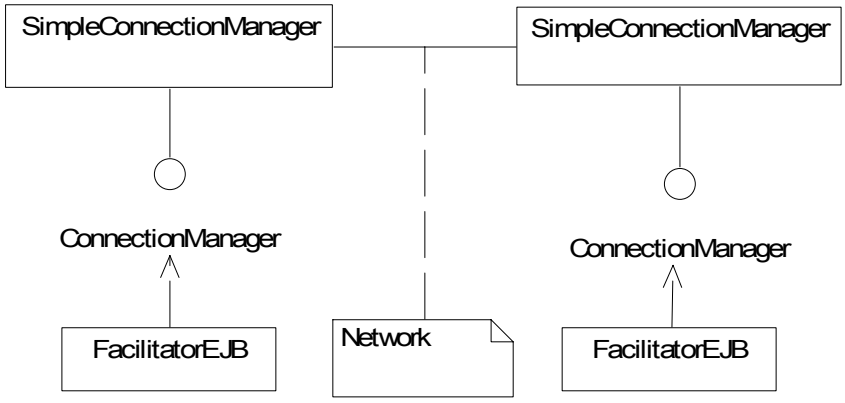

Fig. 8. *ConnectionManager* components provide links between frameworks

3.5. Security Component

The security subsystem is a very important element of an agent framework. The security subsystem provides data and agent code confidentiality and integrity during agent migration or data exchange. To provide for data and code integrity, it is necessary to apply cryptographic and digital signature mechanisms. Framework can encrypt and/or sign agent code before sending it to another agent framework. Also, the agent may use framework security service to encrypt and/or sign data sent to other agents. The access control segment of security subsystem insures integrity of data and code. It provides for integrity of data exchanged between agents and also protects agent framework from malicious agents. To protect a framework from malicious agents, it is necessary to use existing programming language security mechanisms (in the Java programming language, it is implemented in the `java.lang.SecurityManager` class). To establish access control, access control policies need to be defined. These policies contain permissions that allow (or forbid) agents to access different resources in the system.

The security subsystem can be programmed using proprietary solutions, but it is more convenient to use existing solutions provided by application servers. The lowest level of security support that can be used from application servers is user authentication and authorization. This feature provides application server components access control and, therefore, it also provides agent access control.

Most application servers support encrypted communication between clients and server components, as well as encrypted communication between application servers. Digital signatures and verification using certificates are also supported by some application servers. If security support in an existing application server does not exist or is not strong enough, it is possible to

apply proprietary cryptographic methods and plug them in the XJAF agent framework. To do so, it is necessary to implement the *SecurityManager* component of the framework. This component should be a class that implements the `SecurityManager` interface. This interface supports methods for data encryption/decryption, as well as methods for digital signatures/verification and also methods for enforcing access control. The Figure 9 shows an implementation of the *SecurityManager* component.

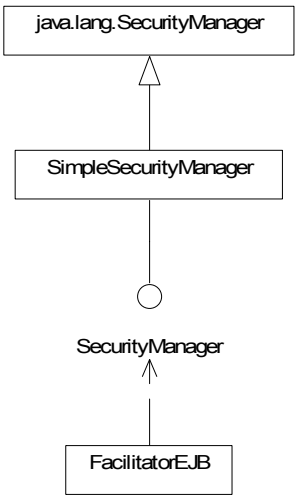

Fig. 9. An implementation of the *SecurityManager* component

The `SecurityManager` interface does not specify the security system. It merely lists all necessary methods to be implemented for full security support of XJAF. In the paper [38], an implementation of the security manager is given. It supports different algorithms for symmetric and asymmetric encryption, as well as digital signatures. Key management is based on PKI (Public Key Infrastructure) infrastructure. The system can use existing PKI implementations to provide key management functionality. Access control is based on the JAAS (Java Authentication and Authorization Service) system. The security policies used by XJAF's JAAS implementation can be kept in XML files or in the database.

By implementing the `SecurityManager` interface, the *SecurityManager* component provides for data and code integrity. However, to protect an agent framework from malicious agents, a different technique is required. Protection from agents consists of protecting the local resources and file system. This level of protection can be implemented by extending the `java.lang.SecurityManager` class. This class provides access control for accessing network resources, object creation, local file system, system attributes, clipboard, threads, etc. By extending this class and implementing

appropriate methods, it is possible to protect the agent framework from malicious agents as shown in [38].

3.6. Message Management

Communication between agents is done using KQML messages. KQML message exchange is managed by the *MessageManager* component. This component uses the JMS system as a low-level layer of communication. The Figure 10 shows how the JMS system is used for message exchange.

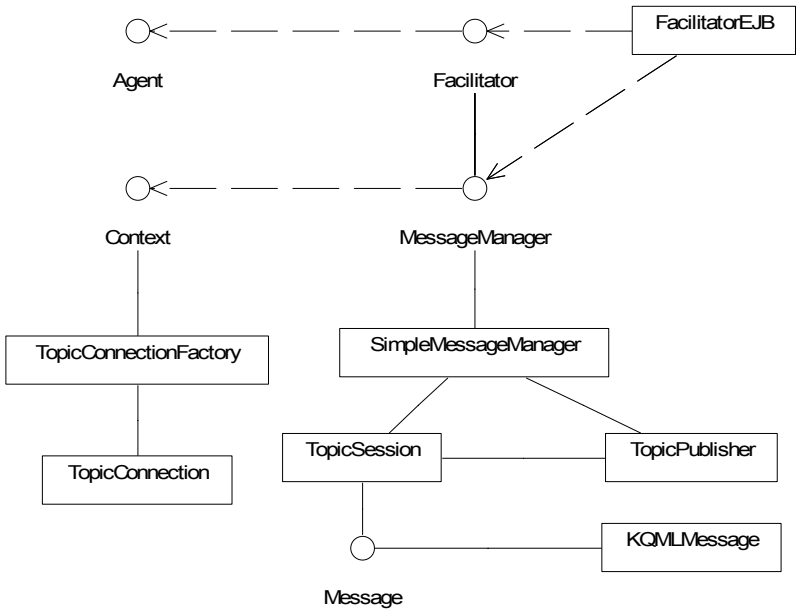

Fig. 10. The *MessageManager* component uses JMS for message exchange

When an agent sends a KQML message to another agent, it is embedded into a JMS message. The JMS message is sent to all agent frameworks subscribed to this service, but only the agent framework having the destination agent will receive the message and extract the KQML message from it. This KQML message is then sent to the agent by invoking the `onKQMLMessage()` method defined in the `Agent` interface. This interface is implemented by all agents in this framework.

3.7. Service Manager

The *ServiceManager* component implements the service directory subsystem. This component manages the set of services available to agents as shown in the Figure 11.

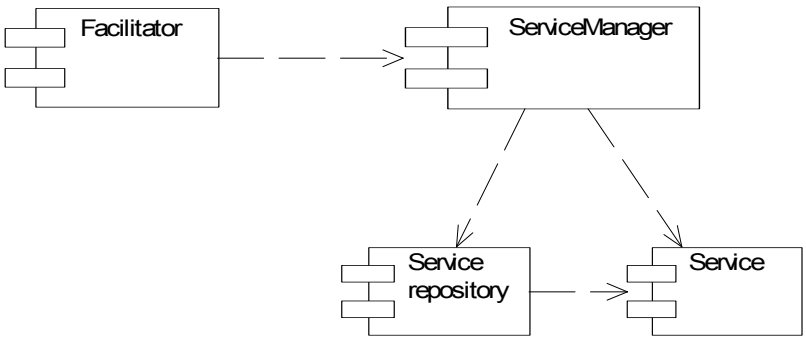

Fig. 11. An implementation of the *ServiceManager* component

The *ServiceManager* component includes the service repository which holds all available services. Services can be added, removed, searched and used. When the service is not needed anymore, it must be returned to the repository. Services are implemented as Java classes which implement the `Service` interface. This interface defines the single method to be implemented – the `return()` method which is invoked when the service is returned to the repository. This enables finalisation tasks to be performed when the service is returned to the repository.

3.8. Implementation

The framework was initially implemented on the two Java EE application servers – JBoss (v.4.0.2) and Orion (v.2.0.5). These application servers were EJB2.1 compliant. It was possible to implement some nodes in the agent network on the JBoss application server and other nodes on the Orion application server. The difference in deployment on two different application servers was in configuration files only. This proved that the XJAF agent framework can operate on different application servers. The current version of the XJAF agent framework is modified to fit the EJB3.0 version and implemented on the JBoss v.4.2.3 GA application server. Transfer to the EJB3.0 version gave the implementation much needed simplicity (offered by the EJB3.0 standard).

4. Two Applications on Distributed Library Catalogues

According to [43], there are several areas where intelligent agents might be used: mediation between the user and information system, virtual reference, automated serials processing, automated interlibrary loan processing, acquisitions and circulation. In addition, cataloguing, and online interactive tutorials are also areas where agents might be beneficial and reduce workload. Since search can be done in heterogeneous environments, it is particularly useful to use agents, since they can perform various tasks in different environments, while being able to communicate to each other and thus being able to refine the search. According to the [33], existing library systems must meet the following three requirements in order to be able to implement agent technology: the domain should be distributed, the system should consist of independent cooperating components, and the system should contain pre-existing or legacy applications. Two implementations presented in this paper (agent-based central catalogue and agent-based metadata harvesting) meet all three requirements.

In the field of the library information systems there is a need for the centralised catalogue in a library network, which would enable users to search through all library records from the network. It can be done in two ways: to create one centralised library records database, which is loaded from all the nodes in the network, or to create a virtual central catalogue, which enables search over those incorporated libraries.

The first solution has the following advantages over the second one: all the records are in a single database. Since all the data are in a single database, that kind of search is faster than the distributed search. However, if the central catalogue is off-line, then no search is possible. Also, if the resulting database is too big [16], then it is not suitable for storing into a single database.

Since the second solution is a distributed search, it corrects last two drawbacks of the centralised solution – the search is distributed over the network, and the load is balanced over the nodes. Also, each node can take the role of the central node, enabling the system to function even if the central node is off-line. The distributed search can be implemented using agent technology since agents can move over the network in order to gather library data. Also, they can communicate to each other or to agent frameworks in order to either transfer the gathered data or send search tasks to other agents. An agent framework offers an automated way of registering library nodes so that topology of the network is not maintained manually. The agent-based distributed search is presented in this paper as an implementation of the central catalogue for library record databases [30, 36]. The agent-based central catalogue implementation uses mobile agents which migrate over the library network collecting bibliographical data that satisfy the query issued at the central node.

In the field of the metadata harvesting [39], there is the need for distributed collection of data. The data to be collected is stored in various catalogues, which are distributed over the network. To collect such data, it is necessary to implement a kind of distributed software entities that would be able to find,

extract and deliver the data as a single result. All these tasks can be performed using agents, since agent infrastructure offers the possibility of discovering the appropriate agents, means of communication between those agents and services that are necessary for accessing different data providers. The agent-based metadata harvesting implementation with features mentioned above is presented in this paper. In this implementation, the central agent tries to find the appropriate agents in the NDLTD network and send them the task of gathering metadata. Partial results are gathered and presented as a single result.

4.1. Agent-based Central Catalogue

The agent-based central catalogue has been designed to enable users (mostly librarians) to be able to search for library records on the library network. Most library information systems have the local library record database. These library information systems can form library network that would incorporate all local library record databases. In this case, it would be necessary to implement the virtual central catalogue for library records that would enable search over those incorporated databases. Library record database search has been implemented using software agents on the XJAF agent framework. For each library server there is one software agent that is capable of searching the database. The query that has been issued to the central catalogue is distributed to all agents, and all of them are executing that query simultaneously. All query results are incorporated into one joint result that is presented to the central catalogue.

Agent implementation of the central catalogue is based on agents that are searching local library record databases. For each local database, there is one agent which will be used to search that database. The central catalogue query is forwarded to all agents and all of them perform database search simultaneously. Search results are gathered in the central catalogue and incorporated into the joint query result.

The Figure 12 displays the virtual central catalogue that consists of two local BISIS library servers. Each local library server is connected to the local agent framework (XJAF 1 and XJAF 2) which provides working environment for the library agents that are used to perform library search. This system incorporates all local library records into one virtual central catalogue.

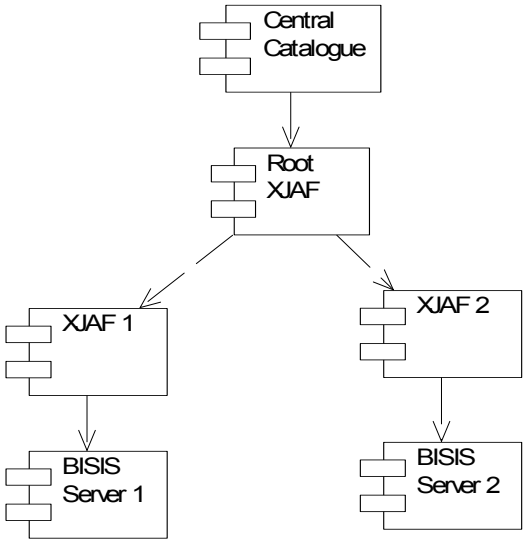

Fig. 12. Component diagram of the Central catalogue with two local library servers

The Figure 13 displays sequence diagram which illustrates agent search execution over the library network. Users post queries to the central catalogue. The central catalogue is connected to the central agent framework (the `AgentFramework` class) that holds library agents (represented by the `LibraryAgent` class). It requests the list of available library agents that will perform local database search. The list of available agents is passed to the central catalogue. It forwards the query to all available agents. The library agents start to migrate (the `moveTo` message) to the local agent frameworks where they will perform database search (using the `LibraryService` class that will receive the `executeQuery` message). Each agent sends the search result to the central catalogue (the `result` message). The central catalogue incorporates all received messages into one result (the `incorporateAllResults` message) and returns that result to the user. The Figure 14 displays the result of the search over the library network, displayed in the librarian application.

The library agent accesses the local BISIS server (the `BISISServer` class) using the library service (the `LibraryService` class). The library service represents a standardised system for accessing an arbitrary library information system. For the particular BISIS system, it has been adjusted to work with it. This means that the central catalogue is not tied to one particular type of library software. Instead, it enables creation of a heterogeneous library server network. Only one condition must be fulfilled: for each type of library software there must be an appropriate library service. This service will provide unified library record access that is independent of the library software type.

This also means that agents do not need to be adjusted to each type of library software, because library service comes between agents and library servers.

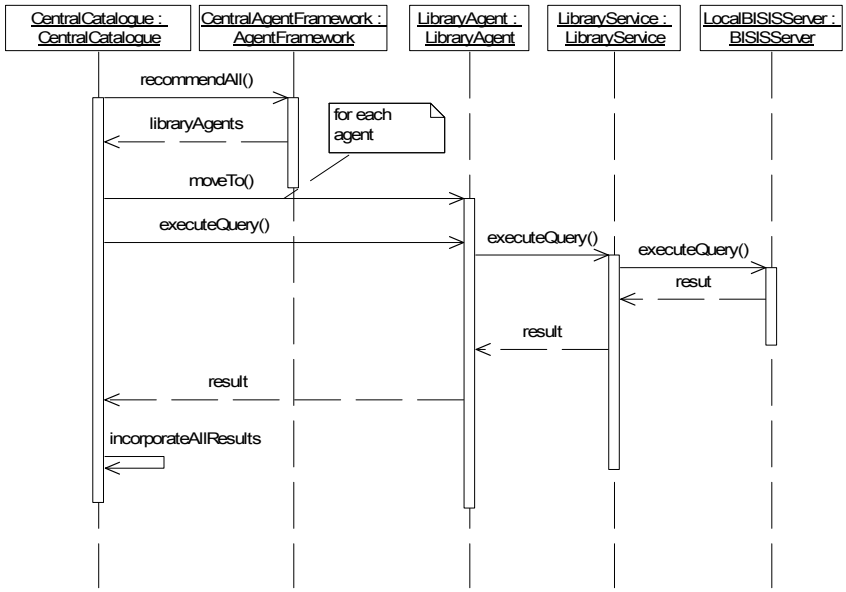

Fig. 13. Agent search execution sequence diagram

Main advantage of the central catalogue over the conventional solutions [30, 10] is the possibility to easily include heterogenous library information systems. This can be achieved by developing the appropriate agent service within the local agent framework. This approach allows code reusability since the same agent code can be used to search different library information systems.

An additional advantage is a dynamic setup of library nodes. It means that a node can be added or removed automatically, without the need to reconfigure existing topology. This is done via the XJAF framework.

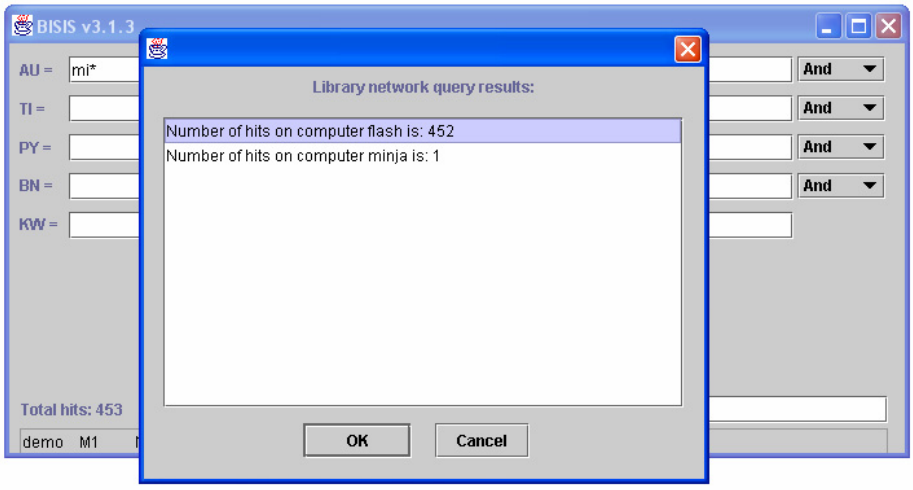

Fig. 14. Query results displayed in the librarian application

4.2. Metadata Harvesting Using Agents

The *Open Archive Initiative* (OAI) [39, 31] is an initiative for foundation of electronic archives, primarily archives of scholarly and scientific papers. In the context of OAI, the term archive means repository of stored information, mainly the repository of scholarly papers. The framework for data interchange is defined by the *Protocol for Metadata Harvesting* (PMH). OAI-PMH metadata harvesting is used as a standard protocol within the *Networked Digital Library of Theses and Dissertations* (NDLTD) [26]. NDLTD aims at building a digital library of *Electronic Theses and Dissertations* (ETD) authored by students of member institutions. Each member of OAI can have both or either of the following roles:

- *Data Providers*: systems which support the OAI-PMH as a means of exposing metadata and
- *Service Providers*: services which use metadata harvested via the OAI-PMH as a basis for building value-added services.

At the University of Novi Sad, the NDLTD implementation [11] was developed as a system for entering and searching electronic versions of theses and dissertations.

The XJAF agent framework is used to harvest metadata from providers. It is done by using XJAF agents, which are actually harvesting metadata in compliance with the OAI-PMH standard. One of the problems in metadata harvesting is the ability to maintain the network of providers, meaning that the network should be aware of the nodes that are added or removed without manual intervention. The approach in this paper is to create a network of

providers via XJAF agent facilitators. There are two distinctive cases: the first one in which it is possible to link a dedicated facilitator to a data provider, and the second case in which there are some data providers which cannot have dedicated facilitator linked. The provider which has a dedicated facilitator (named Local Data Provider) is automatically added to the network without any additional configuring. The providers which do not have dedicated facilitators linked (named Remote Data Providers) are added to the configuration file, maintained manually. Agents harvest metadata using the network of facilitators and corresponding Data Providers as displayed in the Figure 15.

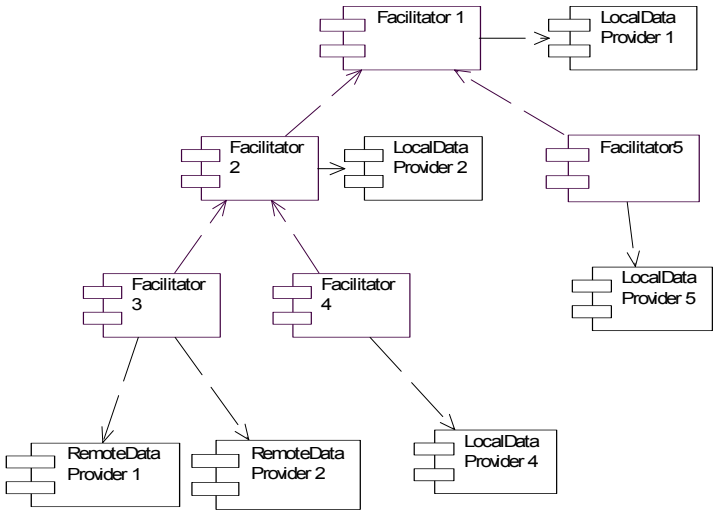

Fig. 15. Agent network forms OAI-PMH network

Each agent framework may have a corresponding *Data Provider* linked. If a *Data Provider* is linked to the agent framework, agents from that framework will search *Data Provider* as a local resource. If it is not linked, agents will be supplied with a list of *Data Providers* to be searched and harvested. In that case, the list of *Data Providers* must be maintained manually.

Agents use the specialised service to access metadata – *OAIPMHService*. This service communicates with the *Data Provider* under the OAI-PMH protocol which comprises both metadata search and acquisition. For communication between *Data Providers* and harvesters, the specialised web service was developed. *OAIPMHService* uses this web service in order to search and obtain metadata. The Figure 16 shows the sequence diagram of metadata harvesting using XJAF agents and the *OAIPMHService* service.

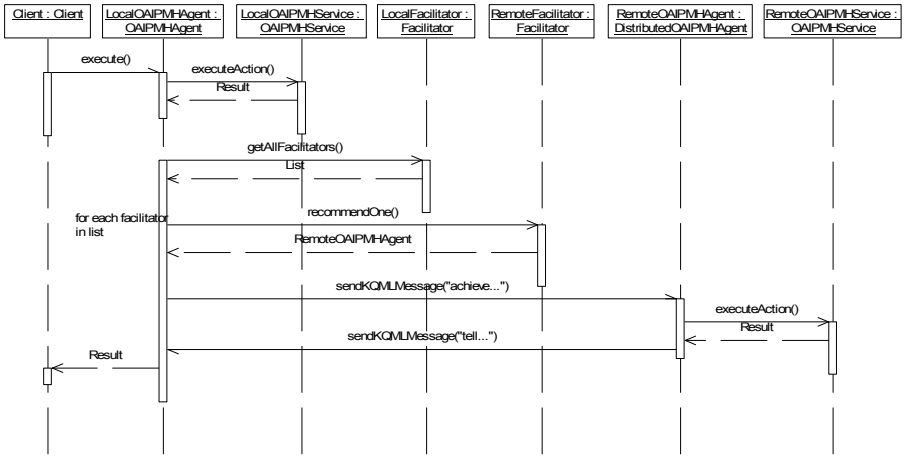

Fig. 16. Sequence diagram of the metadata harvesting

There are two types of agents: a main agent (*OAIPMHAgent*) and a subordinated agent (*DistributedOAIPMHAgent*). The *OAIPMHAgent* agent is designed to receive metadata harvesting task and search the local *Data Provider*. In addition, it engages *DistributedOAIPMHAgent* agents at each agent network node. *DistributedOAIPMHAgent* agents search corresponding *Data Providers* (attached to their local facilitators), gather results and send them to the invoking agent.

There are two types of *DistributedOAIPMHAgent* engagement: local engagement and remote engagement. The former is applied to *Data Provider* with dedicated facilitator. The latter is applied if there is no dedicated facilitator attached to *Data Provider*. In this case, the agent receives the list of *Data Providers* which will be searched and harvested.

The *OAIPMHAgent* agent engages *DistributedOAIPMHAgent* by sending the KQML message containing the task formulation. The distributed agent does the metadata harvesting, gathers the results and sends them to the invoking agent as a KQML message. *OAIPMHAgent* gathers all the results and sends them to the client.

There are two main advantages over the conventional solutions [27]:

1. it is a simple mechanism for automatic dynamic forming of the data provider network;
2. it is a convenient mechanism for building service provider architecture, because the framework itself provides for network interconnection feature, so services and agents can indulge in data processing only – they do not need to deal with interconnection.

Consequently, the source code is considerably smaller. For example, the code producing the same functionality for the agent-based solution is 7 KB, while the code for the conventional solution [2] is 46 KB.

5. Conclusion

This paper describes the agent framework XJAF developed by the authors and its implementation to distributed library catalogues. The framework is based on the Java EE technology. It uses the plug-in technology which provides additional flexibility, since it allows components to be substituted by others without rebuilding the whole framework. XJAF is compliant with the FIPA specification and supports the following concepts: message exchange, agent mobility, security and agent and service directories. Also, this framework proposes additional component of the agent framework: the inter-facilitator connectivity component which defines how separate facilitators form a network. The XJAF framework has the following characteristics: pluggable managers, the inter-facilitator connectivity implementation (*Communication Manager* component) and enhanced security which is implemented in the *SecurityManager* component. XJAF, although based on the Java EE technology, does not depend on the application server that is used. On the higher level of abstraction, this extensible agent framework is not tied to the Java EE technology – it can be implemented in any distributed components framework (.NET, CORBA, etc.).

XJAF has been applied to two particular applications in the field of distributed digital library catalogues: the virtual central catalogue, and metadata harvesting. Two different features of agents have been exploited in those two applications: mobility and agents inter-relationship. In the virtual central catalogue, agents migrate from one node to another searching for the contents specified by the query issued from the central node. In case of metadata harvesting, one central agent delegates tasks to agents in subordinate nodes and collects results.

In both cases, use of the agent framework gave a simple and effective mechanism for dynamic setup of the distributed catalogues network.

The main contribution of this paper is a solution which gives a simple and flexible mechanism for automatic maintaining of the dynamically changing network, as well as an environment suitable for implementation of additional services in distributed libraries.

The future work will be directed towards utilization of the XJAF for value-added services implementation in distributed libraries, interoperability among different platforms and improvement of security.

6. References

1. Aglets Home Page, http://www.trl.ibm.com/aglets/ (current October 2009)
2. Arc harvester and search engine, http://sourceforge.net/projects/oaiarc/ (current October 2009)
3. Bellavista P., Corradi A., Tomasi A., "The mobile agent technology to support and to access museum information", Proceedings of the 2000 ACM symposium on Applied computing 2000, Como, Italy, ISBN:1-58113-240-9, pp. 1006-1013. (2000)

4. Bellifemine F., Poggi A., Rimassa G., "JADE – A FIPA-compliant agent framework", Proceedings of Practical Applications of Intelligent Agents (PAAM'99), London, pp. 97-108. (1999)
5. Bigus J., "The Agent Building and Learning Environment", White Paper, http://www.alphaworks.ibm.com/tech/able (current October 2009)
6. Binder W., Roth V., "Secure mobile agent systems using Java: where are we heading?", Proceedings of the 2002 ACM symposium on Applied computing, Madrid, Spain, ISBN:1-58113-445-2, pp. 115-119. (2002)
7. Blixt K., Öberg R., "Software Agent Framework Technology", MSc thesis, Linköping University, Department of Computer and Information Science (2000)
8. Brugali D., Sycara K, "Towards agent oriented application frameworks", ACM Computing Surveys (CSUR),Volume 32 , Issue 1, ISSN:0360-0300, pp. 21-27. (2000)
9. Chauhan D., Baker A., "JAFMAS: a multiagent application development system", Proceedings of the second international conference on Autonomous agents, Minneapolis, Minnesota, United States, ISBN:0-89791-983-1, pp. 100-107. (1998)
10. COBISS Library Information System, http://www.cobiss.si/cobiss_eng.html (current October 2009)
11. DIGLIB home page, http://www.diglib.ns.ac.yu/frontOffice/index.jsp?newLang=en (current October 2009)
12. d'Inverno M., Luck M., "Development and Application of a Formal Agent Framework", In Proceedings of the First IEEE International Conference on Formal Engineering Methods, Hiroshima, Japan, pp. 222-231. (1997)
13. Ferber J., "Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence", Addison-Wesley Pub. Co., ISBN:021360489. (1999)
14. FIPA Home Page, http://www.fipa.org (current October 2009)
15. He Q., Sycara K., Finin T., "Personal Security Agent: KQML-Based PKI", Proceedings of the second international conference on Autonomous agents, United States, Minneapolis, pp. 377-384. (1998)
16. Jacobs A., The Pathologies of Big Data, Communications of the ACM, Volume 52, Issue 8-9, pp. 36-44, http://queue.acm.org/detail.cfm?id=1563874 (2009)
17. Java Agent Framework Home Page, http://mas.cs.umass.edu (current October 2009)
18. Java Agent Template Home Page, http://java.stanford.edu (current October 2009)
19. Kautz H. A., Selman B., Coen M., "Bottom-up Design of Software Agents", Communications of ACM, Volume 37, Issue 7, pp. 143-147. (1994)
20. Kendall E, Krishna M., Suresh, C., Pathak C., "An application framework for intelligent and mobile agents", ACM Computing Surveys (CSUR), Volume 32 , Issue 1, Article No. 20, ISSN:0360-0300. (2000)
21. Kim Tan H., Moreau L., "Certificates for mobile code security", Proceedings of the 17th symposium on Proceedings of the 2002 ACM symposium on applied computing, Madrid, Spain, ISBN:1-58113-445-2, pp. 76-81. (2002)
22. Lyell M., "Interoperability, standards, and software agent systems", 23'rd Army Science Conference, Orlando, Florida, USA, December 2-5 (2002)
23. Maamar Z., Moulin B., "An Overview of Software Agent-Oriented Frameworks", ACM Computing Surveys, Volume 32, Issue 1, Article No. 19, ISSN:0360-0300. (2000)
24. Nardi B., Miller J., Wright D., "Collaborative, programmable intelligent agents", Communications of the ACM, Volume 41 , Issue 3, pp. 96-104. (1998)
25. Network of Excellence on Digital Libraries, http://www.delos.info/ (current October 2009)

26. Networked Digital Library of Theses and Dissertations Homepage, http://www.ndltd.org (current October 2009)
27. Open Archives Initiative Tools, http://www.openarchives.org/pmh/tools/tools.php (current October 2009)
28. Palaniappan M., Yankelovich N., Fitzmaurice G., Loomis A., Haan B., Coombs J., Meyrowitz N., "The envoy framework: an open architecture for agents", ACM Transactions on Information Systems, Volume 10 , Issue 3, ISSN:1046-8188, pp. 233-264. (1992)
29. Rusbridge, C., "Realizing the Hybrid Library", *D-Lib Magazine*, Vol. 4 No. 9, http://www.dlib.org/dlib/october98/10pinfield.html (current October 2009)
30. Surla, D., et. al., "Distributed library information system BISIS", Group for Information Technologies, ISBN: 96-7444-006-1. (2004)
31. The Open Archives Initiative Protocol for Metadata Harvesting: www.openarchives.org/OAI/openarchivesprotocol.html (current October 2009)
32. University of Michigan Digital Library, http://www.si.umich.edu/UMDL/ (current October 2009)
33. Valeda F. Dent, "Intelligent agent concepts in the modern library", Library Hi Tech, Vol. 25, Issue 1, pp. 108-125, http://lefty64.scc- net.rutgers.edu/dlr/ TMP/rutgers-lib_23854-PDF-1.pdf (current October 2009))
34. Varadharajan V., "Security enhanced mobile agents", Proceedings of the 7th ACM conference on Computer and communications security, Greece, Athens, pp. 200-209. (2000)
35. Vidaković M, "Extensible Java-based agent framework", PhD thesis, Faculty of Engineering, University of Novi Sad, Novi Sad, http://diglib.ns.ac.yu/ndltd/docs/set2/ndltd344/MinjaDoktorat.pdf (current October 2009)
36. Vidaković, M., Konjović, Z., "One Implementation of Central Catalogue for Local Library Record Databases", INFOTHECA: Journal of Informatics and Librarianship, Volume 5, Issue 1-2, pp. 113-119, http://www.unilib.bg.ac.yu/ bibliotekarstvo/infoteka/1_2-2004/RESENJE%20CENTRALNOG%20KATALOGA%201.pdf (current October 2009)
37. Vidaković, M., Konjović, Z., "EJB Based Intelligent Agents Framework", Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002), Cambridge, USA, November 4-6, pp. 343-348. (2002)
38. Vidaković, M., Sladić, G., Konjović, Z., "Security Management In J2EE Based Intelligent Agent Framework", Proceedings of the 7th IASTED International Conference on Software Engineering and Applications (SEA 2003), Marina Del Rey, USA, November 3-5, pp. 128-133. (2003)
39. Vidaković, M., Sladić, G., Zarić, M., "Metadata Harvesting Using Agent Technology", Proceedings of the 8th IASTED International Conference on Software Engineering and Applications (SEA 2004), Cambridge, USA, November 9-11, pp. 489-493. (2004)
40. Vuong S., Fu P., "A security architecture and design for mobile intelligent agent systems", ACM SIGAPP Applied Computing Review archive, Volume 9 , Issue 3, pp. 21-30. (2001)
41. Wilson L., Burroughs D., Sucharitaves J., Kumar, A., "An agent-based framework for linking distributed simulations", Proceedings of the 32nd conference on Winter simulation, Orlando, Florida, ISBN:1-23456-789-0, pp. 1713 – 1721. (2000)

42. Yuh-Jong H., "Some thoughts on agent trust and delegation", Proceedings of the fifth international conference on Autonomous agents, Canada, Montreal, pp. 489-496. (2001)
43. Zick, L., “The work of information mediators: A comparison of librarians and intelligent software agents”, *First Monday*, Vol. 5 No. 5, pp. 1-14 (2000)

Milan Vidaković is holding the associate professor position at the Faculty of Technical Sciences, Novi Sad, Serbia. He received his PhD degree (2003) in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 1998 he has been with the Faculty of Technical Sciences in Novi Sad. Mr. Vidaković participated in several science projects. He published more than 60 scientific and professional papers. His main research interests include web and internet programming, distributed computing, software agents, and language internationalisation and localisation. He can be contacted at: minja@uns.ac.rs.

Branko Milosavljević is holding the associate professor position at the Faculty of Technical Sciences, Novi Sad, Serbia. He received his PhD degree (2003) in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 1998 he has been with the Faculty of Technical Sciences in Novi Sad. Mr. Milosavljević participated in several science projects; in one he was the project leader. He published more than 70 scientific and professional papers. His main research interests include library information systems, document management, multimedia retrieval, and access control. He can be contacted at:mbranko@uns.ac.rs.

Zora Konjović has been holding the full professor position at the Faculty of Technical Sciences, Novi Sad, Serbia since 2003. Mrs. Konjović received her Bachelor degree in Mathematics from the University of Novi Sad, Faculty Science in 1973, Master degree in Robotics from the University of Novi Sad, Faculty of Technical Sciences in 1985, and Ph. D. degree in Robotics from the University of Novi Sad, Faculty of Technical Sciences in 1992. From 1973 till 1980 she was with the Faculty of Science in Novi Sad, and since 1980 she has been with the Faculty of Technical Sciences, University of Novi Sad. Mrs. Konjović participated in 5 scientific and more than 30 professional projects; in 5 she was the project leader. She published more than 150 scientific and professional papers. She is the corresponding author and can be contacted at: ftn_zora@uns.ac.rs

Goran Sladić is a teaching assistant and PhD student at the Faculty of Technical Sciences, Novi Sad, Serbia. His research interests include access control, context-aware computing, XML technologies, document management systems and workflow systems. Mr. Sladić received his Bachelor degree (2002) and Master degree (2006) both in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. He participated in 6

science projects. Mr. Sladić published 25 scientific and professional papers. He is a member of the ACM. He can be contacted at: sladicg@uns.ac.rs

Received: December 01, 2008; Accepted: November 11, 2009.