Mathematical Problems of Computer Science 30, 76--86, 2008.

Agent Interaction Protocols in an Intelligent Agent Server System

Levon H. Aslanyan, David A. Karapetyan

Institute for Informatics and Automation Problems of NAS of RA
e-mail: lasl@sci.am, david@dm-lab.sci.am
url: http://dm-lab.sci.am

Abstract

We study interaction protocols of software agents in an Intelligent Agent Server system, which employs software agents in regard to different applied problems. Four models of agent interaction protocols are proposed. The protocols are evaluated for utility, implementation and applicability. The logical level of system is designed and implemented algorithmically. The test application is the intrusion detection problem.

1. Introduction

The main goal of the research project NetInt (Networked Intelligence) is the design of a distributed application software system to operate in computing networks. The use of software mobile and intelligent agents enables the system to solve a variety of applied problems, such as network management and maintenance, network dynamic optimization and security control. Agents' communication and interaction become a major technical issue of such systems. Agents, which behave autonomously, change their locations. New agents are appearing and others may stop their functioning. Proper communication in this case requires a complete algorithmic model. Similar to this is the known PKI system for security. NetInt is a complex mobile environment which is under the control of a set of servers, where the security reasons are analysed and implemented by means of practical cryptography. The communication system provides functionality, related to data bases (sniffing, log files) and data mining type of analysis and decision support. Typical applications considered are the network management issues and intrusion detection into the systems[1].
NetInt is an extension of SPARTA (Security Policy Adaptation Reinforced Through Agents) system designed within the 5th Framework Programme (FP5) of European Community Framework Programme for Research, Technological Development and Demonstration, 2000-2001.

2. System architecture

The system is basically organized as follows: NetInt agent platforms, or more simple Agent servers, are installed on a number of computers binded together to organize a network. The computers on which the servers are installed are called nodes. Agent servers produce agents as

well as permit and manage agents' access to system resources and their utilization. Mobile agents travel from one host to another and perform the thread of executions. They may assemble and swap information, analyze it and later interchange the analysis results. In this way they try to synchronize the system parameters, reveal pathologies in the system and try to eliminate them. NetInt agent environment is a dialogue system, implemented in Java programming language, which supports transferability of software code across any Java Virtual Machine containing operating system.

Fig. 1 outlines NetInt agent-based system implemented in OMG MASIF standard[2].

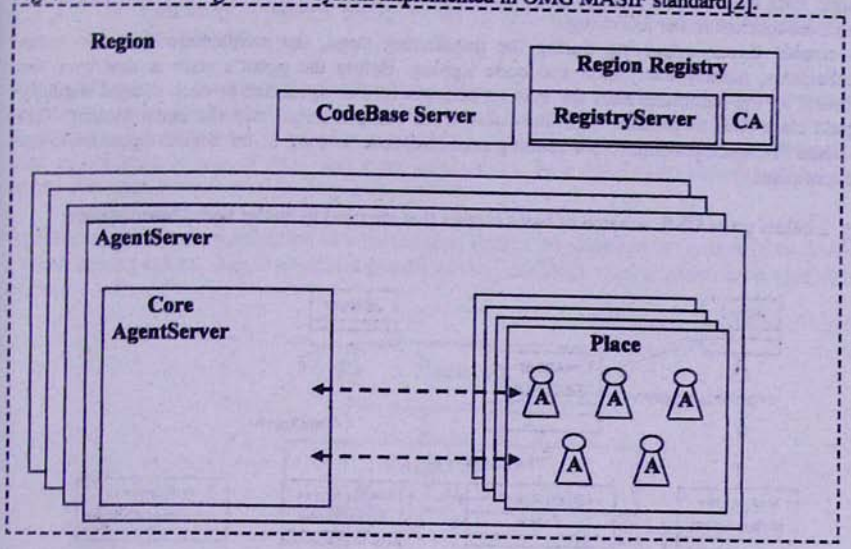

Fig. 1 An outline of functionality of NetInt agent-based system

NetInt agent system involves 3 subsytems:

a. ***AgentServer (AS),*** - the server is basically intended to provide tools to create, run, receive and transfer agents.

b. ***RegistryServer (RS)*** - registration subsystem, where several service delivering subsystems and their components are registered. This subsystem also provides information on necessary agents and agent subsystems. It incorporates internal Certificate Authorities (CA) center as well.

c. ***CodeBaseServer (CBS)*** - this server is mainly designed to provide agent software codes. Whenever there is a need to run an agent, the system seeks the agent software code on the local computer. In case the code is not found, the system requires the code from **CodeBaseServer[3].**

A number of different communicators may exist, that support different transfer technologies (e.g. plain socket connections, RMI or Email). Each Home and Agent Server has at least one communicator present, which the agent can use for jumping. The agent itself does not necessarily know about the mode of transport and most of the time will not be informed. When an agent state has been successfully transferred, the agent's code is loaded from the agent's code base. The code base itself consists of all locally available classes and references to available Code Base Servers. When code for an agent is locally available, it is taken from there, otherwise loaded from a CBS. Sending an agent over the network has a number of security implications, which are touched in the following.

To combat threats appearing during the transferring stage, the architecture uses two main mechanisms, namely encryption and code signing. Before the agent's state is sent over the network by the communicator, the system encrypts it. The agent server only accepts digitally signed class files to prevent malicious code from being inserted into the agent system. This prevents the agent platform from running modified code as long as the digital signature is not compromised.

Fig. 2 below gives UML scheme of basic classes that are used in NetInt agent-based system.

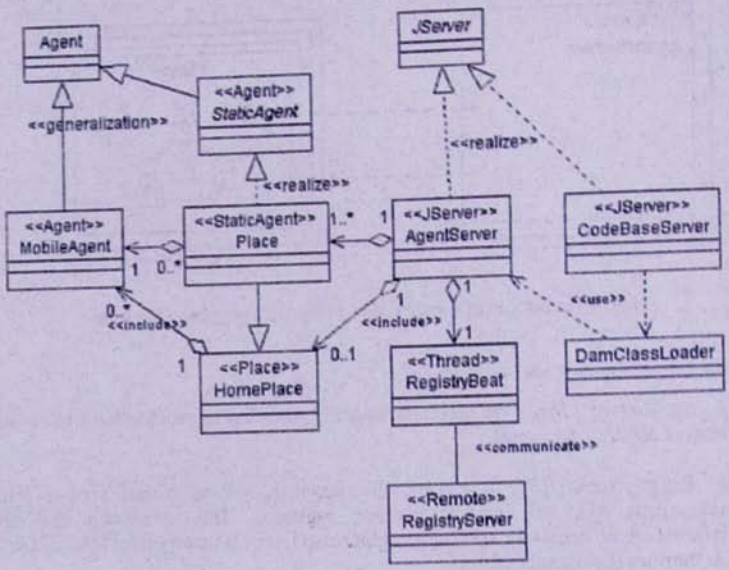

Fig. 2 UML scheme of basic classes used in NetInt agent-based system

Sub-environments in the systems interact via Java RMI communication facility. The latter allows a more effective usage of object-oriented programming (OOP) resources.

Software agents, in accordance with the base concept, should posses the following characteristics to ensure the proper functioning of the whole system:

- **Autonomicy** – agen't ability to act by autonomous, i.e. without meddling of other person or program;
- **Mobility** – agent's ability to travel within the network to search information necessary for task execution;
- **Interoperability** – equal possibilities to interoperate between various software agents;
- **Liability** – the ability of an agent to perform the thread of execution for which the agent is liable for;
- **Flexibility** – agent's ability to act in response to the changes of execution environment

Agent interaction is the major feature that we address when we describe an agent community. Interaction means establishment a form of two-way dynamic communications between two or more agents, trying to reach a mutually acceptable agreement. The term *interaction protocol* is used in reference to sets of rules that guide interactions. In a simple interaction protocol the agents elaborate, accept, or reject proposals.

Agent interaction and coordination in a multi-agent system are based on procedure of exchanging packets among agents. Fig. 3 depicts a pattern of data exchange among agens in an agent-based system:

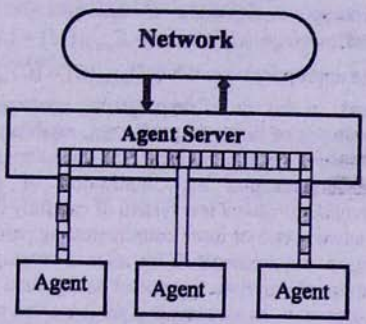

Fig.3 Data exchange diagram in an agent-based system

Data package transferring emerges in situations, when an agent tries to communicate to another agent or to the system. Communication is to be differentiated into the internal (in a local computer and in one Agent Server), local (in a LAN), and global (in a TCP/IP area). In the process of these connections sender determines the address of the counterpart whom it chooses to send a message: it establishes a session with the intended one and sends it a generated packet. In other cases communication needs to apply to the Registry Server.

In the following we address the technical aspects of problems dealing with establishing communications and data exchange between two participants. They can be categorized into that type of problems, which are concerned with achieving secure communications in a medium which is untrustworthy and subject to tampering by potential intruders. These interactions and the consequent need for security, regarding a range of security features and levels, vary widely from application to application. Such issues will be the focus of this study in a context of communication security, which in turn is an issue related to agent community protection.

Within the context of agent-to agent (global) communication, we suppose to obey the general security requirements, including:

- **Confidentiality** – Ensuring that no communication between two parties is revealed to the third party, i.e. that no one can read the message except the intended receiver.
- **Authentication** – The process of proving one's identity. The recipient of a message should be able to securely authenticate its origin; moreover the third party (the intruder) won't be able to act with another name.
- **Integrity** – Assuring the receiver that the received message has not been altered in any way from the original. during transmission, i.e. the intruder would not be able to trap or fake the message.
- **Nonrepudiation** – The sender of message should be unable to deny having sent the message, i.e. the sender should be able to prove that the message is precisely the one issued by it.

Further we shall consider generic public-key encryption algorithms. They basically employ a pair of keys for each participant: one of the keys is designated the *public key* and may be advertised as widely as the owner wants. The other key is designated the *private (secret) key* and is never revealed to another party. It is used to decrypt messages. Suppose participant A wants to send a message to B. Denote A's public key by $pk(A)$ and secret key by $sk(A)$. A message M is encrypted first, by encryption algorithm E that uses the message M and the public key $pk(A)$. The encrypted message (cipher text) $E_{pk(A)}\{M\} = \{M\}_{pk(A)} = C$ is decrypted with the algorithm D using the appropriate secret key $D_{sk(A)}\{C\} = \{C\}_{sk(A)} = M$.

There are several problems in the use of cryptographic systems, such as the problem of key distribution and secure transfer of keys via public net, establishment of intact communication sessions, etc. The crucial role in performing these actions is assigned to keys, however still not a little part has precise organization and application of communication protocols. A *communication /cryptographic/ protocol* is a system of carefully defined actions designed so that it provides interactions between two or more communicating parties according to one or another set of functional /encryption/ requirements. If we refer to cryptographic algorithms as algebraic and logical units that require appropriate theoretical background for proving their security, then cryptographic protocols present to be systems that are subjected to logical analysis to prove their security against malicious actions of the third party. To do these things, we need to make use of appropriate formalisms, such as model checking and analysis. It is easier to detect possible attacks a priori knowing their scenario and having the protocol secureity proof against the attacks. Security provision of a Protocol requires checking for occurence of all events, which is a time and resource consuming procedure. CSP (communicating sequential processes) and FDR (failures-divergence refinemen) prove to be fine tools to implement such analysis, where the first is an appropriate tool to create formalism, and the second to perform global analysis. CSP is used below in analysis of synthesised protocols, and some known results by FDR are taken into account.

3. An outline of Crypto protocols

We shall now schematically explore several cryptographic protocols:

1. *Simpe communication protocol:*

Suppose A knows the public key $pk(B)$ of B and wants to communicate a message M to B:

$A \rightarrow B : \{M\}_{pk(B)}$ *(-> means sending (address : message))*

($\{M\}_{pk(B)}$ means M encrypted by the key $pk(B)$)

a. A encryptes message M as $\{M\}_{pk(B)} = C$ and sends it to B.

b. B decryptes the encryted text C using his private key $sk(B)$.

2. *Public-key distribution protocol:*

$A \rightarrow CA : B$

$CA \rightarrow A : \{pk(B)\}_{sk(CA)}$

A is willing to obtain B's public key from Certificate Authorities (CA) center.

a. A sends B's name to CA's database, which implies that A needs to get B's public key.

b. CA encryptes $pk(B)$ by its secret key $sk(CA)$, i.e. signs it and sends to A.

c. A decryptes the message from CA using CA's key $pk(CA)$.

Several options are available here. B can send A its public key on his own initiative, i.e. B can send the key signed or encrypted, if it knows A's public key.

For generality we shall also present key pair distribution protocol performed at session set-up. Since at this stage the channel is completely unsecure and the only known fact is CA center's public key, then distribution of the key pair could be performed as follows:

Suppose A needs to receive an encryption key pair from CA.

$A \rightarrow CA : A$

$CA \rightarrow A : \{\{pk(A), sk(A)\}_L\}_{sk(CA)}$

a. A requests the key pair from CA.

b. CA generates the public and secret key pair $pk(A)$ $sk(A)$, encrypts it with any key L of length n and sends it to A signed with his secret key $sk(CA)$.

c. A is assumed to know the key L, and recovers its key pair from the encrypted message upon its receipt.

3. *Session set-up protocol:*

$A \rightarrow B: \{k\}_{pk(B)}$

Suppose again that A has learnt (in a way) B's public key $pk(B)$:

a. To initate a session A generates a key k at random to be used with a symmetric cryptographic algorithm. Then A encrypts k with $pk(B)$ and sends it to B.

b. B recovers its session key from the encrypted message.

c. Both parties encrypt messages during the session using their symmetric encryption key which is already known to each of them.

4. Interaction protocols in multi-agent systems

The above-mentioned cryptographic protocols (there are a greatly many such protocols) are exploited in agent-based systems to acheive secure data exchange between agents. They guide and manage every interaction between agents. We propose interaction protocols that provide secure communications between agents, agent servers in a multi-agent system, based on the cryptographic protocols mentioned above. A commentary on each protocol follows:

1. After loading, *AS* agent server addresses to registry server *RS* for registration and to receive its public and secret key pair. The protocol proceeds as follows.
Suppose that the public key *pk(CA)* of *CA* and its general properties are known and accessible to agent community.

AS -> CA : AS
CA -> AS : {[pk(AS),sk(AS)],L}$_{pk(CA)}$

a. *AS* asks *CA* for the key pair.
b. *CA* generates the public and secret key pair $pk(AS)$ $sk(AS)$, encrypts it with the key *L* of length *n* and sends it to *AS* signed with his secret key $sk(CA)$.
c. *AS* recovers its key pair by decrypting the received message with the keys *L* and $pk(AC)$.

2. *Associating or binding an encryption key pair* to agent *A* by AgentServer (AS) at agent creation stage.

AS -> CA : A
CA -> AS : {pk(A), sk(A)}$_{pk(AS)}$
AS -> A : {pk(A), sk(A)}

a. *AS* asks *CA* for receiving *A*'s key pair.
b. *CA* generates key pair *pk(A), sk(A)* and encrypes it by using *AS*'s public key, and sends it encrypted back to *AS*.
c. *AS* recovers the key pair from the encrypted message and passes it to agent *A*.

3. *Session set-up protocol between agents A and B*
Suppose *A* is a registered agent in *AS* agent server and is willing to establish session with agent *B*, having no knowledge of its public key and location.

A -> AS : LOC(B)
AS have B (case of internal communication of agents)
AS -> A : {LOC(B)}
A -> B : {k}
B -> A : {k}
SESSION
Else (global communication)
AS -> CA : LOC(B) (at this stage LOC(B) is just a text string as we see it)
CA -> AS :{LOC(B), pk(B)}$_{sk(CA)}$
AS -> A : {LOC(B), pk(B)}
A -> B : {k}$_{pk(B)}$
SESSION
END

a. *A* asks *AS* for B's address.
b. Agent server *AS* checks whether *B* is located in its environment, if yes, then it sends this address to *A*.
c. *A* sends *B* a key k generated at random which is referred to as the session key.
d. B decrypts the message and gets the session key.
e. Both parties encrypt messages using symmetric encryption key constructed at the session set-up, which is already know to each of the parties.

No encryption problem is created in this case, since messages are exchanged within a certain system i.e. massages do not go through the non-secure network, consequently no third party can reveal their secrets

We now consider the case when agent B is located on another agent server, denoted as AS_1.

a. Agent A asks AS for B's address.
b. AS checks agent B's location and if it is not located within its environment, runs public-key distribution protocol to ask registry server (RS) for B's public key and address.
c. RS creates a massage containing B's address and its public key, signs it with secret key $sk(CA)$ and sends to AS.
d. AS decrypts the message and sends its contents to A.
e. A generates a key k (session key) at random, encrypts it with $pk(B)$ and sends it to B.
f. The same steps a.b.c.d. are performed by B to learn A's address from agent server AS_1.
g. B generates its own session key k_1 and sends it together with the key k, previously sent by A, back to A encrypted with $pk(A)$.
h. Upon receipt A decrypts the message and recovers the session key k_1, which sends back to B.
i. Both parties authenticate each other and encrypt messages using symmetric encryption key pair k, k_1 established at the session set-up, which is already known to each of them.

4. *Agent A from agent server AS to agent server AS_1 transfer protocol*

$A \rightarrow AS: MOV(AS_1)$
$AS \rightarrow CA: LOC(AS_1)$
$CA \rightarrow AS: \{LOC(AS_1), pk(AS_1)\}_{sk(CA)}$
$AS \rightarrow AS_1: \{k\}_{pk(AS1)}$
$AS_1 \rightarrow CA: LOC(AS)$
$CA \rightarrow AS_1: \{LOC(AS), pk(AS)\}_{sk(CA)}$
$AS_1 \rightarrow AS: \{k, k_1\}_{pk(AS)}$
$AS \rightarrow AS_1: \{k_1\}_{pk(AS)}$
SESSION

a. Agent A asks AS for transfer to AS_1.
b. AS runs public-key distribution protocol to ask registry server (RS) for AS_1's address and public key.
c. RS creates a massage with AS_1's address and its public key, signs it using secret key $sk(CA)$ and sends to AS.
d. AS decrypts the message, generates a session key k at random, encrypts it with AS_1's public key $pk(AS_1)$ and sends it to AS_1.
e. AS_1 repeats the steps taken in b.c.d. to request RS for AS's address and public key
f. AS_1 generates its own session key k_1 and sends it together with the key k, previously sent by AS, back to AS encrypted with $pk(AS)$.
g. Upon receipt A decrypts the message and recovers the session key k_1 which sends to B.
h. Both parties authenticate each other and encrypt messages using session symmetric encryption key, which is already known to each of them.
i. Finally AS sends agent A to AS_1, and upon reception AS_1 registers its new address in RS.

Comment: There is a slight weakness in security here. For example an agent, which has been removed and added again, can be removed yet another time by a replay attack. We choose not to do anything about this rather minor problem, which could be solved by including the nonce into the part, which is signed. It is the responsibility of local CA to make sure that no certificate is added more than once. The methods of CA are secure enough, such that they could in principle be remotely callable.

Comment: The methods of protocols have a significant overhead. This is mainly because they involve a high number of cryptographic calculations. This is no big problem since they are very

rare events. The methods of the NetInt must however be optimized for speed. Jumping is a special case because it involves a considerable overhead, but a common operation should preferably be in the framework. The jumping mechanisms in NetInt are optimized for security and flexibility. If they should also be optimized for speed, protocols for negotiating shared keys between hosts with much communication should be included. In this way secure highways can be made available on the most used jumping paths. All these optimizations are possible but non-essential and rather complicated to manage, and have therefore been left out.

Comment: Like any other cryptoprotocol algorithms the proposed solutions also require protocol security proving. *CSP* and *FDR* were applied for checking security properties of such algorithms[4]. A check on Session protocol performed by using these tools revealed that not only the session key but also the sender name should be transferred to achieve the necessary level of protocol security.

To test a protocol like this one with FDR we have to build models of well-behaved nodes (Alice and Bob) and an intruder (Cameron) and see how the latter can interfere with the former.
As we have already said, the encryptions and similar will be modelled as symbolic objects: we create an appropriate data type to contain them. It consists of various constants we will need, public-key (PK) and symmetric-key encryption constructions and a sequencing construct.

datatype fact = Sq. Seq(fact) |
PK. (fact , fact) |
Encrypt. (fact, fact) |
Alice | Bob | Cameron |
Na | Nb | Nc |
pkA | pkB | pkC |
skA | skB | skC |
AtoB | BtoA | Cmessage

The type fact contains various collections of constants, which can be collected together into sets for later use. The three identities used are those of two nodes we will later treat as reliable, plus one (Cameron) for the intruder to assume when it acts as another party.

nodes= {Alice, Bob, Cameron}
publickey = {pkA, pkB, pkC}
secretkey = {skA, skB, skC}
nonces = {Na, Nb, Nc}
sessmess = {AtoB, BtoA, Cmessage}

Almost all possible actions for both parties are modeled. For example, both Alice and Bob can either act as an initiator of the protocol (Send) or as the responder (Resp).

User(id,ns) = if ns == <> then STOP else
Send(id,ns) [] Resp(id,ns)

In a similar manner intruder actions are constructed.
Equally, we would expect the intruder to be unable to learn the secrets AtoB and BtoA, since these are never revealed to Cameron deliberately. The spy or intruder can hear whatever passes between Alice and Bob, can interact with them as Cameron, and can intercept and fake messages. Such data is too bulky to be covered here and much more details could be found in [5]

Based on check results appropriate changes have been made in the protocol (not only the session key but also the sender name should be transferred), which makes a good background to assure

that parties could engage in an intact communication over a non-secure communications media, without running a risk of intrusion.

5. NetInt software

NetInt system is implemented in Java programming language in WINDOWS operating system environment. Moreover, exploitation of Java programming allows NetInt software system to be executed on any operating system containing Java Virtual Machine. The proposed NetInt agent environment is an automated dialogue system that enables direct communication of the user with the system; consequently availability of an appropriate Interface (see Fig 4) is required.

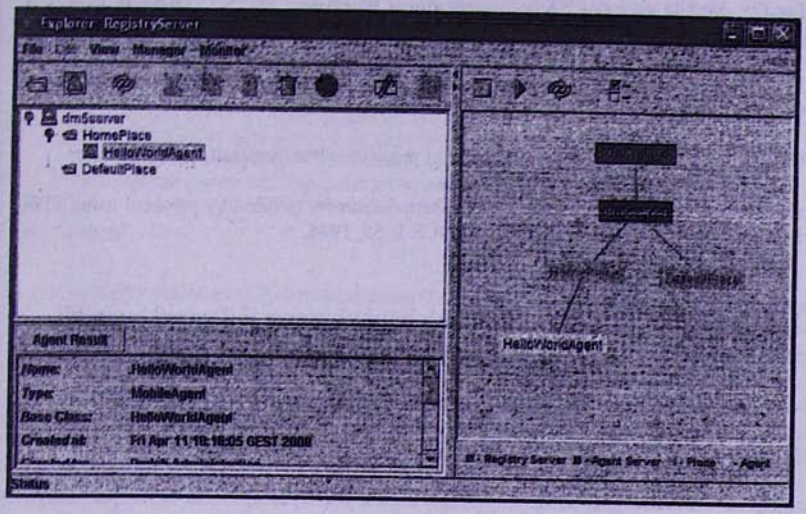

Fig. 4 Interface of NetInt agent system managment

NetInt agent system management interface includes the following main parts. The left part of the interface presents AgentServer of current node with its agents and places. The main tools that enable management (run/remove) of agents in the system are located at the top. The bottom part of the interface provides data on the agent or place such as agent name, type, its creation time and the name of its creator, run time, current state of the system, user descriptions, etc. The right part of Interface displays RegistyServer of NetInt, i.e. the whole NetInt agent system.

The type *Agent* is used to create and transport agents in NetInt system. All agents in the network are inherited from this type. The basic methods responsible for transportation of agents are *getNextLocation*, *getLocation* and, *move*, as well *run* method which is called during initiation stage.

The *Crypto_Methods* type is implemented for generating and distributing the public and secret keys as well as for encrypting data. The following functions *KeyPairGeneration*, *GetKeys* *Message_Encryption*, *Message_Decryption*, *RSASignature*, *Message_Encryption*_and_*Sign*, *SignatureVerification*, respectively, are used in *Crypto_Methods* to perform these procedures.

References:

1. L. Aslanyan, K. Margaryan, H. Sahakyan, Data analysis algorithms in network protection systems, III International Conference on "Digital Information Processing and Control in Extreme Situations", Minsk., May 28-30 2002, ISBN: 985-6453-80-1, pp. 221-225.
2. D. Milojicic, M. Breugst, MASIF - The OMG Mobile Agent System Interoperability Facility. Mobile Agents - Second International Workshop, MA '98 (Stuttgart, Germany, September 1998).
3. D. A. Karapetyan, Intelligent Agent Server (NetInt) System, Mathematical Problems of Computer Science vol. 25, pp. 64-70, 2006.
4. C. A. R. Hoare, Communicating sequential processes, Prentice Hall, 1985.
5. G. Lowe, Breaking and fixing the Needham-Schroeder public-key protocol using FDR, Proceedings of TACAS '96, Springer LNCS 1055, 1996.

Բանական ագենտ սերվերների համակարգում միջագենտային համագործակցության արձանագրություններ

Լ. Ասլանյան, Դ. Կարապետյան

Ամփոփում

Աշխատանքում ուսումնասիրվում են ծրագրային ագենտների վրա հիմնված Բանական ագենտ սերվերների համակարգում միջագենտային համագործակցության ընթացակարգի մասին: Ներկայացված են միջագենտային համագործակցության ընթացակարգերի 4 մոդել: Ընթացակարգերը գնահատված են ըստ արդյունավետության, իրականացման և հնարավոր կիրառությունների: Համակարգի տրամաբանական մակարդակը հանդիսացված և իրականացված է ալգորիթմորեն: Տրված է ներխուժման հայտնաբերման խնդրի տեստային լուծման տարբերակ: