Scale-up and Performance Studies of Three Agent Platforms

Kalle Burbeck, Daniel Garpe, Simin Nadjm-Tehrani
Dept. of Computer and Information Science
Linköping University, Sweden
{kalbu, x03danga, simin}@ida.liu.se

Abstract

With maturing technology agents are now a viable choice for distributed computing, also for systems with requirements on dependability and scalability. Agent platforms provide common services to applications developed as agents. Given the abundance of available platforms it is not easy to select an agent platform given a set of applications requirements. Evaluations of relevant properties of agent platforms are therefore needed, but unfortunately few up-to-date evaluations exist. In this paper we introduce and evaluate the three recent agent platforms JADE, Tryllian and SAP. Focus of the evaluation is the important properties of performance, security and scalability. We conclude that all platforms perform very well, but that platform architecture heavily influences the performance.

1. Introduction

In a multi-agent system (MAS), several agents interact with one another to solve a problem where interaction may consist exclusively of cooperation or competition, or a mixture of both. The benefits of using agents in distributed computing are for example loose coupling, adaptability and support of heterogeneous systems. The software agent technology, at some point considered as hype, has matured as a useful technology in many fields[1]. Agents have been used for a wide range of applications, for example in mobile environments or as adaptable control entities in large complex networks. Regardless of the type of application every agent needs some basic functionality. The agent platform is the distributed middleware providing common services required by MAS applications, such as communication, security, agent life cycle management and directory services.

When the activity and number of agents deployed on the platform increases so does the pressure and the need to distribute the shared resources while keeping up the performance. Comparisons[2] of these aspects of agent platforms are rare and due to rapid development outdated. In this work we will therefore compare three recent agent platforms, with focus on the important properties of performance, scalability, security.

Section 2 explains the initial selection of platforms. The platforms chosen for experimental evaluation is then presented in section 3 and relevant properties analyzed. Section 4 describes the experimental setup followed by the results and a concluding discussion in section 5 and 6.

2. Initial selection of platforms

During the last few years at least 60 different agent platforms[3] have been implemented. In our evaluation a set of 18 platforms were initially selected for further study. Existing surveys[4, 5] supplied some guidance, but the surveys are partially out-of-date with some platform projects abandoned altogether, and newer platforms not included.

After an initial evaluation of material presented in platform documentations, based on criteria such as maturity, security, communication facilities, footprint, scalability and performance, the Java Agent Development Environment (JADE)[6] platform was selected as one of the most promising. JADE is an open source project that has been around since about 1998. Tryllian Agent Development Kit (Tryllian)[7] was selected to be further evaluated together with JADE since the two platforms share many commonalities, but are built on different technologies. Tryllian is a commercial product that entered the market in 2002.

The alternative to use an existing general platform that may or may not be suitable for a given application is to develop a platform from scratch given the application requirements. Safeguard Agent Platform (SAP) is a new agent platform, developed in the Safeguard project[8] during the first half of 2003, to be simple, lightweight and have good performance. The study that is presented here was initiated to evaluate the strengths and weaknesses of SAP in comparison with other available platforms, and to potentially justify further development of the platform. A fourth platform (Grasshopper 2) that passed the other criteria, could not be included in this study, since its license agreement restricted the conducting of benchmarking tests for evaluation of performance.

3. The evaluated platforms

3.1 Basic notions

Software agents have been defined in almost as many ways as there are commentators in the field. There is some consensus, however, about the key features an intelligent software agent should exhibit:

- *Autonomy* – the ability to act without external prompting by a human or another software system.
- *Pro-activity* - the ability to be goal-oriented.
- *Reactivity* – the ability to perceive its environment and act upon that same environment in response to its perceptions and goals.
- *Communication* – social ability is the key to many agent functions, such as cooperating or competing with other software agents. In order to facilitate such interactions, an agent must be able to communicate meaningfully with other entities in the environment.
- *Adaptability* – the ability to learn is desirable but not mandatory.

In this paper we consider the Agent objects defined by the platforms as agents, also when they do not fully exhibit all features above.

The Foundation for Physical Intelligent Agents (FIPA)[9] is a non-profit organization setting up standards for agent applications and systems. We adopt the view of FIPA abstract platform and agent management specifications here. Important components are:

- *The Agent Management System* (AMS) is responsible for agent activities such as agent creation, lifecycle and deletion as well as providing and keeping unique agent addresses. The analogy with the phonebook's 'white pages' comes as no surprise. There can only be one logical AMS on an agent-platform even if it physically spans over multiple machines.
- *The Directory Facilitator* (DF) is a place where agents can register their services for other agents to search for. The facility is also often named 'yellow pages' as in the phonebook.
- *The Message Transport System* (MTS) provides communication facilities between agents.
- The *agent container* is the environment where agents live. An instance of an agent platform may consist of multiple containers, possibly distributed over multiple hosts.

3.2 JADE

JADE is the platform with strongest resemblance to the FIPA specification. This is clearly reflected in the architecture with at least one DF for each container, and one main container with the AMS. The compliance with FIPA also defines the limit of each JADE platform instance since even though it is possible to distribute containers on several hosts there has to be one shared main container.

Communication - JADE supports both synchronous and asynchronous agent communication. Message passing between agents on the JADE platform can be handled in three different ways depending on the location of the two agents. If both agents are placed in the same container, Java's event mechanism is used for in-process communication without any message translation. If the agents share the same platform but are placed in different containers, RMI communication is used. Finally if two different platforms are used the standard IIOP protocol is used.

Concurrency - In JADE concurrency can be viewed as a three level hierarchy. On the topmost level the platform supports distribution of containers, either on the same host or between several hosts. Every container has its own Java virtual machine and the independent behaviour of these at the platform abstraction level is obvious. One step below in this hierarchy is the agent where each agent is assigned its own Java thread. At the lowest level the agent itself supports concurrency in what JADE defines as *agent behaviours*. JADE uses a round-robin non-pre-emptive scheduling policy for executing multiple behaviours inside an agent thread. Behaviours are stateless which means that state variables must be used when breaking up execution of behaviours.

Security - Security in JADE is not a built in standard feature but rather as a plug-in. It provides a security model based on principals, resources and permissions, which enables authentication and authorization of both agents and the owners. Secure communication is also provided in the form of Secure

Socket Layer (SSL). For communication with agents outside the platform border SSL cannot be used.

3.3 Tryllian ADK

The Tryllian platform is based on a peer-to-peer (P2P) library called JXTA [10]. Project JXTA is a P2P-based networking technology from Sun Microsystems, Inc striving for platform independence, interoperability and ubiquity. This library is a dominant building block, pervading the whole Tryllian system. Each platform instance is considered a peer residing on a single host. The platform border may be defined by the host or more appropriately be considered unlimited due to the absence of any main AMS.

Communication - The JXTA pipe is the basis of communication between agent containers. If two agents are located in the same container Java's event mechanism is used for local process communication.

Concurrency - Tryllian, similar to JADE, provides an abstraction for parallel behaviours internal to an agent; in Tryllian called *task*. Even though the concept is basically the same as in JADE, the two approaches have their dissimilarities. In addition to the event model, where the task responds to incoming messages, the task model has a proactive behaviour in what is called the task's heartbeat. This means that every few milliseconds the task is given an opportunity to initiate its own action. Based on the completion of a task, i.e. success or failure, Tryllian provides a logical task scheduler in which tasks can be scheduled according to a state diagram. Besides this logical scheduling ability the underlying task scheduling policy is not explicitly stated. An important feature of Tryllian is that each agent task is run by a separate thread obtained from a common thread pool.

Security – Transport Layer Security (TLS) version 1.0 is used for secure communication between containers. To improve performance only one TLS connection is open between two peers even if many JXTA pipes are used. Tryllian supports a number of cipher suites but which one to use is decided internally.

3.4 SAP

Similar to JADE, the SAP architecture is rather close to the FIPA model but there are no pretensions of complying with the standard specifications. Still the SAP architecture has the basic parts such as management system and name and directory services. For agent name resolution a special lookup server is used and it has to be started separately. This central lookup server defines the platform border.

Communication - In contrast to the other two platforms SAP does not provide any extra transport abstraction layer for the agent communication. Plain TCP/IP sockets are used between system instances and the local in process communication is the choice for agent conversation in the same virtual machine. There exists an alternative with Java Message Service (JMS) that uses a central message queue server but the recommendation is to use sockets.

Concurrency - On the platform level every agent runs in a separate Java thread and every instance of the platform has its own Java virtual machine. SAP does not provide any extra abstraction corresponding to JADE behaviours or Tryllian tasks. The agent developer may use multiple Java threads internal to the agent for parallel behaviours.

Security - In the current version of the platform (version 1.1) no message encryption option for agent-to-agent communication exists yet.

4. Experiment set-up

4.1 Test scenarios

The scenarios centre on pair wise agent-to-agent communication. Different scenarios are realized through change of parameters, which are explained below.

- *Number of hosts* - The most likely scenario for an agent platform is a distribution of the platform between several hosts. The parameter is either one or two hosts.
- *Number of agent pairs* – By increasing the number of agents communicating the general behaviour of the platform MTS as well as scalability are tested. Communication is always considered as a conversation between two agents. So when the numbers of agents are increased it is always done in steps of agent pairs.
- *Message size* – Depending on the usage of an agent platform the size of message will vary. Typically the communication is based on interactions with relatively small messages and therefore the tests with increasing number of agents are using 2 KB messages. Cases with messages of larger size are possible and this has also been tested as single agent-pair conversations with message sizes between 0 and 100 KB.
- *Message encryption* - This is one way to test performance-security trade-offs.

Two additional parameters, Java configuration and platform configuration, were considered in initial experiments and suitable settings were established. For the experiments presented here, those parameters were fixed for each platform.

4.2 Test bed

The tests were done on a single computer and on a Local Area Network (LAN) consisting of two computers connected with a 100 Mbps cable. The two computers share the same fundamental set-up according to Table 1. A separate network was used to avoid influence of other hosts.

Table 1: Hardware and software set-up of the computers used in the experiments.

Operating System	Microsoft Windows 2000 Professional
Java VM	Java SDK 1.4.2 (1.3.1 for Tryllian)
CPU	AMD Athlon-PECM 900 MHz
Memory	512 MB (SDRAM)
Network card	3Com EtherLink XL 10/100

As all three agent platforms are implemented in Java so were also the agents used for the experiments. The latest Java SDK version 1.4.2 with accompanied runtime environment was used but also version 1.3.1, as one of the platforms, Tryllian, did not support any later versions. The Java Virtual Machine and platforms were restarted between each experiment to avoid influence of previous experiments.

4.3 Measurement

Round trip time (RTT) is used to measure communication performance. The RTT is used because of the asynchronous nature of an agent platform, so time is always relative the same agent. The clock used is Java's System.currentTimeMillis() method and the unit is milliseconds. The resolution of the clock (10 ms) is considered good enough since time is measured for 1000 and 10000 repetitions rather than individual messages.

For each experiment the sender agent sends a message to the receiver agent. The receiver agent obtains the payload, and sends an equivalent message back. This is repeated 1000 times for single agent pairs and the mean RTT is used as result. To obtain good confidence in the results, each experiment was repeated ten times. The average is presented here.

Single pair experiment is straight-forward to implement and test. With multiple agent pairs there is a short period in the beginning (end) of each experiment when not all agent pairs have started (finished) exchanging messages. By increasing the number of exchanged messages from 1000 to 10 000 for all experiments with multiple agent pairs the influence on the results becomes insignificant.

5. Results

This section presents some of the results of the experiments. Additional results are available [11] but left out due to space restrictions.

5.1 Implications of agent location

In Figure 1 we use JADE to illustrate the important choice of how to distribute agents. The sender and receiver agent may be located in the same container on one host, in separate containers on one host or on different hosts.

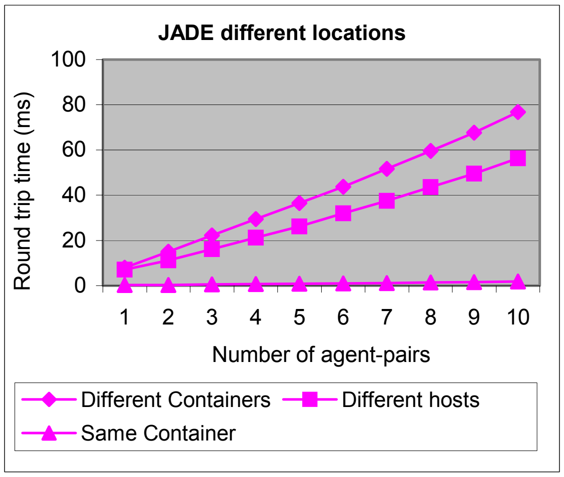

Figure 1: RTT for multiple agent pairs in different locations.

Containers may be used for separating groups of agents in a multi agent application also when all agents run on the same host. If performance is considered this is not a wise choice due to the use of RMI rather then events for communication. The result of Figure 1 where communication over network is faster then local communication between containers is explained by the high bandwidth of the network and that two computers share the processing load in the case with network communication. Naturally the RTT between agents in the same container is significantly less due to use of events rather than RMI.

5.2 Single agent pair comparison

When increasing the load the communication time is expected to rise linearly. This is indeed the case as shown in Figure 2 with sender and receiver located on separate hosts.

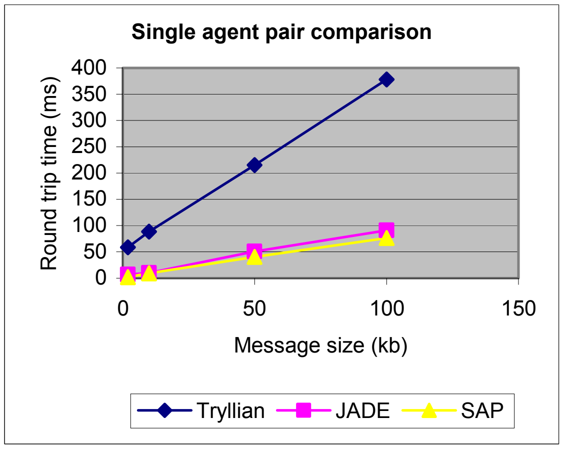

Figure 2: RTT for single agent-pair on LAN.

5.3 Multiple agent pairs comparison

Figure 3 shows the results when increasing the number of agent pairs from one to ten, with senders and receivers distributed on separate hosts.

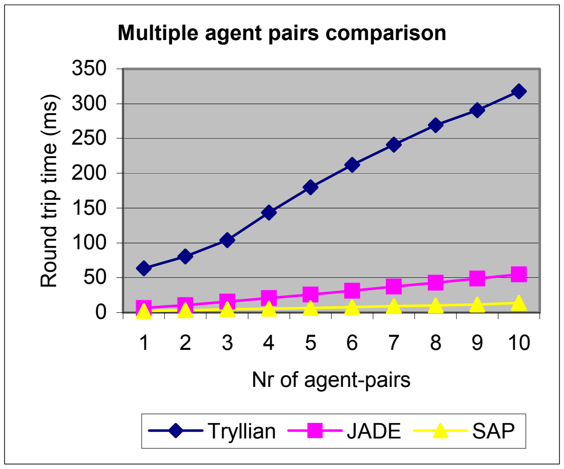

Figure 3: RTT for multiple agent-pairs on LAN.

As explained earlier all tests with multiple agents imply conversations between agent pairs and the round trip time is an average of the results from all involved agent pairs. The tests with only a limited number of agents give a good indication of how the platforms perform in general when the number of agents is increased for applications with a reasonably small number of agents.

5.4 Scalability

To stress the platforms the number of agents was further increased beyond ten as shown in Figure 4. When testing 100 agents on each machine Tryllian failed. The reason was not because of a platform crash but rather a result of agents not finding each other and this indicates a failure of JXTA rather than the Tryllian platform implemented on top of JXTA. This is also proved in the next graph presented where agents are all located in the same platform instance on a single host and 100 agent-pairs works fine.

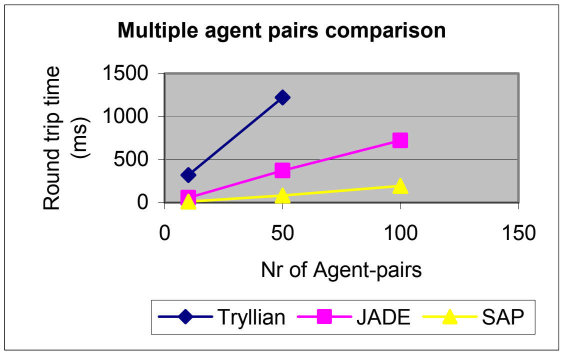

Figure 4: RTT for multiple agent-pairs on LAN.

Figure 5, with results from multiple agent pairs on the same host, shows that the results for the platforms relative each other is similar to the case when the platforms are distributed over two hosts.

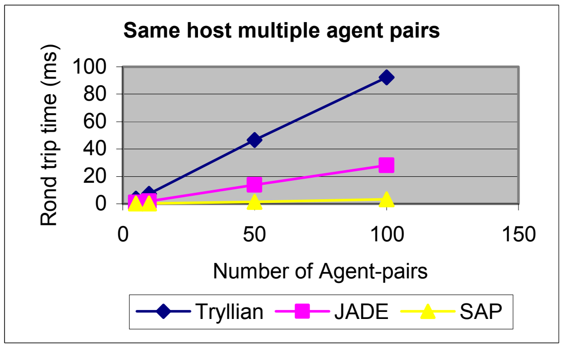

Figure 5: RTT for multiple agent-pairs on same host.

Due to the underlying communication implementation with events and method invocations the platforms handle this situation without any failures. Due to the thread pool of Tryllian, the platform is expected to handle a very large number of agents as long as they are located on the same host using the event mechanism rather then JXTA. The tests for JADE and SAP worked without problem up to 100 agents, which was the maximum used for this parameter.

5.5 Performance and Security

Security and encryption naturally comes with a performance penalty. Figure 6 shows the results for one agent pair with increasing message size. We see that adding security doubles the RTT for both platforms. SAP is not included in this comparison since the platform does not provide encryption yet.

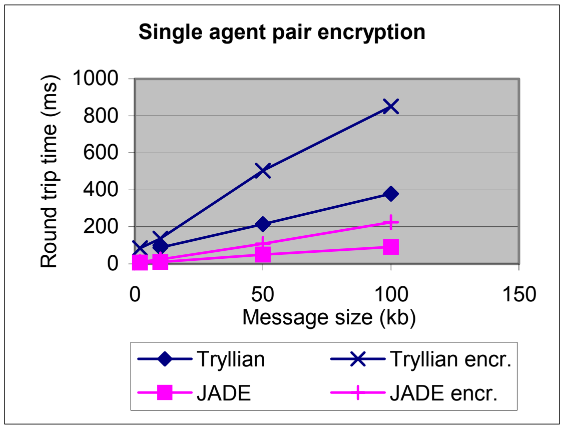

Figure 6: RTT for single agent-pair on LAN with and without encryption.

RTT with and without encryption is also presented in Figure 7. Now the message size is kept constant while instead the number of agents communicating is increased.

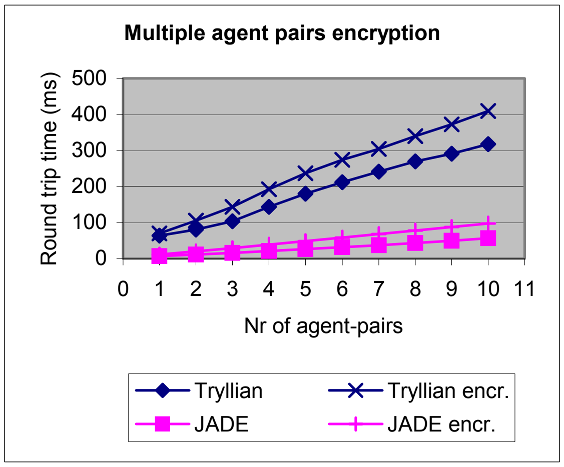

Figure 7: RTT for multiple agent-pairs on LAN with and without encryption.

The platforms handle the increasing number of agents very well, and the slope of the test cases with encryption are close to slope of the test cases without, which must be regarded as good. This is explained by the fact that both platforms use secure connections between agent containers rather then individual agents.

The initial handshake of a secure connection is very expensive. The good result for both experiments using encryption are due to the fact that both platforms establish secure connection between containers and keep the connection open to avoid extensive handshaking.

6. Discussion

When the results from the three platforms are compared it is not hard to see the general tendency of SAP as the platform with better performance followed by JADE and last Tryllian. In fact this pattern, more or less, is found throughout the whole test series. A more detailed analysis of the results and the factors behind them is presented here.

The architectural differences between the three platforms are certainly key factors from a performance perspective. While Tryllian is built on JXTA it has the advantage of being scalable in the sense of distributing platform instances without any limitations. This is not the case for the other two, being restricted by the requirement of a central AMS or a central lookup server. The term scalability was not interpreted as being able to distribute the platform over large networks though, but rather as the ability to keep up good performance when the load is increased. In this perspective the JXTA abstraction cannot be considered as being advantageous.

The extra abstraction found in JADE with RMI, and Tryllian with JXTA pipes compared to SAP with Java sockets should be considered as a factor contributing to the results produced. The RMI and JXTA abstractions have benefits as in the architectural aspect but for the communication between platform instances the performance is suffering. JXTA is still not very mature and when new releases are integrated in Tryllian this could prove beneficial for the platform.

Java 1.4 implements a number of performance enhancements over version 1.3. SAP and JADE benefit from this.

Both Tryllian and JADE provide multitasking behaviours for the agents but not SAP. JADE uses one thread per agent shared by all the agent behaviours, while Tryllian executes each task in a separate thread obtained from a thread pool. When dealing with a very large number (thousands) of agents, the lack of a thread pool will limit scalability for JADE. For Tryllian the thread pool is beneficial as long as extensive communication is not used between different hosts, in which case JXTA will limit scalability anyway. In many applications it should be possible to locate agents with extensive communication on the same host, to avoid scalability problems due to JXTA. The JADE choice of implementation may cause some gain in performance since context switches are not necessary when switching agent activities with behaviours inside the same thread.

Tryllian provides the largest set of configuration options, which makes the use of the platform more flexible. The parameters were all set to provide maximum resources to the platform under the benchmarks but still the RTT was large compared to the other platforms due to JXTA.

6.1 Limitations and future work

For some applications *mobility*, the ability of an agent to move from one place to another may be an important property. This is not considered in this work due to the principal problems related to security when using mobile agents.

The experiments presented here only considered a network with two hosts. It would be interesting to compare a true peer-to-peer based platform such as Tryllian with platforms using a centralized directory service using a more complex scenario. In a more dynamic environment JXTA would prove more beneficial with features such as automatic HTTP tunnelling, handling of dynamic IP-addresses and a graceful response to disconnected mode.

Even though most basic agent functionality is implemented in all the platforms evaluated here, a platform may of course have additional useful functionality (e.g. the agent persistence of Tryllian). However, these are not analyzed unless they are related to performance, scalability or security properties of the platforms.

During this work we realized that the selection of an appropriate platform given a set of application requirements is not easy. A more extensive study including more agent platforms with more parameters (e.g. usability and platform features) would be very helpful for the application developer searching for a suitable platform. This is outside the scope of this paper and is left as future work.

Future work should also include evaluation of additional platforms. Two platforms that deserve to be mentioned are Cougar[12] and Lana[13]. Cougar was developed in a DARPA project now continued as open source with interesting related projects such as UltraLog (concerning logistics information system survivability) and Cougar Micro Edition. Lana is a programming model for autonomous systems, which may be used as an agent platform. Lana implements many interesting features such as asynchronous method calls between programs and protection of programs from each other using protection domains.

Acknowledgements

The work presented in this paper has been supported by the European Safeguard project IST-2001-32685 [8].

References

1. *EUTIST-AMI*, http://www.eutist-ami.org/, accessed 4th Dec. 2003.
2. Silva, L.M., et al., *Comparing the performance of mobile agent systems: a study of benchmarking.* Computer Communications, 2000. **23**(8): p. 769-778.
3. *AgentBuilder - Agent Construction Tools*, http://www.agentbuilder.com/AgentTools/, accessed 4th Dec. 2003.
4. Perdikeas, M.K., et al., *Mobile agent standards and available platforms.* Computer Networks, 1999. **31**(19): p. 1999-2016.
5. Ricordel, P.-M. and Y. Demazeau. *From analysis to deployment: a multiagent platform survey*. in *1st International Workshop on Enginnering Societis in the Agents World (ESAW)*. 2000. Berlin, Germany: Springer Verlag.
6. *JADE*, http://sharon.cselt.it/projects/jade/, accessed 4th Dec. 2003.
7. *Tryllian*, http://www.tryllian.com, accessed 4th Dec. 2003.
8. *Safeguard*, http://www.ist-safeguard.org/, accessed 4th Dec. 2003.
9. *FIPA*, http://www.fipa.org/, accessed 4th Dec. 2003.
10. Eckstein, R., et al., *JXTA in a nutshell*. In a Nutshell, ed. B. Eckstein. 2002: O´Reilly & Associates.
11. Garpe, D., *Comparison of three agent platforms - performance, scalability and security*, Thesis number LiTH-IDA-EX--03/070--SE *Dept. of Computer and Information Science*. 2003, Linköping University.
12. *Cougaar*, http://www.cougaar.org, accessed 4th Dec. 2003.
13. Razafimahefa, C., C. Bryce, and M. Pawlak. *Lana: An Approach to Programming Autonomous Systems*. in *European Conference on Object-Oriented Programming (ECOOP)*. 2002. Malaga, Spain.