

A COMPARISON OF TWO DIFFERENT JAVA TECHNOLOGIES TO IMPLEMENT A MOBILE AGENT SYSTEM

Alf Inge Wang*

Dept. of Computer and Information Science,
Norwegian University of Science and Technology,
No-7491 Trondheim, Norway.

Carl-Fredrik Sørensen†

Dept. of Computer and Information Science,
Norwegian University of Science and Technology,
No-7491 Trondheim, Norway.

ABSTRACT

This paper describes an evaluation of the two technologies Aglets from IBM Japan and JavaSpaces from Sun Microsystems used to implement the same mobile agent system. The Aglets framework was initially designed to support mobile agents, while the JavaSpaces framework was not. In this evaluation we have looked at issues like adaptability, configurability, extensibility, location transparency, and abstraction level. We have also measured the effort that was used (in hours) and the lines of code necessary to implement the same mobile agent system using the two different technologies. We found that there are several advantages by using JavaSpaces instead of Aglets for implementing mobile agent systems even if it was not intentionally designed to support this. Our evaluation does not look at the difference in performance of the two technologies.

Keywords: *Mobile agents, Evaluation, Aglets, JavaSpaces*

1 INTRODUCTION

Mobile computing has been a very hot topic in the last years, where mobile agents have been one of the promising technologies. A mobile agent is mobile code that can migrate from one computer host to another, and resume the execution where it left off. Today, there are many different technologies that can be used to implement a mobile agent system. This paper presents an evaluation of two such technologies: Aglets from IBM Japan, and JavaSpaces from Sun Microsystems. In the evaluation we looked at different aspects of the technologies such as how they cope with a changing execution environment, how easy it was to implement a mobile agent system, the level of abstraction provided, etc. To do the evaluation we chose to identify a number of evaluation criteria (described in section 5.1), then implement the same mobile agent system using two different technologies, and finally evaluate the technologies using the chosen evaluation criteria. The Aglets framework was initially designed to provide an infrastructure for mobile agents, while the JavaSpaces framework was not. We wanted to investigate if JavaSpaces could provide the same support as Aglets for mobile agents. We chose to look at the JavaSpaces framework because it provides a very high abstraction level, it can deal with a dynamic execution environment, and it has support for migration of Java objects

between hosts.

The rest of the paper is organised as follows: Section 2 describes other evaluations of mobile agent technologies, section 3 describes the technologies Aglets and JavaSpaces that were evaluated in this paper, section 4 describes the mobile agent system that was implemented in the two technologies, section 5 describes how we evaluated the technologies and the results from this evaluation, and finally section 6 concludes the paper.

2 RELATED WORK

In [5], Kiniry and Zimmerman evaluate the three Java-based mobile agent technologies Odyssey, Aglets, and Voyager. The main objective of the evaluation was to investigate issues such as ease of installation, feature set, documentation and cost. They found that all these three technologies were rather easy to install, and that Odyssey and Aglets were in lack of sufficient documentation. There is also a noticeable difference in the agent transport mechanisms used in the mobile agent technologies. The Aglets framework provides the simplest agent transport using Java RMI. Voyager can use Java RMI, but CORBA is also supported. Odyssey has explicit support for multiple agent transport mechanisms: Java RMI [9], CORBA [10], and DCOM [7]. The design of Aglets makes it easy for the programmer to code agents by extending the well-know applet concept to aglets where the object also can migrate between sites. Kiniry and Zimmerman have also identified four characteristics most Java-based agent technologies share: 1) They provide an agent server of some kind, 2) Agents can migrate from server to server in some fashion, carrying their state with them, 3) Agents can load their code from a variety of sources, and 4) All are 100 percent pure Java [14]. This paper describes a high-level evaluation of three mobile agent technologies. Our paper describes an evaluation of implementing a mobile agent system using two different mobile agent technologies. This means that our evaluation is a more hands-on evaluation.

In [3], Ismail et al. present an evaluation of mobile agent technology in terms of communication costs. This paper compares the performance results from running four different minimal implementations for searching two databases and for fetching documents from a remote server. The first implementation

*Phone: +47 73594485, Fax: +47 73594466, Email: alfw@idi.ntnu.no

†Phone: +47 73590731, Fax: +47 73594466, Email: carlfrs@idi.ntnu.no

is based on their own mobile agent platform, the second uses the classical client-server paradigm (Java-RMI), the third uses Aglets from IBM, and the fourth uses a mobile agent framework called AAA. The results from the evaluation showed that there is a lowest break-even point (in terms of number of database records or documents) from which mobile agents are beneficial compared to traditional client-server architecture. The results also showed that mobile agents could outperform client-server implementations of the same application. As in our evaluation, Ismail et al. also evaluate an application that has been implemented using different mobile agent technologies. The main difference between the two evaluations is that we have focused on how easy it was to implement and change the application, while Ismail et al. have focused on performance.

In [4], Altmann et al. present an evaluation and a survey of mobile agent platforms. The criteria used in this evaluation can be divided into five main categories: Security (authentication, encryption, certificates, authorization, and access restriction), availability (available evaluation version, and state of platform), environment (documentation, and supported operating systems), development (programming language, graphical administration tools, monitoring, debugging, RAD, deployment, and architecture), and characteristics properties (mobility, and standards like FIPA and MASIF). The result of this evaluation ranks eleven mobile agent platforms according to the criteria, where Grasshopper, Jumping Beans and Aglets were the top three. Grasshopper got the best score because of the integrated security mechanisms and the graphical administration tools. This evaluation is also a high-level evaluation that ranks the different functionality provided in mobile agent platforms. However, the evaluation does not provide any experiences from using the mobile technologies to implement an application.

3 MOBILE AGENT TECHNOLOGIES

This section presents the two technologies that were used to implement the mobile multi-agent system.

3.1 AGLETS

The Aglets framework [6] developed by IBM Japan was designed to make mobile agent systems in Java. The Java aglet extends Java applets, by conserving execution state and stacks when moving between network hosts. A Java applet requires a web-browser or an applet-viewer to run on a host. In the same way, an aglet requires an *aglet host* running on the computer prior it can visit this host. The aglet host provides a secure environment to protect the host against untrusted aglets, and can upload aglets through class loaders (both class files and state) from another host.

There are some primary operations that can be used to control the life of an aglet:

- **Create aglet:** A new aglet is created. The aglet has an initial state and will execute its main thread.

- **Clone aglet:** A copy of an aglet will be created with the same state and code as the original.
- **Dispatch aglet:** An aglet will move from one host to another with the state preserved.
- **Retract aglet:** An aglet will be brought back to the initial host with the state preserved.
- **Deactivate aglet:** An aglet will be paused and the state will be saved to disk.
- **Activate aglet:** Restore a paused agent with the state that was stored to disk.
- **Dispose aglet:** Kill an aglet including its state.

The lifecycle of an aglet is shown in figure 1.

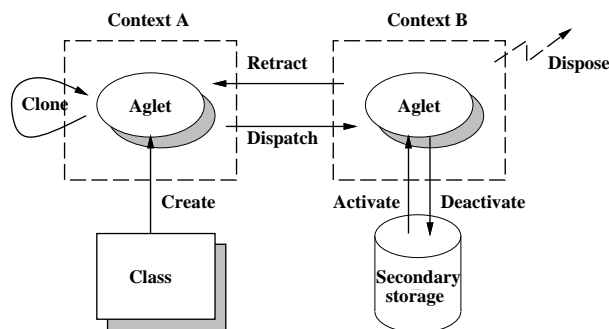


Figure 1. The lifecycle of an aglet

To enable preservation of execution state of Java objects, serialisation is used. Serialisation was provided as a part of JDK 1.1 and makes it possible to transform the state of an aglet object to a stream of data (as well as the execution thread and stack).

3.2 JAVASPACE

JavaSpaces [8] from Sun is based on Java RMI and JINI [15], and is a framework to manage exchange of distributed objects. JavaSpaces technology uses a unified mechanism for dynamic communication, coordination, and sharing of objects between Java technology-based network resources like clients and servers. A JavaSpace is a virtual space between providers and requesters of network resources or objects. This allows Java applications in a distributed execution environment to exchange tasks, requests, and information in the form of Java objects. There are four primary operations that can be used in a JavaSpace (illustrated in figure 2 [1]):

- **Write** an entry (Java Object) into a JavaSpace.
- **Read** an entry from a JavaSpace that matches some specified parameters.
- **Take** an entry from a JavaSpace that matches some specified parameters (removing it).
- **Notify** a specific object when entries that match some specific parameters are written into this JavaSpace.

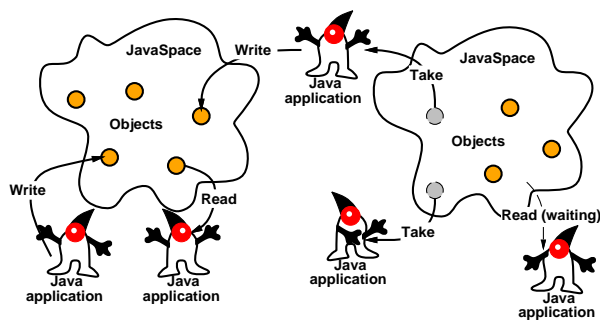


Figure 2. The four primary JavaSpaces operations

An *entry* is in JavaSpace terminology a typed group of objects, described as a class in Java. JavaSpace technology offers much of the same functionality as required in a mobile multi-agent system such as migration of objects, message handling, sharing of objects. Maybe the most useful functionality in this respect is the support for search of objects with certain properties.

4 MOBILE SOFTWARE AGENT SYSTEM FOR COOPERATIVE SOFTWARE ENGINEERING

Our mobile software agent system is named Distributed Intelligent Agent System (DIAS), and can be used to implement process support for cooperative software engineering (CSE). By cooperative software engineering we mean large-scale software development and maintenance work in a distributed organisation where people must cooperate to create their products. DIAS can be used to implement support for distributed, cooperative activities between roles such as resource negotiation, brainstorming, voting, coordination of artefacts etc. The DIAS consists of the following four main components (also shown in figure 3):

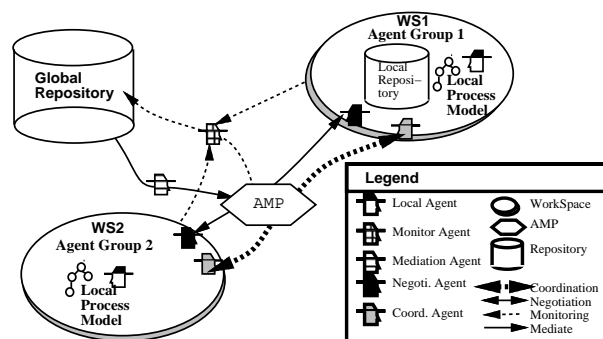


Figure 3. The Distributed Intelligent Agent System

- **Agent:** An agent is set up to achieve a modest goal, characterised by autonomy, interaction, reactivity to environment, as well as pro-activeness. We have identified three main types of agents: (1) *User agents* (interface agents) that interact with users and perform the tasks they are defined to do for the end-users, (2) *Interaction agents* that assist with cooperative work between workspaces (negotiation agents,

coordination agents), and (3) *System agents* that give system support to other agents and Agent Meeting Places (monitor agents, mediation agents, etc.).

- **Agent Meeting Place (AMP):** AMPs are where agents meet and interact. AMPs support agents in doing efficient inter-agent communication. There can be different types of AMPs for different purposes. Each AMP will have a defined ontology, which the agents have to follow. We can also consider special AMPs for negotiation, coordination, information exchange, selling and buying services etc.
- **Workspace:** A workspace is a temporary container for relevant data (artefacts, models etc.) in a suitable format to be accessed by tools, together with the processing (work) tools. It can be private, as well as shared. Files stored in a repository can be checked in and out of a workspace.
- **Repository:** Repositories can be global, local, or distributed, and are persistent storage of data. Experience bases are one specific type of repository, that we can use in our multi-agent system to support community memory.

A more detailed description of the multi-agent system can be found in [18, 11, 2, 13].

4.1 THE REQUIREMENTS USED IN THE EVALUATION

Prior to the implementation of DIAS in Aglets and JavaSpaces, we established some requirements that both implementations had to follow. These requirements describe the basic functionality of the agent system like agent localisation, agent communication etc. In addition, we implemented a simple test case for both implementations. This test case provides support for resource negotiation between two parties, where the mobile agents were instructed by the user what to achieve from the negotiation. The negotiation agents then moved to an AMP, where the negotiation took place. After a successful negotiation the agent returned to the user with the result, or possibly to ask what to do if there were problems when negotiating with the other agents. Here is an outline of the functional requirements that were used to implement the two systems (the requirements for the test-case is not included here) [16]:

- R1 **Unique Identification:** An agent and an AMP must have unique identifiers.
- R2 **System Agents:** System agents are stationary agents that reside on an AMP and provide services to the agent system. System agents are responsible for creation and deletion of AMPs, monitor agent behaviour, receive agents arriving at the AMP, registration of agent of the AMP, localisation of agents' whereabouts, and move agents between AMPs.
- R3 **Participation Agents:** Participation agents provide aid the user agents in the interaction with other agents. These agents are responsible for handle communication between two agents, and for negotiate between user agents or directly between users.

R4 **User Agents:** User agents are defined and created by the user using a pre-defined AMP.

R5 **Agent Meeting Places (AMPs):** AMPs are agent places that facilitate services for agents to interact. An AMP should provide an environment for executing agents. AMPs should provide all system agents and in addition have the capability to store agent information and exchange agent information with other AMPs.

R6 **Agent match making:** One agent can address and exchange services with other agents using an agent identifier for finding the agent. An agent can also use agent ontology, defining the services and abilities to be used of the different agents in the search for other agents.

5 THE EVALUATION

This section describes the evaluation of the DIAS implementations using Aglets and JavaSpaces respectively.

5.1 EVALUATION CRITERIA

One of our goals with the evaluation was to investigate how two different technologies can cope with a changing execution environment and how easy the technologies are to configure. The evaluation criteria we ended up with were very similar to typical non-functional requirements. Most of these criteria cannot be measured directly by metrics, but are given a grade (low-medium-high) based on observations. The only measurable criteria are effort spent and lines of code. The following evaluation criteria were used to compare the two different implementations of the DIAS:

- E1 **Effort spent:** How much time was spent to implement the DIAS measured in man-hours?
- E2 **Lines of code:** How many lines of code were required to implement DIAS?
- E3 **Support for mobile agents:** Were mobile agents supported directly or indirectly?
- E4 **Adaptability:** How easy was it to adapt the technology to changes in the execution environment?
- E5 **Configurability:** How easy was it to configure the technology to be ready for use?
- E6 **Extensibility:** How easy is it to extend the implementation using the technology?
- E7 **Localisation transparency:** Does the technology need to address hosts in the system explicitly, or is this taken care of by the provided technology layers?
- E8 **Abstraction layer:** At what level of abstraction does the developer operate?
- E9 **Coverage of DIAS:** How well does the technology provide support for all services needed in DIAS?

5.2 THE IMPLEMENTATION

The implementations were done by two groups of two graduate master students. Both groups spent two months in learning the technology before they started to implement the DIAS.

- **The Aglets implementation:** The Aglets implementation of DIAS was named DIAS II and required JDK 1.1.7 and Aglets 1.1 to run. After the installation of the necessary packages, DIAS II required that the user configured all the AMPs involved in the system. This means that the user for each AMP explicitly must enter the network addresses to the other AMPs to establish communication with them. In Aglets the network address is given similar to http, e.g. “*atp://testmachine2.server.com:4432/AMP*”. Although the Aglets framework was designed to provide infrastructure for mobile agents, 1072 lines of code was necessary to implement the most fundamental services of DIAS. This is due to the low abstraction level of Aglets.
- **The JavaSpaces implementation:** The JavaSpace implementation of DIAS was named DIAS III and required JDK 1.2, JINI Technology Starter KIT 1.0.1, and JavaSpaces Technology Kit 1.0.1 to run. After the installation of JDK, JINI, JavaSpaces and DIAS III, the prototype could be run from any machine without further configuration of hosts or networks. This is made possible by JINI’s plug-and-play facility of network resources, detecting JavaSpaces that appear or disappear dynamically. Mobile agents are not directly supported in the JavaSpace framework, so some additional design and code had to be added to provide support for this. Because of the high abstraction level of JavaSpaces, only 621 lines of code were necessary to implement the basic functionality of DIAS.

5.3 COMPARISON

Since there was a difference in the completeness of the implementations of DIAS II and III, we decided to only consider the parts of the agent system that was implemented in both versions. We have therefore only considered the requirements R1 (Identification requirements), R2 (System agents), and R5 (agent Meeting Place) when comparing the two implementations.

If we first look at *E1* (effort spent), we can notice that 1280 man-hours were spent on DIAS II, and 850 man-hours on DIAS III. This means that the effort-ratio between DIAS II and III is approximately 1,5. If we further take a look at *E2* (lines of code), we can see that DIAS II consists of 1072 lines of code, and DIAS III only 621. Here the ratio between the two implementations is 1,7 that is more significant than the man-hour ration. Even though the lines of code do not tell the whole truth, we can see that the Aglets implementation required much more work. We believe *E8* (abstraction level) is the main reason for the significant differences. In the JavaSpace implementation, the underlying technology layers in the JavaSpace framework and JINI did a lot of the work. Even if Aglets directly supported mobile agents (*E3*), the low ab-

straction level resulted in much more work in configuration, routing, and communication management. Since the JavaSpace framework provides an infrastructure for moving Java objects between spaces (including preservation of the state of the objects), it was very easy to extend the framework to support mobile agents (more details about how this was done can be found in [17]).

To check the adaptability (*E4*) of the two technologies, we initiated some events during execution of DIAS II and III. The events we initiated were to halt an AMP, to start a new AMP, and to move one AMP from one machine to another. These tests showed that JavaSpaces performs very well in a changing execution environment because of the underlying support of JINI. JINI polls the network regularly to check if there are any JINI network services running, making it possible to detect new services and remove services that are not connected anymore. Also moving an AMP from one node in the network to another did not cause any problem. JavaSpaces provides location transparency (*E7*) through JINI. Initiating the same events in the Aglets implementation did not go that well. E.g. when moving one AMP to another network node, the AMP was not automatically detected by the other AMPs. To make DIAS II work, it was necessary to manually remove the moving AMP from all other AMPs registers. Then register the AMP again with the new network address. Another way to improve the adaptability of the Aglets implementation could be to implement dynamic network service detection as an add-on to the Aglets framework. This would basically mean to implement JINI or integrate JINI into Aglets.

Configurability (*E5*) was another area where JavaSpaces performed much better than Aglets due to the same reasons as given in previous paragraph. Since the network configuration is done automatically in JavaSpaces, no extra configuration is needed when starting DIAS III. For DIAS II, it was necessary to explicitly register all AMPs network addresses before the agent system could be used.

Extensibility (*E6*) can be split into two parts: Dynamic extensibility and static extensibility. By dynamic extensibility, we mean the ability to extend the system when running. As expected, the dynamic extensibility of JavaSpaces is superior to Aglets again because of JINI. We also tried to extend the DIAS II and III implementations statically. We found that the difference between Aglets and JavaSpaces was not so substantial for static changes of the agent system.

In the last evaluation criteria the two agent implementations was checked for coverage of DIAS (*E9*). Here Aglets more directly provided the facilities needed to implement DIAS because of direct support of mobile agents. By using JavaSpaces we had to implement an extra infrastructure for receiving and activating moving agents. But if we look at E1 (effort spent), we can see that the JavaSpace implementation required less effort despite the additional elements that had to be implemented.

A summary of the score of DIAS II and III is shown in table

1. Apart from E1 and E3 we have used low, medium and high to indicate the score.

Evaluation criteria	Aglets	JavaSpaces
E1. Effort spent	1280 man-hour	850 man-hour
E2. Lines of code	1073	621
E3. Support for mobile agents	High	Medium
E4. Adaptability	Low	High
E5. Configurability	Low	High
E6. Extensibility	Medium	High
E7. Localisation transparency	Low	High
E8. Abstraction layer	Low	High
E9. Coverage of DIAS	High	Medium

Table 1. Evaluation Summary

5.4 DISCUSSION

If we only look at table 1, JavaSpaces is a superior technology for mobile agents compared to Aglets. It should be noted that our evaluation criteria focus on coping with a changing execution environment, and easy configuration. We did not do any extensive performance tests on moving agents between hosts. However, from initial tests we noticed that it was more resource demanding to run several AMPs at one machine using JavaSpaces compared to Aglets, especially in memory usage. Also moving agents between hosts was slightly faster using Aglets compared to JavaSpaces.

Another issue that should be mentioned is migration of agents over great network distances (through many network nodes). By using Aglets, network distance is not a problem since agents' hosts are explicitly identified. However, JavaSpaces uses JINI's discovery protocol to look for other JINI services in the network. This means that JINI cannot guarantee JINI service discovery beyond LAN. We have tested DIAS III where AMPs were on different networks, and did not had any problems. We also discovered JavaSpaces running in the US and Germany from our network in Norway. But to ensure mobility over larger distances, we had to design and implement some extra services for such communication. This means that for mobility over large distances, the AMPs must know the http address of each other before exchange of agents. This will work similar to the Aglets implementation.

In the previous section we pointed out that there were significant differences in effort required to implement DIAS II and III. We can also see that there is a correlation between lines of code and time spent on implementation. If we also consider the effort to maintain and extend DIAS II and III, we expect the difference to be even more significant. This is due to the fact that less lines of code are much easier to maintain and understand. Since it is also easier to understand, it is easier to expand DIAS III. We believe that the main difference between Aglets and JavaSpaces is the provided transparencies. In section 5.3, we mentioned location transparency as one example. We can also say that JavaSpaces in addition provides network

transparency. The network transparency makes it possible to use network resources without to care about configuration and the handle of network facilities. But the location and network transparencies provided in JavaSpaces can also be a disadvantage if you want to make your own configurations, e.g. to improve the performance or scalability.

6 CONCLUSION

In this paper we have evaluated two different technologies to implement a mobile agent system. We have found that JavaSpaces, not particularly designed to support mobile agents, is a very promising technology in this respect. The main benefits from using the JavaSpaces framework are location transparency, dynamic configuration and a high abstraction level. These benefits make it more efficient to implement and maintain mobile agent systems, and make agent systems that can handle a changing execution environment.

Further work should look more into analysis of performance of the two technologies. Here it is important to investigate the memory and CPU usage required to run agent servers. It is also important to measure how much network resources are required to dispatch an agent from one host to another. Also it is interesting to see what are the minimum required resources to run an agent. Mobile agents could be very useful in an environment consisting of mobile devices such as PDAs and Java enabled mobile phones. However, to be able to execute mobile agents on such devices, it is necessary that the agents and the agent execution environment consume minimal memory and CPU. In the project MOBILE Work Across Heterogeneous Systems (MOWAHS) [12] we will address these issues.

ACKNOWLEDGEMENT

This paper is a result of work in a project called MOBILE Work Across Heterogeneous Systems (MOWAHS) [12] sponsored by the Norwegian Research Council's IKT-2010 program.

We would like to thank Anders Aas Hanssen, Bård Smidsrød Nymoen, Terje Salvesen, and Jan Waage for implementing DIAS. We would also like to thank professor Reidar Conradi for useful comments.

REFERENCES

- [1] E. Freeman, S. Hupfer, and K. Arnold. *JavaSpaces Principles, Patterns, and Practice*. Addison Wesley, June 1999. Sun Microsystems, Inc.
- [2] Anders Aas Hanssen and Bård Smidsrød Nymoen. DIAS II - Distributed Intelligent Agent System II, January 2000. EPOS TR 372 (diploma thesis), 324 p., Dept. of Computer and Information Science.
- [3] Ismail L. and Hagimont D. and Mossiere J. Evaluation of the Mobile Agents Technology and Comparison Studies. Technical report, ESPRIT Project C3DS Deliverable, January 1999.
- [4] Josef Altmann and Franz Gruber and Ludwig Klug and Wolfgang Stockner and Edgar Weippl. Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms. In *2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents*, Montreal, Canada, 29 May 2001.
- [5] Joseph Kiniry and Daniel Zimmerman. A Hands-On Look At Java Mobile Agents. In *IEEE Internet Computing*, pages 21–30. July - August 1997.
- [6] Danny Lange and Mitsuru Oshima. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, 1998.
- [7] Microsoft. Distributed Component Object Model (DCOM). <http://www.microsoft.com/com/tech/dcom.asp>, 1998.
- [8] Sun Microsystems. JavaSpaces TM Specification. White paper, Sun Microsystems, January 25 1999. Available on web: <http://www.sun.com/jini/specs/js.pdf>.
- [9] Sun Microsystems. JavaTM Remote Method Invocation (RMI). <http://java.sun.com/products/jdk/rmi/>, 2002.
- [10] OMG. CORBA BASICS. <http://www.omg.org/gettingstarted/corbafaq.htm>, 2002.
- [11] Geir Prestegård, Anders Aas Hanssen, Snorre Brandstadmoen, and Bård Smidsrød Nymoen. DIAS - Distributed Intelligent Agent System, April 1999. EPOS TR 359 (pre-diploma project thesis), 396 p. + CD, Dept. of Computer and Information Science, NTNU, Trondheim.
- [12] MOWAHS project. MOBILE Work Across Heterogeneous Systems. web: <http://www.mowahs.com>, 2001.
- [13] Terje Salvesen and Jan Waage. DIAS III - Distributed Intelligent Agent System Using JavaSpaces, April 2000. EPOS TR 390 (pre-diploma project thesis), 118 p. + 21 p. App.
- [14] Sun Microsystems. Java[tm] Technology. <http://www.sun.com/software/java/>, 2002.
- [15] Sun Microsystems. Jini[tm] Network Technology. <http://www.sun.com/software/jini/>, 2002.
- [16] Alf Inge Wang. *Using a Mobile, Agent-based Environment to support Cooperative Software Processes*. PhD thesis, Norwegian University of Science and Technology, Dept. of Computer and Information Science, NTNU, Trondheim, Norway, February 5th 2001.
- [17] Alf Inge Wang. Using JavaSpaces to Implement a Mobile Multi-Agent System. In *In Proc. IASTED International Conference on Applied Informatics (AI2002)*, page 6, Innsbruck, Austria, 18-21 February 2002.
- [18] Alf Inge Wang, Chunian Liu, and Reidar Conradi. A Multi-Agent Architecture for Cooperative Software Engineering. In *Proc. of The Eleventh International Conference on Software Engineering and Knowledge Engineering (SEKE'99)*, pages 1–22, Kaiserslautern, Germany, 17-19 June 1999.