# DESIGN PRINCIPLES FOR A MOBILE, MULTI-AGENT ARCHITECTURE FOR COOPERATIVE SOFTWARE ENGINEERING

ALF INGE WANG * ANDERS AAS HANSSEN† BÅRD SMIDSRØD NYMOEN‡
Dept. of Computer and Information Science,
Norwegian University of Science and Technology (NTNU), N-7491 Trondheim, Norway.

## ABSTRACT

The paper describes experiences we have achieved from implementing a mobile multi-agent system for cooperative software engineering, based on the Aglets technology from IBM. When implementing the mobile multi-agent system, we faced problems dealing with locating agents, inter-agent communication, registration of agents etc. Based on our experiences, we present some design principles for how to how to locate agents, how agents should communicate, how to manage connection to the agent system, how to register agents and agent places, how to move agents, how to remove agents, and how to give CORBA-agent interaction support. These design principles should be applicable for others wanting to design mobile multi-agent systems using the Aglets technology.

**Keywords:** Mobile agents, Multi-agent architecture, Software design.

## 1 INTRODUCTION

There are a number of mobile agent technologies available today, both commercial and as research prototypes. Many of these agent technologies only provide a framework for building a mobile agent system, and do not offer extensive solutions for e.g., inter-agent communication, keeping track of agents, locating agents, etc. In this paper we describe some design principles we have extracted from experiences building the mobile multi-agent system called DIAS, based on the mobile agent technology Aglets from IBM. DIAS is one important component for an architecture that supports Cooperative Software Engineering. Agents are used to represent actors in a cooperative effort, and give the users of the agent system support for doing efficient negotiation, cooperation and exchange of data.

When we started building our mobile multi-agent system, we had only a high-level architecture consisting of agents and agent places (more on this in section 2). We wanted to allow both stationary and mobile agents, where the mobile agents could move between agent places. User clients to the system can connect and disconnect to the agent system dynamically, while the rest of the agent system can run continuously. As we started to do low-level design, we discovered that mechanisms for dealing with mobility of agents, and dynamic user client connections were not directly supported by the Aglets framework. These mechanisms have been carefully designed to cope with scalability, security, and stability. The rest of the paper describes the results we gained from our low-level design of the DIAS system.

In [12], Aridor and Lange report several design patterns they have found when creating mobile agent applications. In general, agent design patterns are used to capture good solutions to recurrent problems to make agent applications more flexible, understandable, and reusable. The patterns Aridor and Lange present, focus on agent travelling, agent tasks, and agent interaction. *Travelling patterns* give solutions for routing agents among destinations, forwarding newly arrived agents automatically to another host, and using tickets to encapsulate quality of service and permissions needed to move an agent. *Task patterns* are concerned with the breakdown of tasks and how these tasks are delegated to one or more agents. *Interaction patterns* are concerned with locating agents and facilitating their interactions.

Mobile agents have become a very popular research topic lately. Many technologies facilitate moving objects between hosts. Voyager [3] developed by ObjectSpace is a product family consisting of an ORB and an application server supporting mobile agents. Grasshopper [5] is an agent development platform launched by IKV in 1998. It enables the user to create agent applications enhancing electronic commerce application, dynamic information retrieval, telecommunication services and mobile computing. Grasshopper is completely implemented in Java giving the benefit of high distributed integration. Jumping Beans [1] builds on the Java platform and provides a framework for Java programs to "jump" from computer to computer. The Jumping Beans architecture is based on client-server architecture, where programs moving from one host to another must do this through a central management server. The central management server has a very strict security system. By using a central management server, the Jumping Bean framework is best fitted for small distributed net-

*Phone: +4773594485, Fax:+4773594466, Email: al@finge.com
†BEKK Consulting AS, Palekaia 1, 0150 Oslo, Norway, Email: anders.aas.hanssen@bekk.no
‡Mogul.com, Nedre Bakklandet 60 N-7014 Trondheim, Norway, Email: BardN@numerica-taskon.no

works rather than WANs (bottleneck).

# 2 THE DISTRIBUTED INTELLIGENT AGENT SYSTEM (DIAS)

This section is a short introduction to the Distributed Intelligent Agent System (DIAS) which is a part of CAGIS Multi-Agent Architecture for Cooperative Software Engineering. [11] The CAGIS architecture uses agents to represent co-operative participants in a cooperative effort and supports coordination, negotiation and communication through agents. DIAS provides a foundation for creating a mobile multi-agent system through high-level agent API and multi-agent services. The DIAS architecture is based on a more general multi-agent architecture for supporting a distributed information technology application [9].

## 2.1 DIAS COMPONENTS

We have tried to keep the DIAS architecture simple and it consists of only two main components. By combining these main components, we get a fully functional mobile multi-agent system. The main components are:

- **Agent** A piece of software acting on behalf of a user. The agent is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to the environment, as well as pro-activeness. There are three main groups of agents in DIAS: *System agents, Participation agents*, and *User agents*. System agents administrate the DIAS architecture. We have identified the following types of system agents:
  *Manager agents* are responsible for managing Agent Meeting Places (AMPs), *Facilitator agents* facilitate communication between agents, *Monitoring agents* log events and manage security in AMPs, *Repository agents* retrieve information from, add information to, and query repositories, and *Interface agents* provide and interface between the agent system and other applications.

  Participation agents can either be stationary or mobile, and they provide system support for cooperative processes in agent places. Typical participant agents are:
  *Communication agents* that bring messages or data from one agent to another agent situated on a different agent place, *Negotiation agents* that help other agents to reach an agreement, *Mediation agents* that will act on behalf on higher management and use prior experience to solve locked negotiation processes, and *KQML agents* that make it possible to directly send a KQML message from a user to agent. One advantage with the latter agent, is the ability to communicate directly with other agents without having to implement a specific agent for this purpose.

  User agents are created by a user or by a vendor of agent applications, and can be either mobile or stationary. The DIAS agent API is used by the developer to create user agents.

- **Agent Place** An agent place is where software agents meet and interact in the DIAS architecture. The agent places can be distributed on different hosts, and facilitate means for efficient inter-agent communication based on KQML [2]. Agent places can also have CORBA-support, facilitating external applications to communicate with the agent system through interface agents. The OMGs standard, Mobile Agent System Interoperability Facilities (MASIF), is used to provide a bridge between CORBA applications and DIAS. There are two main types of agent places:

  - *Agent Meeting Place (AMP)* This is the place where the agents advertise their capabilities, communicate with other agents. AMPs are where agents that represent different users can come to and interact. An AMP can be addressed as a DIAS Service Provider (DSP) if an Agent Docking Place (ADP) is connected to it. The AMP will then provide services to the ADP.
  - *Agent Docking Place (ADP)* This is the user-client where the user interacts with his/her agents, and where agents can be created and killed. Agents can also communicate locally in ADPs.

The DIAS architecture was implemented in Java using Aglets Software Development Kit from IBM [8] to provide agent mobility. *Only KQML-layer* in JATLite [6] was used to support the KQML communication language in our architecture. Since the contents of a KQML message can be anything, we have chosen to use XML for this purpose. XML gives us an easy way of representing and wrapping data, and there are several XML tools available for Java as well.

# 3 DESIGN PRINCIPLES FOR MOBILE AGENT SYSTEMS

This section describes the adopted principles in designing a mobile multi-agent system. A more detailed description of our design principles and how these principles are implemented can be found in [4], while an overview of the technology and high-level design issues in DIAS is discussed in [10].

## 3.1 AGENT LOCATION

In a mobile agent system, there most be a mechanism to locate mobile agents. A user agent will start at the users ADP, and can visit to different agent places on different hosts. Participant agents are also mobile and can move between agent places on behalf of a user request or another agent. In DIAS, the ability to communicate with these mobile agents is essential. Thus, a mechanism to locate mobile agents is needed. We have considered three alternatives for locating agents in DIAS:

1. Each agent knows where other agents are situated.

2. Agents make footprints where they go.

3. Agents report to an AMP where they go.

The *first alternative* will require agents to hold much updated information. This will cause problems with performance and scalability, since agents can be mobile. The *second alternative* implies that agents leave footprints to agent places when they are moving to another agent place. When an agent wants to locate a certain agent, it must therefore follow the footprints of this agent. This option can be quite complicated to implement and is not especially efficient. Especially, when agents are killed, all the footprints must be followed and deleted. If an agent has made a far journey, this removal of footprints can be a time and resource consuming process. In DIAS, we have used the *third alternative* to solve the agent location problem. In this approach, an agent must inform his new location to an AMP before moving. We use the term *DIAS Service Provider* (DSP) to name where an agent should report its localisation. By using this alternative, the process of updating agent's locations is distributed to several AMPs. The agent will only inform its new location to the DSP when before moving to another agent place. Replication of DSPs can also be used to achieve better availability of the agent system. Two rules are used to decide the DSP of an agent:

1. Agents created in an AMP will get the current AMP as its DSP.

2. Agents created in an ADP will use the DSP of this ADP. When an ADP is created, an AMP must be specified as DSP. AMPs are used as DSPs because they are persistent while ADPs can be temporary.

All agents have their DSP as a part of their agent ID, making it possible to locate agents' DSPs from an agent ID

## 3.2 AGENT COMMUNICATION

Because it was very hard to combine JATELite's message router with Aglets, we had to create our own agent communication mechanism. When agents communicate, the DIAS system has to locate where the receiver of the message is situated. If communicating agents are at the same agent place, they can exchange messages locally. If communicating agents are on different agent places, the message will be carried by a communicating agent. Here is a simple example showing Agent A at AMP1 sending a message to Agent B located at ADP1. Note that system agents (not shown in the figure) are used to bring the message from Agent A to Agent B.

These steps describe figure 1:

1. The message from Agent A to Agent B is first sent to a system agent locally at AMP1.

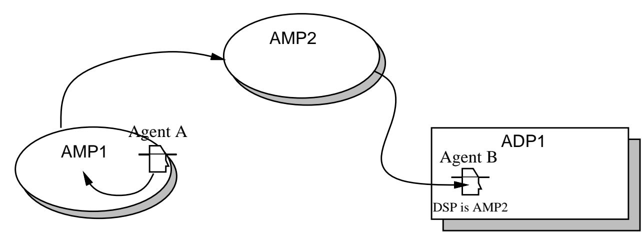


*Figure 1. Communication example*

2. A system agent at AMP1 finds the DPS address of agent B, and sends the message to this DSP (here AMP2).

3. A system agent at AMP2 looks up the location of Agent B, and sends the message directly to Agent B on ADP1.

4. Agent B replies the message with an acknowledgement (sent directly back to Agent A)

As the example above illustrates, the DSP of the receiving agent is used to find this agent. AMPs are dynamically exchanging ID information about agents in the system. AMPs that also are DSPs, hold information about agents whereabout. There are two types of agents that are important when communicating in DIAS:

- **Facilitator Agent** Every AMP and ADP has a facilitator agent. All messages in the DIAS system have to be sent through the facilitator agent, deciding if a message should go directly to the receiver agent (locally) or via communication agents to find the correct remote receiver.

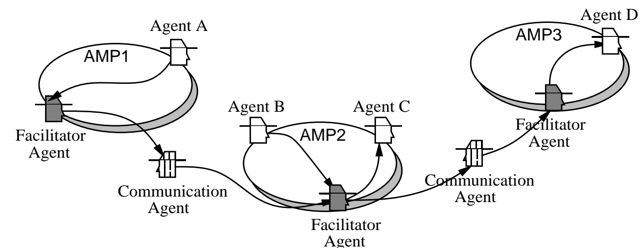


*Figure 2. An example of how the facilitator agent works*

Figure 2 illustrates two examples for how facilitator agents are used in DIAS. First in figure 2, Agent A want to send a message to agent D. The facilitator agent in AMP1, forwards the message using a communication agent to AMP2. The facilitator agent in AMP2 then forwards the message to AMP3, where AMP3's facilitator agent gives the message to the requested agent D. The second example in the same figure is when agent B wants to send a message to agent C. The facilitator agent can then just give the message to agent C, since both are located in AMP2. The facilitator agent will act according to available receiver agent information, which is the agent identifier, agent's DSP identifier and ontology. If the agent and DSP identifiers are missing, the facilitator agent tries to find an agent with matching ontology.

- **The Communication Agent** The communication agent is responsible for bringing a message from one agent place to another. A communication agent is triggered by a facilitator agent, carries the message to the target destination, and delivers the message to facilitator at the target agent place.

## 3.3 CONNECTION OF AGENTS AND AGENT PLACES

Users of the DIAS are not connected to the agent system all the time, therefor mechanisms for handling user connection and disconnection are needed. When a user wants to connect to the agent system, an ADP (user client) must be initiated. The ADP must then connect itself to an AMP to establish the connection with the rest of the agent system. This AMP will provide services to the ADP, and thus is called DIAS Service Provider (DSP). An illustration of how an ADP1 connects to AMP2 is shown in figure 3.

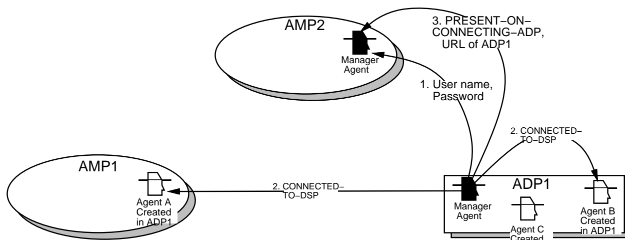


*Figure 3. Illustration connection between ADP and AMP*

The procedure used to handle connection of ADPs, ensures that all parties influenced by the connection are notified. The procedure follows three steps:

1. The connecting ADP1 sends user name and password to the Manager Agent in AMP2 (DSP of ADP1).
2. CONNECTED-TO-DSP tells all agents created in the ADP1 that this ADP is connected to the agent system. This notification tells agents in other agent places (like Agent A in figure 3) that it is possible to return home to ADP1.
3. The Manager Agent at the DSP (AMP2) will then receive the message PRESENT-ON-CONNECTING-ADP, along with all needed information from ADP's Manager Agent (the ADP's ID etc.).

When an ADP is disconnecting, the agent system must make sure that all involved parties (agents and agent places) are notified, and status information must be updated. A procedure involving three steps is used when an ADP disconnects:

1. DISCONNECTED-FROM-DSP notifies agents created in the ADP that the ADP is going to disconnect. This makes it possible for these agents to return to home, before the connection with the user is closed down. The user can configure the ADP to wait for a certain time (default five seconds) before it will close down.
2. PRESENT-ON-DISCONNECTING-ADP is sent to all agents that are present on the disconnecting ADP, to notify that they have to move themselves, or they will be stopped.
3. DISCONNECT-FROM-DSP tells the DSP's Manager Agent that the ADP is going to disconnect. This Manager Agent can then delete registered information about the ADP and disconnect it.

## 3.4 REGISTRATION OF AGENTS AND AGENT PLACES

In DIAS, agent places have registries about agents and other agent places in the system. These registries are essential to make it possible for agents to communicate, and they contain information about ID for agents and agent places. The registration of agents, ADPs and AMPs is done as following respectively:

- **Registration of agents** First, agents will be registered as *origin agent* in the agent place they were created . The origin agent registry hold information about the agent ID, and it is used to inform agents (listed in origin agent registry) that an ADP has moved. A user can choose to start his/her ADP at another site, and the users agents must be informed about current location of the ADP. Second, all agents living in the agent system should be registered as *registered agents* in agent places in the system. This registry makes it possible to get an overview of agents in the system. If agents also have this agent place as a DIAS Service Provider (DSP), agent location is also registered. Third, agents present at an agent place are registered as *present agents* as well. When the facilitator agent is looking for a receiver of a message, the present agent registry is checked first to see if the message can be delivered locally.
- **Registration of ADPs** A ADP can is only registered in DSPs as "registered ADP", where general information about the ADP, information about location of the ADP, and ADP's user information is given.
- **Registration of AMPs** When an AMP wants to register in another AMP, both AMPs exchange information about themselves. In this way the AMPs will have information about each other and be able to recommend each other to agents searching for an AMP hosting a specified agent with given ontology.

## 3.5 MOVING AGENTS

In DIAS, mechanisms for moving agents between hosts are supported in the underlying technology provided by Aglets from IBM. However, the system must also be

aware of agents' location any time. When an agent is moved to another agent place, the agent has to give information to three agent places: (1) To the DSP about where it will move, (2) to the agent place the agent will move from that it has left, and to the agent place the agent will move to that it has arrived. An agent must be registered in the agent place it tries to move to. If not, the agent tries to register itself before moving.

### 3.6 REMOVING AGENTS

In DIAS, agents can be created and killed at run-time. However, system agents cannot be killed individually since they must provide their services as long the agent place is running. If the agent places are shut down, the system agents associated with this agent place will also be killed. Participation agents have often a dynamic nature, and can be created and killed on demand. Since these agents are not registered any places, they can be killed immediately. The user agents' destiny is decided by the user, and it is up to the user to kill them. These agents will unregister themselves from agent places before they are killed.

### 3.7 CORBA AGENT INTERACTION

DIAS offers external programs to access the multi-agent system through CORBA to enable programs in programming languages other than Java or other agent systems to interact with DIAS. In addition, it should be possible to connect to an AMP without having a local ADP installed on the host. A CORBA client can either interact directly with a CORBA AMP, or interact with a CORBA ADP. The former means that we do not need an ADP installed on the host, while the latter is a practical solution, since both the CORBA programs will be on the same host as the ADP.

The *Interface Agent* plays the main role when CORBA programs interact with DIAS. This agent is created as a system agent in its agent place, and connects to the ORB in the agent place when created. In this way the Interface Agent is an agent as well as being a CORBA object on the ORB. The following interfaces are used for CORBA interaction:

- **MAF AgentSystem interface** Defines agent operations like receive, create, suspend, and terminate.
- **MAF Finder interface** Defines operations for registering and unregistering of agents, and finding the location of agents, places and agent systems.
- **DIAS interface** Defines the communication method used by CORBA applications (can be extended by the developer)

The Interface Agent is the "glue" between the CORBA client and the agents. The operations in the Interface Agent are invoked by the CORBA client. In DIAS, KQML is used to express the communication between agents. When a CORBA client wants to send a KQML message to an agent place (e.g., to initiate an agent) the following will happen:

1. The KQML message is translated to a string.
2. A CORBA call with the KQML string as a parameter is executed.
3. The CORBA object (the Interface Agent) on the CORBA server will then translate the KQML string into a KQML message.
4. The KQML message will then be sent from the Interface Agent to the receiver agent.

## 4 DISCUSSION

The Aglet framework [8] provides underlying services for building a mobile multi-agent system, but it lacks of high-level services to enable a fully functional agent system. The design principles in section 3 describe how we added these high-level services to provide a better way to manage mobility and distribution of agents, as well as agents interaction. Dealing with mobility and distribution is mainly a technical problem to ensure that the system is reliable and scalable. However, agent interaction, it is not only a technical problem. Our current implementation provides only simple means for agent interaction, and should be replaced with a more advanced mechanism for dealing with this problem. Domain models should be used as ontology to define as limited set of words agents can understand and how these words are related and used. We are currently working on incorporating a document model for modelling ontology. In [7], Gulla and Brasethvik offer a framework for creating a graphical model of an ontology and for exporting this model as an XML document. A starting-point for an ontology model is generated using a tool that parse through documents typical for the ontology domain chosen, resulting in a list of domain specific words. A graphical modelling tool can then be used to refine the ontology model and define relationships between words. The last step is to export the ontology model to an XML document that can be used in DIAS.

## 5 CONCLUSION

In this paper, we have described some design principles applicable when designing a mobile multi-agent system using Aglets technology. The principles are not closely related to the underlying technology, and could therefore also be used for other agent technologies. We found that our design principles will make it easier to generate agent applications, since the agent system takes care of things as routing messages, locating agents, connecting and disconnecting user clients (ADPs). We are now exploring our agent system, building agent applications, to gain more experiences of what advantages and shortcomings our architecture has.

## ACKNOWLEDGEMENT

We would like to thank Reidar Conradi for useful comments and suggestions. The CAGIS project is sponsored by the Norwegian Research Council's Distributed InformationSystems (DITS) Programme.

## REFERENCES

[1] Ad Astra Engineering Inc. Jumping Beans - The Mobility Framework. web: http://www.JumpingBeans.com, 1999.

[2] Tim Finin, Yannis Labrou, and James Mayfield. *Software Agents*, chapter KQML as an agent communication language. MIT Press, Cambridge, 1997. Ed. Jeff Bradshaw.

[3] Graham Glass. ObjectSpace, Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing. White paper, ObjectSpace, 1999. Availeble on web: http://www.objectspace.com/products /documentation/VoyagerOverview.pdf.

[4] Anders Aas Hanssen and Bård Smidsrød Nymoen. DIAS II - Distributed Intelligent Agent System II. Technical report, Norwegian University of Science and Technology (NTNU), January 2000. Technical Report, Dept. of Computer and Information Science.

[5] IKV. GrassHopper - The Agent Platform. web: http://www.ikv.de/products/grasshopper/, 2000.

[6] Heecheol Jeon, Charles Petrie, and Mark R. Cutkosky. JATLite: A Java Agent Infrastructure with Message Routing. *IEEE Internet Computing*, 4(2), March/April 2000.

[7] Jon Atle Gulla and Terje Brasethvik. On the Challenges of Business Modeling in Large-Scale Reengineering Projects. In Chen and Embley and Kouloumdjian and Liddle and Roddick, editor, *Fourth International Conference on Requirements Engineering (ICRE'2000)*. Schaumburg, Illinois, June 2000.

[8] Danny Lange and Mitsuru Oshima. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, 1998.

[9] M. Matskin, M. Divitini, and S. Petersen. An Architecture for Multi-Agent Support in a Distributed Information Technology Application. In *International Workshop on Intelligent Agents in Information and Process Management*, page 12, Bremen, Germany, 15-17 September 1998.

[10] Alf Inge Wang. Experience paper: Implementing a Multi-Agent Architecture for Cooperative Software Engineering. In *Twelfth International Conference on Software Engineering and Knowledge Engineering (SEKE'2000)*, Chicago, USA, 6-8 July 2000.

[11] Alf Inge Wang, Chunnian 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.

[12] Yariv Aridor and Danny B. Lange. Agent Design Patterns: Elements of Agent Application Design. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 108–115, St. Paul, Minnepolis, USA, May 9-13 1998. ACM Press New York.