Master's Thesis Report

# Data Collection in Distributed Command and Control Systems with Mobile Agents

**by**


## Andreas Johansson



LiTH-IDA-Ex-00/65

2000-06-09

Master's Thesis Report

# Data Collection in Distributed Command and Control Systems with Mobile Agents

**by**


## Andreas Johansson



LiTH-IDA-Ex-00/65



Supervisors: Per Wikberg and Håkan Söderberg
Examiner: Nancy Reed



2000-06-09

FOA
The Swedish National Defence
Research Establishment

Linköping, June 2000
LiTH-IDA-Ex-00/65

# Data Collection in Distributed Command and Control Systems with Mobile Agents

by
Andreas Johansson

Version 0.9992ida

# Abstract

In modern military command and control, the demands for faster analysis and less expensive exercises grows all the time. One result of this is the desire to get as much valuable knowledge out of each exercise as possible for evaluation purposes. The log files in the military Command and Control systems are one area with a lot of rarely used data.

To extract this information different techniques can be used. *Mobile agents* are one approach that has been used increasingly in the last few years. This report describes a feasibility study on the use of mobile agents in distributed military Command and Control systems for acquiring and analysing data in log files. Different commercial off the shelf agent development tools where tested to examine the mobile agent technology and evaluate if and how it can be used to collect data from log files in distributed military Command and Control systems.

The result shows that the mobile agent concept is promising but that further work is needed. The mobile agent technique has benefits compared to the usual client-server model, especially from a military perspective.

This work was done at FOA (the Swedish National Defence Research Establishment) in the Division of Human Sciences and resulted in prototype agents and this report.

# Sammanfattning

I modern militär ledning så ökar ständigt kraven på snabbare analyser och mindre kostsamma övningar. Ett resultat av detta är önskan att från de genomförda övningarna få ut så mycket värdefull kunskap som möjligt för utvärderingsändamål. Loggfilerna i de militära ledningssystemen är ett område med mängder av sällan utnyttjad data.

För att få fram denna information kan olika tekniker användas. *Mobila agenter* är ett tillvägagångssätt som har börjat att användas allt mer under de senaste åren. Denna rapport beskriver en genomförbarhetsstudie av användandet av mobila agenter i distribuerade ledningssystem för att förvärva och analysera data i loggfiler. Några olika kommersiella agentutvecklingsverktyg har testats för att utforska den mobila agentteknologin och utvärdera om och hur denna kan användas för att samla data från loggfiler i distribuerade ledningsystem.

Resultatet visar att det mobila agentbegreppet är mycket lovande men att ytterligare studier behövs. Mobil agentteknik har fördelar jämfört med den traditionella klient-server modellen, speciellt utifrån ett militärt perspektiv.

Detta arbete har gjorts på FOA (Försvarets forskningsanstalt) på avdelningen för humanvetenskap och resulterade i prototypagenter och denna rapport.

# Table of contents

# 1 Introduction

Today's military are no longer based on the same premises and prerequisites as they were 50 years ago. From once being categorised by its size in number of men and that a war could last for tens of years, the situation is now different. Through technical developments, wars are now shorter and faster, meaning that the Command and Control[1] processes for the military must also become faster. What matters now is that the commanding staff has a good view of the situation and therefore can make good and fast decisions and inform the concerned units and people.

## 1.1 Background

This work was done at FOA (the Swedish National Defence Research Establishment) in Linköping. FOA is a research institute that mostly does research for the Swedish defence industry and have over 700 researchers in academic positions.

The thesis work was done in the Division of Human Sciences and within the project *LedFram*. *LedFram* is a project that performs research on support for the development of Command and Control systems for military purposes. The emphasis is on improving current systems and developing better systems in the long term.

## 1.2 Purpose

The purpose of this work is to perform a feasibility study to explore and evaluate the mobile agent concept for use in distributed Command and Control systems for data collection. The focus will be to enhance the information gained and the utility of the exercises performed. One part of the work is also to evaluate some development tools for creating mobile agents and see how they perform in a similar environment.

Our aim is to produce some simple agent prototypes, with the purpose to highlight possible benefits, shortcomings and possibilities/difficulties when using mobile agents and mobile agent development tools. We will not focus on the security aspects in this work. The main purpose for the work is as a basis for further work with this report as the main source.

## 1.3 Methodology

First the agent domain will be studied with emphasis on mobile agents. Then the different agent development tools will be described and evaluated. To do that some example agents will be defined and created with the tools. Finally the results and conclusions will be analysed and discussed in this report.

[1] Command and Control will be abbreviated to C². In the rest of the document all abbreviations and descriptions can be found in appendix A.

# 2 Military domain

The military domain has its own demands, rules and problems. This chapter tries to present an overview of them.

## 2.1 Command and Control systems

Military $C^2$-systems are used to command the units in the field. The systems are often a combination of different hardware and software tailored for the military. In the last few years COTS-products (Commercial off the shelf products) have become increasingly important to use as modules to lower development costs of these systems [National research council, 1998].

As the military becomes even more computerised the systems become more and more complex and their users are spread over larger and larger geographical areas. This is also an effect of the new kind of mobile military units and the new kind of faster command chains that are needed. Because of this, it is becoming increasingly hard to effectively validate and verify the $C^2$-systems and the way they are used.

The C²-systems are both heterogeneous, made up of several different systems from different manufactures and years, and distributed which makes it hard to survey what is happening in the complete technical system.

## 2.2 Model verification

To be able to understand and adapt to future warfare different business models are being developed. These business models contain goals, objects and processes included in modern command, with the purpose of specifying future C²-systems.

If the business models should be developed further they must be verified and validated. Since it is hard to compare them with future warfare they are compared with today's warfare instead. The primary business model used in this work is HP ATLE's model *Markstrid 5.0* that concerns ground battle [Gardelius, 2000].

## 2.3 Military exercises

The military not only has exercises to train their units but it is of equal importance to train the command staff that leads the units. This training is done on so-called development exercises. During these exercises the evaluation of the business models are also conducted.

These exercises are run and supervised by staff instructors. These are experienced officers that create and afterwards evaluate the scenario that the officers being trained will be faced with. But what is even more important is that they draw conclusions about how well the trained officers performed in different situations and what could be improved.

During the exercise the staff instructors observe the activities and ask questions to get the needed information to further develop the models, methods and tactics that form the base of the command and control work procedures.

The development exercises are often simulated, i.e. there are no, or only a few, real units in the field and the rest are simulated either by computer systems or staff instructors.

One problem is to get enough facts about the outcome of the exercise to be able to draw any conclusions about the models. Today this is done mostly by interviewing the staff instructors and letting them fill in questionnaires. The problem with this is that it is hard or maybe even impossible to get all the relevant information. Another problem with this approach is that it is very time consuming both for the staff instructors and the developers of the models.

## 2.4 Staff instructors

The questions that staff instructors wish to answer are many and are of varied kinds. These can include how the system performs, who is sending which document to whom in the $C^2$-network, how long that takes, what happens with the document, etc. Of course all of these questions can be tied to the business model *Markstrid 5.0* on some level.

### 2.4.1 Questions tied to the model

How can questions derived from the model be answered with the help of log files? And is it possible to use a log alone instead of observations or maybe as a complement to them? The questions below are a few selected from a report template [Wikberg et.al, 2000] and freely translated from Swedish. The numbering (A2 etc.) corresponds to the numbering of the questions in the report template.

A2. To what extent does the right recipient assesses the right information at the right time?

A6. To what extent are defined consequences relevant with regard to predetermined goals?

A9. To what extent is the method for scrutinisation and establishment of different alternatives chosen with regards to disposable time?

A12. To what extent are execution orders about suppression acknowledged?

Questions A2 and A12 seem more easily answered through computer analysis of data while the other two questions seem more difficult to retrieve from log files.

The examples above show that not all questions can be answered with only the data from log files. Often some human must analyse the data and draw conclusions (like in A6). Other questions are more easily and faster collected from log files by a program then by manually searching (like in A12).

### 2.4.2 Obstacles for the staff instructors

The time pressure on the staff instructors is usually high during the development exercises, since there are many questions that need follow up. Often the questions also require high mobility of the staff instructors, as they usually need to visit several geographically different locations to be able to answer the questions. This can be a big problem since the exercise might be spread over a very large area, sometimes up to 400 square kilometres.

Even if the staff instructors are at the right location at the right moment it still might be the case that the question can not be answered. That can depend on many factors, the staff instructor does not want to interrupt a pressured decision-maker, it can be hard to observe the staff members as they work in a narrow staff vehicle or difficulties with observing who is sending what to whom in the $C^2$-system. But to be able to conduct correct observations and to be able to fully answer the questions it is often a necessity

that the staff instructor knows what is happening in the system. This information is often available today in different log files but not easy accessible to the ones who need it, the staff instructors.

One reason for this is that a high level of computer knowledge is often needed to retrieve information from a system (for example a database). The interfaces are often not especially user friendly and the use of them might even require programming knowledge (like asking advanced SQL-questions). But to put the right question to the system, knowledge about the military activities is necessary. These two skills are not often combined in the same person.

## 2.5 Log files

Since the exercises are carried out with the help of computer tools and software, a lot of log files about what has been done with the systems are created. In these large files there is a lot of information that could be used to get an understanding of what has happened. The problem with log files is that they are very hard to understand by reading them manually and it also consumes a lot of time to do so.

They might also be saved in some special format or in a database and if so, reading them would require specialist knowledge. Different systems also produce different kinds of log files for different purposes, which entails integration difficulties. It is also hard to "get the whole picture" from log files, even if the information exists in them and they are in a regular, readable text format and not so long. This can be seen in a demonstration log file in appendix B.

## 2.6 Summary of problems

To be able to get the results from the development exercises, more efficiently and faster, tools are needed to collect and present the large amount of digital information that is stored automatically. The problem today is that no good tools exist either to collect or to present the log files that are generated during the exercises. This problem is growing all the time as systems are computerised and the log files grow larger and larger because more details are logged and more complex manoeuvres are performed [Watson R, 1996].

One problem with collecting data from the log files is that they are scattered on many different (heterogeneous) computer systems distributed in many different places. Some files can for example be located locally on a computer in a staff vehicle that lacks constant connection to the other units. That can be because they often have to regroup to other locations and then temporarily break the connections or because they want to keep a low profile so the enemy will not spot them and therefore deliberately break the connections with the other units. This means that all computers are not necessarily available all the time or can only be contacted for a very short period of time and then the connection is lost again.

What would be needed is some tool to get the useful information out of all the big log files. And to be useful that tool must have an easy to use interface. It could be the same as a search engine on the Internet where one enters one or several search words and gets a result based on those words.

It could also be desirable if it was possible to "browse the $C^2$-system" and with simple means during times of high pressure and without expert knowledge, ask all types of "ad hoc questions" about the events, document handling, etc.

# 3 The agent concept

To try to solve the problems outlined in the previous chapter, the use of software agents will be evaluated. But what is an agent?

## 3.1 Agents

There exist many definitions of the concept of agents. Milojičić [Milojičić et.al, 1999] defines agents according to their attributes: *"Active, autonomous, goal-driven, and typically acting on behalf of a user or another agent."* An agent should therefore have a goal or task that it independently, without user guidance, tries to solve on behalf and in the interest of someone else.

Of course this is just one definition of a *software agent* (hereafter just called agent) and almost every researcher has their own definition. Agents can also be characterised more specifically according to their emphasise. If they collaborate with each other to solve their task they can be called *collaborative agents*, and if they can move around they can be called *mobile agents*. A longer list can be found in [Nwana & Ndumu, 1998].

## 3.2 Mobile agents

A *mobile agent* is an agent that independently and autonomously (by itself) moves from one computer to another in a network to perform its task. Compared to a regular data transmission it is not just transferring its code and its data but most importantly, it starts to run again when it reaches each destination.

One task for a mobile agent could be to find some data in log files distributed at several different locations as mentioned in the previous chapter. One benefit with using mobile agents is that they can be sent away when a connection is present, then the connection can be broken, but the agent remains at the site where the data is stored so it can still continue to work. When a connection is re-established the agent can return and present what it has found as a web page, for example. This is a large benefit compared to the client-server approach where there must exist a constant connection for the same thing to work.

For this to work a so-called *Agent platform* is needed. An *agent platform* is a server that must run on every computer that should be able to send or receive mobile agents. The agents run on top, or inside, of this server. When an agent is finished at one host and should be transferred this is handled solely by the two agent platforms at the sending and receiving hosts. Mobile agents can therefore not be received unless there is an agent platform running at the receiving host, this is done for both practical and security reasons. For some words about security concerning mobile agents, see *8.4 Security of agents*.

## 3.3 Agent development tools

Mobile agent development systems are tools to develop and run distributed applications by creating mobile agents. The agents are often small Java programs that travel in the network to perform their function. They can be tailored to perform a specific task or be made more general to be able to cope with different situations.

There exist quite a lot of agent development tools, both to create static and mobile agents depending on the purpose of the agent. Since this work will focus on mobility the static agents systems are of less importance and will not be discussed further. What is usually included with mobile agent development tools is the agent platform to run the

agents and some programs to create the agents alternatively some API to add agent functionality to your own programs.

The basis for mobile agent technology includes three important things [Cockayne & Zyda, 1997]. The first is a programming language that makes the development of agents possible and allows them to be co-ordinated with an agent platform and to be used in a system. The majority of the agent developer tools commonly use Java for this purpose. The second is an execution environment (in the case of Java, a "Virtual Machine") that can run the developed code. And the last one is a communication protocol (TCP/IP or RMI for example) that connects different systems and gives the agent the ability to move between or within systems.

# 4 Experimental Method

The goal of this work is specified and initial delimitations are given in this section.

## 4.1 Goals

The purpose of this work is to perform a feasibility study to explore and evaluate the mobile agent concept. The focus will be on evaluating if mobile agents are suitable to be used as a support mechanism for data collection in distributed military $C^2$-systems. The reason for this is to enhance the information gained and the utility of the exercises performed. Another goal is also to test and evaluate some COTS-products for developing mobile agents and see how they perform. The main purpose for the work is as a basis for further work in the area.

### 4.1.1 Performance

The purpose of this work is to evaluate mobile agent technology and evaluate if and how it can be used to collect data from log files in distributed military $C^2$-systems. The work will produce some simple agent prototypes, with the purpose of highlighting possible benefits, shortcomings and possibilities/difficulties when using mobile agents and mobile agent development tools.

To fulfil the purpose, this work has been divided into three parts:

- Investigate how well COTS-products can be used to construct mobile agents and describe advantages and drawbacks with using them.
- Through making some small prototype agents, evaluate the agent concept and decide if it is suitable for data collection in an experimental demonstration distributed $C^2$-systems.
- Discuss the results of the two points above from the military point of view.

To test this, a scenario will be designed and tested concerning data collection related to military $C^2$-issues. After that, one or several agents will be created that can solve different types of search questions. The important part is the methods and concepts to examine what can be done with mobile agents and how it can be done (with focus on the data collection).

### 4.1.2 Military benefits

To effectively use the stored information in the log files some better method than just letting the staff instructors find and open the files manually and then read them must be used. The exercises are often very expensive because there are a lot of people needed to perform a realistic exercise. Often the officers are also from different places in the country. For these reasons the exercises are short and intense, which means that the staff instructors have a lot to do and that the military wants to get as much feedback as possible from each exercise. Here, a mobile search agent can help the staff instructor to collect information that normally is just left unused because of time pressure or lack of necessary technical knowledge.

### 4.1.3 Evaluation scenario

To be able to in an equivalent way compare agents created with different COTS-products a scenario was constructed. It includes creating one or a few different mobile

agents and testing their abilities. The primary one is a search agent that will take a word to search for and move over to another computer. There it will search through the text files on that computer and retrieve a list of the files that contain the search word and return to the sender and present the result. The data collection questions in the scenario correspond to questions that staff instructors on a tactical level are believed to ask.

## 4.2 Initial delimitation

The product AgentBuilder was predetermined to be the platform for developing the agents. Another delimitation is that the programming language Java should be used as the implementation language for the agents. The agents created would only be run in an experimental local network and not in the actual C²-system. They will therefore not solve the current problems the military struggles with; the intention is to help in the long term with this study.

The work will result in a prototype used to highlight some benefits and possibilities of using agents. It will not be a complete system and therefore it will not have all the functionality, error control, security features, etc that a real system must have. Also the presentation of the collected data will not be dealt with since that is covered by another thesis project [Albinsson 2000].

## 4.3 Literature study

To get to know the area a literature study was performed, that included reading different books as well as articles about agents and agent development with focus on COTS-products. The main emphasis has been on real-time systems and also integration of old and new systems. Other literature used was Java reference literature [Weber, 1998] and user guides for the different tools. The Internet was also used as an information source; particularly the tool's home pages were frequently used. Also articles regarding military subjects were read to get an overview of the area.

## 4.4 Environment

The development and testing of the agents was primary made on a Dell Workstation 400 with Windows NT 4.0. To be able to send the agents out to different computers, four more computers were used. All of them had Windows 98 and were Pentium II or Pentium III Dell machines. The computers were connected through FOA's internal network. The software used was AgentBuilder Lite 1.2 from Reticular Systems Inc, Visual Café 4 from Symantec. And later on Concordia 1.1.4 (build 34) from Mitsubishi Electric Information Technology Center America and Jess 5.0 by Ernest Friedman-Hill at Sandia National Laboratories.

# 5 COTS software evaluations

The COTS-products that were tested are AgentBuilder and Concordia which both are agent development systems and Jess, which is a rule-based expert system. Both the agent development systems claimed to be able to make mobile agents and they were tested by running the supplied agents and by creating example agents. The impressions and pros and cons with each of them are in the following sections. Also comments regarding implementation will be given where appropriate.

## 5.1 AgentBuilder

AgentBuilder is a system made by Reticular Systems Inc [Reticular, 1999] to create agents. This is done by using a semi-graphical develop environment. In other words, to create agents a combination of traditional programming and graphical "select and fill in fields" are used. For more discussion about how this is done see AgentBuilder's user guide [Reticular, 1999]. Figure 1 shows a snapshot of AgentBuilder's development environment.

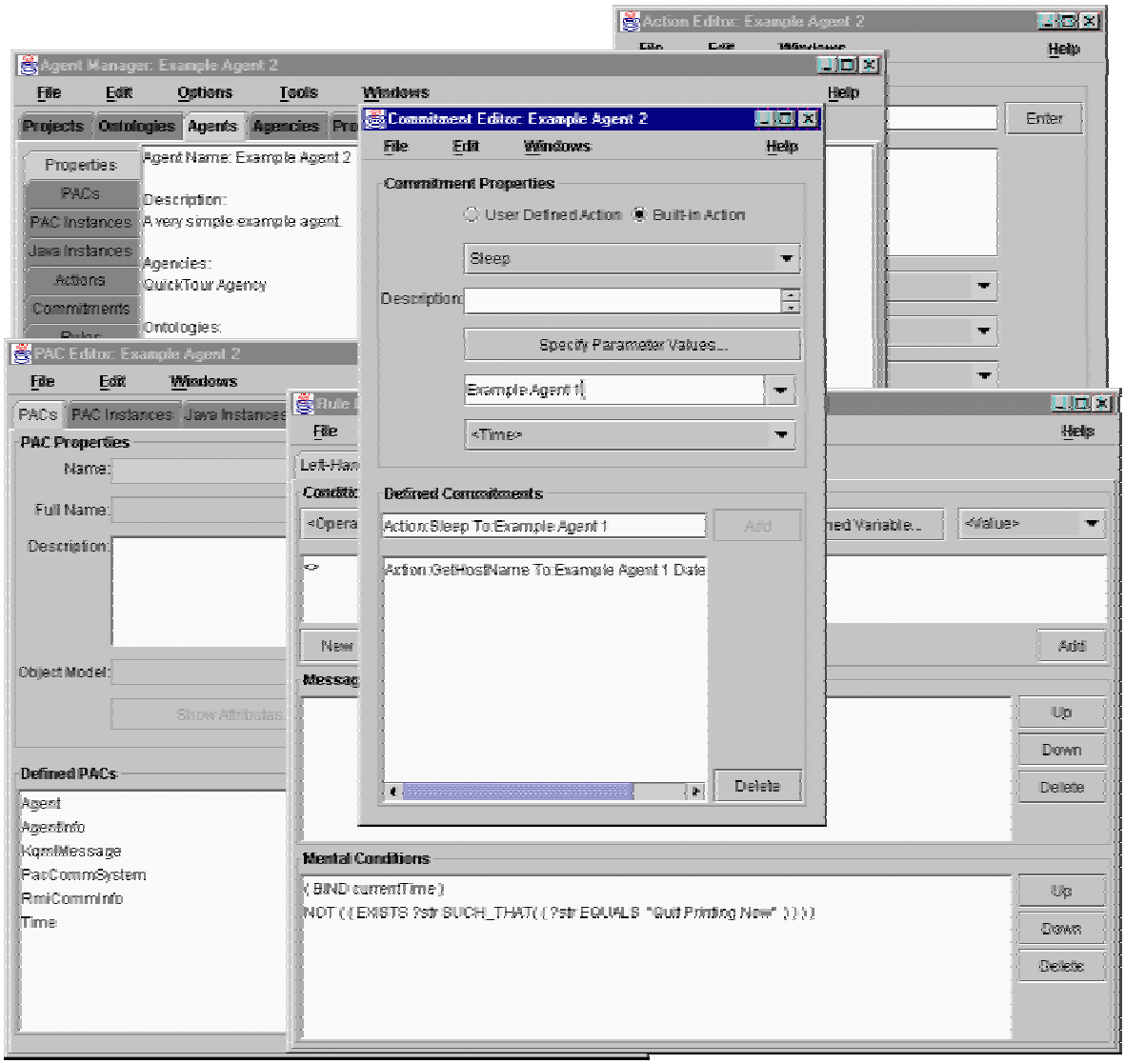


**Figure 1.** *AgentBuilder opened with different windows.*

First AgentBuilder was installed and tested by running the accompanying example agents. The agents that were run were "Hello World" and "Example agent 1" to "Example agent 4", all supplied with the product. After that was done the enclosed tutorial was followed. According to a step by step instruction the user created the same agents as the example agents to see how it is done.

There were some problems using AgentBuilder in general. But the worst problem was that the AgentBuilder version tested (AgentBuilder Lite 1.2) did not support mobility at all, contradictory to what they claimed. With AgentBuilder you can only make static agents and not mobile agents and it must therefore be classed as a static (planning) agent system. But this is not enough according to the specified demands and AgentBuilder therefore lacks support for an essential capability for solving the problem.

The following points are some problems that were discovered during its usage:

- The tool is a bit awkward to use. There is a lot of clicking to get anything done and as a user, you neither get a good overview of the system nor of how to build an agent.
- The help system does not work (in the version used). When using it the program crashes. That can be solved by reading the help files with a browser. But then you miss the context sensitive help.
- You cannot create or remove agents dynamically so all agents must exist from the beginning. This is a problem if you want the agent to be able to clone itself and then transfer to different computers to more quickly or more flexibly solve its task.
- AgentBuilder can not automatically generate all necessary code and it does not hide the implementation details. That means that one must fill in the missing parts for the tool or write some code to get a functioning agent. If the coding of the functions for the agent had been separated from the behaviour specification, it could have been a very good system. Now programming skills are needed (to a certain degree) to use the tool.
- An IP-address (which is static during the lifetime of the agent) can be given but it is only used if an agent wants to communicate with other agents. This led to confusion in the beginning when we were under the impression that the agents created were able to be mobile.
- The communication is made through KQML, which at first seems to be a good thing as it is standardised, but according to *Jönsson and Nordberg* [Jönsson & Nordberg, 2000] this could be changed in the future with the advent of XML.
- The enclosed example agents only did quite simple things (for example print out a message when a time interval has passed). However, even if they where simple there was a great deal of work to create them when following the tutorials.
- Several bugs were also found, mostly GUI related.

Although this shows a lot of negative things with AgentBuilder there exists good things also. The main one is that AgentBuilder uses abstract terms and tools. Instead of specifying exactly what to do in the code one specifies rules and commitments for the agent through different managers and editors (for example the *Ontology Manager* or the *Rule Editor*). KQML is used which right now is the standard for agent communication and there are several finished modules that you can buy and use in your agents.

We did not use AgentBuilder much after we concluded that it did not meet the requirements for mobile agents. Since many good things about a tool are discovered

during its usage, they are not reported here. To create a static (planning) agent you can possibly try out AgentBuilder, but we should suggest to also testing other alternatives as well, since they might be better. But for the purpose of developing mobile agents AgentBuilder is of no use at all. Therefore the conclusion must be, despite the initial delimitation that AgentBuilder should be used, that another agent development system for this project must be chosen.

## 5.2 Concordia

Concordia is a commercial product made by Mitsubishi Electric Information Technology Center America [Mitsubishi Electric, 2000] designed as a platform and development tool to create mobile agents. The version used (1.1.4) is an evaluation kit (that does not expire) and it has almost all the features that the full product has.

AgentBuilder is a more complex system than Concordia since it has reasoning and is partly GUI-based for making the agents. Concordia on the other hand is just a Java API in a package that is imported when the agents are coded. Almost the only requirement is that the class for the agent that the programmer creates must be inherited from Concordia's *Agent* class.

Why was Concordia chosen then? There exists many other agent development tools on the market. Just to name a few of them they are *Aglets*, *Grasshopper* and *Zeus*. A longer list with all the links can be found at the AgentBuilder's site [AgentTools, 2000]. I only tested a few of them and therefore can not say that Concordia is the best choice. The first suitable one found was chosen and that was Concordia. To put a lot of effort scanning the very turbulent agent development tools market would have conflicted with the goals of the project. Concordia works and has a simple to use API so it suited this initial project well.

### 5.2.1 Overview

To get a better idea of what Concordia is Mitsubishi describes it on their homepage [Mitsubishi Electric, 2000] in the following way:

> Concordia is a full-featured framework for development and management of network-efficient mobile agent applications for accessing information anytime, anywhere and on any device supporting Java. With Concordia, applications:
>
> - Process data at the data source
> - Process data even if the user is disconnected from the network
> - Access and deliver information across multiple networks (LANs, Intranets and Internet)
> - Use wire-line or wireless communication
> - Support multiple client devices, such as Desktop Computers, PDAs, Notebook Computers, and Smart Phones

As stated above with Concordia you are fully able to create agents that can move around in the network. The itinerary for the agent can be changed at any time and there is no limit on the number of hosts an agent can visit. It is possible to both create single agents and multiple agents that communicate to solve their task. Only single agents were tested in this work. Mitsubishi also has a full version that supports security, reliability and better administration of the agents, which would be needed in a final system. But of course

other systems can also be investigated and more thoroughly tested. For security issues, see *8.4 Security of agents*. The server control panel for the evaluation version is shown in figure 2.

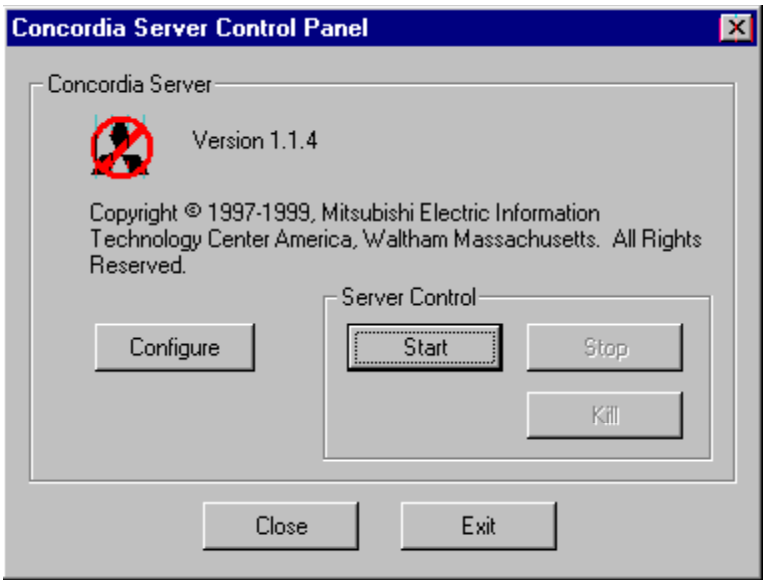


**Figure 2.** *The server control panel for the evaluation version of Concordia.*

Concordia includes several interesting techniques. One is support for co-operation between multiple agents. This is achieved by using the mechanisms for collaboration and distributed events. Another feature is the *Agent Transporter*, which is a kind of minimal Concordia server that can be included in any Java application (or applet). With that the application can send, receive, and execute agents. A useful feature for the programmer is that agents can bring selected class-files with them that the programmer knows will be needed (for example some data storage object). If the agent needs a class-file that it did not bring and can not find locally Concordia will automatically retrieve it from some source (like a web server). In this way the programmer can decide that an agent should not bring some big class-file with it that is seldom needed, but if it is, it will be fetched to the current host of the agent.

*Service bridges* are application specific service providers that operate within the Concordia server and provide functionality to agents. Service bridges are Java modules that can be installed into a Concordia server such that they are automatically instantiated at server start-up and are registered with a naming service within the server. Mobile agents can then travel to the server and interact with the service's public interface to gather the information they need or to perform the tasks they need to accomplish. A *service bridge* gives a uniform interface between different agents and some machine-specific resources (for example some legacy system, like a database).

### 5.2.2 Implementation details

To launch an agent with Concordia is easy and does not require so much written code (but for the agent to do something useful more code is of course needed). This is a short example of how it is done:

A simple Agent could be constructed like the following:

First one must have an agent that derives from Concordia's base class *Agent*:

```
class TestAgent extends Agent{
       public void arrive(){
              System.out.println("The agent has reached its destination");
       }
}
```

This very simple agent with one function (that prints out a message on the screen at the computer were it has arrived) can be launched in three ways. The first is by *the Agent launch wizard* (figure 3 below) and then no more coding has to be done. The second version is by a script (which means that agents can be made to launch automatically, for instance when a computer is started up). And the third one is the most powerful; the agent can be launched through some other Java code. An example of that follows below:

```
TestAgent agent = new TestAgent();
Itinerary path = new Itinerary();
path.addDestination(new Destination("host1", "arrive"));
path.addDestination(new Destination("host2", "arrive"));
path.addDestination(new Destination("host3", "arrive"));
agent.setItinerary(path);
agent.setHomeCodebaseURL("http://webserver/Agents");
agent.launch();
```

A short description of the interesting parts:

Line 1 creates a new `TestAgent` to be launched.

Lines 2-6 create an itinerary for the agent and fill it with the destinations. Each `Destination` containing two things. First **where** the agent should go, and then **what** it should do when it arrives there. "Host1" can be a logical name in the network or an IP-address. "Arrive" is the name of the method to be run, in this case `TestAgent` only has a single method which is run at every new destination.

Line 7 sets up the code base for the agent. This is where it can fetch classes that it needs, but can not find at its current location.

Line 8 finally launches the agent through the Concordia server.

The agent then travels to host1, host2 and host3 in turn and at each location it prints out its message. And once the Itinerary is completed, the Agent will be garbage collected and will disappear from the network. Normally an agent in the end returns to the computer it started from to present its result.

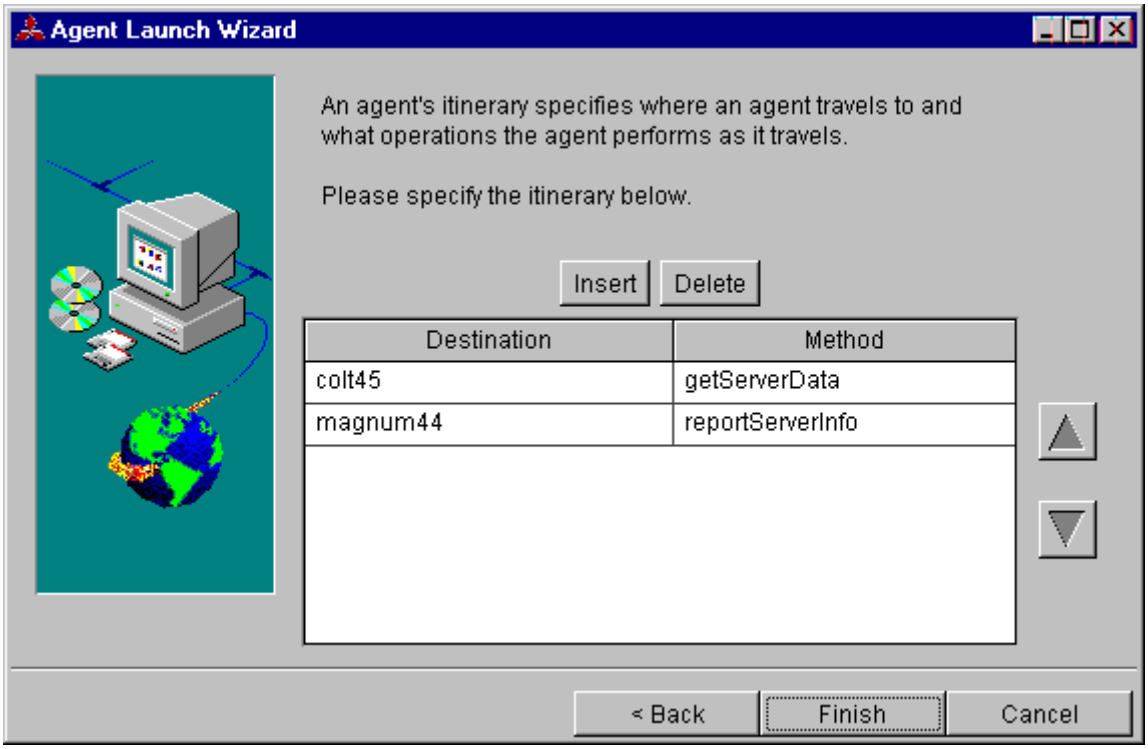


**Figure 3.** *Concordia's Agent Launch Wizard where the destinations the agent should visit are specified and the method of the agent to be run at the destination.*

One problem was found though. Agents would not be launched when executing the *agent.launch()* command but instead of that a cryptic error message appeared. After some email correspondence with Mitsubishi *Michael J. Young* answered that "Concordia 1.1.4 does not work with Java2 at this time".

This was a total surprise since in the documentation they claim that Concordia will work with "Any operating system supporting the Java Development Kit (JDK) 1.1.6 or higher", which I assumed included Java2. And as in any modern Java development I had used Java2.

I then sent a reply asking when they supposed they will have Java2 support and *David Wong* answered "We are expecting to have support for Java2 at the end of this calendar year", which sounds promising.

### 5.2.3 Summary

The impression of Concordia is that it is a very stable and easy to use platform. The only problems were the problem concerning Java2 incompatibility and the handling of the Concordia server, which is a bit limited. Fortunately the Java2 support will come this year and the server handling will probably be better with the full product that includes the enhanced administration tools.

What is missing compared to AgentBuilder is the reasoning part. Concordia just supplies the transfer mechanism, but the creator of the agents has to decide where the agent should go and what it should do when it reaches its destination. To remedy this you could build your own reasoning system or use a finished one. Our choice was the latter, as described next.

## 5.3 Jess

Jess is a system developed by Ernest Friedman-Hill at Sandia National Laboratories [Friedman-Hill, 2000]. Jess is a rule based expert system written entirely in the Java language. Therefore it is possible to easily combine it with mobile agents written in Java. Using Jess, you can build Java applications that have the capacity to reason about the world around them using knowledge supplied in the form of declarative rules and facts.

Jess is a tool to build programs of the type of "intelligent" software called Expert Systems. An Expert System has a set of rules that can be repeatedly applied to a collection of facts about the world. Rules that apply are fired, or executed. Jess uses a special algorithm called *the Rete algorithm* [Jess user guide, 2000] to match the rules to the facts.

Maybe the term "intelligent" should be used with care since a system built in Jess is no more intelligent than the facts and rules. But compared to an ordinary program one might call it intelligent since it has the ability to strive for a goal, adapt to dynamic events and change its plans if it can (or must) do them in another way to reach its goal.

### 5.3.1 Overview

Jess is a production system, which is a more precise name for the special type of rule-based expert systems that applies rules to data [Watson M, 1997]. A rule-based program can have hundreds or even thousands of rules, and the inference engine in Jess will continually apply them to data (facts) in the form of a knowledge base. The rules can represent the heuristic knowledge of a human expert in some domain, and the knowledge base will then represent the state of an evolving situation (an interview, an emergency). In this case, they are said to constitute an expert system. Expert systems are widely used in many domains. The newest applications of expert systems are in the reasoning part of intelligent agents, in enterprise resource planning (ERP) systems, and in order validation for electronic commerce.

With good rules the system can reason about the world around it and make plans to reach its goals. One goal could be for example to decide for a mobile agent which computers it should visit to find some specified data. It could first use its rules to decide which computers might have the data and which do not. Then it orders the mobile agent to move to each of the possible interesting computers and doing a search on each of them. The results are noted in Jess' knowledge base. When the search finds the data a rule fires that says that the goal is fulfilled and the system can then continue with the next goal instead. In this way the system can be made goal driven instead of the more traditional sequentially controlled programs. Goals can also be divided into sub-goals.

The Jess language is a re-implementation in Java of the CLIPS expert system shell, written in the C language. CLIPS means *C language integrated production system*, and these kinds of production systems were originally in most cases implemented in Lisp, therefore the name. For more theory about expert systems and also a description of the CLIPS language (which Jess users also can benefit from since they are almost similar) see [Giarratano & Riley, 1998]. Jess started as an internal research project at the Sandia National Laboratories, but has grown to a full freeware (for non-commercial use) system and is constantly improved by input from the users.

### 5.3.2 Implementation details

Jess is a set of ordinary Java classes that can be used in other programs, for example in agents. The `jess.Rete` class is the inference engine itself, the "brain" of Jess. It works by

loading rules and facts about the world and then matching the facts to rules and fires the matching rules (which then can give new facts and so on).

Facts and Rules can be given to Jess in two ways. Either they can be loaded from a script file (since Jess has its own scripting language) or they can be given in the Java code through Jess Java API. The previous method can be used to "load" the intentions and goals for an initialising agent and the latter when the agent is running.

Jess language syntax looks like Lisp (structured in the same way with parentheses). Here is an easy example:

```
(defrule small
     (fire)
     =>
     (assert (send out firetrucks)))
```

If the fact `(fire)` exists in the knowledge base then the first part of the rule `(fire)` will match, and the rule will be fired once when the Jess command `(run)` is given. This will result in the fact `(send out firetrucks)` being inserted into the knowledge base, which in turn can fire other rules that match the new fact.

What would be recommended is that the Jess classes (and any other classes commonly used by agents) are placed with the Concordia server on every computer that should have the server. In that way the network transit time can be reduced by only transporting the classes that are not used or needed by every agent.

One problem that was discovered with Jess (version 5.0) in combination with Concordia is that it was not straightforward to connect the two of them. Since Concordia depends on Java's serialization mechanism to transfer the agents I was pleased to see that all the Jess classes implemented the `Serializable` interface and therefore should be able to be included in the agent and travel with the agent.

Then it became apparent that it was not that easy. After including the Jess engine in the agent code and compiling it, everything seemed to work fine. The agent could move and bring the engine with it and execute rules after it reached its destination. But every time the agent moved the engine lost all of its present rules and facts (context).

After some questions to the Jess mailing list Ernest Friedman-Hill (that develops Jess) answered that in the current version it was not possible to do that quite so easily. What had to be done was to save all rules and facts to a file (or two files to be exact), transfer the files with the agent, and then load them into the engine at the new destination. We had to work at it a bit to get it to work but finally the agent could transfer itself and Jess and all its contents without problem.

To quote Friedman-Hill: "*I'm hoping that by the Jess 6.0 release, I'll have things worked out such that the straightforward process you tried will be sufficient.*" Since version 6.0 is out as a very early alpha version it is hard to say when it will be finished, but the workaround will be fine until then.

See appendix C for the code of the agent. The important sections concerning the transfer of Jess are `prepareForTransport`, that is automatically run by the Concordia server directly before the agents are sent away, and `completedTransport`, that is run before the agent resumes execution at the new host.

### 5.3.3 Summary

Jess is an expert system shell based on facts and rules. With those it reasons about the world and tries to accomplish its goals. Expert systems using rules can be used to make decisions and do planning on a higher lever than in ordinary code. Jess is written entirely in Java and therefore it was possible to connect it with the mobile agents created with Concordia. In future work it will be plausible to have Jess expert systems control what an agent does when it transfers between systems.

## 5.4 Comparison between tools

The tools used were AgentBuilder, Concordia and Jess. The reason that the system was changed from AgentBuilder to Concordia was that AgentBuilder lacked support for mobile agent development.

After installing Concordia it seemed to fulfil the demands better. Which tool that was chosen is not that important, the important thing was the agent and mobility concepts and therefore the first suitable tool was selected.

One thing that the created Concordia agents lacked compared to agents built with AgentBuilder was the ability to reason about their world and according to their knowledge decide what to do. Not to say that the agents could be made "intelligent", but that they can better handle different situations and therefore have a higher chance of fulfilling their goals.

To fill that need in Concordia, the search started for some system that could reason. Jess is an expert system that could do that. It is written in Java and based on facts (contains information about the world) and rules (matches the facts to see if the rules patterns are fulfilled, if so the rule triggers).

Another reason that Jess was utilised is the desire to, where possible, make a more flexible and general foundation for an agent system without ad hoc solutions. A search agent could be tailored to work very well for one task, but if the demands or environment changes more work is needed to correct it than if the agent was controlled by Jess.

To sum up, building agents with Jess and Concordia combined have much the same reasoning strength as AgentBuilder agents but with the extra benefit that they are also mobile. Therefore it is shown that it is feasible to create agents that are both mobile and also have planning and adaptability capabilities.

# 6 Evaluation of the agent concept

This chapter reports the results of the evaluation of mobile agents.

## 6.1 Agent benefits and drawbacks

Data communications nowadays often take place through the client-server model, which usually works fine. So what does mobile agents have to offer that is new?

Here are seven benefits of using mobile agents [Lange & Oshima 1998] compared to client-server solutions:

1. **They reduce network load –** Distributed systems often rely on the client-server model where multiple interactions between the client and the server are required to accomplish a given task (looking at www pages is a good example). Especially when security measures are enabled. This can result in a lot of network traffic. And particularly when the servers have more information that can reasonably be transferred back to the client for processing there [Huhns & Singh, 1998]. Mobile agents on the other hand take the request or question and transfer with it to the server. Data can then be processed (for example searching) locally and therefore reducing the flow of raw unprocessed data in the network. When very large volumes of data (like log files) are stored at remote hosts, these data should be processed on that host rather that transferred over the network.
2. **They overcome network latency –** In critical real-time systems (such as robots) processes need to respond to changes in their environment in real time. Controlling such systems through a large factory network involves significant latencies, which might not be acceptable. Mobile agents offer a solution, since they can be dispatched from a central controller to act locally and directly execute the controller's directions.
3. **They encapsulate protocols –** Many (legacy) systems have their own protocols or they can have different versions of the same protocol. The agent can then be made to communicate with different protocols on different hosts and that can overcome a legacy problem that otherwise may arise.
4. **They execute asynchronously and autonomously –** When the agent is sent away it needs no connection with or guidance from its sender. It is totally independent and only needs a connection to the sender when it is finished with its task so that it can return home to the sender. This is also good if the network connections are expensive or fragile.
5. **They adapt dynamically** – Mobile agents can sense their environment and react to changes dynamically. They can therefore move around in the network in such a way as to maintain a good position for solving their task.
6. **They can be heterogeneous –** A mobile agent is generally computer and transport-layer-independent and dependant only on its execution environment. It therefore provides good conditions for seamless system integration.
7. **They are robust and fault-tolerant** – The ability of mobile agents to react dynamically to unfavourable situations and events makes it easier to build robust and fault-tolerant distributed systems. If a host is being shut down, agents can be warned and given time to transfer to another host in the network. A large task can also have backup agents in case something goes wrong with the primary agents.

Of course there exist some drawback with using mobile agents (or rather problems to solve):

- **Security –** This is an area that still is open for research. For our discussion of future work on security issues, see *8.4 Security of agents*.
- **Lack of infrastructure –** For agents to work efficiently, when for example doing a search, additional infrastructure is needed. To find data, the agent needs to know what computer to search and how to search. Some kind of servers might be needed so that the agents can make such queries. Servers can be used for name lookups, locating data, controlling agents and communication between agents.

In the market right now there seems to exist two varieties of agent systems. One of them is focused on planning, which tries to make a plan to solve its task or goal using the knowledge it has about the world around it. The other focus is on mobile agents, which can autonomously transfer from computer to computer in a network and thereby get closer to the data source or continue to use the data source even if there exists only a temporary connection with it [Milojičić et.al, 1999].

Since agent technology is not fully mature yet, these two types of focus are seldom included in one product or if done give a poor result. The best thing would be to have a combination of both so that the agent can reason about its world and make active decisions about what to do next, and also transfer between hosts if needed in the process.

## 6.2 Example agents

To test the COTS-products and also show some possible applications agents have, different example agents were constructed. They were focused on demonstrating some possibilities rather then creating finished and fully functionally agents. The agents were all programmed in Java with the use of the Concordia classes through their *Agent API*. Some of the agents also include the Jess engine as an integrated part of it.

Below is a description of the different example agents that are most relevant for this work. This is not an exhaustive list, since different agents accompanying the tools also were run and tested. All of them except the *Simple information agent* (that agent was included with the Concordia evaluation kit) were constructed during this work.

**Simple information agent –** A quite simple agent that transfers to another computer (host 2), collects some system information and returns back to host 1 to present it. The interesting thing with this agent is that it creates a new data object at host 2, fills it with the data and brings it back. And if the class for the object can not be found at host 2 the Concordia server automatically sends a request to fetch that class file from host 1.

**Search agent –** This was the first agent made. It transfers to another computer and there searches the file tree looking for a specific filename. If the file is found it saves the path for the file and then returns home.

**Picture agent –** This agent was solely made as a demonstration agent to visually show how the agents move. It transfers to another computer, opens a window and shows a picture there.

**Advanced search agent –** First this agent asks for a word to search for in text files on a remote computer. It then transfers to that computer and scans it for text files and zip-files. If it finds a plain text file, it opens it and searches the file for the search

word. Every time it finds one occurrence of the word the agent copies that line from the file and also saves the entire path to the file. After all files have been searched it can move to another computer (or back to the first of course) to present its results.

**Mobile Jess agent –** This is the most advanced of the agents. It is an agent that contains the rule based expert system Jess integrated with it. It contains the same functions as the picture agent and advanced search agent with the addition that it can keep facts and rules about the world. As this agent works right now it only takes the Jess engine and all its facts and rules with it when the agent moves. This agent could be expanded so that the expert system decides for the agent what it should do in different circumstances. The code for this agent can be found (somewhat edited for readability) in appendix C.

The reason to create our own agents was that the enclosed example agents in AgentBuilder and Concordia were not suitable. The agents in AgentBuilder were too simple and the ones in Concordia often depended on some database connection (which was not present). Also the enclosed sample agents were not especially focused on data collection.

Overall, the agents worked very well and they were easy to construct with the help from the Concordia documentation and its Java API. Both the agents and the system were stable so our personal opinions are that Concordia is a suitable system for creating mobile applications.

## 6.3 The LSÖ exercise

The LSÖ exercise is a yearly command and control training exercise and this year it was situated in Enköping between the 11th and 17th of May 2000. The exercise included several thousand persons (trained, staff instructors and support personal) and is one of the larger military exercises in Sweden.

During this exercise FOA had an exhibition case where this work and another related thesis work about visualisation of the collected data [Albinsson 2000] where shown to the military staff. Four computers were connected in a local network and two of the created agents were demonstrated. The agents were the *Picture agent* and the *Advanced search agent* (see the previous section). The *Mobile Jess agent* was also discussed but not shown.

The people visiting the stand were mostly officers involved in command and control issues, but also people from different departments (ATLE, FMV, etc.). The demonstrations were made on a personal basis when people had time. Slightly more than 40 people got demonstrations.

The personnel suggested many questions and ideas and we had several discussions, which was the purpose. Often questions regarding the security of mobile agents were asked. Overall the presentations got a good reception from the audience and many wanted to know more about mobile agents and if the work would later produce a final system.

# 7 Discussion

Since the main purpose for this work is to form a basis for further studies and development it is interesting to reflect on how mobile agents can be exploited by the military in the future.

## 7.1 COTS-products

The COTS-products used are of two kinds, commercial and freeware. AgentBuilder and Concordia are of the first category and Jess of the later.

One interesting question is if it is always better to buy a product and therefore get the right to full support than to use a freely developed product? Often the argument for commercial systems is that they are better and have faster future development and much better support. This is my experience with the COTS-products used in this work:

First I must say that the help from the Jess mailing list is very fast and competent. Both questions about the Jess system and the Jess scripting language were answered quickly and competently. Concordia on the other hand was not as good or fast when answering questions. You can then of course argue that I only used the evaluation kit and not the full commercial version. But I tried to ask for details (price etc.) for the full version but since their interest only seemed modest I did not put more effort into that.

When it comes to future development the new versions are coming much faster for Jess than for Concordia. Since I started this work there has been new versions of Jess, both a new bug fix (5.1) and an alpha version for the new Jess 6.0 system. Concordia still has the same version even though they still do not have full support for Java2.

Jess also comes with full source code; Concordia only comes as binary code and that was not so good since I wanted to change the *Agent launch wizard* a bit. The reason for that is that you can only launch one single agent with the wizard and then the program exits. This results in having to fill in everything again even if you just wanted to send out the same agent with the same itinerary.

To conclude, my impression is that a commercial product does not necessary have better support than a freely developed product and therefore I think that freely developed products can also be considered as a possible alternative (at least some of them) in real systems.

## 7.2 The mobile agent concept

One benefit with mobile agents that might not be so obvious at first is that it is easier to develop large advanced distributed applications. The reason for this is that the programmer does not need to be concerned with the distribution issues of the application, that part is almost transparently handled by the agent platform.

Mobile agents, as the ones built, are constructed in a strait-forward manner. But in a real system they might be constructed from different parts, or modules. One module could be a logging module that logs everything the agent does, another could be the Jess reasoning system. To make a specialised agent for some task, different Lego-like modules could be assembled into a template agent. With many universal modules not much Java coding is required to create a new agent.

The universal modules that build up the agents do not necessary have to always be transferred around with the agent and therefore increase the network load. Instead of

that the modules can be stored at every host with the server. And if they are installed as a package with the Concordia server no extra work is needed.

Jess could also add one level to the control and construction of useful agents. An agent can be pre-made with different functions included. To control what the agent does a Jess script file with the agent's behaviour (the facts and rules) can be loaded in its launch phase. And to have another agent that behaves differently, other rules can be written and loaded into the agent. This gives the agents much more flexible and adaptable behaviour and could also more easily solve integration problems (for example new systems that are included in the network).

One potential problem that we found during the LSÖ exercise it that the response time could be quite different from the client-server model. When a request is sent out with the client-server model the user often has to wait while the request is processed. To make the user notice that it will take some time some kind of "Searching, wait..." message is often displayed. This is not the case with agents since they execute asynchronously in their current environment which means that it is hard to tell how long a request will take (if the agent do not have a time limit in which it should return). Agents are therefore more suitable for tasks where the user can send out an agent and then continue to work with something else until the agent returns. Here lies the potential problem, which is essentially a feedback problem. If the user sends out an agent and nothing appears to happen for a while, a normal user will probably send it out again and again until something happens or the user gives up. This will clog the network and frustrate the user. Some possible ways to solve this could be that periodically some message is displayed saying that the agent is working and also by instructing the users about how mobile agents' function.

Another issue that needs some more thought is that of the infrastructure for the agents. The Concordia system is good for its purpose, but not enough in a real distributed $C^2$-system. What it lacks is some infrastructure to help the agents locate the data. It can be different servers that the agent or a *Service bridge* in Concordia knows about. The agents can then query these servers about which computers that are accessible and what kind of data they are supposed to contain. One example could be that the agent wants to find messages from a company and only those messages that contain orders that are given. It can then query a locating server to get the IP-addresses for all computers in the company. Or even better is if it queries for the staff computers of the company, since the orders most likely are given there and the agent then has fewer computers to search through. The decision to narrow the search question could be made by Jess' rules. Instead of having the expert system in every agent one possible solution is to have *traffic management agents* or servers with that functionality. They can then direct other agents to the right computers in the network.

Finally, the mobile agent concept appears to be suitable for data collection tasks, especially when we have large amounts of distributed data and have advanced queries and maybe do not know precisely what we are looking for or how to specify the query for it.

## 7.3 Military applications

Use of mobile agents can have many benefits as stated in *6.1 Agent benefits and drawbacks*. From a military point of view some of them are especially interesting and the motivation for them will be discussed below:

**They reduce network load –** This is important if you want to keep a low profile. From/to a staff-command area a lot of network traffic is usually generated. If this is spotted the staff can be a target for enemy fire. Mobile agents can help to reduce unnecessary load on the network.

**They encapsulate protocols –** Agents (and especially Concordia's Service bridge functionality) can help to solve the problem of co-existence between both new and old systems. Computer systems used in the military often have a long lifetime and new systems must be able to interact with the old ones. Therefore the use of one single internal agent format can help the introduction period.

**They execute asynchronously and autonomously –** During a regroup of a staff vehicle the network connections can be broken and then the client-server model will not work. Agents can on the other hand transfer to the computers in the vehicle just before the connection is closed and then continue their work (like a data search) without the network connection. When the regrouping is finished the agent can move away. This is a great benefit with agents in many other situations, that they are totally independent from its home host.

**They adapt dynamically –** A planning agent (like an agent that is directed by Jess) can sense its host and adapt to it. One way to take advantage of this in military systems is to sense which other systems are installed on the host to be able to interact with them. Another way is if the agent got a notification before a regrouping (and network disconnection) so it could decide if it wanted to continue or transfer directly.

All these military benefits combined with the regular benefits from using mobile agents call for a deeper and more throughout study of how the military can utilise this new technique in the best and most beneficial way.

## 7.4 Conclusions

The combination of the mobile agent system Concordia and the expert system Jess together meet all the requirements to give a flexible and good solution for the data collection task in a distributed Command and Control system.

The result shows that the mobile agent concept is promising but that further work is needed before the design of a data collection system can be made. The mobile agent technique has benefits compared to the usual client-server model, especially from a military perspective.

# 8 Future work

Since one of the goals of this thesis was to form the basis for future work, this section describes ideas and thoughts that could, or in some cases must, be further studied. The different sections below show promising areas that further work can examine.

## 8.1 Initial list of points to study

In the beginning of the work a list of interesting points that could be studied were made. Since it is a long and quite comprehensive list many points are still untouched. The list can probably be used as a starting point for ideas about how to continue the work. The points were:

- How will a diverse, vague and/or complex question be turned into a search question?
- How should example and/or template agents be designed to work in a distributed $C^2$-system?
- How should problem knowledge be handled?
- In which form should the answers from a search question be assembled?
- How should the agent keep knowledge about the world around it?
- How will the agent find the right data?
- How can the agent choose a good path in the network?
- Can the agent learn which way to take in the network and where specific data are?
- To which degree should the agent collect parts of files or whole files?
- To which degree should there be different kinds of agents for different tasks?
- What information can NOT be collected with agents?
- What if the agent has a restriction that it must be finished before a specified time and when the time is reached it hasn't. Should it then return with a partial answer or clone itself so the clone can continue with the search and go back and notify the user how far it has progressed right now?
- Should one/how should one check the correctness of the data that the agent collects?
- What if something goes wrong with the agent/network during its task? How should that be handled?
- Which factors influence the design of the user interface/work area?
- Visual tools for "making" an agent, what should they look like and what functions do we need them to have?
- Where in a larger $C^2$-system, should one place the agent functionality?
- Should the agents log everything they do/everywhere they go?
- How can agents be put together for different search questions? Template agents? Agents with modules that are plugged-in when the functionality is needed? Different agents for different questions?
- If users ask questions by assembling an agent with "Lego"-type parts, how can we be sure that the user and the agent have the same idea of what the question is about?

## 8.2 Data mining

Data mining is also a growing area that might be used together with the mobile agent concept to enhance the data collection (for an introduction to data mining, see [Adriaans & Zantinge, 1996]). Mobile agents can actively go and look for different data in the log files in advance. This collected information can be used to create directories and indices of contents that either hold all the data or only pointers to where it can be found. In that way, there is no "search" remotely with the query, and the answer is returned within fractions of a second (if all the data is stored locally, of course). This method is currently used in large databases of terabytes of legal documents in the United States, for example the Westlaw system [West Group, 2000].

These "data mining agents" can then constantly travel around in the network to keep the indices updated all the time. And to avoid having the indices at every host, a set of a few servers providing the functionality can be placed at strategic locations in the network to be queried by the search agents. For this to be efficient the C²-systems can specify where it stores its log files (the file path) and also what kind of data and how the data is stored in some (for the data mining agents) universal format.

This solution can be used if the importance is that the queries are answered quickly, at the cost of larger storage usage. In a query system operating in real-time on current data (and not afterwards, when all data from the exercise are already generated) this approach is probably both faster and more efficient.

## 8.3 Implementation and infrastructure

This section explores the ideas for future work concerning the practical implementation of a mobile agent data-collection system. Questions regarding infrastructures, like file formats and needed servers are also placed here.

- **Integration –** To get a useful system it must be easy both to specify which information one is interested in (the search) and also how it should be returned and displayed. The agents must therefore be integrated with a search form (a good idea would be if it was web based) and some presentation tool that shows the data in an easily understandable format (for example as a time line [Albinsson, 2000]). Another related question is if the presentation tool is integrated with the agents in such a way that it should be able to send out more agents to collect missing data in the presentation?
- **Data parsing –** How much should the agent parse (find and sort) information during the time it tries to answer the search question? Should most of the work be done at the host with the data or after the agent returns?
- **Concordia –** The version that was used is only an evaluation version and should therefore be replaced with the full version that has better support for security and robustness. Other alternative agent development tools might also, if necessary, be investigated.
- **Study advanced possibilities –** Both Concordia and Jess have more advanced features than are shown by the example agents. One can also use the so-called *Service Bridges* and Collaboration between agents in Concordia. Service Bridges can be used in a general way to communicate among agents and different systems (like databases or other programs) at the host. The agent can then easily use newly developed systems just by adding a new Service Bridge. And with the help of Jess the agent could make active decisions, thus partly taking its own initiative.

- **File format specifications –** These agents have only used text files. Since the military uses a lot of different formats (like for instance the Microsoft Office formats) the agents must be able to scan these formats. To do that specifications for the formats are needed or modules that can read them.
- **Universal agent internal data format –** One question is if the agent should have its own internal data format. As almost every system has its own format for the data the problems with compatible formats can be a big problem. But if the agent has a universal internal data format and all the different Service Bridges translate (or keep track of the format) from the host format (from a database or some other source) to the agents universal format this problem can be solved.
- **Network route planning –** The agents must be able to plan which computers in the network they should visit. In a network with hundreds of computers it is not feasible to visit them all. Since some selection must be made, this could be a good task for Jess to solve. In a time limited task the order of the visits is very important so that some criteria find the best computers to visit first, in case time is running out.
- **Infrastructure for data collection –** What kind of servers and services are needed to support data collection with mobile agents? One obvious function is location servers which given a question (for example a name of a unit) returns a list of IP-numbers of computers that are likely to be of interest for the agent. But other, maybe not so evident, services might also be needed.
- **Data mining –** Should data, and if so how, be categorised and indexed in advance to speed up the queries? See section *8.2 Data mining*.

## 8.4 Security of agents

One important matter that has not been covered in this work about mobile agents but is very crucial in military applications is security. The impact of security in the military is obvious and so the concerns of malfunction or hostile mobile agents must also be considered. So why has this not been a central part of this work then? One reason is that the priority was to evaluate the possibilities of mobile agents and not the potential problems. The other, maybe more important, reason is that a future mobile system is intended to be used during exercises and not necessarily in a real situation. Therefore the demands on security might be lowered if the exercise is conducted in a closed local network without connection to the Internet.

To quote Milojičić about myths regarding mobile agents [Milojičić et.al, 1999]:

> **Myth:** Mobile agents are risky to use.
>
> **Fact:** The largest concern of potential mobile agent users is related to security. Users are afraid to accept agents at their computers. The concern about accepting agents at servers is even higher. However, accepting mobile agents is not necessarily different from allowing remote access, as enabled by Java applets, or accepting e-mail that contains active entities, such as Word documents with macros. If agents were allowed to communicate with the servers through a set of limited, well-defined interfaces, and if they were contained within a clearly defined sandbox (as the Java Virtual Machine supports), the risk would be significantly reduced.

As this shows, mobile agents do not necessary present a security hole, if treated correctly. Concordia also has security support built into the full version (which we have not tested). For more information on the security issues in Concordia see [Walsh et.al, 1998].

## 8.5 Domain analysis

Domain analysis is a very important factor for any system to be successful. The reason is that if the system produced can not fulfil the demands of its contemplated users then it will, to a large extent, be a waste of money.

- **Type of questions –** What kinds of questions should the agent be able to answer that helps the staff instructors? This is related to which questions they want to answer and maybe later visualise.
- **Requisite analysis –** No analysis of what kind of information the staff instructors require has been done in this initial study. A thorough requisite analysis must be done to specify which kind of data the agents should be able to find and in the prolongation what kind of data the systems must store in their log files.
- **Logged data specifications –** Since the agents have not been used together with a real system the assumption has been that all needed information is logged. This might not be the case in a real system and a study of what is logged in the existing systems and what would be needed to be logged in future systems to present the information must be done.
- **Real-time or "after action review" –** Should the mobile agents be used only after an exercise (and if so, all the data are known to be already logged and accessible) or should they be used during a running exercise? This highly effects how the agents should act and be implemented. Also the interface with the user might differ.
- **User perception of mobile agents –** It is important to make sure that the possible users of mobile agents understand broadly how they work and should best be used. Also to what extent feedback is necessary for the user to feel that he or she knows enough about the progress of the agent.

## 8.6 Additional considerations

Here follows some additional considerations about mobile agents that could be made.

- **Install and test the agents in a more realistic environment –** At FHS (the Swedish National Defence College) they have a C²-training laboratory that could be a good isolated place to start. Maybe it also could be done at some smaller exercise, like ESÖ.
- **Analyse enemy use of mobile agents –** Enemy mobile agent code is a potent threat. To ascertain the possibilities of how enemy mobile agent intelligence technology might work, the use of agents for data collection is a good way to learn about that.
- **Connections to other areas –** This analysis of the use of mobile agents is restricted to data collection in the C²-system with the purpose of evaluating the ground warfare model. But mobile agents have a wide range of possible uses. Some of them are in information distribution, user adaptable agents and surveillance agents.

# 9 References

## 9.1 Literature

[Adriaans & Zantinge, 1996]
Pieter Adriaans and Dolf Zantinge (1996). *Data mining*. Harlow: Addison Wesley Longman (ISBN 0-201-40380-3)

[Albinsson 2000]
Pär-Anders Albinsson (2000). Visualisering av loggfiler – ett hjälpmedel för utvärdering av versamhetsmodeller. Linköping: Linköpings Universitet, Department of Computer and Information Science, LiTH-IDA-Ex-00/62. Also to appear as a report from Försvarets Forskningsanstalt. (in Swedish)

[Cockayne & Zyda, 1997]
William R. Cockayne and Michael Zyda (1997). *Mobile Agents*. Greenwich, CT: Manning Publications Co.

[Gardelius, 2000]
HP ATLE Projekt 1, Ove Gardelius (2000). Verksamhetsmodell Markstrid 5.0. Enköping: ATLE, Försvarsmakten (in Swedish)

[Giarratano & Riley, 1998]
Joseph Giarratano and Gary Riley (1998). *Expert Systems – principles and programming*, third edition. Boston: PWS Publishing Company (ISBN 0-534-95053-1)

[Huhns & Singh, 1998]
Michael N. Huhns and Munindar P. Singh (1998). *Readings in agents*. San Francisco, CA: Morgan Kaufmann Publishers, Inc. (ISBN 1-55860-495-2)

[Jönsson & Nordberg, 2000]
Eric Jönsson and Jens Nordberg (2000). Intelligent Agents in an Electronic Auction Context. Linköping: Linköpings Universitet, Department of Computer and Information Science, LiTH-IDA-Ex-00/37

[Lange & Oshima, 1998]
Danny Lange and Mitsuru Oshima (1998). "Mobile Agents with Java: The Aglet API" in *Mobility : Processes, Computers and Agents*, p 495 - 512. Reading, Massachusetts: Addison-Wesley Longman, Inc. (ISBN 0-201-37928-7)

[Milojičić et.al, 1999]
Dejan Milojičić, Frederick Douglis and Richard Wheeler (1999). *Mobility : Processes, Computers and Agents*. Reading, Massachusetts: Addison-Wesley Longman, Inc. (ISBN 0-201-37928-7)

[National research council, 1998]
National research council (1998). Commercial multimedia technologies for twenty-first century army battlefields. Washington D.C: National research council

[Nwana & Ndumu, 1998]
H.S. Nwana & D.T Ndumu (1998). "A Brief Introduction to Software Agent Technology" in *Agent Technology*, p 29 - 47. Berlin, Heidelberg: Springer-Verlag. (ISBN 3-540-63591-2)

[Walsh et.al, 1998]
Tom Walsh, Noemi Paciorek and David Wong (1998). "Security and Reliability in Concordia" in *Mobility : Processes, Computers and Agents*, p 525 - 534. Reading, Massachusetts: Addison-Wesley Longman, Inc. (ISBN 0-201-37928-7)

[Watson M, 1997]
Mark Watson (1997). *Intelligent Java Applications for the Internet and Intranets*. San Francisco, CA: Morgan Kaufmann Publishers, Inc. (ISBN 1-55860-420-0)

[Watson R, 1996]
Richard Watson (1996). *Data mangement – an organizational perspective*. New York: John Wiley Sons Inc.

[Weber, 1998]
Joseph L. Weber (1998). *Special edition Using Java 1.2*, Fourth edition. Indianapolis: QUE. (ISBN 0-7897-1529-5)

[Wikberg et.al, 2000]
Per Wikberg, Håkan Söderberg and Arne Worm (2000). Modellbaserad utvärdering: Datainsamling med hjälp av förbandsinstruktörer. Linköping: Försvarets Forskningsanstalt, FOA-R-00-01462-505-SE, ISSN 1104-9154 (in Swedish)

## 9.2 Internet links

[AgentTools, 2000]
Reticular Systems Inc. (2000). A list with different agent construction software and tools (Acc. 2000-05-26), *http://www.agentbuilder.com/AgentTools/index.html*

[Friedman-Hill, 2000]
Ernest Friedman-Hill (2000). *Jess*, Sandia National Laboratories (Acc. 2000-06-08), *http://herzberg.ca.sandia.gov/jess/*

[Jess user guide, 2000]
Ernest Friedman-Hill (2000). Jess user guide, chapter 10, *The Rete Algorithm*, Sandia National Laboratories (Acc. 2000-06-08), *http://herzberg.ca.sandia.gov/jess/*

[Mitsubishi Electric, 2000]
Mitsubishi Electric Information Technology Center America (2000). *Concordia* (Acc. 2000-06-08), *http://www.meitca.com/HSL/Projects/Concordia/*

[Reticular 1999]
Reticular Systems Inc. (2000). *AgentBuilder* (Acc. 2000-05-26), *http://www.agentbuilder.com/*

[West Group, 2000]
West Group (2000). *Westlaw* (Acc. 2000-06-08), *http://www.westlaw.com/*

# Appendix A. Abbreviations and descriptions

Here are all used abbreviations and descriptions of the conceptions used.

*API* – Application programming interface

*ATLE* – see *HP ATLE*

*C²* - Command and Control

*CLIPS* – *C Language Integrated Production System*. It is a well-known expert system shell written in the C language. See also Jess.

*COTS* – Commercial off the shelf (products)

*ESÖ* – A much smaller Command and Control exercise than LSÖ

*FHS* – the Swedish National Defence College

*FMV* – the Defence Material Administration

*FOA* – the Swedish National Defence Research Establishment

*HP ATLE* – *Huvudprojekt Arméstidskrafternas Taktiska Ledningsystem*, the main project for the armed forces tactical Command and Control system

*Jess* – *the Java Expert System Shell*. Jess is a rule-based expert system written entirely in the Java language and descends from the CLIPS system.

*KQML* – Knowledge Query & Manipulation Language

*LedFram* – *Ledningsförmåga i den Framtida striden*, Commanding in the future battlefield. It is a project that researches support for the development of Command and Control systems for the military authorities, now and in the long term.

*LSÖ* – *LedningsSystemÖvning*, Command and Control exercise

*RMI* – Remote Method Invocation

*SQL* – Structured Query Language

*XML* – Extensible Markup Language

# Appendix B. Log file for scenario

One 45 minutes long scenario was constructed by two former officers that now work at FOA. From this scenario a demonstration log file was created with the messages concerning five military units in battle during a period of about 45 minutes. The whole log file (in Swedish) can be found below. As is apparent that even small files with little information like this can be hard to understand without help from computers.

The categories are (from left to right); reference, time, seconds, unit, role, activity, type of message, from/to, unit, role, file format, time number and contents.

| Ref | Tid | s | Enhet | Person | Aktivitet | Typ | Från/Till | Enhet | Person | Filformat | Tids nr | Innehåll |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 859 | 37 | 514 | UndC | Skriver | rapport | | | | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 2 | 900 | 4 | 514 | UndC | Sänder | rapport | till | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 3 | 900 | 33 | L1 | UndC | tar emot | rapport | från | 514 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 4 | 901 | 2 | L1 | UndC | öppnar | rapport | från | 514 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 5 | 902 | 11 | L1 | UndC | sänder | rapport | till | 513 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 5,01 | 902 | 11 | L1 | UndC | sänder | rapport | till | L3 | UndBef 2 | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 5,02 | 902 | 11 | L1 | UndC | Sänder | rapport | till | Div | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 5,10 | 903 | 0 | L1 | UndC | Skriver | rapport | | | | | | |
| 6 | 903 | 30 | L1 | UndC | Sänder | rapport | till | 514 | UndC | txt | 901 | Med TidsNr. 0855 konfirmeras |
| 6,1 | 904 | | 514 | UndC | tar emot | rapport | från | L1 | UndC | txt | 901 | Med TidsNr. 0855 konfirmeras |
| 6,2 | 903 | | Div | UndC | tar emot | rapport | från | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 7 | 904 | | Div | UndC | öppnar | rapport | från | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 7,1 | 902 | 19 | L3 | UndC | tar emot | rapport | från | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 8 | 905 | | L3 | UndC | öppnar | rapport | från | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 8,1 | 903 | 33 | 512 | UndC | tar emot | rapport | från | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 9 | 906 | | 512 | UndC | öppnar | rapport | från | L1 | UndC | txt | 900 | 0855 Två strv.Komp vid STRÄNGNÄS FLYGPLATS frr mot GRUNDBRO |
| 10 | 905 | | 514 | UndC | öppnar | rapport | från | L1 | UndC | txt | 901 | Med TidsNr. 0855 konfirmeras |
| 11 | 905 | 50 | 514 | StriC | Skriver | oleat | | | | ppt | 906 | Karbild över nya läget |
| 11,1 | 907 | | 512 | StriC | Skriver | Ori | till | L1 | StriC | txt | 907 | ORI: ingen kontakt med Fi |
| 12 | 908 | | 512 | StriC | Sänder | Ori | till | L1 | StriC | txt | 907 | ORI: ingen kontakt med Fi |
| 13 | 908 | | 512 | StriC | Sänder | Ori | till | L3 | StriC | txt | 907 | ORI: ingen kontakt med Fi |
| 13,1 | 906 | | Div | UndBef 1 | Skriver | oleat | till | L1 | UndBef 2 | ppt | 905 | Karbild över Div. Läge |
| 14 | 908 | | Div | UndBef 1 | Sänder | oleat | till | L1 | UndBef 2 | ppt | 905 | Karbild över Div. Läge |
| 15 | 908 | | Div | UndBef 1 | Sänder | oleat | till | L3 | UndBef 2 | ppt | 905 | Karbild över Div. Läge |
| 15,1 | 909 | 1 | L3 | Undbef2 | tar emot | oleat | från | Div | UndBef 1 | ppt | 905 | Karbild över Div. Läge |
| 16 | 909 | 12 | L3 | Undbef2 | öppnar | oleat | från | Div | UndBef 1 | ppt | 905 | Karbild över Div. Läge |
| 16,1 | 909 | 33 | L1 | Undbef2 | tar emot | oleat | från | Div | UndBef 1 | ppt | 905 | Karbild över Div. Läge |
| 17 | 912 | | L1 | Undbef2 | öppnar | oleat | från | Div | UndBef 1 | ppt | 905 | Karbild över Div. Läge |
| 17,1 | 909 | 24 | L3 | UndBef 2 | Skriver | Förfrågan | till | L1 | Undbef 2 | txt | 909 | Fråga: har Ni fått Div. Lägebild med tidsnr0 905 |
| 18 | 910 | 2 | L3 | UndBef | Sänder | Förfrågan | till | L1 | Undbef | txt | 909 | Fråga: har Ni fått Div. Lägebild med tidsnr0 905 |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 2 | | | | | | 2 | | |
| 19 | 910 | 45 | L1 | Undbef2 | tar emot | Förfrågan | från | L3 | UndBef 2 | txt | 909 | Fråga: har Ni fått Div. Lägebild med tidsnr0 905 |
| 20 | 913 | 55 | L1 | Undbef2 | öppnar | Förfrågan | från | L3 | UndBef 2 | txt | 909 | Fråga: har Ni fått Div. Lägebild med tidsnr0 905 |
| 20,1 | 914 | 9 | L1 | undBef2 | Skriver | förfrågan | till | L3 | Undbef 1 | txt | 909 | Svar på meddelande med tidsnr 0909: JA |
| 21 | 915 | 19 | L1 | undBef2 | sänder | förfrågan | till | L3 | Undbef 1 | txt | 909 | Svar på meddelande med tidsnr 0909: JA |
| 21,1 | 915 | 55 | L3 | UndC | tar emot | rapport | från | L1 | UndC | txt | 909 | Svar på meddelande med tidsnr 0909: JA |
| 22 | 916 | 6 | L3 | UndC | öppnar | rapport | från | L1 | UndC | txt | 909 | Svar på meddelande med tidsnr 0909: JA |
| 22,1 | 909 | 32 | div | StriC | Skriver | förfrågan | till | l3 | StriC | txt | 908 | läget vid mekb11 frågas |
| 23 | 910 | 11 | div | StriC | Sänder | förfrågan | till | l3 | StriC | txt | 908 | läget vid mekb11 frågas |
| 24 | 911 | | l3 | StriC | Sänder | oleat | till | 512 | chef | ppt | 905 | Kartbild över div läge |
| 24,1 | 912 | | 512 | chef | tar emot | oleat | från | l3 | StriC | ppt | 905 | Kartbild över div läge |
| 25 | 912 | | 512 | chef | öppnar | oleat | från | l3 | StriC | ppt | 905 | Kartbild över div läge |
| 26 | 912 | | 514 | StriC | Sänder | oleat | till | l3 | undbef2 | ppt | 906 | Kartbild över nya läget |
| 26,1 | 913 | | l3 | undbef2 | tar emot | oleat | från | 514 | StriC | ppt | 906 | Kartbild över nya läget |
| 27 | 913 | | l3 | undbef2 | öppnar | oleat | från | 514 | StriC | ppt | 906 | Kartbild över nya läget |
| 28 | 913 | | 512 | StriC | skriver | order | | | | txt | 913 | Bataljonsorder |
| 29 | 915 | | Div | StriC | radio | förfrågan | till | l3 | StriC | | | läget vid mekb11 frågas |
| 30 | 915 | | l3 | StriC | radio | svar på förfrågan | från | Div | StriC | | | läget vid mekb11 redogörs |
| 30,1 | 915 | | 514 | StriC | Skriver | Ori | till | l1 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 31 | 915 | | 514 | StriC | Sänder | Ori | till | l1 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 32 | 915 | | 514 | StriC | Sänder | Ori | till | l3 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 33 | 915 | | 514 | StriC | Sänder | Ori | till | 512 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 33,1 | 916 | | l3 | StriC | tar emot | Ori | från | 514 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 34 | 916 | | l3 | StriC | öppnar | Ori | från | 514 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 34,1 | 916 | | l1 | StriC | tar emot | Ori | från | 514 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 35 | 916 | | l1 | StriC | öppnar | Ori | från | 514 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 35,1 | 917 | | 512 | StriC | tar emot | Ori | från | 514 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 36 | 917 | | 512 | StriC | öppnar | Ori | från | 514 | StriC | txt | 914 | ori: fientligt fältarbetsfordon bekämpat vid BRÅTORP. Fortsätter framryckning |
| 37 | 910 | | l3 | StriC | tar emot | förfrågan | från | div | StriC | txt | 908 | läget vid mekb11 frågas |
| 38 | 913 | | l1 | StriC | skriver | Order | | | | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 39 | 916 | | l1 | StriC | Sänder | order | till | 512 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 40 | 916 | | l1 | StriC | Sänder | ori | till | 514 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 41 | 916 | | l1 | StriC | Sänder | ori | till | l3 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 41,1 | 917 | | 514 | StriC | tar emot | ori | från | l1 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 42 | 917 | | 514 | StriC | öppnar | ori | från | l1 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 42,1 | 917 | | l3 | StriC | tar emot | ori | från | l1 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 43 | 917 | | l3 | StriC | öppnar | ori | från | l1 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 43,1 | 917 | | 512 | StriC | Skriver | förfrågan | till | l3 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |
| 44 | 917 | | 512 | StriC | Sänder | förfrågan | till | l3 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |
| 44,1 | 918 | | l3 | StriC | tar emot | förfrågan | från | 512 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 45 | 918 | I3 | StriC | öppnar | förfrågan | från | 512 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |
| 46 | 919 | I3 | StriC | sänder | förfrågan | till | I1 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |
| 46,1 | 920 | I1 | StriC | tar emot | förfrågan | från | I3 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |
| 47 | 920 | I1 | StriC | öppnar | förfrågan | från | I3 | StriC | txt | 916 | Föranleder div lägesbild tnr 0905 någon ny order för vår del? |
| 48 | 921 | I1 | StriC | Sänder | order | till | 512 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 48,1 | 922 | 512 | StriC | tar emot | order | från | I1 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 49 | 922 | 512 | StriC | öppnar | order | från | I1 | StriC | ppt | 913 | Oleatorder till 512 baserad på div lägesbild tnr 0905 |
| 49,1 | 923 | 512 | StriC | Skriver | ori | till | I1 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 50 | 923 | 512 | StriC | Sänder | ori | till | I1 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 51 | 923 | 512 | StriC | Sänder | ori | till | I3 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 52 | 923 | 512 | StriC | Sänder | ori | till | Div | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 53 | 923 | 512 | StriC | Sänder | ori | till | 514 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 53,1 | 923 | I1 | StriC | tar emot | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 54 | 923 | I1 | StriC | öppnar | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 54,1 | 925 | I3 | StriC | tar emot | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 55 | 925 | I3 | StriC | öppnar | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 55,1 | 926 | div | StriC | tar emot | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 56 | 926 | div | StriC | öppnar | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 56,1 | 924 | 514 | StriC | tar emot | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 57 | 924 | 514 | StriC | öppnar | ori | från | 512 | StriC | txt | 923 | Ori: 512. Påbörjar framryckning |
| 57,1 | 925 | 514 | StriC | skriver | ori | till | Div | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 58 | 925 | 514 | StriC | Sänder | ori | till | Div | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 59 | 925 | 514 | StriC | Sänder | ori | till | I1 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 60 | 925 | 514 | StriC | Sänder | ori | till | I3 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 61 | 925 | 514 | StriC | Sänder | ori | till | 512 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 61,1 | 928 | Div | StriC | tar emot | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 62 | 928 | Div | StriC | öppnar | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 62,1 | 926 | I3 | StriC | tar emot | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 63 | 926 | I3 | StriC | öppnar | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 63,1 | 927 | I1 | StriC | öppnar | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 64 | 927 | I1 | StriC | öppnar | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 64,1 | 926 | 512 | StriC | öppnar | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 65 | 926 | 512 | StriC | öppnar | ori | från | 514 | StriC | txt | 924 | Ori: Vi är i strid med fientligt pansarskyttekompani vid GRUNDBRO |
| 65,1 | 927 | 514 | StriC | skriver | ori | till | I1 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 66 | 927 | 514 | StriC | Sänder | ori | till | I1 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 67 | 927 | 514 | StriC | Sänder | ori | till | I3 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 68 | 927 | 514 | StriC | Sänder | ori | till | 512 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 69 | 927 | 514 | StriC | Sänder | ori | till | Div | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 69,1 | 929 | div | StriC | tar emot | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 70 | 929 | div | StriC | öppnar | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | utslagna. Begär understöd av 512. |
| 70,1 | 929 | I3 | StriC | tar emot | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 71 | 929 | I3 | StriC | öppnar | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 71,1 | 930 | I1 | StriC | tar emot | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 72 | 930 | I1 | StriC | öppnar | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 72,1 | 928 | 512 | StriC | tar emot | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 73 | 928 | 512 | StriC | öppnar | ori | från | 514 | StriC | txt | 926 | Ori: Vår Framryckning är hejdad. 3 egna vagnar utslagna. Begär understöd av 512. |
| 74 | 928 | 512 | StriC | radio | förfrågan | till | 514 | chef | | | Samverkan avseende understöd. Begäran om skiss över läget. |
| 75 | 928 | 514 | chef | radio | förfrågan | från | 512 | StriC | | | Samverkan avseende understöd. Begäran om skiss över läget. |
| 75,1 | 930 | 514 | StriC | Skriver | oleat | till | 512 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 76 | 930 | 514 | StriC | Sänder | oleat | till | 512 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 77 | 930 | 514 | StriC | Sänder | oleat | till | I1 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 78 | 930 | 514 | StriC | Sänder | oleat | till | I3 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 78,1 | 930 | 514 | StriC | tar emot | oleat | från | 514 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 78,2 | 930 | 514 | StriC | öppnar | oleat | från | 514 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 78,3 | 931 | I3 | StriC | tar emot | oleat | från | 514 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 79 | 931 | I3 | StriC | öppnar | oleat | från | 514 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 79,1 | 931 | I1 | StriC | tar emot | oleat | från | 514 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 80 | 931 | I1 | StriC | öppnar | oleat | från | 514 | StriC | ppt | 930 | Förslag till stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 81 | 931 | 512 | StriC | radio | | till | 514 | StriC | | | Bekräftar förslag till stridsplan tnr 930 |
| 82 | 931 | 514 | StriC | radio | | från | 512 | StriC | | | Svarar på Bekräftar förslag till stridsplan tnr 930 |
| 83 | 929 | I1 | StriC | Sänder | order | till | 512 | StriC | txt | 928 | Order: 512 panbat understöder 514 pansskyttbat anfall mot GRUNDBRO |
| 83,1 | 930 | 512 | StriC | tar emot | order | från | I1 | StriC | txt | 928 | Order: 512 panbat understöder 514 pansskyttbat anfall mot GRUNDBRO |
| 84 | 930 | 512 | StriC | öppnar | order | från | I1 | StriC | txt | 928 | Order: 512 panbat understöder 514 pansskyttbat anfall mot GRUNDBRO |
| 84,2 | 931 | 512 | StriC | skriver | ori | till | I1 | StriC | txt | 930 | Ori: Order med tnr 928 redan påbörjad efter samverkan med 514 |
| 85 | 931 | 512 | StriC | Sänder | ori | till | I1 | StriC | txt | 930 | Ori: Order med tnr 928 redan påbörjad efter samverkan med 514 |
| 85,1 | 931 | I1 | StriC | tar emot | ori | från | 512 | StriC | txt | 930 | Ori: Order med tnr 928 redan påbörjad efter samverkan med 514 |
| 86 | 931 | I1 | StriC | öppnar | ori | från | 512 | StriC | txt | 930 | Ori: Order med tnr 928 redan påbörjad efter samverkan med 514 |
| 86,1 | 932 | I1 | StriC | skriver | order | till | 512 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 87 | 932 | I1 | StriC | Sänder | order | till | 512 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 88 | 932 | I1 | StriC | Sänder | order | till | 514 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 88,1 | 933 | 512 | StriC | tar emot | Order | från | I1 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 89 | 933 | 512 | StriC | öppnar | Order | från | I1 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 89,1 | 934 | 514 | StriC | tar emot | Order | från | I1 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 90 | 934 | 514 | StriC | öppnar | Order | från | I1 | StriC | txt | 931 | Order: Stridsplan från tnr 928 gäller. |
| 90,1 | 930 | I1 | undc | skriver | förfrågan | till | 514 | undc | txt | 929 | Läget vid 514 frågas. |
| 91 | 930 | I1 | undc | Sänder | förfrågan | till | 514 | undc | txt | 929 | Läget vid 514 frågas. |
| 91,1 | 931 | 514 | undc | tar emot | förfrågan | från | I1 | undc | txt | 929 | Läget vid 514 frågas. |
| 92 | 931 | 514 | undc | öppnar | förfrågan | från | I1 | undc | txt | 929 | Läget vid 514 frågas. |
| 93 | 931 | 514 | undc | radio | | till | I1 | undc | | | Redogörelse för läget vid 514. |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 94 | 931 | | l1 | undc | radio | | från | 514 | undc | | | Redogörelse för läget vid 514. |
| 94,1 | 932 | | 512 | StriC | skriver | ori | till | l1 | StriC | txt | 931 | Ori: Framryckning enligt stridsplan tnr 928 på börjad. |
| 95 | 932 | 38 | 512 | StriC | Sänder | ori | till | l1 | StriC | txt | 931 | Ori: Framryckning enligt stridsplan tnr 928 på börjad. |
| 95,1 | 933 | | 512 | StriC | skriver | ori | till | l1 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 96 | 933 | 49 | 512 | StriC | Sänder | ori | till | l1 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 97 | 934 | | 512 | StriC | Sänder | ori | till | l3 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 98 | 934 | | 512 | StriC | Sänder | ori | till | 514 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 99 | 934 | | 512 | StriC | Sänder | ori | till | div | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 99,1 | 933 | 57 | l1 | StriC | tar emot | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 100 | 935 | 5 | l1 | StriC | öppnar | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 100 | 936 | | l3 | StriC | tar emot | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 101 | 936 | 52 | l3 | StriC | öppnar | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 101 | 934 | 11 | 514 | StriC | tar emot | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 102 | 934 | 51 | 514 | StriC | öppnar | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 102 | 936 | | div | StriC | tar emot | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 103 | 937 | 48 | div | StriC | öppnar | ori | från | 512 | StriC | txt | 933 | Ori: Vi är i strid. Fientligt pansskyttbkompani bekämpat. |
| 103 | 938 | 49 | 514 | StriC | skriver | rapport | till | l1 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 104 | 940 | | 514 | StriC | sänder | rapport | till | l1 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 105 | 940 | | 514 | StriC | sänder | rapport | till | l3 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 106 | 940 | | 514 | StriC | sänder | rapport | till | 512 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 107 | 940 | | 514 | StriC | sänder | rapport | till | div | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 107 | 940 | 35 | l1 | StriC | tar emot | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 108 | 942 | | l1 | StriC | öppnar | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 108 | 940 | 34 | l3 | StriC | tar emot | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 109 | 943 | | l3 | StriC | öppnar | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 109 | 940 | 52 | 512 | StriC | tar emot | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 110 | 941 | 2 | 512 | StriC | öppnar | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 110 | 940 | 32 | Div | StriC | tar emot | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 111 | 944 | | Div | StriC | öppnar | rapport | från | 514 | StriC | txt | 939 | Rapport: Enligt Stridsplan tnr 928: Järvägsövergång vid GRUNDBRO tagen. Fortsätter framryckning. |
| 111 | 932 | 21 | l1 | undb1 | skriver | ori | till | l3 | StriC | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 112 | 933 | 15 | l1 | undb1 | Sänder | ori | till | l3 | StriC | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 113 | 933 | | I1 | undb1 | Sänder | ori | till | Div | StriC | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 113 | 933 | 8 | I3 | StriC | tar emot | ori | från | I1 | undb1 | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 114 | 934 | | I3 | StriC | öppnar | ori | från | I1 | undb1 | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 114 | 933 | 21 | Div | StriC | tar emot | ori | från | I1 | undb1 | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |
| 115 | 935 | | Div | StriC | öppnar | ori | från | I1 | undb1 | ppt | 928 | Ori: Stridsplan för 512:s understöd av 514 strid vid GRUNDBRO |

# Appendix C. Agent sample code

This is the code for the mobile Jess agent. It has been edited for readability.

```
/*
 *  MobileAgent
 *
 *
 *  DESCRIPTION
 *  A Java Mobile Agent. This agent travels to a Concordia Server
 *  and retrieves information from the server and the machine
 *  it is running on.  When finished the agent can report the
 *  information.
 *
 */

import java.util.*;
import java.util.zip.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

import COM.meitca.concordia.*;
import jess.*;


/**
 * A Java mobile agent that retrieve files from another computer that has files
 * that includes a keyword.
 * @see          Agent
 * @author       Andreas Johansson
 */
public class MobileAgent extends Agent implements JessListener {
   /**
    * A list of information the agent has retrieved.  As the
    * agent travles this member variable PERSISTS from machine
    * to machine.  No special action must be performed by the
    * programmer to save state between stops
    */

    String[] keywords;
    Vector found = null;
    Rete r = null;
    ReteTmpObject rto = null;

    /**
     * Constructs a MobileAgent.  The constructor will be called
     * by the AgentLaunchWizard on the machine where the
     * wizard is run.
     */
    public MobileAgent() {
        found = new Vector();
        keywords = new String[3];
        r = new Rete();
        try{
            DumpFunctions df = new DumpFunctions();
            r.addUserpackage(df);
        }catch(Exception e){
            System.out.println("MobileAgent.constructor: An error occurred.\n" +
                               e.getMessage());
        }
    }

    /**
     * This method gathers information about the server and
     * its host machine.
     */
    public void getServerData() {
        try {

            System.out.println();
            System.out.println();
            System.out.println("Agenten kör");
            System.out.println();

            keywords[0] = "Andreas";
            keywords[1] = "O-998202";
           keywords[2] = "Sune";
           File dir = new File("C:\\Andreas");

           scanDir(dir);

            System.out.println();
            System.out.println("Jess innehåll:");
            r.executeCommand("(assert (framme hos den andra datorn))");
            r.executeCommand("(defrule test (tillbaka igen) => (assert (regeln aktiverad)))");
```

```
          Enumeration rules = r.listDefrules();
          while(rules.hasMoreElements()){
              System.out.println("Rules: " + rules.nextElement().toString());
        }
        Enumeration facts = r.listFacts();
          while(facts.hasMoreElements()){
              System.out.println("Facts: " + facts.nextElement().toString());
        }
        System.out.println();

        } catch (Exception error) {
            System.out.println("MobileAgent.getServerData: An error occurred.\n" +
                               error.getMessage());
        }
      }

/**
 * This method reports the information gathered during the trip.
 */
public void reportServerInfo() {

      OccurenceInfo foundData;
      String oldPath = "";

      try{
          System.out.println();
          System.out.println("Agenten är tillbaka.");
          System.out.println();
        System.out.print("Rader med ");
        int n=0;
        while(n < keywords.length && keywords[n] != null){
           if(n > 0)
              System.out.print("eller ");
           System.out.print("\'" + keywords[n++] + "\' ");
        }
          System.out.println("i:");

        Enumeration enum = found.elements();
          while(enum.hasMoreElements()){
              foundData = (OccurenceInfo)enum.nextElement();

             if(!oldPath.equals(foundData.getPath())){
                System.out.println("Funnet i filen " + foundData.getPath());
                oldPath = foundData.getPath();
             }
             System.out.println(foundData.getLineNumber() + ": " + foundData.getLine());

          }

          System.out.println();
          System.out.println("Jess innehåll:");
          r.executeCommand("(assert (tillbaka igen))");
          r.run();
          Enumeration rules = r.listDefrules();
          while(rules.hasMoreElements()){
            System.out.println("Rules: " + rules.nextElement().toString());
        }
        Enumeration facts = r.listFacts();
          while(facts.hasMoreElements()){
            System.out.println("Facts: " + facts.nextElement().toString());
        }
        System.out.println();

        }catch (Exception error) {
           System.out.println("Mobile.reportServerInfo: An error occurred.\n" +
                              error.getMessage());
       }
     }

/**
 * This method recursivly scans the given directory after the keywords.
 */
private void scanDir(File dir) {
      try {

        if(dir != null){

           String[] list = dir.list();
           File file;

           int n = 0;

           while(list.length > n){
              file = new File(dir.getPath() + dir.separatorChar + list[n]);

              if(file.isFile()){
                 if(file.getName().toLowerCase().endsWith(".zip")){
```

```
                ZipFile zip = new ZipFile(file);
                System.out.println("Zip-fil funnen: " + zip.getName() +
                              " Packar upp den och undersöker filerna i den...");

                 Enumeration enum = zip.entries();

                 while(enum.hasMoreElements()){

                    ZipEntry entry = (ZipEntry)enum.nextElement();
                    System.out.println(file.getCanonicalPath() + "#" + entry.getName());
                  LineNumberReader lnr = new LineNumberReader(new BufferedReader(new
                                              InputStreamReader(zip.getInputStream(entry))));
                 parseFile(lnr, file.getCanonicalPath() + "#" + entry.getName(),
                             found, keywords);

                 }
               zip.close();

             }
              else{

                 System.out.println("Scannar fil " + dir.getPath() + dir.separatorChar +
                                    list[n]);
                 LineNumberReader lnr = new LineNumberReader(new BufferedReader(new
                                                                  FileReader(file)));
                 parseFile(lnr, file.getCanonicalPath(), found, keywords);

              }
          }
            else{
              System.out.println(dir.getPath() + dir.separatorChar + list[n] +
                                 " är en katalog. Rekurserar ner...");
                 scanDir(file);
              }
          n++;

            }
        }
        else
           System.out.println("dir var null!");

        } catch (Exception error) {
            System.out.println("MobileAgent.scanDir: An error occurred.\n" +
                               error.getMessage());
        }
    }


    /**
    * This method parse a file (sent to a LineNumberReader) looking for lines that
    * contains the keywords.
     */
    private void parseFile(LineNumberReader lnr, String filePath,
                           Vector found, String[] keywords){
        try{

           String line;
           int n = 0;
           int m;
           boolean foundInLine;

           while ((line = lnr.readLine()) != null){
              foundInLine = false;
              m=0;

              while(m < keywords.length && keywords[m] != null){
                 if(line.toLowerCase().indexOf(keywords[m++].toLowerCase()) > -1)
                    foundInLine = true;
              }
             if(foundInLine){
                OccurenceInfo oi = new OccurenceInfo();
                oi.setPath(filePath);
                oi.setLine(line);
                oi.setLineNumber(lnr.getLineNumber());
                   found.addElement(oi);
                   n++;
                }
             }
             lnr.close();

        } catch (Exception e) {
            System.out.println("MobileAgent.parseFile: An error occurred.\n" +
                               e.getMessage());
        }

    }
```

```
public void prepareForTransport()
{
    try{

        r.executeCommand("(save-facts \"C:/facts.tmp\")");
         r.executeCommand("(bsave \"C:/bsave.tmp\")");
        File facts = new File("C:\\facts.tmp");
        File bsave = new File("C:\\bsave.tmp");
        BufferedInputStream factsIS = new BufferedInputStream(new FileInputStream(facts));
        BufferedInputStream bsaveIS = new BufferedInputStream(new FileInputStream(bsave));
        rto = new ReteTmpObject((int)facts.length(), (int)bsave.length());
        factsIS.read(rto.facts);
        bsaveIS.read(rto.bsave);
        factsIS.close();
        bsaveIS.close();
        facts.delete();
        bsave.delete();
        r = null;
    }catch(IOException e){
        System.out.println("MobileAgent.prepareForTransport: An IO-error occurred.\n" +
                           e.getMessage());

    }catch(JessException e){
        System.out.println("MobileAgent.prepareForTransport: An Jess-error occurred:\n" +
                           e.getMessage());

    }
}

public void completedTransport()
{
    try{

       r = new Rete();
       DumpFunctions df = new DumpFunctions();
        r.addUserpackage(df);
        df.addJessListener(this);

        File facts = new File("C:\\facts2.tmp");
        File bsave = new File("C:\\bsave2.tmp");
        BufferedOutputStream factsOS = new BufferedOutputStream(new
                                           FileOutputStream(facts));
        BufferedOutputStream bsaveOS = new BufferedOutputStream(new
                                           FileOutputStream(bsave));
        factsOS.write(rto.facts);
        bsaveOS.write(rto.bsave);
        factsOS.close();
        bsaveOS.close();

        r.executeCommand("(bload \"C:/bsave2.tmp\")");
        r.executeCommand("(load-facts \"C:/facts2.tmp\")");
        facts.delete();
        bsave.delete();

    }catch(Exception e){
        System.out.println("MobileAgent.completedTransport: An error occurred.\n" +
                           e.getMessage());
    }
}

/**
 * Used to reattach the parser to the Jess engine after a bload.
 * @param je
 */
public void eventHappened(JessEvent je) throws JessException
{
    if (je.getType() == JessEvent.BLOAD){
        // Reattach parser to engine
        r = (Rete) je.getObject();
    }
}


/**
 * The data object that holds information about one found occurence of the keyword.
 */
private class OccurenceInfo implements Serializable
{
    String path;
    String line;
    int lineNumber;

    public void setPath(String path){
        this.path = path;
    }

    public void setLine(String line){
        this.line = line;
```

```
            }

            public void setLineNumber(int lineNumber){
                this.lineNumber = lineNumber;
            }

            public String getPath(){
                return path;
            }

            public String getLine(){
                return line;
            }

            public int getLineNumber(){
                return lineNumber;
            }

        }

        private class ReteTmpObject implements Serializable
        {
            public ReteTmpObject(int size_facts, int size_bsave)
            {
                facts = new byte[size_facts];
                bsave = new byte[size_bsave];
            }

            byte[] facts;
            byte[] bsave;

        }

}
```