THE UNIVERSITY *of* LIVERPOOL

Information Management, Condition Monitoring and Control of Power Systems over Internet Protocol Networks

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy

in

Electrical Engineering and Electronics

by

D. P. Buse, BSc

September 2003

Acknowledgements

This work was supported by an EPSRC doctoral studentship. Additional support was provided by the National Grid Company PLC. The facilities of the Department of Electrical Engineering and Electronics and the National Instruments E-Automation Laboratory at the University of Liverpool were essential to the completion of my studies.

I would like to thank my supervisor Professor Q. H. Wu for his valuable direction and support throughout my PhD studies. I would also like to thank John Fitch of the National Grid Company, for supporting this project and providing assistance with the substation simulator and IMU, and Brian Baker, also of the National Grid Company, for his support of the project during its initial stages. Finally, I would like to thank Pu Sun, who worked on the hardware architecture and the data acquisition system for the prototype, and with whom I collaborated on the writing of several papers based on the project, and Jun Qiu Feng, who worked on the human-machine interface for the prototype system.

Abstract

This thesis presents an architecture for the information management, online monitoring and control of distributed power system substations over computer networks using the Internet Protocol suite. A modern substation automation system is made up of a number of different monitoring, control and protection devices (known as Intelligent Electronic Devices or IEDs), each having a microprocessor, analogue and/or digital input/output connections and a network interface. These devices collect and make available a large quantity of data such as condition monitoring data and fault and event records. In addition to this data, there is also information, including maintenance records and technical documentation, stored at various locations in the power system. However, the amount of data available makes it difficult for engineers and management to retrieve the data required for a particular purpose and to make use of it in an appropriate manner. The aim of this thesis is therefore to develop an open architecture for access to power system data and devices, which provides the functionalities of information management and online control, as well as supporting further work to create a fully integrated and intelligent automation system.

The architecture proposed and evaluated in this thesis is based on the concepts of multi-agent systems and mobile agents. Multi-agent systems provide a useful methodology for the modelling and implementation of distributed systems, particularly those consisting of a number of largely autonomous components. The extensive use of multi-agent systems in various areas, including industrial and manufacturing systems, suggests that the multi-agent systems methodology may also be appropriate for the design of power system automation systems.

Within this architecture, software agents are used to represent the various data sources and components of an electricity substation. This representation consists of multiple levels, including substation plant (transformers, switchgear and other components), data acquisition and control systems and information infrastructure (databases and document repositories). The use of agents provides a standard communications interface for access to data sources which use different data models or communications protocols. By using the directory facilities provided by a Foundation for Intelligent Physical Agents (FIPA)-compliant agent platform, it is possible to reconfigure the system by adding and removing new agents, without having to change the other system components. The multi-agent system also provides a basis for the development of intelligent cooperative control schemes in the future.

A mobile agent is a software program with the capability to suspend its execution and resume it on another computer. Mobile agents may be distinguished from mobile object technologies, such as Java applets, in that mobile agent transfer is initiated by the agent itself, rather than by the client computer, and also in that when a mobile agent is transferred it takes with both data and code, whereas in mobile object systems only code is transferred. Mobile agents have been successfully applied to a range of problems including information retrieval and software update, and various experiments have shown that they can provide performance gains in these applications. In this thesis, mobile agents are applied to the retrieval and analysis of substation data, and to remote operator intervention. Experimental and theoretical results demonstrate that mobile agents can provide a performance advantage for these applications in comparison to client-server or static agent methods.

This thesis also describes a prototype implementation of the architecture, in the form of a substation information management system, which has been demonstrated and evaluated using a substation simulator. This simulator is used to test substation control equipment and provides the same analogue and digital I/O as would be provided by the substation plant. Therefore, the simulator provides a realistic test environment. The prototype demonstrates agent-based data acquisition, information management and data analysis using mobile agents. Data is gathered by a data acquisition system and stored in the National Grid Company's Information Management Unit (IMU) database. It is then accessed via the database agents and the user interface agent. The implementation of the prototype system demonstrates the viability of the proposed architecture and provides various insights into its advantages and disadvantages. The architecture is also evaluated theoretically with respect to its performance, modifiability, functionality and reliability.

Contents

List of Figures

List of Tables

Definitions and Abbreviations

ACC: Agent Communications Channel. The component of a FIPA platform which provides inter-agent messaging services.

ACL: Agent Communications Language. A language for use in inter-agent communications. The specific ACL used in this project is the Foundation for Intelligent Physical Agents Agent Communication Language (FIPA ACL).

ActiveX: A software component model developed by Microsoft and used in Windows programming.

AMS: Agent Management Service. The component of a FIPA platform which provides agent management and lifecycle services (e.g. start agent, delete agent, move agent).

API: Application Programming Interface. A set of functions in a library or software system that may be called by another program.

Architecture: In this thesis, the term "architecture" refers to a software architecture (the structure of the software system and how the components of that system interact).

BDI: Belief - Desire - Intention. An architecture for intelligent agents.

CIM: Computer Integrated Manufacturing.

CORBA: Common Object Request Broker Architecture. A distributed object system developed by the Object Management Group, a standards consortium consisting of approximately 800 companies and organisations.

CPU: Central Processing Unit.

CS: Client/Server.

DAQ: Data Acquisition.

DB: Database.

DCOM: Distributed Component Object Model. A distributed object system developed by Microsoft.

DF: Directory Facilitator. In a FIPA platform, the DF allows agents to register descriptions of the services they provide. Other agents may then search the DF to locate agents providing a particular service.

DLL: Dynamic Link Library. On the Windows operating system, a library that can be loaded by a program at runtime.

DNP: Distributed Network Protocol. A protocol for SCADA communications managed by the DNP users group, an industry consortium.

EMS: Energy Management System.

FIPA: Foundation for Intelligent Physical Agents. A non-profit consortium of approximately 25 members including companies, universities and research institutes which aims to produce standards for heterogenous, interacting agents and agent-based systems.

FIPA platform: A FIPA platform is an agent platform which implements the FIPA standards.

GOOSE: Generic Object Oriented Substation Events.

GOMSFE: Generic Object Model for Substation and Feeder Equipment.

HMI: Human Machine Interface.

HTML: Hypertext Markup Language. The language in which World Wide Web pages are written.

HTTP: Hypertext Transfer Protocol. The protocol used by the World Wide Web to retrieve web pages. HTTP is used in combination with TCP and IP.

IEC: International Electrotechnical Commission. An international standards body for electrical, electronic and related technologies.

IED: Intelligent Electronic Device. A monitoring or control device having a central processor and a number of analogue or digital input or output channels.

IIOP: Internet Inter-ORB Protocol. The communication protocol used by CORBA.

IMU: Information Management Unit. A system developed by the National Grid Company for the storage of substation data.

IP: Internet Protocol. The network layer protocol used for Internet communications, which handles the movement of data packets around the network. IP may be used with different transport layer protocols, which provide the required services to transfer data between hosts using IP. These include UDP and TCP.

IPMT: Internal Platform Message Transport. The component of a FIPA platform which transfers messages between agents on the same platform.

Java RMI: Java Remote Method Invocation. A distributed object system included as part of the Java platform.

JADE: Java Agent Development Environment. A free, open source FIPA platform developed by Telecom Italia Laboratories and other contributors. Widely used for agent and multi-agent system development.

JDBC: Java Database Connectivity. A set of software libraries used to access a database and included in the Java platform.

Jini: A distributed systems toolkit developed by Sun Microsystems. Jini is based on Java and Java RMI and provides service location and service discovery facilities.

JNI: Java Native Interface. A system which allows Java programs to call libraries written in other languages such as C.

Kbit/s: Kilobits per second.

Kbps: Kilobits per second.

LAN: Local Area Network.

LN: Logical Node (in IEC 61850 standard).

MA: Mobile Agent.

Mbit/s: Megabits per second.

Mbps: Megabits per second.

MTS: Message Transport Service. Transfers messages between FIPA platforms.

Ontology: A data model defining a hierarchy of related concepts for use in an information system.

ORB: Object Request Broker. In a CORBA system, the ORB permits client objects to call methods on server objects regardless of where the sesrvers are located and in what programming language they are implemented.

PRS: Procedural Reasoning System. An implementation of the BDI architecture.

PXI: PCI Extensions for Instrumentation. An interface specification, based on Compact PCI, developed by National Instruments for PC-based instrumentation devices.

PICOM: Piece of Information for Communication (in IEC 61850 standard).

RTU: Remote Terminal Unit. A data collection device situated at a remote site and connected to a host SCADA system.

SCADA: Supervisory Control and Data Acquisition. A SCADA system consists of a central supervisory computer and a number of subordinate units situated at remote sites to perform monitoring and control functions.

SCS: Substation Control System.

SDI: Selective Dissemination of Information. An SDI system sends documents to a user as they become available based on a profile of the user's interests.

SICAP: Substation Information, Control and Protection. A National Grid Company strategy for the application of substation equipment and systems to provide improved information management and integrate different plant functions.

SL: Semantic Language. A logic-based language developed by the Foundation for Intelligent Physical Agents for use in the content of Agent Communications Language messages.

SQL: Structured Query Language. The standard language used for database queries.

TCP: Transport Control Protocol. A connection-oriented, reliable, transport protocol used for the transmission of data on the Internet and in local networks. TCP is used in combination with IP.

TF/IDF: Term Frequency / Inverse Document Frequency. A method of determining the relevance of a document to a query.

Thread: A thread is a single path of execution through a program. If a program has multiple threads of execution, these may proceed in parallel. On a multi-processor machine, multiple threads may be executed simultaneously. On a single processor machine, thread execution will be interleaved by the operating system.

UCA: Utility Communications Architecture. A set of standards for substation communications developed by the Electric Power Research Institute (EPRI).

UDP: User Datagram Protocol. A non connection-oriented transport protocol which provides a simpler data transfer mechanism than TCP.

UML: Unified Modelling Language. A set of standard diagrams and notations developed by the Object Management Group, the industry consortium responsible for CORBA, for modelling object-oriented software systems.

URL: Uniform Resource Locator. A string that identifies a resource (e.g. Web page or file) in the World Wide Web.

WAN: Wide Area Network.

Mathematical Definitions

The term "server" used here refers to a computer holding either data or a mobile agent server. The more general term "computer" includes both server computers and client computers which may not necessarily have these facilities.

B_{a}: Size of a mobile agent a.

$B_{\mathrm{interactions}}$: Total amount of data transferred for all interactions between a client agent or program and a server agent or program, not including mobile agent transfer time.

B_j: Amount of data transferred for the j^{th}interaction between a client agent or program and a server agent or program.

B_{msg}: Size of a message *msg*.

B_{rep}: Size of reply message *rep*.

B_{results}: Size of results message.

B_{req}: Size of request message *req*.

$B_{\mathrm{RPC}}(L_1, L_2, B_{\mathrm{req}}, B_{\mathrm{rep}})$: Amount of data transferred when a client program at location L_1performs a remote procedure call to a server program on location $L_2$with message sizes $B_{\mathrm{req}}$for the request and $B_{\mathrm{rep}}$for the reply.

$O_{\mathrm{dereg}}(a, s)$: Overhead (time) of registering agent $a$on server s.

$O_{\mathrm{R}}(msg, dest)$: Overhead (time) involved in receiving message msgon destination computer $dest$.

$O_{\text{reg}}(a, s)$: Overhead (time) of registering agent a on server s.

$O_{\text{S}}(msg, src)$: Overhead (time) of sending message $msg$from source computer src.

T_{CS}: Total time taken to perform a task using a client-server system.

T_{MA}: Total time taken to perform a task using a mobile agent.

$T_{\text{msg}}(m, src, dest)$: Time to transfer message m from computer src to computer $dest$.

T_{msg}: In general, the time to transfer a message across the network.

$T_{\text{proc}}(s)$: Time to perform processing on server$s$.

$T_{\text{ret}}(s)$: Time to retrieve data for an analysis operation from server s. May be abbreviated to T_{ret}if only one server is present.

$T_{\text{RPC}}(L_1, L_2, B_{\text{req}}, B_{\text{rep}})$: Time for a client program at location $L_1$to perform a remote procedure call to a server program on location $L_2$with message sizes $B_{\text{req}}$for the request and $B_{\text{rep}}$for the reply.

$T_{\text{transfer}}(a, src, dest)$: Time to transfer agent $a$from computer $src$to computer $dest$.

T_{transfer}: In general, the time to transfer an agent across the network.

$\delta(a, b)$: Network latency (delay) between computer $a$and computer b.

$\tau(a, b)$: Bandwidth between computer $a$and computer b.

Chapter 1

Introduction

A modern power system contains a large number of monitoring and control devices, for example, the National Grid Company (NGC) in the UK operates a transmission network of 244 substations at 275kV or 400kV, with a further 82 substations at 132kV and below [1]. Whereas previously each substation had a centralized control system, modern substations are being equipped with many distributed Intelligent Electronic Devices (IEDs) performing various tasks [2]. Each of these IEDs is capable of sending and receiving data, often via a network, resulting in a large quantity of data becoming available. It is claimed that utilities are "among the largest users of data" and "the largest users of real-time data" [2]. However, engineers now have more data available than they are capable of managing in the time available to them [3]. In order to manage this amount of data, and allow the utility engineers and management to make use of it in an appropriate manner, various systems and architectures have been proposed and developed which aim to integrate the data from different IEDs and make it available to users [2]. Many of these (for example [4][5][6]) are based on client-server methodologies and protocols such as Hypertext Transfer Protocol (HTTP).

In comparison to client-server and object-oriented systems, multi-agent systems have several claimed advantages. Jennings [7] states that "the natural way to modularize a complex system is in terms of multiple autonomous components that can act and interact in flexible ways in order to achieve their set objectives", and also that agents provide a "suitable abstraction" for modelling systems consisting of many subsystems, components and organizational relationships. Ferber [8] describes how agents, as a form of distributed artificial intelligence, are suitable for use in application domains which are themselves widely distributed. The modern power grid, with substations distributed throughout a wide area, falls into this category of systems.

This thesis describes a substation automation architecture based on the multi-agent systems methodology. This chapter begins by presenting the historical background of substation automation systems, along with the newer network-based approaches and architectures. Agents, multi-agent systems and mobile agents are also introduced, and the main contributions of the thesis are presented.

1.1 Introduction to Industrial and Power System Automation

The term *industrial automation* covers a range of systems used to improve the productivity, safety or product quality of an industrial concern [9]. Normally, the main function of any industrial automation system is to control a process being performed.

Industrial automation systems may be applied to a wide variety of industries, which may be approximately grouped into the two categories of *process industries* or *continuous process industries*, such as electric power systems and other utilities, and *discrete manufacturing industries,* which include industries in which individual items, such as motor vehicles or electronic goods, are produced. The type of automation system that is appropriate to a process industry may differ from the type of automation system that is appropriate to a discrete manufacturing industry [10].

There are several models of industrial automation systems in common usage. One of the more well-known is the *Computer Aided Manufacturing (CIM)*[1]

[1]In the 1980's, the term *computer integrated manufacturing (CIM)* was used to describe

pyramid model, in which the system is viewed as a series of layers, ranging from low-level data acquisition and control functions to high-level functions such as plant and process management [9]. This model is shown in Figure 1.1.

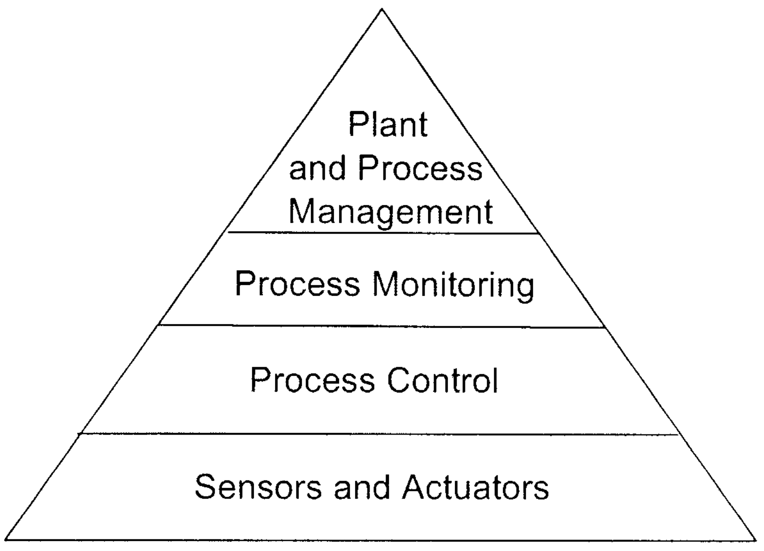


Figure 1.1: "CIM pyramid" Model of an Automation System

A more detailed model of industrial automation systems is the CIM reference model developed by the International Purdue Workshop on Industrial Computer Systems [11]. This model describes in a generic fashion the tasks, and to some extent the implementation, of an "integrated information management and automation system", with most of the description aimed at the manufacturing industries. As with the CIM pyramid model, the outline structure of this model is hierarchical, consisting of a number of *levels*. The five levels included [11] are:

- Operational management: This level is responsible for overall production scheduling, coordination and reliability assurance.
- Section/Area: the duties of units at the area level include production scheduling, maintenance and local cost optimization for a particular area. The area level is also responsible for generating production reports and analysis of operational data.

a range of industrial automation systems based on computers.

- Supervisory control: Units at the supervisory control level are responsible for responding to emergency conditions in the plant, optimizing the operation of the controllers and maintaining "data queues" for lower-level units under their control.
- Control level: the duties of this level include direct control of plant, human/machine interface and the collection of information for transmission to higher levels.
- Equipment: this level includes individual machines, sensors and actuators.

However, it is stated that "the number of levels used in a factory model is arbitrary", and that the six levels serve only to assist the standardization process [11]. Within the hierarchical structure, control flows either within a level or downwards through the levels, and information / data flows upwards [11].

It has been suggested [11] that the usual method of implementation of a CIM system is to use a "hierarchy of separate computers". In support of this statement, the authors of [11] state that hierarchical systems provide the ability to implement distributed control of the plant, with each computer controlling a local area, and that the hierarchical structure follows the usual human management structure of a plant. It is conceivable that a multi-agent approach might be one way to implement such a hierarchical structure (see the description of hierarchical multi-agent systems, Section 1.2.2).

1.1.1 Automation Systems in Electricity Transmission

Electricity transmission networks consist of a number of substations interconnected by transmission lines. Each substation contains transformers, switchgear (disconnectors and circuit breakers) and other items of plant and protective equipment [12].

In the transmission industry three types of automation system are used: Supervisory Control And Data Acquisition (SCADA) systems, Energy Management Systems (EMS) and Substation Automation Systems (SAS) [13]. These form a hierarchical structure, with the EMS on the top level, the SCADA system directly subordinate to the EMS, and, at the lowest level, the individual SASs of each substation.

Energy Management Systems

An energy management system controls the overall operation of the power system [14]. The components of an EMS are [15]:

1. Network analysis, including state estimation, load flow optimization, dispatching and voltage control.
2. Generation scheduling and control.
3. Data storage and retrieval.
4. SCADA, including data acquisition, alarms and HMI.

Therefore, a SCADA system forms one component of an energy management system. In the conceptual model of an EMS presented in [15], the SCADA system connects the plant, or "external equipment" to the data base, with the other two functions (network analysis and generation scheduling) operating on the stored data.

SCADA systems

The SCADA system of a power system, shown in Figure 1.2, is responsible for data acquisition, human-machine interface functions and alarm/event processing [15]. In a SCADA system, a centrally located master computer is connected via some form of network to a number of Remote Terminal Units (RTUs), located in the substations and connected to the local substation automation system or substation control system (SCS). The master periodically polls the RTUs to retrieve status information, and can send commands back to the RTUs for execution by the substation automation system.

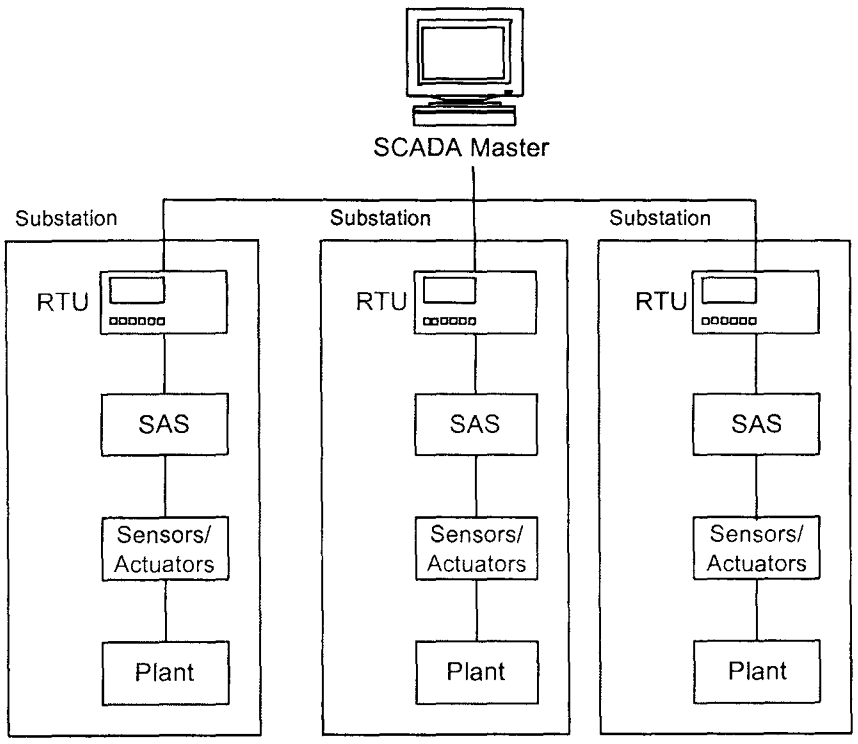


Figure 1.2: Supervisory Control and Data Acquisition (SCADA) System

Substation Automation Systems

A substation automation system (SAS) is used to monitor and control a single substation, and to collect data for transmission to the overall SCADA system [16]. The main functions required of an SAS are *control* and *protection.* Control involves the operation of the plant, either locally or from a remote location. Protection, normally performed at the lowest level of the system by a number of relays, prevents damage to the system in the event of a fault. Various protection functions are provided, such as overcurrent, overvoltage and thermal overload. A more modern SAS will also provide a graphical human-machine interface, condition monitoring and historical data logging [5], along with remote access for information.

1.1.2 Network-Based Power System Automation

Traditional substation automation systems have a number of drawbacks. The interoperability of devices is hampered by "an excess of incompatible hardware interfaces and protocols" [17], and more access to substation information is required in order to make business decisions [2]. In recent years a number of new architectures and products have been developed which aim to address one or more of these drawbacks. In particular, in the last 10 years there have been moves to integrate local area networks (LANs) into industrial systems. This has led to the development of several industry-oriented network architectures, such as fieldbuses, Modbus and Profibus [18]. Also, a number of systems now use standard Ethernet networks [19][20] and the TCP/IP Internet protocol suite.

The use of Ethernet and TCP/IP in an automation system has an important advantage in that it allows control systems to be connected directly to office and enterprise networks, which themselves usually use Ethernet and TCP/IP. This means that data collected by the control system can be shared with other systems such as databases and Enterprise Resource Planning (ERP) systems, and can be viewed by users who are not located at the production site. It is also possible to make process information accessible over the World Wide Web [6], which removes the need for a specific client program to be installed on a user's system as data can be viewed with a Web browser. However, even when a web browser is used it may still be necessary to download a large software component to the user's machine to allow online monitoring data to be displayed.

A recent system installed in an Australian substation, described in [16], consists of a number of distributed Remote Terminal Units (RTUs) connected by a fibre-optic ring network. The RTUs communicate with "intelligent relays" and input-output devices, and there are duplicate HMIs and communication links to the SCADA system at the control centre.

Client-Server and Distributed Object Systems

Many current networked automation systems employ client-server technology, similar to that shown in Figure 1.3. A number of devices, either simple sensors or actuators or, more commonly, *intelligent electronic devices* (IEDs), which incorporate an embedded processor, are connected via a network, such as Fieldbus or Ethernet, to one or more servers. These servers normally run the applications which perform the centralized control functions of the system, including supervisory control, alarm and event management and data storage. A number of clients can then connect to the servers either using client programs or a Web browser[19] [4].

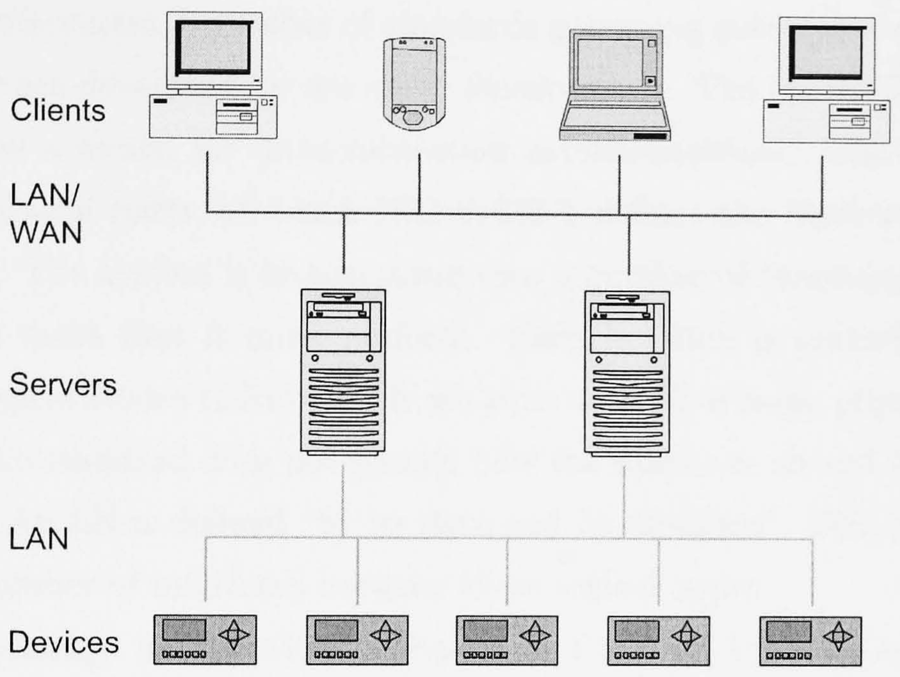


Figure 1.3: Client-Server Automation System

A client-server system employs a "request-reply" method of interaction. The client system sends a request to the server, which must then carry out a specified action, such as retrieving a Web page for display, and transmit a reply to the server. An example of a client-server protocol is HTTP, which is used to display information stored on a Web server.

Distributed object systems, such as CORBA [21] [22], Microsoft D/COM and Java RMI, provide an alternative methodology for distributed programming. A distributed object system extends the object-oriented programming methodology to cover objects located on multiple computers. Server objects may expose a number of methods via some standardized interface description, which other objects may invoke by sending a message to the server [23]. Unlike agent-based systems, the interface of a distributed object is pre-defined (in terms of its methods), and the interactions are normally synchronous (an object must wait for a method to complete before it can continue execution).

Relevant Standards

To address the problem of interoperability between IEDs produced by different manufacturers, a number of standards governing substation communications have been developed, or are under development. The IEC 61850 standard [24] specifies a model for intra-substation communications. The standard is split into several parts, of which IEC 61850-5 defines the basic structure for the system. The system is broken down into a number of functions, which are the various tasks that it must perform. Each function is performed by one or more Logical Nodes (LNs), which are situated within some physical device. However, the standard does not specify how the functions should be allocated to devices. An LN is defined "by its data and its methods". IEC 61850-5 also defines a number of interfaces between these logical nodes.

Data exchange in IEC 61850 is based on Pieces of Information for Communication, or PICOMs. A PICOM consists of an item of data to be transferred, along with information about its data type, permissible transmission time, source logical node and destination logical node or nodes. The standard also includes an object model (based on GOMSFE: Generic Object Model for Substation and Feeder Equipment) [25], and an event system for use in communications between protection devices (GOOSE: Generic Object Oriented Substation Events) [26].

Earlier substation communications standards include the Distributed Network Protocol (DNP) [27] and Utility Communications Architecture (UCA) [28].

1.1.3 The National Grid Company System

The substation control systems currently operated by the National Grid Company [29] are distributed control systems with a centralized substation control system (SCS) and a number of bay level remote terminal units (RTUs). However, in most of these systems the processing is centralized at the SCS. The substations are currently linked via "private wire" circuits to the Grid Control Centre. These will eventually be replaced by an optical fibre network, providing a TCP/IP based Wide Area Network between substations and other locations, as well as a dedicated control network running an IEC protocol. Each substation will also have a local area network installed [30], based on optical fibre Ethernet and the TCP/IP protocol.

The National Grid Company has also instigated trials of Internet technology and servers within its substations [6]. The system used is based on a server called the Information Management Unit (IMU). The IMU contains a Microsoft SQL Server database, in which data from IEDs is stored. This data is then made available to clients via a Web server situated on the IMU. The company found that the use of Internet technology provided "a way to minimize routine, repetitive tasks and also to plan pre-emptive maintenance" [6].

1.2 Introduction to Agents, Multi-Agent Systems and Mobile Code

While there is no fixed definition of an *agent* (or *software agent*), the concept is typically used to refer to software components that have their own thread of control (and hence may act autonomously), and are capable of sensing and reacting to changes in some environment. Often software agents have other properties, such as the ability to communicate with other agents. Recently, software agents have become widely used in the modelling of complex, distributed, problems [7]. This section discusses in more detail the various types of agents that are in use, along with multi-agent systems, which are systems in which agents interact in order to solve some problem or achieve a set of goals, and mobile agents, which are agents capable of moving from one server to another during their execution.

1.2.1 Intelligent / Autonomous Agents

There are a number of definitions of an *intelligent agent.* One of the more widely used is that put forward by Wooldridge and Jennings [31], which defines an agent as a system that "enjoys the following properties:

- autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
- social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language
- reactivity: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it
- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative"

In the same paper, Wooldridge and Jennings go on to describe the notion of a *strong agent,* used by researchers in the artificial intelligence field, which is an agent that "is either conceptualized or implemented using concepts that are more usually applied to humans". An example of a strong agent is one based on a *mental state* described in terms of beliefs, desires, intentions and commitments.

Strong agents can also be known as *cognitive* agents, in comparison to simpler *reactive agents*, which are agents that act only in response to changes in the environment [8]. One of the simplest type of reactive agent is an agent having a series of IF-THEN rules, mapping from input states to actions.

Agent Architectures

There are several different architectures for intelligent agent implementation. A well-known cognitive architecture is the **Belief-Desire-Intention**

(BDI) architecture [32, 33], in which the agent's knowledge base is described by a set of *beliefs,* (those facts which an agent considers to be true) *desires* (those conditions which the agent wishes to bring about) and *intentions* (actions which the agent has committed to perform). These are explicitly represented in the knowledge base; for example, the Procedural Reasoning System (PRS) implementation [33] represents beliefs and goals as ground literals (sentences containing no implications, binary operators or variables) in first-order logic [34]. As described in [34], a BDI agent is capable of both reactive and deliberative behaviour. On each execution cycle of the interpreter, the agent retrieves new events from the environment. It then generates a set of *options*, which are plans or procedures that the agent is capable of carrying out, both in response to events and in order to achieve its goals. The agent will then execute, or partially execute, one or more of the selected options. This process is repeated for the agent's lifetime.

The **subsumption architecture** [35] is an example of a reactive architecture which does not employ an explicit knowledge representation. A subsumption agent consists of a number of concurrently-executing *behaviours* [36]. These are arranged in a number of layers, with lower layers representing simpler behaviours, which have a high priority, and higher layers representing more abstract behaviours, and having lower priority. Low-level behaviours are unaware of the presence of the high-level behaviours. It is therefore possible to construct an agent using the subsumption architecture starting with the lowest-level behaviour and working upwards, with the agent being functional, at least to a certain extent, after each layer is constructed. For example, Brooks [35] describes a mobile robot with a number of layers, performing tasks such as "avoid objects" (the lowest layer), "wander", etc, up to "plan changes to the world" and "reason about behaviour of objects" (the highest layer).

Machine learning methodologies, such as reinforcement learning [37], genetic programming [38], or inductive logic programming [39], may be used to enhance the performance of an agent. While it is possible to use learning to improve the capabilities of an agent using an architecture such as BDI (for example, [40] used machine learning methodologies to recognize plans being undertaken by other agents in a BDI architecture and [41] uses case-based reasoning in a BDI agent for information retrieval), it is also common to incorporate learning into a much simpler agent architecture. Learning agents have been applied in a number of domains, including user interfaces [42], telecommunications [43], control and robotics [44].

Layered architectures such as TouringMachines [45] and INTERRAP [46] are cognitive architectures consisting of one or more layers. According to Ferguson, the advantage of a layered architecture is that a layered agent, by having different levels of behaviour operating concurrently, is capable of reacting to changing circumstances while planning its future actions and reasoning about the behaviour of other agents. Both of the architectures mentioned have three layers: TouringMachines has a reactive layer, modelling layer and planning layer, while INTERRAP has a behaviour-based layer, local planning layer and cooperative planning layer. In TouringMachines, all three layers are connected to the agent's sensors and effectors. The three layers operate concurrently and are unaware of each other, while a control mechanism is used to filter the inputs and outputs and prevent conflicts. In INTERRAP, the sensors and effectors are connected only to the lowest layer (the behaviour-based layer). Activation requests are passed upward through the layers, and commitments are passed downwards. Unlike the subsumption architecture (which is a form of layered architecture), both TouringMachines and INTERRAP are based on explicit knowledge representation [45].

1.2.2 Multi-Agent Systems

Ferber [8] defines a multi-agent system as a system consisting of an environment, a set of objects which exist in that environment and can be acted upon by agents and a set of agents, which represent the "active entities" of the system. Agents are related to objects by relations, and may act on those objects by means of operations.

Ferber defines two extreme classes of agent, a *purely communicating* agent, which is one which has no physical environment, and acts only by communicating with other agents, and a *purely situated* agent, which has no communication with other agents, but is situated in a physical environment and acts through that environment. Many multi-agent systems also employ agents that have features of both of these types [8].

Multi-Agent System Architecture

Several architectural styles have been used in the development of multi-agent systems. Shehory [47] describes four such organizations:

- Hierarchical multi-agent systems, in which agents communicate according to a hierarchical structure, such as a tree. A system such as the Open Agent Architecture [48], which uses brokers, is a hierarchical system, as each agent communicates only with a broker or facilitator agent. Shehory gives the disadvantage of such a system as the reduction in autonomy of the individual agents, as lower levels of the hierarchy depend on and may be controlled by higher levels. However, hierarchical architectures can greatly reduce the amount of communications required, and also the complexity and reasoning capabilities needed in the individual agents.
- Flat multi-agent systems, in which any agent may contact any of the others. These provide the greatest agent autonomy, but result in more communications between agents. Also, agents in a flat structure must either know the locations of their communications partners, or be provided with agent location mechanisms such as yellow pages services. Many smaller multi-agent systems appear to be developed using a flat organization.
- A subsumption multi-agent system is a system in which agents are themselves made up of other agents. In this system, the subsumed agents are completely controlled by the containing agents. This is similar to the subsumption architecture for an individual agent. According to Shehory, the fixed structure of a subsumption multi-agent system provides efficiency but restricts the flexibility of the system.

- A modular multi-agent system is comprised of a number of modules. Each module normally employs a flat structure, while inter-module communications is relatively limited. A modular multi-agent system might be useful in a situation such as power system automation, in which each substation could be categorized as a single module. Most communications within a power system are either within a substation or between a substation and the control centre, and so this might be an appropriate multi-agent system structure.

1.2.3 Mobile Agents

Mobile agent systems are systems which involve the transfer of a currently executing program, known as a *mobile agent*, from one location to another. Fuggetta, Vigna and Picco [49] state that "in the mobile agent paradigm a whole computational component is moved to a remote site, along with its state, the code it needs, and some resources required to perform the task." Mobile agents were first discussed in the early 1990s, and applications to a wide range of areas have been proposed or implemented. For example, [50] describes a distributed calendar application implemented using mobile agents, [51] describes a military information retrieval application and [52] describes the application of mobile agents to network monitoring.

There are a number of reasons why mobile agents might be used in any particular application:

- Mobile agents can provide performance improvements by reducing network load [53, 51].
- Using mobile agents can allow servers to be made more flexible, with components being added and removed at runtime [54].
- Mobile agents permit disconnected operation, in which a client can "launch" a mobile agent into the network, disconnect, and then reconnect to retrieve the results of the mobile agent's task [55].

However, because of concerns regarding mobile agent security and a lack of motivation to deploy mobile agents, there has so far been limited use of mobile agents in real applications [56]. The mobile agent security issue consists of two problems: protecting a host and its data from a malicious agent or other attacker, and protecting an agent and its data from a malicious host or another agent [57]. While the first of these problems may be, at least partly, solved, there is still ongoing research into the second [58].

1.3 Previous Agent-Based Automation System Architectures

There has been much previous work in the field of application architectures for multi-agent systems. For example, the RETSINA [59] architecture is a 3-tier architecture consisting of user agents, wrapper agents representing information sources and "middle agents" which transfer data between the two. Wrapper agents both "agentify" the data sources, allowing them to be queried using the agent communication language and convert data from the data models (ontologies) used by the individual data sources into a global ontology used for querying. Therefore, the wrappers assist in integrating data from heterogeneous data sources. RETSINA has been applied to several problems including financial portfolio management and visit scheduling. A similar information architecture, consisting of an ontology agent and database agents, is described in [60]. These architectures provide a basis for the development of a multi-agent information management architecture. However, they do not include the other functions used in power system automation, such as information management and control. Therefore, it is necessary to significantly extend these architectures to include this functionality.

A number of multi-agent systems have been employed to handle various aspects of industrial automation. For example, the ARCHON system [61] was used to perform fault identification and service restoration in a power transmission network. Seven agents were used, based on both existing and new expert systems in the control centre. Each agent was responsible for a particular task, for example, blackout area identification or control system interface. ARCHON agents consisted of two layers: the *ARCHON layer*, which was responsible for local control, decision making, agent communications and agent modelling, and the *AL-IS interface*, which handled communications between the ARCHON layer and the "intelligent system" being wrapped by the agent [62]. The application of ARCHON described was used only in the control centre, and was not a full substation automation system. It lacks much of the information management functionality required, for example, the storage and analysis of historical data. However, some of the general principles of the ARCHON system, including the use of wrappers, may be applied to the design of such a system.

There are many applications of agent technology in the manufacturing industries. These are similar in some ways to applications in the process industries and utilities, but often focus on machine control and task allocation. For example, the PABADIS project [63] aims to develop a system for agent-based manufacturing. The PABADIS system contains agents representing machines and products [64]. The machine agents register descriptions of their capabilities with a lookup service, which can be used by the mobile product agents to locate machines capable of carrying out the tasks involved in manufacturing a particular product. The PABADIS architecture demonstrates the use of a multi-agent system in an industrial process. However, the architecture of a manufacturing system differs from that of a utility. A utility system is a continuous process, whereas a manufacturing system contains discrete parts and outputs. Also, the area covered by a distributed utility system is much wider than a single factory. In this work it is hoped to make use of the general agent-oriented principles used by PABADIS (use of directories and representation of components of the plant as agents) but to design an architecture more suited to the power systems and continuous process field.

Bussmann and Schild [65] used a multi-agent system in the control of a flexible manufacturing system. The system was used to manage the flow of material between different machines, and to allocate tasks to machines. It was applied to automobile manufacturing. The approach taken by this system was based on auctions, in which workpieces auctioned off tasks to machines. It was found that this system provided both improved throughput and increased robustness compared to traditional methods [66]. As with PABADIS, the relevance of Bussmann and Schild's work is restricted by the fact that their application is in the manufacturing domain.

Leito and Restivo [67] describe a multi-agent architecture under development for "agile and cooperative" manufacturing systems. As well as controlling the manufacturing system, the architecture supports re-engineering of products. The agent architecture consists of Operational, Supervisor, Product, Task and Interface agents. This architecture is also intended for use in manufacturing industries. However, it might be possible to use agents corresponding to the operational agent (which Leito and Restivo define as corresponding to the "physical resources") and supervisory agent in a utility system. Also, as the paper states, "only preliminary results are presented", and further work is therefore required.

Mangina et al [68] describe the use of a multi-agent system and modal logic for gas turbine condition monitoring in a power generating station. By reasoning about what causes a gas turbine to move from one state to another, the agents are able to identify the causes of faults. The same group have also developed the Condition Monitoring Multi-Agent System (COMMAS) architecture [69], which uses three layers of agent. Attribute Reasoning Agents (ARAs) monitor and interpret sensor data, Cross Sensor Corroboration Agents (CSCAs) combine data from different sensors, and Meta Knowledge Reasoning Agents (MKRAs) provide diagnostics based on the information provided by the other agents. Mangina's architecture is relevant to the power systems domain. However, it provides only the single task of condition monitoring, and does not contain information management and remote control or operation functionality.

These applications have been relatively successful, suggesting that the multi-agent approach is a promising method for the implementation of industrial automation systems. However, the previous work described does not provide a single architecture for providing all the functions required by a power system automation system, either because it focuses on a single application or because it is intended for use in manufacturing industries. The work described in this thesis is intended to provide such an architecture.

1.4 Problem Statement

The increasing use of intelligent electronic devices (IEDs) and networks in power system substations has led to the availability of a large amount of data and information of various types, and standard protocols such as IEC61850 [24] have improved the interoperability of different devices. However, it remains difficult to effectively manage the amount of data produced [70], and to convert this data into knowledge to enable engineers to make use of it [3]. A framework is required to provide open access to substation information via the power company's wide area network and to integrate previously separate functions such as protection, control and information management [29]. Another drawback with current automation systems is that they are inflexible and cannot easily accommodate new requirements or changes to the substation plant and monitoring equipment. It is hoped that a new architecture might be able to address this shortcoming.

The client-server model, used by most current systems, is widely supported and therefore provides a simple means to develop a distributed application. However, it is more suited to centralized applications, in which one server serves a number of clients, or one client controls a number of servers, than true distributed applications [71], and is lacking in flexibility. Distributed object systems, such as CORBA, provide a more suitable representation, in which the system is composed of separate objects. However, in a distributed object system objects do not usually have their own thread of control, which means that it is not possible for different parts of the system to act simultaneously.

Also, message passing in a distributed object system is usually synchronous, which means that an object that invokes a method on another object must wait for that object to respond before it can continue with any other tasks that it is involved in. The multi-agent approach provides increased autonomy by giving each agent its own thread of control, and provides asynchronous message-passing. Multi-agent systems also provide a high-level communications language (Foundation for Intelligent Physical Agents Agent Communication Language or FIPA ACL[72][73]) with a clearly defined semantics, which is useful in information management and integration. Therefore, this thesis will attempt to apply a multi-agent systems approach.

In the development of this framework, emphasis should be placed on its suitability for a wide range of applications in the industrial automation domain, in particular to the process industries and utilities, which have a similar structure. If possible, it would be desirable to be able to transfer the architecture and the basic implementations of its individual components from one situation to another without alteration.

1.5 Thesis Outline

This chapter has introduced current industrial and power system automation systems and different techniques used in the construction of such systems, as well as multi-agent systems and mobile agents.

Chapter 2 gives an overall view of the proposed architecture. Firstly, the tasks that must be performed by an industrial automation system are considered. These tasks are then mapped into a multi-agent system, in which each task is performed by an agent or a number of agents, also taking into account the physical structure of the power system. The collaboration between agents to perform the various tasks is then discussed. The representation of knowledge for agent communication is described and a basic ontology for automation systems is defined. Finally, the implementation of the agent platform, upon which the multi-agent system is constructed, is considered.

Chapter 3 provides a more detailed examination of the static agents that make up the architecture. The basic agent architecture used by all agents is presented, and the sensors, effectors and knowledge of each agent are then considered individually. Implementation issues relating to particular agents, for example the plant agents which control and monitor items of substation plant, are discussed in detail.

Chapter 4 examines the use of mobile agents and mobile code. Previous research into mobile agent performance is discussed, and used to define a performance model for the mobile agent applications described. Experiments in remote control and data analysis are presented, including agent algorithms and detailed performance results. Other proposed applications such as remote monitoring are also considered.

Chapter 5 describes a substation information management system implemented using the proposed architecture. This system is used to provide online monitoring, historical data querying and analysis and remote control for a substation simulator provided by the National Grid Company. The particular agents and ontology used in the implementation of the system are described and detailed examples of the system's use are given.

Chapter 6 provides an evaluation of the whole architecture, using the criteria of functionality, modifiability, performance and reliability. The architecture is evaluated with reference to other industrial automation systems, criteria for substation automation systems and other possible software architectures. Variations on the proposed architecture are also considered with respect to their expected performance in data acquisition.

Chapter 7 presents the conclusions of this thesis and a summary of the contributions made.

Appendices provide detailed results tables for the mobile agent experiments, describe some of the details of the implementation of the prototype system, and give definitions of the data file formats used.

1.6 Contributions to Knowledge

The main contribution of this thesis is to examine the use of multi-agent, mobile agent and other computing technologies in the context of power system automation. The main aims are to determine the applicability of existing computer and information systems techniques to this domain, and to examine how different methodologies such as multi-agent systems and mobile agents can be combined into an integrated automation system. To this end, the following contributions have been made:

- A multi-agent architecture for power system information management, monitoring and control has been developed. By using agents to represent components of the automation system, it is possible for the architecture to more closely match the distributed nature of the system, and the flexibility of the system is increased by allowing components to be added and removed at runtime.

- As part of the development of the information management architecture, the representation of various aspects of power system knowledge in a form suitable for use in multi-agent system communications has been examined, and various examples of this have been provided.

- A prototype implementation has been produced which demonstrates the feasibility of the architecture. The prototype consists of agents which perform the data acquisition, information management and remote control functions of an industrial automation system, for a single substation. Mobile agents are used for data analysis and report generation. The prototype also provides an extensible platform for further research into intelligent applications in the power system.

- The use of mobile agents for a number of applications in power systems, as part of the overall architecture, is discussed and evaluated. Experimental results are provided which suggest that for the applications of data analysis / report generation and remote control, it is possible for mobile agents to provide significantly increased performance compared to static agents or client-server systems. This is particularly true when the user is connected to the substation by a network of high latency or, in the data analysis case, low bandwidth.

- The functionality, performance and flexibility of the architecture have been evaluated, providing insight into the utility of the multi-agent approach to the design of industrial automation systems, and the design decisions involved in developing a multi-agent based automation system architecture. Particular attention has been paid to the architecture of the data acquisition system due to its performance requirements, and various possible multi-agent configurations for data acquisition have been evaluated.

Chapter 2

An Agent-Based Architecture for Power System Automation

This chapter describes in detail a proposed multi-agent software architecture for power system automation. The overall structure of the system consists of WAN (Wide area network) and LAN (local area network) components, as shown in Figure 2.1. The LAN component represents those components of the architecture that would be installed at a substation, while the WAN component represents those components of the architecture found at other locations, such as an office or on a client's computer. There are multiple LAN components, one for each substation, and there may also be multiple WAN components.

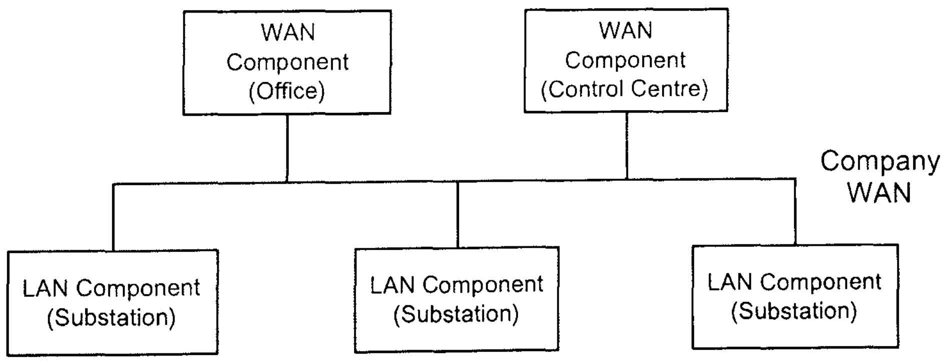

WAN Component (Office)
WAN Component (Control Centre)
Company WAN
LAN Component (Substation)
LAN Component (Substation)
LAN Component (Substation)

Figure 2.1: Outline View of Architecture

2.1 Tasks Performed

As discussed in the Introduction, a power system automation system must perform a variety of tasks, which operate at different timescales, have different characteristics and involve the transfer of different types of data. Figure 2.2 gives a set of common tasks which must be performed by an automation system, including the flow of data between the different processes of the system. This is based on the models of industrial automation discussed in Chapter 1.1, on the requirements for the SICAP substation automation system laid out in [29], and on experience with existing data acquisition and SCADA systems. The "system boundary" divides those processes which are implemented by the system from the external interactors (databases, users and plant). The processes shown are defined as follows:

User interaction involves taking commands and queries from the user, and translating them into an appropriate form for submission to the other components of the system. As shown in Figure 2.3, the task of interacting with the user involves handling queries, requests and online data display. The task of online data display is driven by events from the control system, while queries and requests originate with the user.

Intervention provides the ability for a user to send commands which change the state of the system, for example, to open or close a circuit breaker or set the tap position of a transformer.

Output data interpretation translates from a desired state, expressed in a command, to the appropriate setting (or sequence of settings) of digital and analogue outputs required to achieve this state

Data acquisition takes raw data from sensors and translates it into numerical values. This task corresponds to that of the "sensor process" described by Somerville [23].

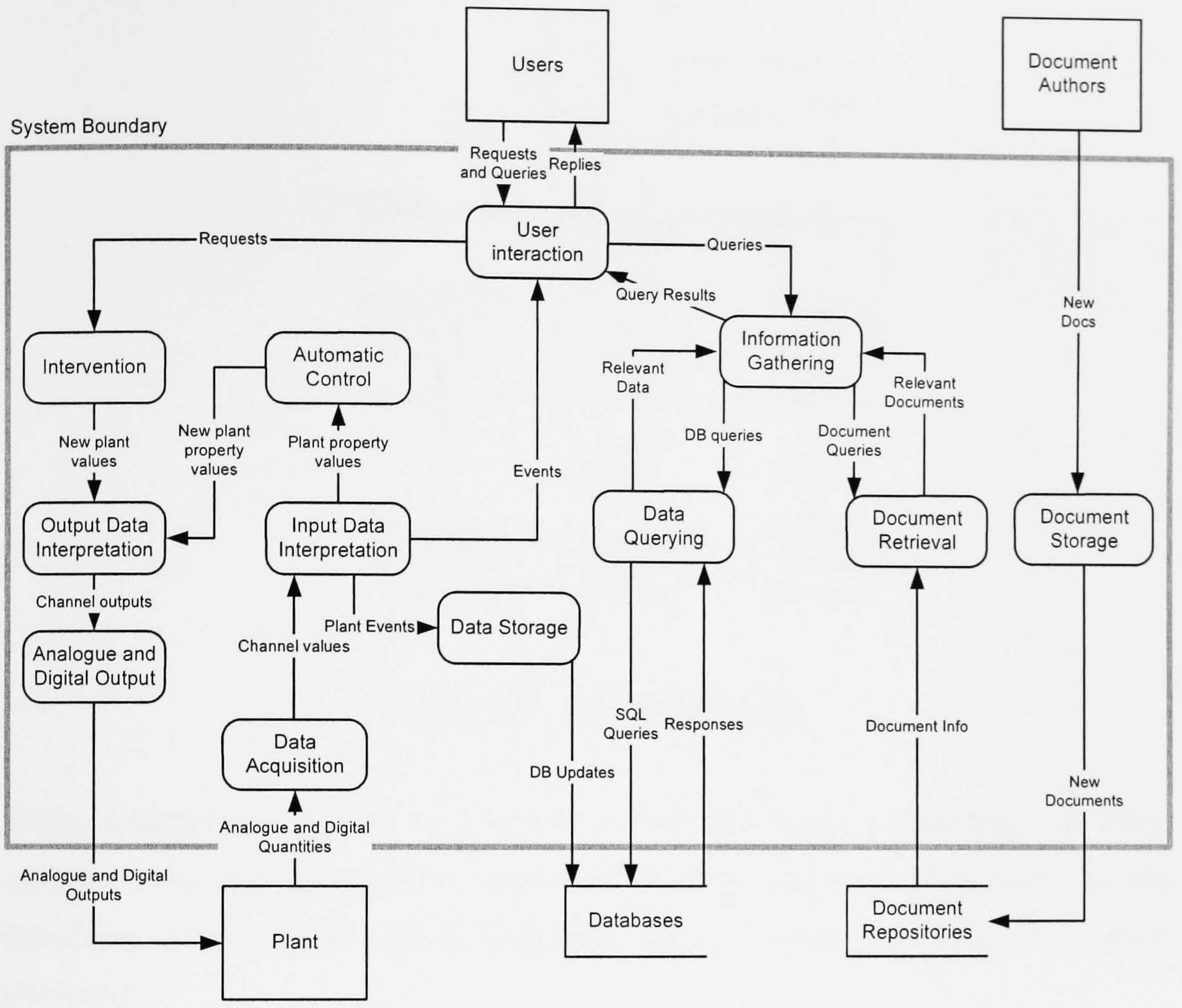


Figure 2.2: Functional Decomposition of Power System Automation System

Input data interpretation takes the data gathered by the data acquisition process and converts it into a machine-understandable representation of the state of the power system. This task corresponds roughly to that of the "process data" process described in [23].

Automatic control involves closed-loop or open-loop control of the system without the participation of a user.

Data storage takes event and other data from the plant (via the input data interpretation process) and stores it as historical data in a database.

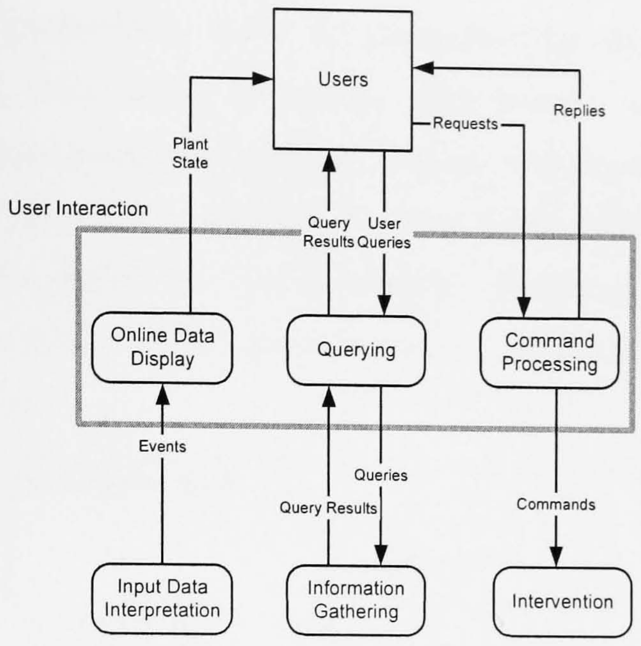


Figure 2.3: User Interaction

Data querying is used to retrieve stored data from a database. It takes queries from the information gathering module and transmits them to the database as Structured Query Language (SQL) queries, returning the results obtained.

Document retrieval provides an information retrieval facility for documents related to the power system. These include maintenance records for substation plant, technical documentation etc.

Document storage is used to allow users (either document authors or system administrators) to add new documents to a document repository.

Information gathering is responsible for taking queries from the user interaction component, or from other components, and retrieving data or documents that are relevant to those queries.

These tasks are now considered in three groups: data acquisition / control, information management and user interaction. For the purpose of this discussion, the output data interpretation, input data interpretation, automatic control and intervention tasks are performed by the data acquisition / control system and information gathering, data storage and document management by the information management system. As shown in Figure 2.4. all tasks, including information management, user interaction and data acquisition/control must be performed in a substation. However, wide area network locations (control centres/offices) perform only information management and user interaction tasks.

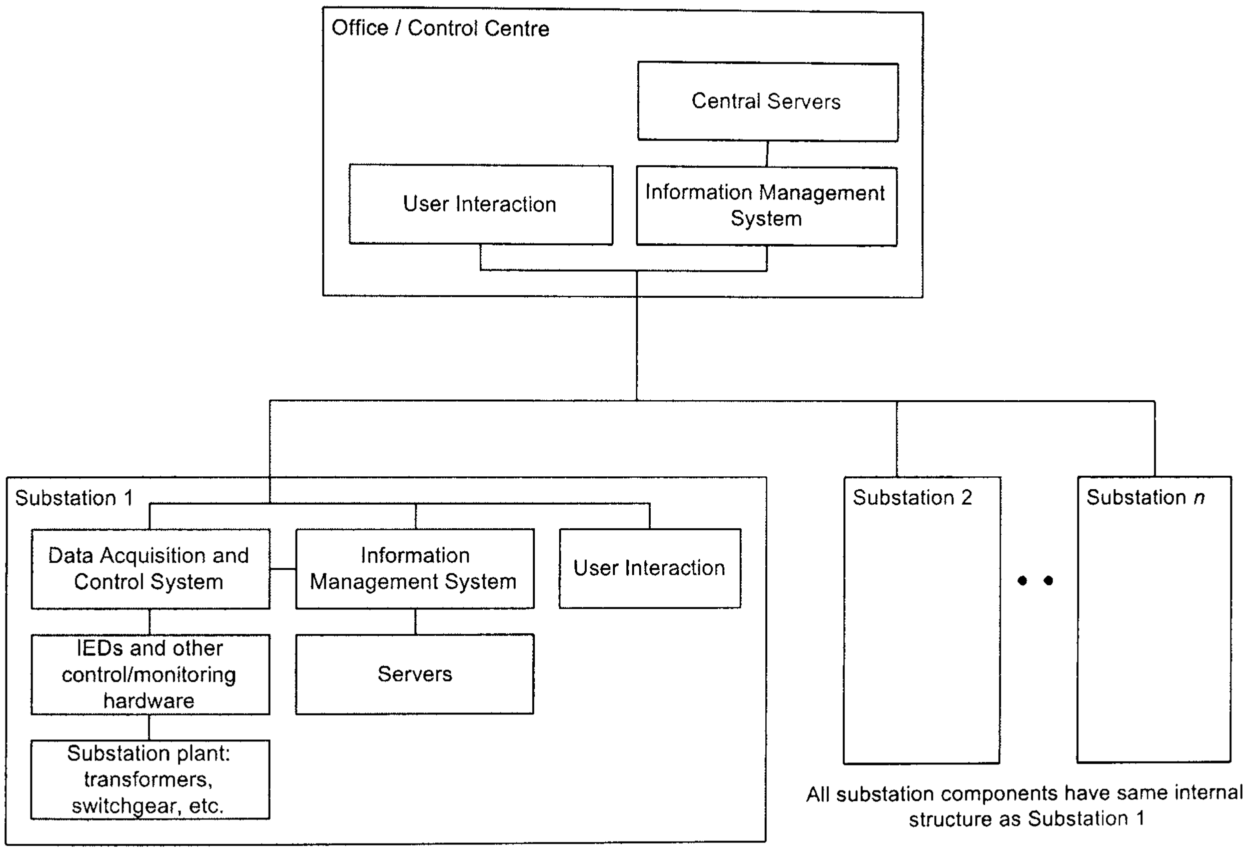

Office / Control Centre
Central Servers
User Interaction
Information Management System
Substation 1
Data Acquisition and Control System
Information Management System
User Interaction
IEDs and other control/monitoring hardware
Servers
Substation plant: transformers, switchgear, etc.
Substation 2
Substation *n*
All substation components have same internal structure as Substation 1

Figure 2.4: Allocation of Systems to Substations and Wide Area Network

2.2 Multi-Agent System

This section describes one possible multi-agent system to fulfil the requirements of the tasks described in the previous section. In designing multi-agent systems, a common approach preferred by several authors in the field [74][75] is to use a *physical decomposition* process, in which each object in the system is represented by an agent. This provides the derived system with a structure which is representative of the physical system being modelled and hence is easy to understand. Also, a physical decomposition may increase the ability of the system to cope with change. For example, if an item of plant is removed from a physically decomposed system it may only be necessary to alter the agent associated with this item of plant. In a functionally decomposed system, if an item of plant is removed, all the functional agents which make use of this item of plant will need to be altered. In the design of the multi-agent system described here, this principle is applied to derive plant agents which represent substation plant, device agents representing monitoring devices and user agents representing users.

However, physical decomposition may not always be the best approach to take. For example, in other work the functional approach is still used in agents such as brokers and mediators, even when the main multi-agent system uses a physical decomposition [75]. In the system described in this thesis, the main application of the functional decomposition approach is to mobile agents. There are separate mobile agents for monitoring, remote control and information gathering. This functional decomposition is useful for mobile agents because it allows relatively small mobile agents to be developed (the agents only have to perform a single specific task), which reduces the amount of data that must be transmitted across the network when a mobile agent is moved.

Another aspect to consider is the level of intelligence and autonomy to be exhibited by the agents. Again, a high level of intelligence might be detrimental to the mobile agents, as the large amount of code required would increase the transmission size of the agent. However, intelligence and autonomy might be beneficial to other agents in the system.

This section discusses the design of the system by considering its two main components: data acquisition/control and information management. However, these components are not completely separate. Some agents perform more than one role and are present in multiple components. For example, the plant agents both control the plant and act as a source of plant information to the information management system.

2.2.1 Agent Platform

An agent platform provides a basis for the implementation of a multi-agent system, and the means to manage agent execution and message passing. It is intended that the architecture should be implemented using an agent platform based on the Foundation for Intelligent Physical Agents (FIPA) specifications [76]. These specifications, being developed by an industry consortium of around 25 organisations, are the most prominent attempt to standardise multi-agent systems technology and are implemented in a number of publically available agent platforms. The specifications define an abstract agent platform, a number of services that must or may be provided by such a platform and a standard communications language. It is assumed that agents have access to the FIPA-specified directory facilitator (DF) service for publishing their capabilities and locating other agents capable of providing a specific service. The DF uses "service descriptions", defined in [76], to allow agents to identify the services provided by each other. In the system described here, an agent might need to use the directory to find out, for example:

- Whether an agent is capable of providing the answer to a particular query. For example, suppose an agent knows that it can obtain the low voltage current of a transformer SGT1 by querying channel 0 of device 0 on node IED1. The agent may then use the directory facilitator to obtain the identity of any agent that may provide it with the ability to query this channel.
- Whether an agent is capable of performing a particular action.

Use of the directory facilitator permits agents to be added and removed at runtime, as an agent providing a service may be substituted with another agent providing the same service. A directory entry for an agent includes (among other items) the agent name, service name, service type, ontologies (data models) used, protocols used and a set of properties, which may be defined by the user, describing that service. Similarly to the mechanism used in [77] to register agents with brokers, in this system agents register the actual queries that they may answer with the directory facilitator. The service type "query-service" is used to denote a registration containing such information. For example, in the implemented device agent described in Chapter 5, the agent registers with the DF the actual information that it can provide using the system ontology, for example, in the FIPA Semantic Language (SL) [78] language, the following might state that the device agent can provide the value of a channel "tp1":

```
(any ?a (value tp1 ?a))
```

Another agent may then use a unification procedure (e.g. that found in Prolog) to match the information provided by information sources with a specified query[1]. This is only a basic mechanism, and in a complex system additional information (e.g. preconditions for a query to be answered) should be provided to enable agents to choose between two or more agents providing the same or similar services. In the system described here, a similar method is used for agents to register actions that they are capable of carrying out. The service type used for this is "request-service". All agents providing a service will register either a query service or request service. Agents may additionally register more specific services, for example, the ontology agent registers a "fipa-oa" service, which is a service defined in the FIPA standards for ontology agents[79].

Inter-Agent Communications

The use of the FIPA platform also provides a standard agent communication language, FIPA ACL [72]. FIPA ACL is a high-level agent communication language based on speech acts. An ACL message consists of an outer message

[1]In most cases, it should be possible to ignore the *any, all* or *iota* part of the expression and match only on the inner expression. However, in order for the expression above to be a legal term in FIPA SL, there must be no free variables, and so *any, all* or *iota* must be included.

structure, providing information such as the sender and receiver of the message, and a message content, expressed in some language understandable to both the sending and receiving agents [80]. Each message has a *performative,* or *communicative act,* which determines its type and hence the effect that it is intended to have on the receiver. The FIPA ACL specifications define a message structure [72], standard communicative acts [81] and interaction protocols. Wherever possible, the standard interaction protocols are used in this architecture. In addition, the language used for all inter-agent communications is the FIPA SL language [78].

2.2.2 Data Acquisition and Control System

An object model showing the components of a typical data acquisition system is given in Figure 2.5. The system consists of a number of data acquisition *nodes*, for example, PCs equipped with input or output hardware or standalone Intelligent Electronic Devices (IEDs). If connected to a network, which we assume is the case for the development of this automation system, the node will have one or more network interfaces, represented by the *interface* class. This has various subclasses representing the different types of network available, including the *IPBasedInterface* shown on the diagram that represents an address on an Internet Protocol network, having a protocol name, host name or IP address and port number.

Each node is equipped with a number of *devices,* which represent a physical or virtual unit of I/O capability, for example, a PCI data acquisition card or a (non-modular) IED. It is possible that a node contains only one device - for example, a computer may have only one data acquisition card. However, nodes with 0 devices are not of interest to the data acquisition system, as no input or output facilities are provided. A device may contain a number of *channels*, each of which is capable of the input (*InputChannel*), output (*OutputChannel*) or both (*InputOutputChannel*) of a single analogue or digital value. Each channel has a data type, which defines the type of data input or output, for example, *Boolean* or *float.*

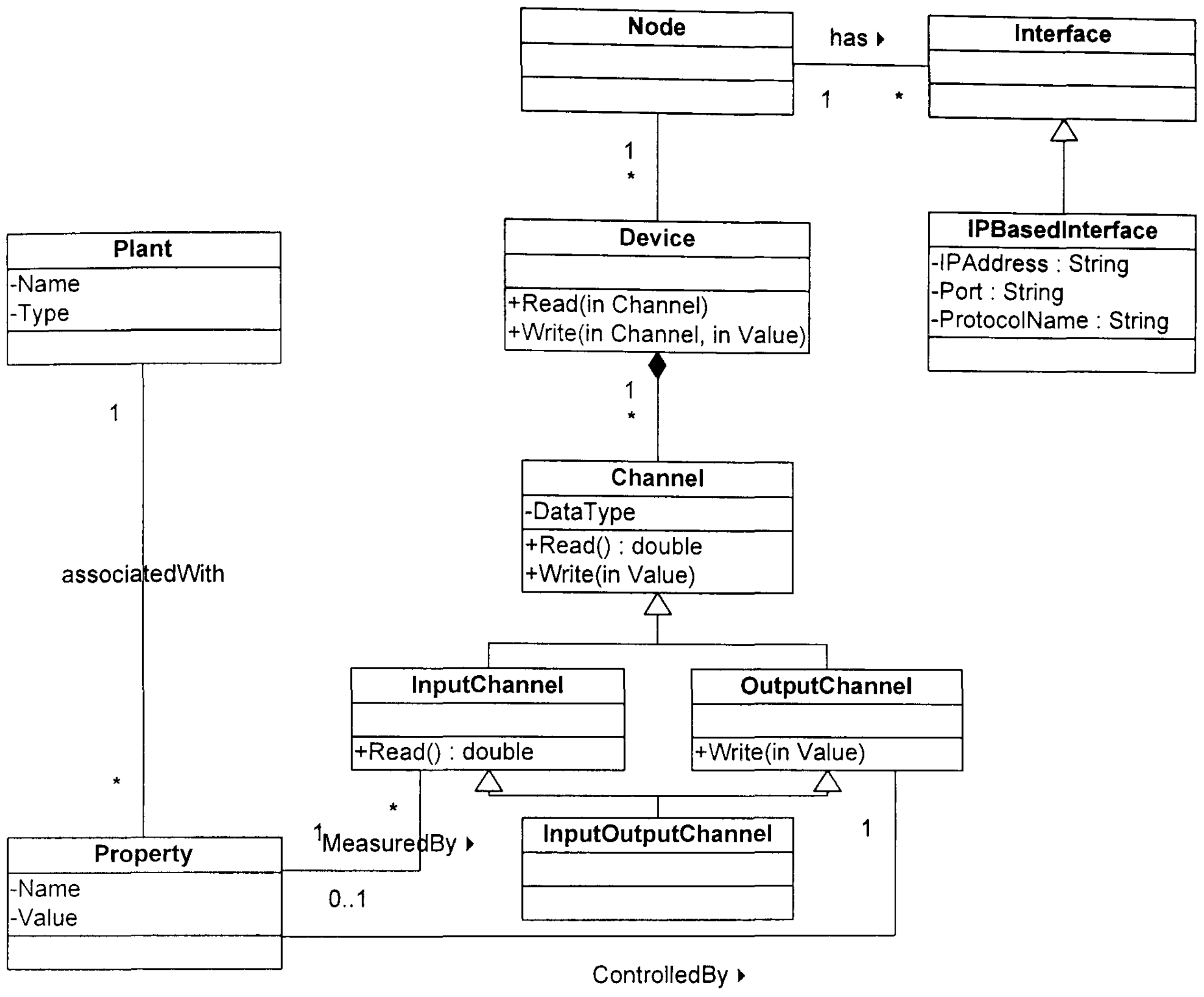


Figure 2.5: Generic Object Model for Data Acquisition System (using UML class diagram notation)

Each input channel measures a particular *property* of an item of plant via sensors and actuators (not shown on the diagram as they are external to the data acquisition system) . For example, an analogue input device might use a thermocouple sensor to measure the oil input temperature of a transformer. It might be possible that a property is measured by more than one channel. However, because a channel is capable only of the input or output of a single quantity it is not possible for it to measure more than one property. If a property is directly controllable (e.g. the status of a circuit breaker), then it is *controlled by* one or more output channels. The simplest method of control is that in which writing a value to a channel directly sets the value of the plant property, for example, writing a 0 to a digital output to close a relay. However, it is also possible that properties may be controllable, but are set indirectly via actuators. For example, writing a value to the "tap up" channel of a transformer would cause the tap position of that transformer to be increased by 1, but it is not possible to directly write a tap position value to the "tap position" channel.

Agents

From the model shown in Figure 2.5, and using the principle from [82] that agents should correspond to things within the problem domain rather than "abstract functions", it is possible to derive several possible agent communities, depending on the desired granularity of the decomposition. For example, considering the data acquisition system, which consists of nodes, channels and devices, should each channel be represented by a separate agent, or should a single agent represent all channels on a device, or even all channels on a node? Van Dyke Parunak [82] states that an agent should be "small in mass" (representing a small portion of the entire system), "small in scope" (having limited sensors and action capabilities) and "small in time" (able to remove or "forget" outdated information in its memory) - in other words, that in general agents should be kept small. This would suggest the use of separate agents to represent nodes, channels and devices. However, this may not always be simple to implement in practice, and would result in a very large number of agents, which might prove difficult to manage. Therefore, the proposed architecture specifies the use of one agent per device. However, because agents use Directory Facilitator (DF) entries to locate other agents capable of performing an action or responding to a query, it should be possible to substitute a channel agent for part of the functionality of a device agent without disturbing the rest of the system. All DF entries relating to the relevant channel would be removed from the description of the device agent, and put into the description of the channel agent. Therefore, "client" agents looking for, for example, the value of the channel would determine from the DF that the agent now providing that service was the channel agent rather than the device agent. An alternative architecture would be to have a hierarchical structure consisting of one device agent and multiple channel agents, with the device agent providing the external interface to the system. Other agents would not see the channel agents, and would pass requests to the device agent, which could then forward them to the appropriate channel agent.

Considering the plant and properties of plant, it follows from the principle of agents representing entities that each item of plant should be represented by a single agent. A property of an item of plant is not actually an entity, although it is shown on the object model as such to allow the link to be made between channels and properties. Each plant agent will be aware of all of the properties of its respective item of plant and, where appropriate, able to control them.

Figure 2.6 shows the derived multi-agent data acquisition and control system. The device agents are responsible for performing data acquisition and output on a single device. Plant agents obtain data from the device agents, and are responsible for converting that data into a representation of the current state of the relevant item of plant. They are also responsible for automatic control tasks relating to that item of plant, and for providing information to the information management system. Cooperative, distributed control schemes may be implemented by communications between the plant agents. For a typical substation, there will be plant agents representing transformers, circuit breakers, disconnectors and any other items of plant.

For most control and information management purposes, only the plant agents need be visible to the rest of the system, as users are normally interested in the functioning of the substation, rather than the details of the data acquisition system. However, as well as allowing the agents to be as "small" as possible, the use of intermediate data acquisition system agents between the plant and the plant agents, rather than solely using the plant agents, is necessary because there may not be a 1:1 mapping between plant and devices - a device may control or monitor multiple items of plant, and an item of plant may be monitored or controlled by a large number of devices. The device agents

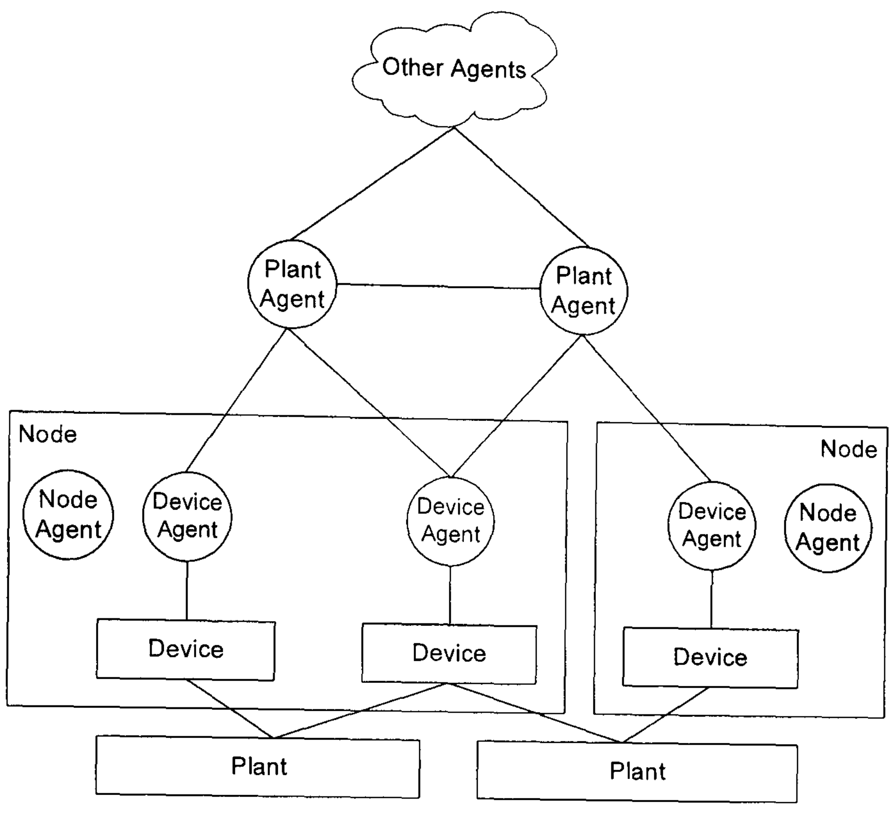


Figure 2.6: Multi-Agent System for Data Acquisition and Control

hide the different implementation details of these multiple data acquisition devices from the plant agents, making it possible to re-use the same plant agent implementation (with configuration changes) for different items of plant of the same type, even where the systems monitoring these items of plant are different. This architecture also simplifies the task of changing a data acquisition device or adding a new one. However, it is also conceivable that similar information hiding could be accomplished by the use of a library of drivers, each for a specific device but sharing a common interface, in place of device agents. Another advantage of the multi-layered structure is that functions such as the configuration of the data acquisition system (carried out by device agents) are separated from the control functions (carried out by plant agents). This should make the implementation of the individual agents simpler, as they have fewer tasks to carry out.

Based on the principle of "small agents" discussed in [82], it is possible to conceive that the agents representing complex plant items such as transformers, which have many components, should themselves be split into several "sub-agents". For example, a transformer agent might be responsible only for monitoring the windings of the transformer (voltage, current etc) and would be associated with a cooling system agent, monitoring the cooling fans and oil temperatures, and a tap changer agent, responsible for the operation of the tap changer. It is difficult to evaluate whether or not this would be useful in practice, and for reasons of time it has not been implemented in the current prototype of the architecture. However, it is an issue for further research.

2.2.3 Information Management System and User Interface

The architecture of the multi-agent information management and user interaction system extends previous agent-oriented information system architectures such as RETSINA [59], which is described in Section 1.3. This 3-tier architecture consists of user agents, wrapper agents representing information sources and "middle agents" which transfer data between the two. Wrapper agents allow heterogeneous data sources to be queried using the agent communication language (which in this system is FIPA ACL), and convert data from the data models (ontologies) used by the individual data sources into a global ontology used for querying.

The architecture for substation information management, shown in Figure 2.7, contains 3 basic types of database: data logging databases, which are used for storing status and event information, "static" databases which hold configuration data, and the ontology database, which holds the system ontology (data model). Each database has its own database agent, with the ontology database having the ontology agent. Additional data is provided by plant agents themselves and by substation document repositories. The plant agents do not require wrappers, as they are already agents. The other sources of information are represented in the system by wrapper agents: the databases by database agents and the document repositories by document management agents.

A mobile server is a server which is added to the system for a period of time (perhaps to carry out data acquisition for an experiment) and is then removed. Data stored on a mobile server may be accessed via the multi-agent system in the same way as data stored on any other server, providing that appropriate agents are available on the mobile server. The type of agent to be used would be determined by whether the mobile server contained a database (in which case a database agent would be used) or a data acquisition system (in which case device agents and a node agent would be provided). In the second case, the mobile server should be integrated into the data acquisition / control system, and pass data to a plant agent, rather than into the information management system.

The information transport (middle) layer of the system is composed of brokers, task-oriented agents and mobile agents. Task-oriented agents use the services and information provided by the service agents (database agents, document agents and plant agents) to provide a specific service to user agents. For example, an alarm and event agent might use monitoring data provided by the device agents to generate alarms and events for viewing by users. These agents provide a convenient encapsulation of particular forms of data. It is also possible to conceive of a variety of different task-oriented agents to perform various functions such as modelling, prediction or decision support. These agents are task-oriented rather than physically oriented. The use of task-oriented agents permits a variety of services to be implemented by introducing new agents rather than by modifying a large portion of the existing system.

The user interface layer consists of user agents, with each agent representing a particular user of the system, and a human-machine interface. The user agents perform data transformation between the multi-agent system and the human-machine interface, similar to the function played by a wrapper agent for a data source.

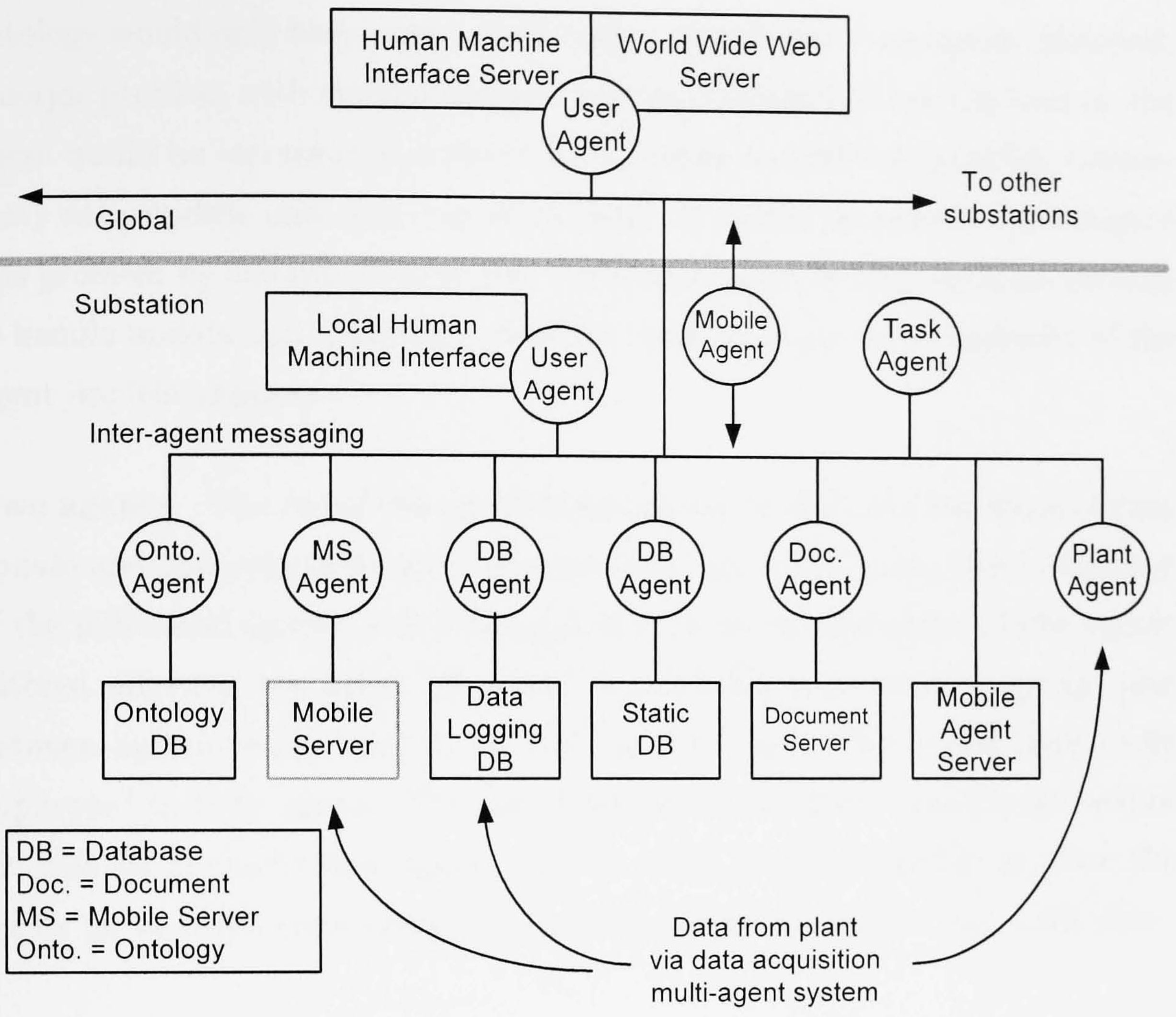


Figure 2.7: Information Management Multi-Agent System

Database Agents: Input and Output Agents or Input/Output Agent?

The database agent as described above must perform two tasks: data storage and data querying. However, it would also be possible to use separate storage and querying agents. The advantages and disadvantages of each configuration are now considered.

One agent The use of one database agent provides the ability to view the database, which is a single software system, as a single agent, and is therefore conceptually attractive. Also, the mapping rules (discussed in Section 3.1) used by a database agent to translate data from its database schema into the global ontology would only have to be stored within a single database agent. However, a major problem with the single-agent approach would be that the load on the agent would be increased as a result of its having to perform (possibly concurrent) data update and querying operations. It might be possible to mitigate this problem by the use of threading (with each agent having separate threads to handle update and querying). However, this increases the complexity of the agent implementation.

Two agents The use of two separate agents would allow the functions of data update and querying to be split between these agents, reducing the complexity of the individual agents, and making it simpler to modify either of the agents without affecting the other. However, a mechanism to share mapping rules between agents would have to be used, or else these rules would have to be duplicated in both agents. The use of two separate agents should also reduce the load on the individual agents, and it would also be possible to place the agents on separate computers, further improving performance and scalability.

2.2.4 Combined Multi-Agent Architecture

By combining the components of the multi-agent system obtained in the preceding sections, an integrated architecture, the outline of which is shown in Figure 2.8 is obtained[2]. In the information management system, all agents may communicate with each other. However, the task agents may be situated in the substations, and in this case would not need to use mobile agents to retrieve data.

The interface between the data acquisition / control system and the information management system occurs at two points: the plant agents (for online monitoring and operator intervention) and the data logging database (for data querying). Mobile servers are not shown on the combined architecture, as it is envisaged that they would contain either a database and database agent, or a

[2]In order to make the diagram readable, the architecture has been simplified in comparison to the diagrams of its individual components.

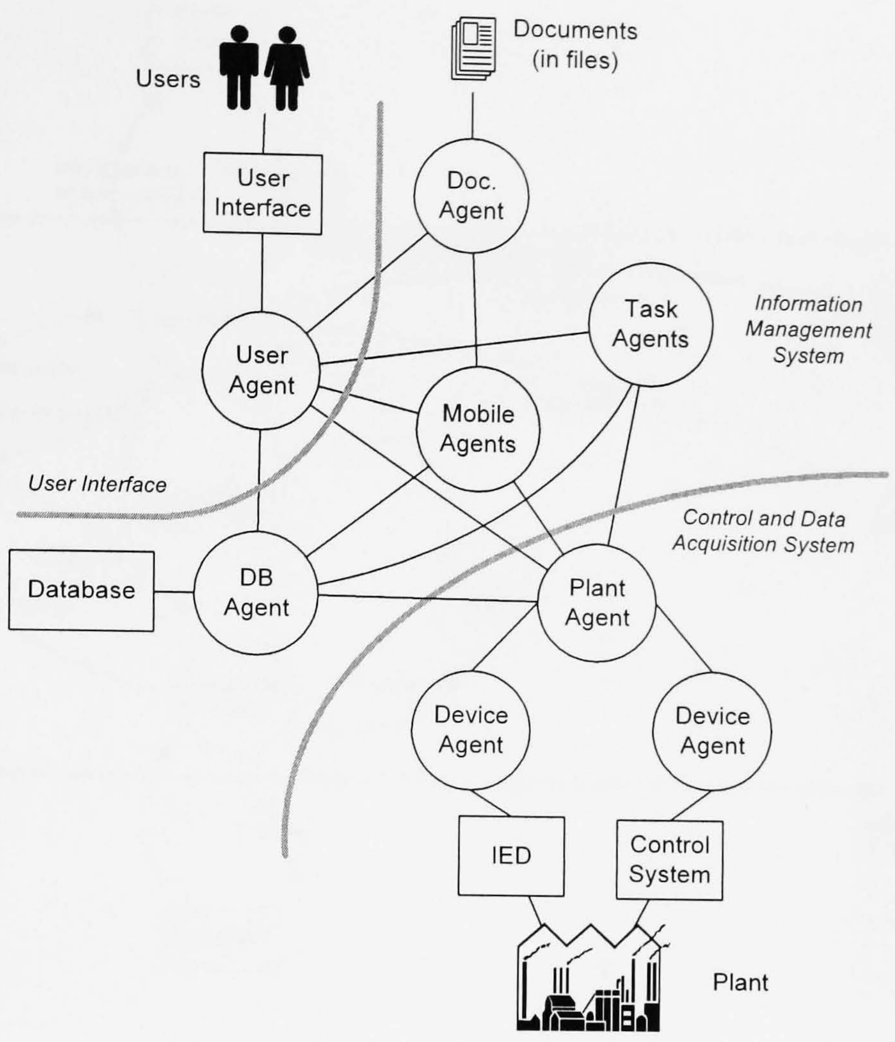


Figure 2.8: Combined Multi-Agent System

data acquisition system and plant / device agents.

Agent Collaboration

Figure 2.9 shows how the different types of agent in the system collaborate. The multi-agent system implements the generic system shown in the data flow diagram (Figure 2.2) and therefore the external interactions of the system (those which cross the system boundary) are the same.

Multiple Substations (Modular Architecture)

The descriptions of the architecture in this chapter have so far considered a single substation, along with its connections to the wide area network. How-

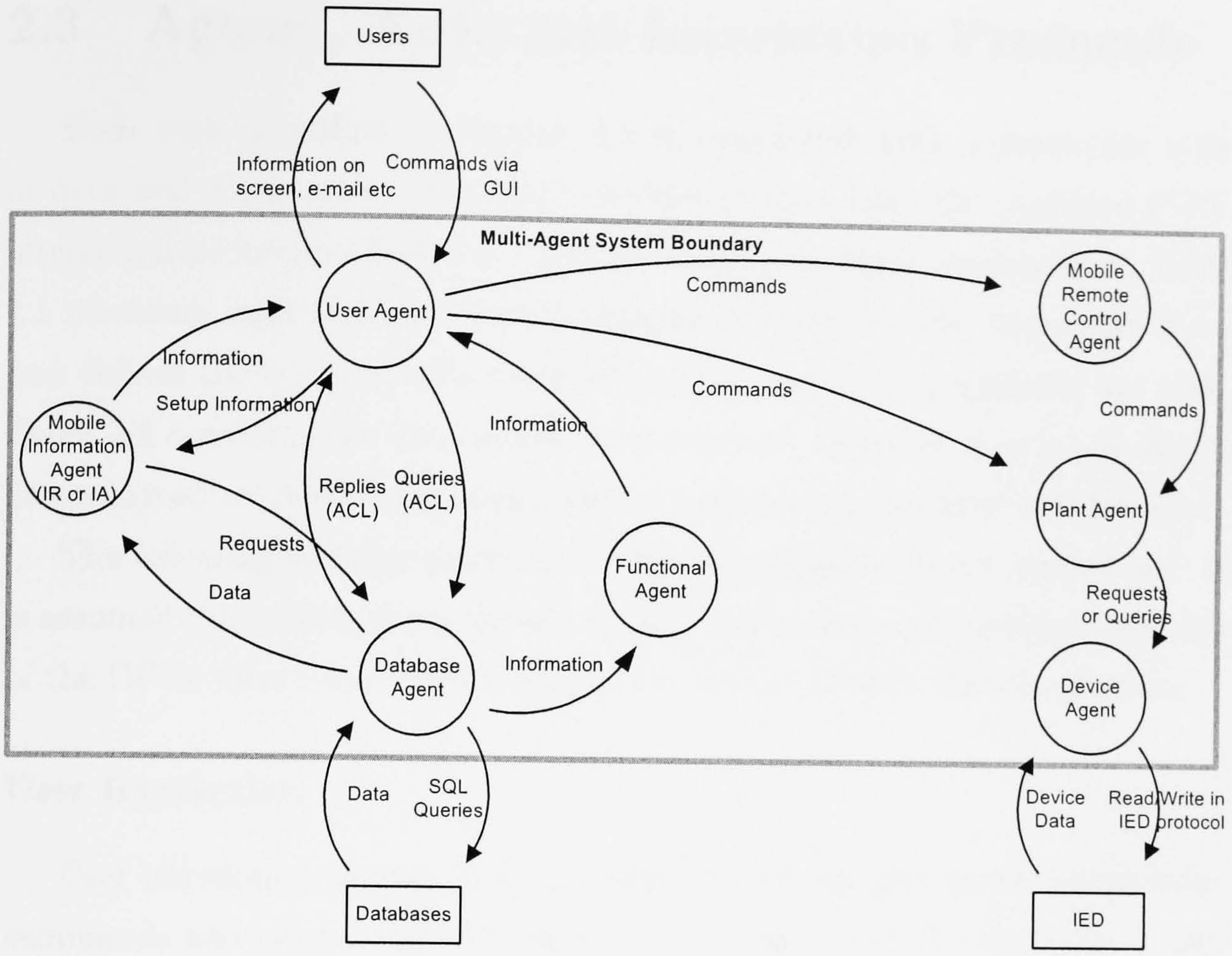


Figure 2.9: Agent Collaboration

ever, a real power transmission system contains a large number of substations.

It is envisaged that in the architecture described each substation would contain a multi-agent system consisting of a data acquisition system and information management system. This would produce a modular architecture consisting of individual substation modules and WAN modules. While possible, communications between substations would be limited, as many functions do not require direct inter-substation communications. Each substation should have its own agent platform and directory services, reducing the amount of data transmission across the wide area network. However, it would be necessary to allow user agents, and other agents located on the wide area network, to have access to any / all substations.

2.3 Agents, Tasks and Interaction Protocols

Each task described in Section 2.1 is associated with a particular type of data and a particular interaction protocol, taken from the standard FIPA interaction protocols. Tables 2.1 and 2.2 summarize these associations. Table 2.1 considers each task as a transformation from input data to output data, and defines the types of data involved and the agents that perform the task. Table 2.2 considers the characteristics of the task (whether it is event-driven or performed on demand by some user or system) and the protocols involved.

The following sections describe how the individual tasks are carried out. It is assumed that agents have already located each other and therefore searches of the DF to locate appropriate agents are not included in the descriptions.

User Interaction

User interaction (Figure 2.10) is carried out by the user agent, which takes commands and queries from the user (via the HMI) and translates them into appropriate ACL messages for transmission to other agents.

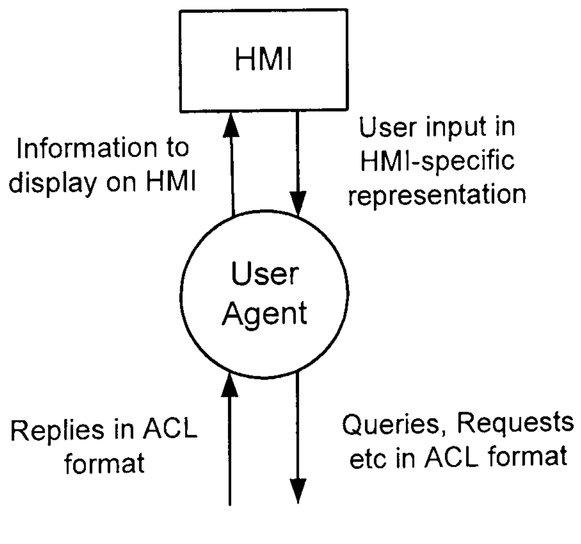

HMI
Information to display on HMI
User input in HMI-specific representation
User Agent
Replies in ACL format
Queries, Requests etc in ACL format

Figure 2.10: User Interaction

Task	Input Data	Output Data	Agents
User Interaction			User Agent
- Queries	HMI input	ACL queries	
- Requests	HMI input	ACL queries	
- Online Display	Events or Status	HMI display	
Intervention	Requests	Requests	Plant Agent
Output D.I.	Required plant status	DAQ actions	Plant Agent
Automatic Control	Current plant status	Required plant status	Plant Agent
Input D.I.	DAQ Events/ Status	Plant Events/ Status	Plant Agent
Data Acquisition	Sensor data	Events / Status (DAQ system)	Device Agent
Data Storage	Data in ACL format and global ontology	Data in SQL statements and DB schema	DB Agent or DS Agent
Data Querying	Queries (ACL)	Responses (ACL)	DB Agent
Document Retrieval	Queries (ACL)	Binary Data	Document Agent
Document Storage	Binary Data in ACL	Binary Data in files	Document or Document Storage Agent
Information Gathering	ACL queries	Information in ACL	Mobile Agents
Analogue / Digital Output	New values (ACL)	DAQ Channel Values	Device Agent

Table 2.1: Tasks, Data Types and Agents (DB = Database, DS = Data Storage)

Intervention (Including Output Data Interpretation)

Intervention (Figure 2.11) is performed by the user agent and plant agent. Upon receiving a command from the user via the graphical interface, the user agent searches the DF to locate a plant agent capable of carrying out that command. It then forwards the command to the appropriate agent as a FIPA ACL *request* message. The plant agent generates a sequence of read or write operations to carry out in order to fulfil the request, and transmits these to

Task	Characteristics	Protocols
User Interaction		
-Queries	On-demand	FIPA Query
-Requests	Request/Reply	FIPA Request
-Online Display	Event-driven	FIPA Subscribe
Intervention	Request/Reply	FIPA Request
Output D.I.	On request	FIPA Request
Automatic Control	Event-driven	FIPA Subscribe (input) FIPA Request (output)
Input D.I.	Event-driven	FIPA Subscribe
Data Acquisition	Event-driven	Device-dependent(input) FIPA Subscribe (output)
Data Storage	Event-driven	FIPA Subscribe (data gathering)
Data Querying	On-Demand	FIPA Query
Document Retrieval	On-Demand	FIPA Request, FIPA Query
Document Storage	On-Demand	FIPA Request
Information Gathering	On-Demand	FIPA Query, (FIPA Request)
Output (A or D)	On-Demand	FIPA Request

Table 2.2: Tasks and Interaction Protocols

the device agent.

Data Acquisition

Data acquisition (Figure 2.12) is performed by the device agents. The device agent must first establish a connection to its IED or other device, and acquire the values of its channels either by polling or by an event-driven method.

Input Data Interpretation

Input data interpretation (Figure 2.13) is performed by the plant agent, in cooperation with the device agents. The device agent must first register the channels that it is monitoring in the DF. Plant agents may then search the DF to locate device agents monitoring those channels that are connected to their items of plant. Once a plant agent has found the relevant device agent or agents, it then establishes subscriptions with these agents in order to be notified

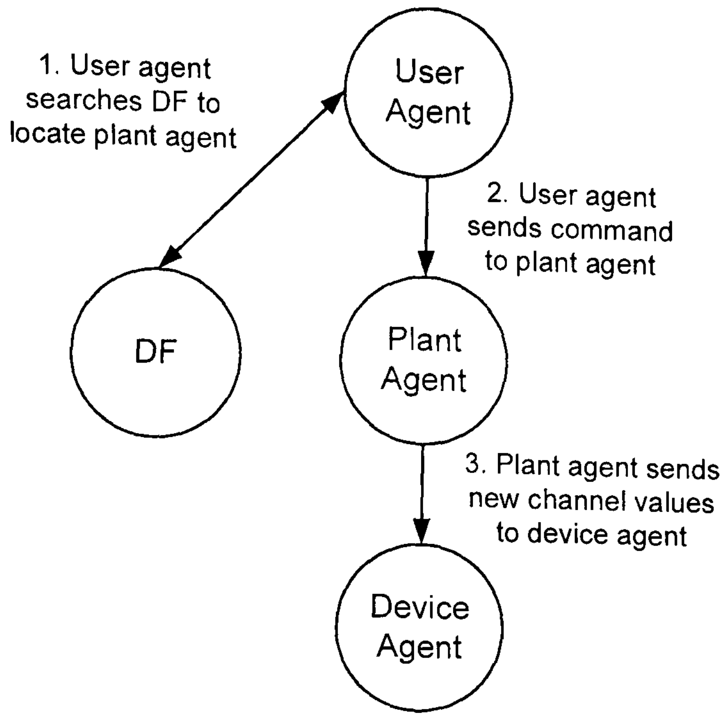

1. User agent searches DF to locate plant agent
User Agent
2. User agent sends command to plant agent
DF
Plant Agent
3. Plant agent sends new channel values to device agent
Device Agent

Figure 2.11: Intervention (including output data interpretation)

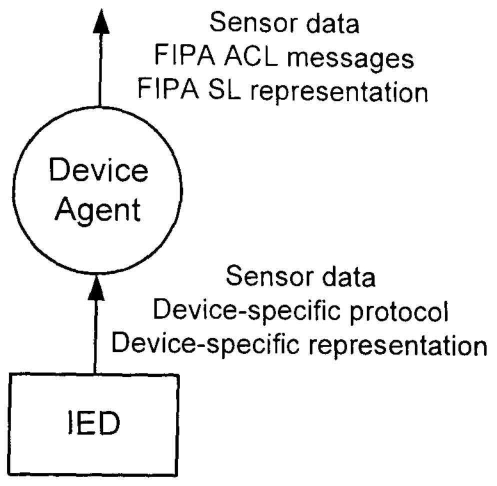


Figure 2.12: Data Acquisition

whenever the value of a relevant channel changes. After these subscriptions are established, the device agent must notify the plant agent whenever the value of a subscribed channel changes. This notification will be a FIPA ACL message with FIPA SL content, giving the name of the channel and its current value. The device agent then uses this information to derive the state of its item of plant. The mechanism used to do this is described in Section 3.5.

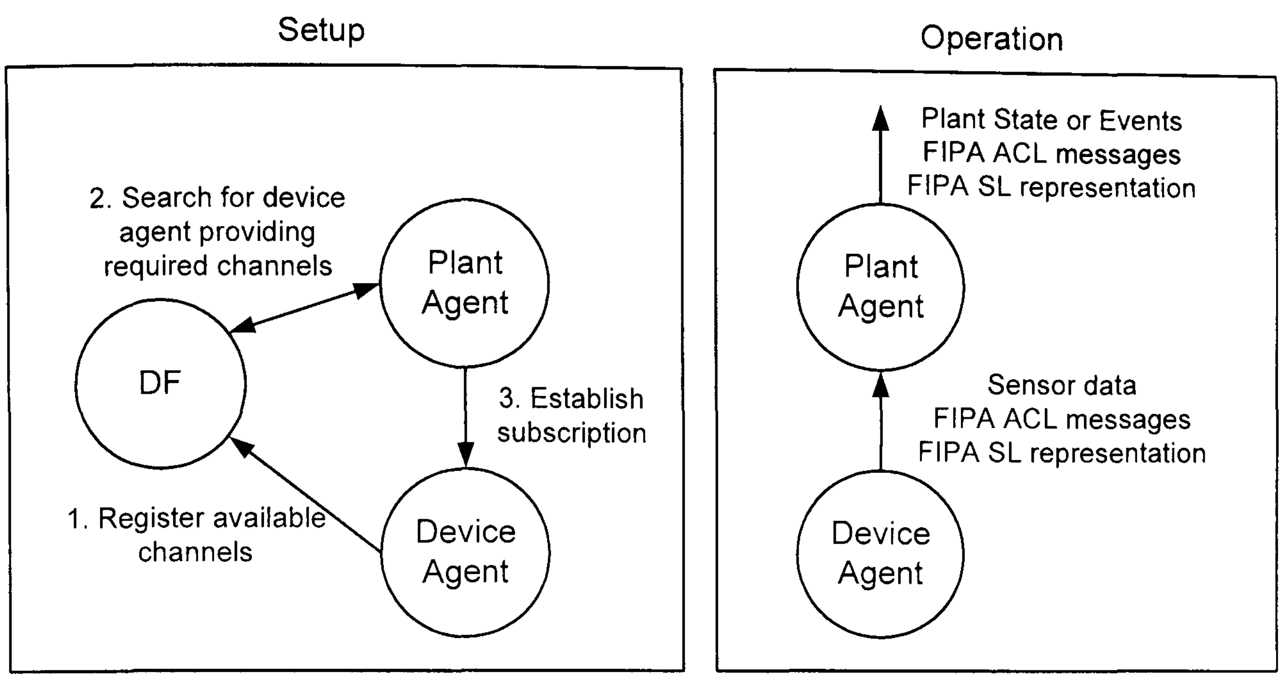


Figure 2.13: Input Data Interpretation

Automatic Control (Including Output Data Interpretation)

Control (Figure 2.14) is performed by the plant agent. New channel values are passed to the plant agent by the device agent. The plant agent then performs input data interpretation on these values to generate the state of the item of plant being controlled. The agent then uses this state and the desired state to generate a set of outputs to be written to the plant, which are sent to the device agent.

Data Storage

Data storage (Figure 2.15) is performed either by the database agent or by a specialized data storage agent (this topic is discussed in Section 2.2.3). The database agent must first establish a subscription with appropriate provider agents in order to be notified of new data. As events arrive, it generates SQL update statements from these events, and enters the new information into the database.

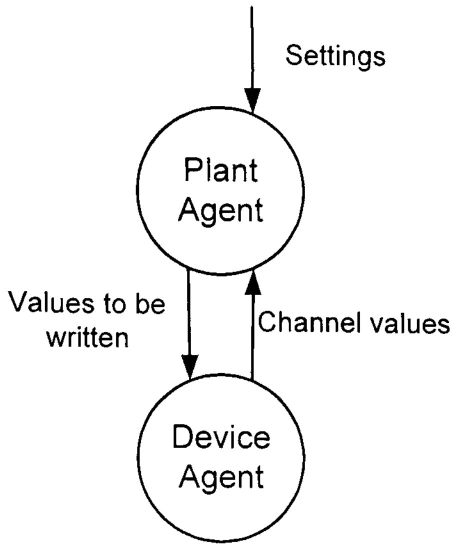


Figure 2.14: Automatic Control

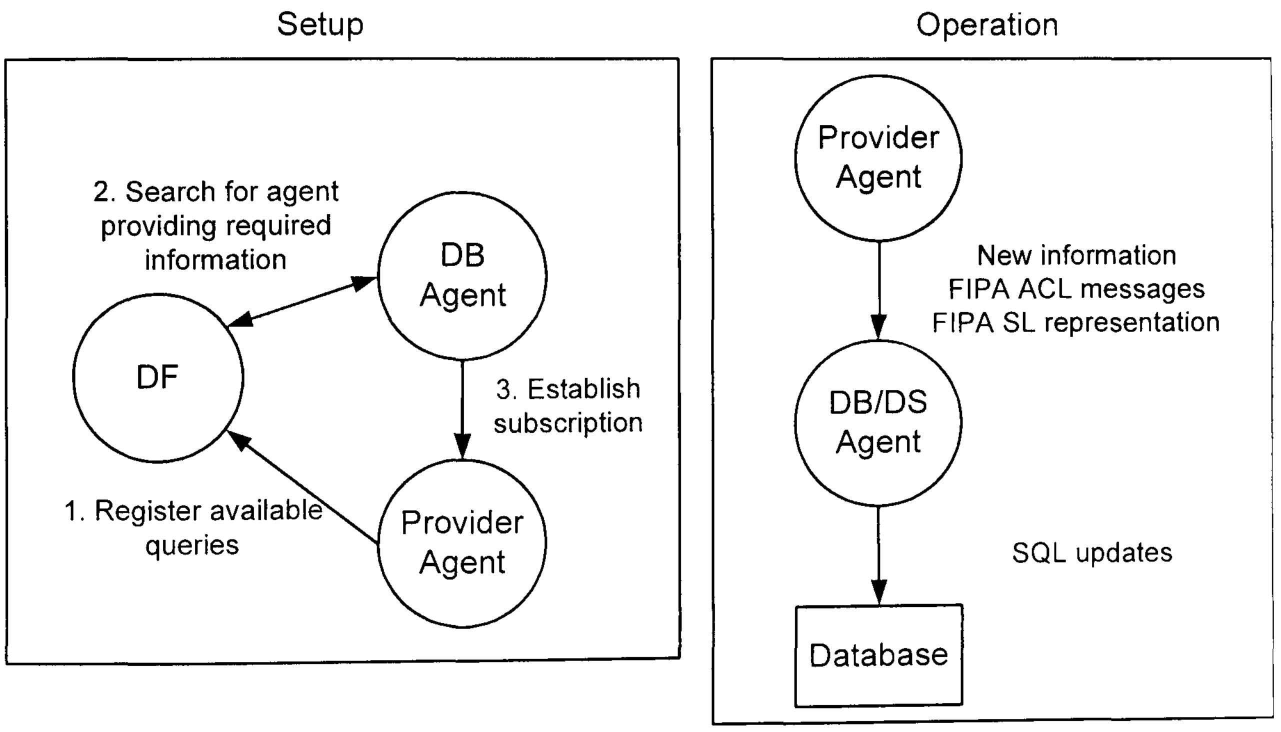


Figure 2.15: Data Storage

Data Querying

Querying (Figure 2.16) is performed by the database agent. A client agent (usually the user interface agent or a mobile agent) generates a query in FIPA ACL / FIPA SL format. The database agent then maps that query into an SQL query which is sent to the database. The results of the query are converted into FIPA ACL / FIPA SL and returned to the client. Integration of data from multiple databases may also be performed at this stage. If a querying agent determines (using the directory facilitator) that multiple agents have information relevant to the query, it may query both of these agents and then merge the results retrieved. Because all database agents convert data into the global ontology, this should be a relatively simple process in most circumstances. A similar procedure may also be used for answering complex queries, for example, to retrieve all transformers exceeding their load rating. In this case, the querying agent might retrieve the list of transformers and load ratings from the static database, and then search the data logging database to determine the maximum load of each one.

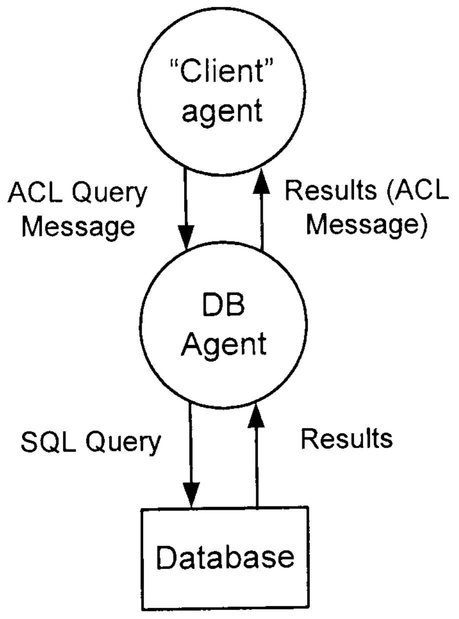


Figure 2.16: Querying

Document Retrieval

The document retrieval process (Figure 2.17) consists of two operations. First, the client agent must determine which documents are relevant to a particular query. Then it may retrieve some or all of these documents.

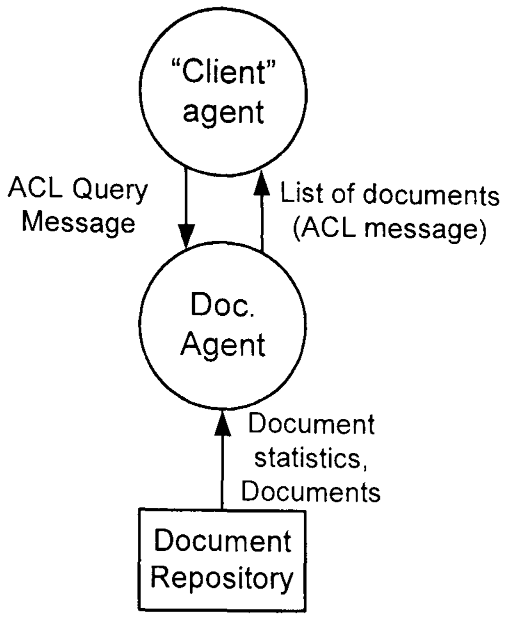

"Client" agent
ACL Query Message
List of documents (ACL message)
Doc. Agent
Document statistics, Documents
Document Repository

Figure 2.17: Document Retrieval

Document Storage

A document may be added to a document repository either directly or via a document agent. To add a document directly, its file is placed in the filesystem of the document repository. To add a document via a document agent, it may be sent to the agent as the content of an ACL message, using Base 64 encoding to convert a binary document into a text-based representation suitable for transmission.

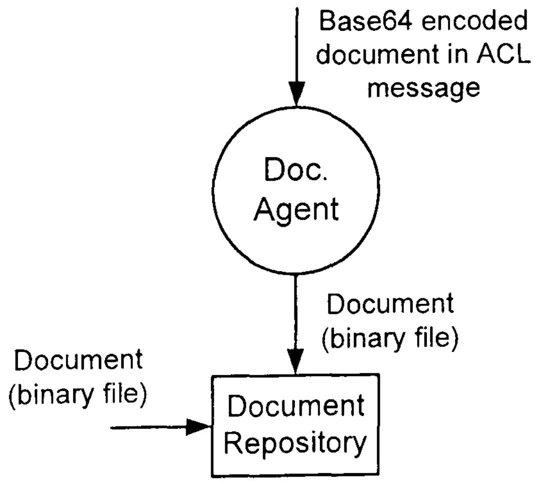


Figure 2.18: Document Storage

Information Gathering

The information gathering process performed by mobile agents is described in Chapter 4.

Analogue/Digital Output

Output to devices is performed by the device agent, which translates between the ACL representation used by other agents and the device-specific representation.

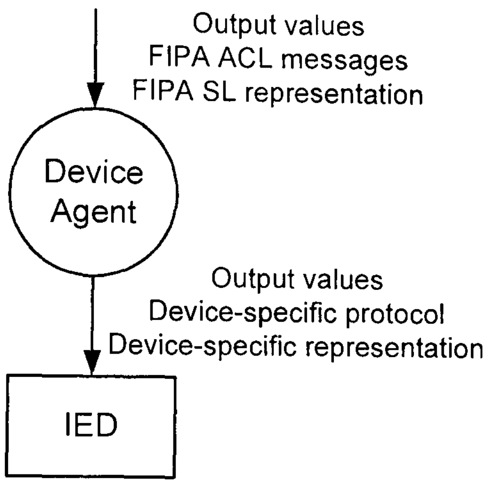


Figure 2.19: Analogue / Digital Output

2.4 Data and Knowledge

In order to permit exchange of knowledge between agents, it is important to have a standard representation of the knowledge available in a substation. The representation chosen in this system is based on first-order logic, and in particular on the form used by FIPA SL [78]. This section first discusses the different types of knowledge available in the power system, and then considers its machine-understandable representation.

2.4.1 Available Data and Knowledge

In a substation automation system, the data available largely consists of status information, "streamed" data, configuration information and other fixed data sources such as documentation.[3]

Status and event information consists of both state information, which gives the current state of some part of the substation plant, and event information, which describes changes in the state of the plant. Often, a data point gives the value of some input, which corresponds to a property of an item of plant. For example, "the low voltage current of transformer 1 is 137.5 kV". Events are usually associated with a time tag.

When real-time information is stored for processing at a later time it is referred to as *historical* data. Historical data includes, for example, logs of events and stored waveforms.

Streamed data If a data point is monitored in real-time by another process, the waveform data will be transferred, a single item at a time, from one process to another. This is referred to as "streaming", and may be used in a variety of real-time monitoring applications.

Configuration information Configuration information consists of information both about the configuration of the substation and about the configuration of the automation system itself. For example, topology information regarding the connections between different items of plant is included in this category.

Non machine-understandable data sources make up a further category of information. Currently, non machine-understandable data sources include documentation and images. This data differs from the real-time and configuration information in that it is not immediately usable by a computer system in a reasoning process. However, it is possible that various processes, for

[3]These categories are partly based on those described by the IEC 61850 standard[24, p41]

example text mining or image recognition, might be able to derive machine-understandable data from non machine-understandable data.

Derived knowledge As well as the "raw data" discussed above, which is considered to be the inputs to the automation system, the system may itself be capable of deriving additional knowledge. For example, National Grid Technical Specification (NGTS) 2.7 states that a substation control system is required to "maintain a running total of operations for each of the primary plant items" [83]. Another example of derived knowledge is the output of a condition monitoring system which uses real-time information (e.g. temperature data) to determine whether or not a transformer is in a good operating condition.

2.4.2 Knowledge Representation

The representation of each of the types of knowledge described above in first-order logic, and hence in the FIPA Semantic Language (SL), is now considered. The notation from Russell and Norvig [84], with mathematical expressions and sets denoted as in [84], and not FIPA SL, is used here. In this notation, object and relation names start with capital letters, and variable names with small letters. Where frames or identifying expressions are used the FIPA SL notation is used instead, as these are not included in the standard notation. FIPA SL is more difficult to read and produces longer expressions than standard logical notation. However, conversion to the FIPA SL notation from the notation used here is relatively simple, as there are only syntactic and not semantic differences. The main difference is that FIPA SL uses an infix notation in which the entire expression is enclosed in parentheses. Also, variable names start with a question mark, and all other names may start with either small or capital letters. For example, "if x is a transformer, then x is an item of plant" could be written in first order logic as follows:

$$\forall x \, Transformer(x) \Rightarrow Plant(x)$$

and represented in FIPA SL as:

```
    (forall ?x (implies (transformer ?x) (plant ?x)))
```

In the above expression, the ∀symbol (for all) means that the statement applies to all possible values of the variable x. The $\Rightarrow$ symbol (implies) is used to state that if the statement on the left ($Transformer(x)$) is true, then the statement on the right ($Plant(x)$) is also true. FIPA SL also includes modal extensions in the form of the operators B (believes), U (uncertain), I(intends) and PG(is a goal of). Each of these operators takes two arguments: an agent and a logical statement. For example:

```
    (B agent1 (status x110 open))
```

means that the agent "agent1" believes the statement "(status x110 open)", which might mean "the circuit breaker X110 has status open". The use of modal logics in reasoning about agents is discussed in [85].

Representation of Status information

If we assume that a single data point of status information gives the value of some property of a plant item (or other item of equipment) at a particular time, it is possible to represent a data point by a binary relationship, or subject-predicate-object triple. For example, suppose that we have the statement "The low voltage current of transformer SGT1 is 12.3 volts". In first-order logic, this could be represented by:

$$LvCurrent(Sgt1, 12.3)$$

Representation of Events

An event must be able to represent the fact that at a given time, some change occurred in the status of the plant, or some action was performed. Due to the requirements of the various processes in the power system, the time represented in such an event must be explicit, rather than defined using operators such as "PAST" or "NEXT" as may be done in temporal logic [86]. We adopt an event representation based on the event calculus representation of time described in [84], Chapter 8. A formal semantics and logic for such a representation is established by Shoham in [87]. An event is represented by the T (or $TRUE$) "predicate", with two arguments: the event itself, and a timestamp, the format of which is discussed below, giving the time at which that event occurred.

For example, the event "at 12:30 pm on 03/02/2003, circuit breaker X1 was opened" might be represented by:

$$T(Opened(X1), 20030203T123059000)$$

Events may also involve changes in the value of a quantity, for example, suppose the low voltage current of the transformer SGT1 was 239.39 volts at a given time:

$$T(LvCurrent(Sgt1, 239.39), 20030203T123059000)$$

Russell [84] states that an argument to the T predicate, such as $LvCurrent(Sgt1, 239.39)$ in the example above, must not be treated as a sentence in predicate calculus (the T predicate, as with other predicates in first-order logic, may take only terms as its arguments, and the statement $LvCurrent(Sgt1, 239.39)$ is a sentence rather than a term). Therefore, $LvCurrent(Sgt1, 239.39)$ must be treated as a function, generating an event as its result. However, Shoham [87] disagrees with this "reification" approach (treating a statement as a function) because he believes that its semantics are unclear, and proposes that a different semantics based on modal logic be used (although both approaches share the same syntax and the T or $TRUE$ notation).

Several alternative syntactic representations of the event calculus may be found in the literature, such as the representation used by Kowalski and Sergot [88], which denotes each event by an identifier.

Timestamps In the examples above, the timestamp was given using the FIPA SL [78] standard time representation. This time format consists of a four digit year, two digit month and two digit date, followed by the letter "T", followed by a two digit hour, two digit minute, two digit second and three digit milliseconds value. This is accurate (assuming absolute accuracy of the device that produced the timestamp) to within 1 millisecond, which is also the most commonly used accuracy for time tagging of substation events within the power industry [89]. However, for certain functions such as phase angle measurement [90, 89], more accurate time tagging is required. It might also be useful to represent the accuracy of a particular timestamp, for example, to state that the timestamp of a particular event was accurate to +/- 1ms. For this purpose, a timestamp object, represented as a FIPA SL functional term with parameters, is proposed. This has two components: the timestamp itself, consisting of year, month, day, hour, minute, second, millisecond components, and its accuracy. Smaller time components than 1 millisecond will be represented by a floating point millisecond value, e.g. 123.45 milliseconds for 123 milliseconds and 450 microseconds.

An example of a timestamp frame is shown below:

```
(timestamp
    :time (time :year 2002 :month 12 :day 10
                :hours 10 :minutes 5 :seconds 2
                :milliseconds 20.34
          )
    :accuracy (time :milliseconds 10))
```

However, as can be seen from the example, this produces a very large textual representation which might reduce performance in transferring data. A possible abbreviation would be to use the standard FIPA time representation for the time and an integer value (representing milliseconds) for the accuracy, adding decimal places to the time representation as follows:

```
(timestamp
        :time 20020204T123456789.500
        :accuracy 0.1)
```

The drawback of this is that the adding of decimal places to the timestamp does not comply with the FIPA standard.

Representation of Non Machine-Understandable (Binary) Data

It is assumed that non machine-understandable data, such as documents and images, is stored on disk in a binary format. However, in order to transfer and display this information, it is necessary to represent some basic facts regarding it. This information is known as *metadata* (data about data). A standard set of terms, the Dublin Core vocabulary [91], are available for use in the representation of metadata and are widely used on the Internet, and are adopted for the system described here.

Representation of Derived Knowledge

Unlike event data, which can be put into a single, fixed format, it is not possible to do so in general for derived knowledge, as many different types of knowledge may be provided. For example, "transformer sgt1 is in good condition" can be considered as a binary relationship giving the value (good) of a property (condition) of a particular object (transformer sgt1), and might be represented by $Condition(Sgt1, Good)$. "The oil input temperature of transformer Y is between 30 and 40 degrees" is similar, but the value of the property is an interval rather than a single value. This might be represented by the statement "There is some x such that the input temperature of sgt1 is x and x is less than 40 and x is more than 30":

$$\exists x\, InputTemperature(Sgt1, x) \wedge (x < 40) \wedge (x > 30)$$

However, problems might exist when reasoning with this representation or storing it in a database, as many programming languages do not allow interval values for variables.

Finally, there are predictions made by condition monitoring systems, for example"if the ambient temperature does not rise then transformer Y will not exceed its operating requirements if the load increases by 50%" (This scenario is based on the system described in [92]). The representation of this statement is outside what may be represented in standard first-order logic. Roughly speaking, it might be interpreted as "Agent 1 (the agent making the prediction) believes that if the load on transformer Y is equal to Z and nothing else changes from its current situation, then transformer Y's condition will be good" (this assumes that transformer Y's condition is already good). The "Agent 1 believes" part of the statement might be represented using a modal logic of belief, as supported by FIPA SL. For example (using B to denote the "believes" predicate), it is possible to state "Agent 1 believes that the load on transformer SGT1 being l does not imply that the condition of SGT1 is bad":

$$B(Agent1, \neg(Load(Sgt1, l) \Rightarrow Condition(Sgt1, Bad)))$$

This is not sufficient to conclude that if the load on sgt1 is l then the condition of sgt1 will be good. In order to do this, the other conditions affecting sgt1's condition must also be stated (e.g. ambient temperature).

A simpler representation of a prediction might be to represent it as the result of a "predict" action taken by some agent. For example, an agent might predict that the hotspot temperature of a transformer at time <time> under load <load> would be 92.5 degrees centigrade:

```
(result
    (action
        (agent-identifier :name prediction-agent)
        (predict-hotspot-temperature
            :time <time>
            :load <load>
        ))
    92.5)
```

2.4.3 Ontologies

In addition to a common knowledge representation, a common *ontology* is also required for interoperability between agents. The ontology provides a defined set of terms to be used for communication, and hence eliminates problems caused by the use of different terms for the same physical object or quantity, for example, one agent referring to the temperature of a transformer as "temperature" and another as "temp". For example, we may define that "low voltage current" of a transformer is to be represented by a binary relation named *lv-current.* Once such an ontology is agreed upon, it is possible to use the multi-agent system to integrate data from heterogeneous sources, by converting the heterogeneous knowledge representations into that of the global ontology [93].

The ontology used in this system must consist of at least the following sub-ontologies (Figure 2.20), implementations of which in a prototype system are described in Chapter 5.

- The FIPA meta ontology, and in particular, that agents understand the *slot-of*, *template-slot-of*, *subclass-of*, *instance-of* relations, and where appropriate their inverse relations. This allows agents to use the concept of inheritance to derive facts about objects from their classes and facts about classes from their superclasses. It also allows, for example, the user interface agent to determine the available properties of an item of plant (using *slot-of*), which permits it to generate a list of properties that may be queried or used in a data analysis operation.

- An ontology describing substation plant, including the different classes of plant (circuit breaker, transformer etc) and the properties of those classes (for example, the tap position of a transformer). This ontology is used for the exchange of data and events regarding the substation. This is an example of a *domain ontology* (as defined in [93]).

- An ontology describing the components of the data acquisition and automation system. This permits agents to describe components of the data acquisition system, and perform operations such as system configuration. As with the plant ontology, this is a domain ontology.

- An ontology for information management, including concepts such as information resources, queries, and the relevance of an information resource to a query. This is a *task ontology* (as defined in [93]).

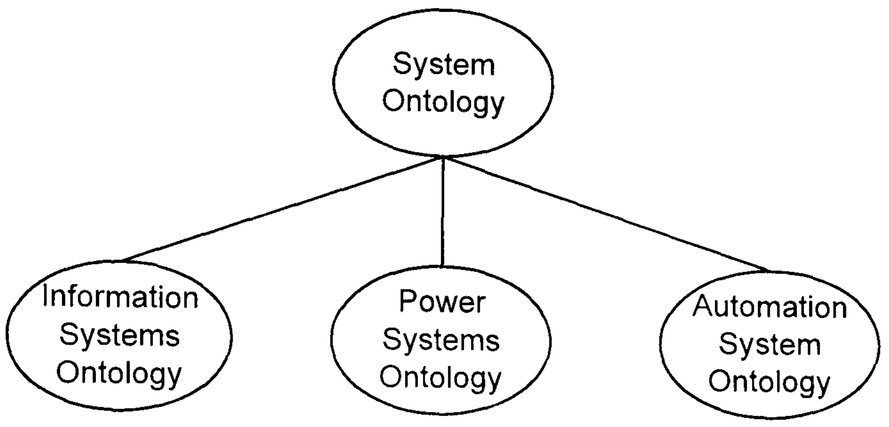


Figure 2.20: Hierarchy of Ontologies

Different agents use different subsets of the global ontology. For example:

- The *plant* class, which is the superclass of all substation automation equipment in the substation plant ontology, is shared by the user interface agent and the static database. The user agent queries the static database for subclasses of *plant* (to generate a list of types of substation plant), and then generates a list of instances of a chosen subclass using the *instance-of* relation. However, when programming the user interface agent, it is unnecessary to include any more of the substation plant ontology, as this may be obtained from the database agents. This lack of implicit knowledge of the substation domain embedded in the user interface agent could allow it to be used in a variety of different (industrial automation) domains, providing that a graphical user interface for the alternative domain was available.

2.5 Agent Platform Implementation

An agent platform is a software system providing a set of standard services (e.g. AMS, DF and ACC) to perform agent lifecycle management, communications and service discovery [76]. All agents are associated with a particular agent platform. Many agent platforms also provide a set of libraries in some programming language or languages (usually Java) which simplify the task of the agent programmer. The FIPA specifications define the services that must be provided by an agent platform, and how these services should be accessed from outside the platform. However, the internal architecture of the platform is dependent on the implementation. There are several agent platforms available which conform to the FIPA specifications.

For the initial prototype stage of the project, a mobile agent platform was implemented based on the FIPA [94] specifications. However, for the main prototype described in Chapter 5, the JADE [95] platform, which is a Java-based agent platform compliant to the FIPA specifications, was chosen, because at the time this choice was made it was the most commonly used FIPA platform, and was the only freely available FIPA platform with mobile agent support. However, the architecture itself is not dependent on any particular platform. This section examines some of the implementation choices that would have to be made in the development of a system based on this architecture.

2.5.1 Standard FIPA Platform

There are a number of standard FIPA platforms available [96], and one of these could be used for the development of the system, as was done for the prototype described in Chapter 5.

The main advantage of using such a platform is the reduction in the amount of implementation effort required to implement the architecture through the use of an off-the-shelf agent platform. Also, the platform would be able to inter-operate with other FIPA systems if that was required.

The major disadvantages of using a FIPA platform are the fact that no

FIPA platform provides inter-platform mobile agent support (JADE supports intra-platform agent mobility), and this would have to be added, if required, either by modifying the platform or by implementing a simple mobile agent system which would interoperate with this platform through the FIPA IIOP interface and be used solely for hosting mobile agents. Also, most current FIPA platforms are based on Java and CORBA. This means that their use for real-time applications, or for hosting agents on substation devices, is restricted, although there are PersonalJava or Java 2 Micro Edition compatible platforms available.

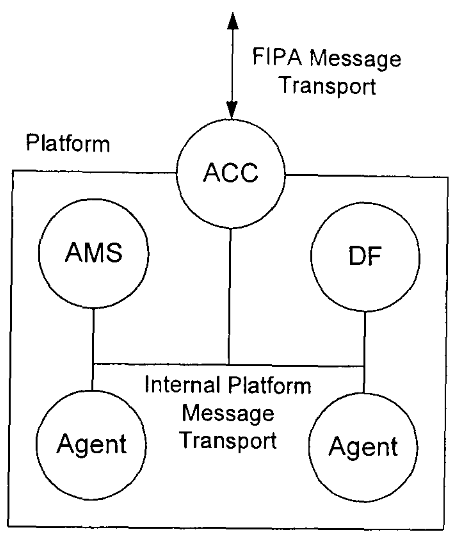


Figure 2.21: Abstract FIPA Agent Platform (Simplified)

Figure 2.21 shows the abstract architecture of a FIPA agent platform. The standard services (AMS, ACC and DF) provide lifecycle management, communications and messaging facilities and service lookup respectively. All agents, including these, communicate via the Internal Platform Message Transport (IPMT). For communications with agents on other platforms, the ACC communicates with a remote platform ACC via the FIPA Message Transport service. However, when considering implementation issues it is simpler to discuss a particular instantiation of this architecture. Figure 2.22 shows the architecture of the JADE agent platform, which is FIPA-compliant and is developed by Telecom Italia Laboratories and the University of Parma [95]. JADE's architecture is based on a modular structure, in which a platform is split into a number of containers, which may run on different machines, and communicate via Java RMI. Communication between agents on the same container uses Java method calls.

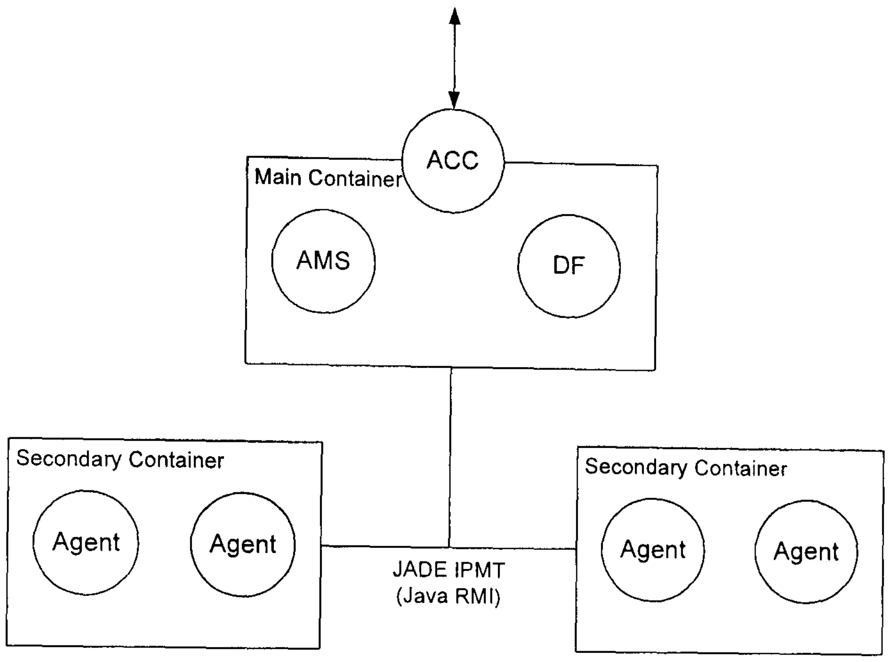


Figure 2.22: Java Agent Development Environment Architecture

2.5.2 Jini Based Platform

Jini is a system for service management and discovery developed by Sun Microsystems and based on Java and Java RMI. It would be possible to implement an agent platform based on Jini. In fact, there are several such platforms in existence, for example, the Ronin platform [97], and the platform used by the PABADIS project [63]. Such a Jini-based platform might be implemented as follows:

- A standard Jini service is defined for an agent, which has a single method, allowing agents to send messages to that agent. This is based on the IIOP interface specified by FIPA

- A gateway provides FIPA services and allows the platform to interoperate with FIPA systems.

A design for this platform is shown in Figure 2.23.

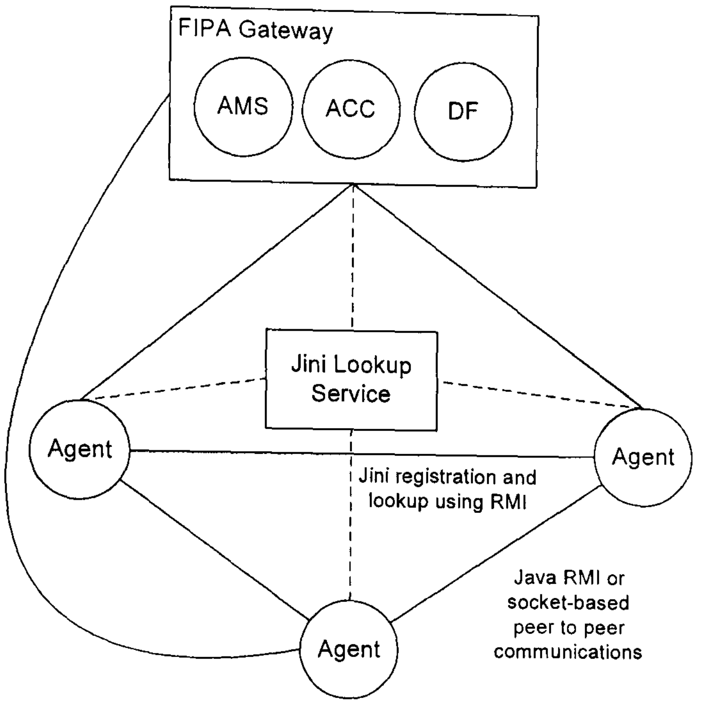

FIPA Gateway
AMS
ACC
DF
Jini Lookup Service
Agent
Agent
Jini registration and lookup using RMI
Java RMI or socket-based peer to peer communications
Agent

Figure 2.23: Jini-based Agent Platform

The main interesting feature of the implementation described here would be its highly decentralized nature, in which there are very few essential platform services. The only requirement would be the Jini lookup server. Other services such as brokers and the FIPA gateway would themselves be agents, rather than components of the platform. The Jini architecture provides high reliability compared to other agent platforms, as it includes features such as leasing, which prevents references from remaining in the lookup service after a service has terminated. It is claimed that leasing allows Jini to "self-heal" a distributed system [98].

The problem with creating a Jini based implementation is that a substantial amount of implementation work would be required to make such a platform FIPA compatible. It would be necessary to provide "wrappers" for the Jini lookup service to allow it to be used as a FIPA Directory Facilitator, and a service would have to be provided, either by each agent individually or by a gateway agent, to permit agents to send and receive FIPA ACL messages.

2.5.3 UDP Based Platform

A very simple agent platform, with limited infrastructure, would be one based on the use of the User Datagram Protocol (UDP) to send messages between agents. Each agent would listen for messages on a UDP port, and agents would also be able to listen for broadcast messages on a broadcast port. This allows for both peer-to-peer and broadcast messaging. In the context of this infrastructure, a "platform" would be a single subnet of the network, in which all agents were within the range of each others' broadcast messages.

The structure of the UDP-based platform is identical to that of the Jini-based platform, apart from that the protocol used both for inter-agent and agent to lookup server communications is UDP rather than Java RMI. The central lookup server would be used to enable agents outside the platform, who were not capable of sending broadcast messages into the network, to locate server agents inside the platform. A FIPA gateway would also need to be provided, which would also act as a message router and lookup service for inter-platform communications.

Problems might be caused when implementing this platform by the lack of ability of certain hubs and switches to transmit broadcast messages. This would have to be examined further were this platform to be implemented.

The main advantage of the UDP-based platform is that the UDP protocol provides for extremely fast message transmission. Therefore this platform might be better suited for the implementation of agents which were required to work under timing constraints, such as device agents. Also, because the agents do not execute on a platform server, they may be written using any programming language. However, this can also be achieved to a lesser extent when using a FIPA platform, as agents can interoperate with and register on the platform using the IIOP interface.

The drawbacks with this platform, as with the Jini platform, mostly concern the amount of implementation work required. Also, the platform may not be as robust as the Jini platform because UDP does not provide reliability features. although these could be implemented at platform level.

2.5.4 Combined FIPA and UDP Based Platform

A possible solution to the real-time and limited device difficulties of a FIPA platform and the scalability problems and implementation overhead of a UDP-based platform would be to combine the two. This combined platform might have a similar conceptual architecture to that of the Lightweight and Extensible Agent Platform (LEAP) system [99], which extends the JADE FIPA platform by providing a "container" based on the Java 2 Micro Edition, which hosts JADE agents and communicates with a host platform using a proprietary socket-based protocol. However, the platform proposed here is slightly different, in that it consists of a FIPA-based platform along with a gateway agent, which allows any number of independent agents to join the system. This agents are implemented as described for the stand-alone UDP-based platform.

The function of the gateway is to translate messages from the format used by the UDP-based agents to FIPA message objects as defined by the FIPA platform, and to pass them to the FIPA platform's Agent Communications Channel (ACC) agent for handling. This means that all platform services for the UDP-based platform can be provided by the FIPA platform, but the UDP-based agents retain the ability to communicate with each other in a peer-to-peer manner, without using the platform, when required.

2.6 Summary

This chapter has described a generic multi-agent software architecture for power system automation systems. The agent architecture is derived from the structure of the power system and data acquisition system. The device agents provide a view of the system based on the monitoring and control system, which consists of data acquisition devices. Meanwhile, the plant agents provide

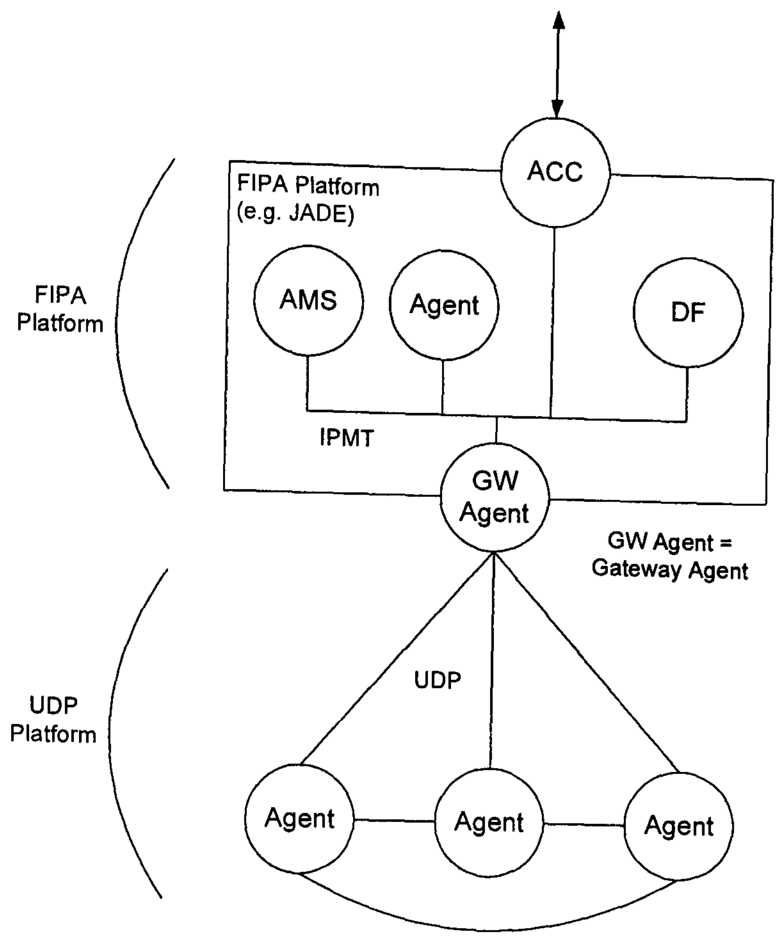


Figure 2.24: Combined Platform

a representation based on the substation plant, which is convenient for the implementation of distributed control, and also for acquiring information about a specific item of plant. The use of one agent for each device or plant item provides a highly distributed architecture, and a natural representation of the physical system being controlled and monitored.

The representation of power system knowledge within the multi-agent system and the methods of communication between agents are also discussed. The system uses the logic-based language FIPA SL to encode events, historical data and commands. A basic ontology of automation systems and power systems is provided for this purpose. Agents communicate using various FIPA standard protocols. The FIPA subscribe protocol is used for regular monitoring, the FIPA query protocol for database and document querying, and the FIPA request protocol for the transmission of commands. The use of these three protocols permits a wide variety of interaction styles and allows many different tasks to be implemented.

Finally, this chapter has investigated the implementation of the agent platform. This is an important aspect of the system as it provides the basis for all agent execution and communications. Various possible platforms were described, including standard FIPA platforms and platforms based on Jini and UDP. For the purposes of the prototype, the FIPA platform was chosen, as it was the most convenient and provided a simple method of implementation. However, for a full system, a hybrid UDP and FIPA platform might be more suitable, as the use of UDP provides for higher-speed communications, which are important in certain industrial applications.

The overall architecture having been considered in this chapter, the next chapter discusses the design and implementation of the individual static agents. The tasks that each agent must perform are used to derive the required capabilities, sensors and knowledge of the agent. Where appropriate, issues regarding agent implementation are also described.

Chapter 3

Static Components of Architecture

This chapter describes the individual software agents that make up the architecture described in Chapter 2. All agents are based around the same basic architecture, shown in Figures 3.1 and 3.2. This is derived from standard descriptions of agents such as those in [36][8][84], and also draws on the Belief-Desire-Intention architecture, described in Section 1.2. Each agent consists of sensors, which allow the agent to perceive its environment, a knowledge base containing the agent's beliefs and goals, and effectors, which allow the agent to take actions. A reasoning engine takes inputs from the sensors and knowledge base, and determines the actions to carry out. This may be done in different ways, for example, in a BDI agent plans stored as part of the agent's knowledge are used. It is intended that the architecture used should allow different forms of reasoning to be used and does not commit the agent to using any specific methodology. However, individual agents may be, for example, BDI agents, in which case the reasoning engine of the agent would be a BDI interpreter. By creating agents with different reasoning processes, knowledge, sensors and effectors, different tasks may be carried out.

The agent specifications provided in this chapter state the knowledge, capabilities, sensors, effectors and interaction protocols of each agent. The sensors

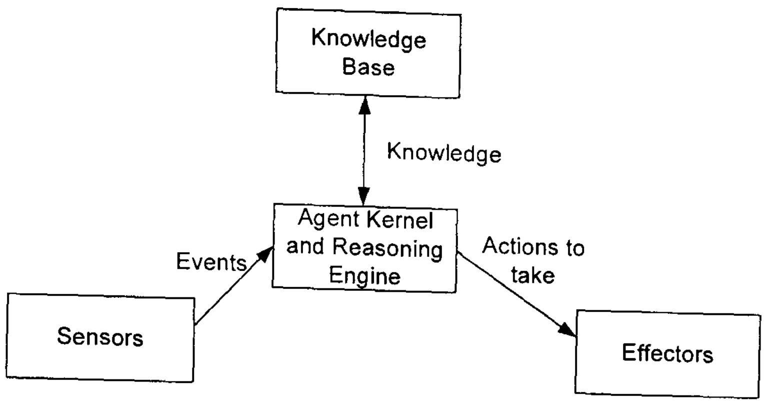


Figure 3.1: Agent Architecture

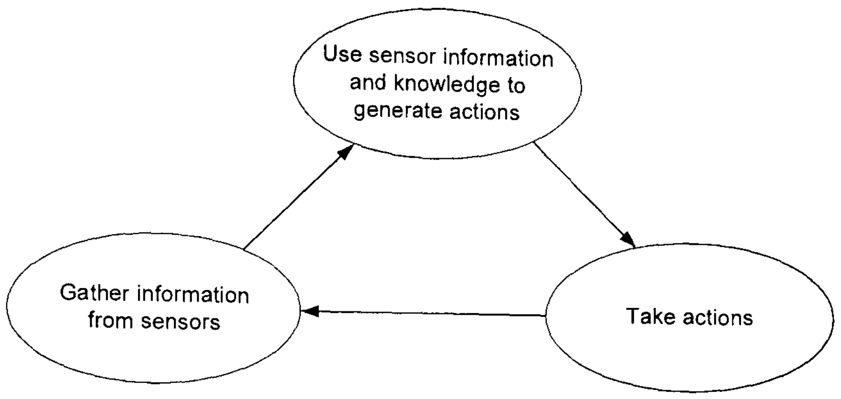

Use sensor information and knowledge to generate actions
Gather information from sensors
Take actions

Figure 3.2: Agent Control Loop

provide input to the agent, the capabilities (effectors) allow the agent to perform actions and the interaction protocols provide the means for the agent to engage in standard conversations and therefore to collaborate with other agents.

In addition to this, the agent specifications also detail which components of the system ontologies (described in Section 2.4.3) are used in the conversations of each agent. It is assumed that all agents use the *FIPA Agent Management* ontology (for registering with and searching the Directory Facilitator to enable agents providing a particular service to be located).

As well as the specifications of the individual agents, this chapter also discusses some of the implementation and knowledge representation issues involved with certain types of agent. These include the implementation of distributed database querying in the database agents, the use of BDI mental components to represent an agent, and the representation of mapping rules for translating sensor data into a plant state representation in the plant agents.

3.1 Database Agents

3.1.1 Description

The database agents provide Foundation for Intelligent Physical Agents Agent Communication Language (FIPA ACL)-based access to a database. Database agents of this type have previously been used in a wide range of information management applications and architectures, for example [100]. The structure of the agent is shown in Figure 3.3. In order to fulfil its task, the database agent must be able to insert and retrieve information to and from the database and to convert this information to FIPA ACL and into the system's global data model (ontology). This conversion is performed by the agent's reasoning engine using mapping rules stored in the knowledge base of the agent, which specify, for example, that a particular property of an item of plant corresponds to a given column in a database table. The format of these depends on the particular implementation of the agent used. The agent's communications sensor and effectors allow it to send and receive information to other agents.

Unlike most of the other agents in the system, the database agent does not implement the *fipa-subscribe* protocol, which means that it is unable to provide subscription-based access to a database, in which subscribing agents would be notified of new data as it arrived. While it would be possible for it to do so, the only method of implementation available with most available database management systems would be to "poll" the database at regular intervals to determine if new knowledge had been added. There are two exceptions to this statement:

1. If the database agent is the only entity capable of inserting information into the database, it will be able to provide subscription functionality as

the new data will be passed through the database agent.

2. If an "active database" [101] is used, it might be possible to add a rule to the database requiring it to generate an event on the arrival of particular data and pass that event to the database agent by some means (possibly specific to the particular database).

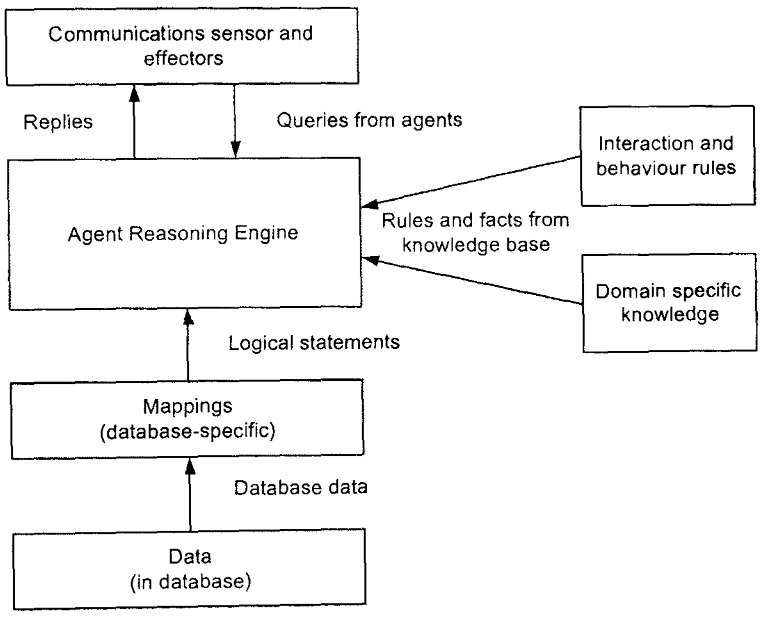

Communications sensor and effectors
Replies
Queries from agents
Interaction and behaviour rules
Agent Reasoning Engine
Rules and facts from knowledge base
Domain specific knowledge
Logical statements
Mappings (database-specific)
Database data
Data (in database)

Figure 3.3: Structure of Database Agent

3.1.2 Agent Specification

This specification defines the knowledge, capabilities, sensors, interaction protocols and ontological knowledge required by the database agent. Using this information, it is possible to modify a generic agent to act as a database agent.

- Knowledge
  - Transformation rules from database schema to global schema.
  - Database configuration (Java Database Connectivity driver, username, password).

- Capabilities
  - Query database
  - Append data to database[1]
- Sensors
  - No explicit sensors (apart from communications). This agent exists only in a software environment, and has no connection to the outside world except through other agents.
- Interaction Protocols
  - FIPA query (responder) - used by other agents to query the database
  - FIPA request (responder) - if data is to be added to database on the initiative of another agent rather than via a subscription.
  - FIPA subscribe (initiator) - this allows the database agent to establish a subscription with a data providing agent (e.g. a plant agent) so that new events are transferred into the database as they occur.
- Ontology
  - The agent converses using components of the global ontology relevant to the knowledge contained in its own database. This will differ for different database agents and is implicit in the mapping rules of the agent.

3.1.3 Agent Implementation

It is possible to implement the reasoning of a database agent in a number of different ways. This is particularly relevant to the need to integrate information from multiple databases. Different implementations allow this to be

[1]In some applications, for example the substation information management application described later in this thesis, the database agent is not able to alter the database, but can only query it.

performed in different ways. Also, certain implementations may provide additional functionality which might be useful in a particular application. Here three alternative implementations are considered, a simple agent which only acts in direct response to queries from other agents, a simple collaborative agent which is capable of querying other agents, and a BDI agent.

Simple Implementation

In a simple implementation, the agent implements the FIPA query protocol, but does not have any autonomy: it will not provide information unless it is asked for it. The agent must wait until it receives either a *query-if* or *query-ref* message. It then uses the content of that message to generate an SQL query to the database.

Simple Collaborative Implementation

This implementation, based on previous work such as [77] and on the techniques used by distributed Prolog implementations, extends the capabilities of the database agent by allowing it to communicate with other database agents and integrate information. The agent uses the DF to determine queries that may be posed to other agents, represented as referential expressions (as defined in the FIPA SL specification [78]). For example, an agent might advertise that it has knowledge about the status of circuit breaker "h13" using the referential expression "(iota ?a (status h13 ?a))". When performing a query, a database agent will retrieve information from its own database where possible, and if not possible, will forward the query, or the part of the query that it is unable to answer, to another agent that is capable of answering it.

This might employ a distributed backtracking procedure, as has been implemented in distributed Prolog systems such as those discussed by [102]. Alternatively, it might be better to implement this collaborative capability as a separate broker agent, as is done in the Infomaster and RETSINA systems [100, 59]. This would reduce the load on the database agent and reduce the need for parallel query processing.

BDI Implementation

In a BDI implementation, we give the agent explicit goals regarding the information it is to provide to other agents, as well as allowing it to reason about other agents' information needs. This can either be used to extend the behaviour of the agent, for example to provide the agent with the ability to remember past queries as proposed by [103], or just to provide an alternative conceptualization. In the BDI implementation, whenever a query with content p arrives from another agent a a goal is added to achieve "a knows p". Thereafter, unless the database agent comes to believe that a already knows p (perhaps because the database agent has already informed a of p), or that it is impossible that a will come to know p, the database agent will attempt to make a know p. This allows for two responses from the database agent: either to notify a immediately of the answer (if it is available), or at a subsequent time, as the information becomes available, to notify a. The agent may also decide to perform some action that would lead to it knowing p, perhaps querying another agent.

Possible alternative methods to achieve this "persistent query" behaviour are the use of the subscribe interaction protocol (although this results in the agent being notified whenever a value changes rather than just the first time it becomes available) or the *request-when* communicative act. However, these methods require the querying agent to explicitly specify that it wishes to be informed of something at a later time.

Choice of Implementation In the prototype system, it was decided to use the simple collaborative implementation. This permitted agents to query each other for information that was not present in their particular database. For example, the static database agent was able to retrieve the properties of its items of plant using the ontology database agent. The BDI implementation was not used because of the amount of time required to implement it and because the functionality provided was not required. However, it would be considered as a possible enhancement for the system in the future. The simple implementation could also be used, but in this case a broker agent would be required to perform the information integration, or alternatively this functionality could be added to the user agent. However, this would increase the complexity of the system either by adding another agent or by making the user agent more complex.

3.2 Document Agents

3.2.1 Description

The document agent is responsible for the management of documents stored in a particular location, such as a directory of a filesystem. The agent has two main responsibilities: to ensure that it stores up-to-date statistics regarding its document collection, and to carry out queries on behalf of other agents. The structure of the document agent is shown in Figure 3.4. This is similar to that of the database agent. However, the document agent requires a sensor to notify it when new documents are added to the repository, and the ability to count the words in a document and generate metadata for use by an information retrieval algorithm. This algorithm can then be used to determine the relevance of a document to a particular query. In the prototype system, the information retrieval algorithm used is the standard Term Frequency / Inverse Document Frequency ranking algorithm[104]. The reasoning engine used by the agent is relatively simple. When queried it passes the query to the document ranking algorithm, which generates a set of matching documents. These are then returned to the querying agent. Agents can also use the document agent to retrieve the full text of a document.

A possible extension of the document agent would be to enable it to forward new documents as they arrive to interested parties, based on past queries. This would be relatively simple, as the agent already has a sensor notifying it when new documents arrive. The required addition would be a list of past queries from users. The agent could then run a user's past queries against a new document to determine whether or not it is relevant to any of them, and if so, forward the document to the user. This is similar to the basic methodology used by Selective Dissemination of Information (SDI) systems, as described in [105].

As with the database agent, the document agent might be conceptualized as a BDI system. The definition and implementation would be similar to that for the database agent.

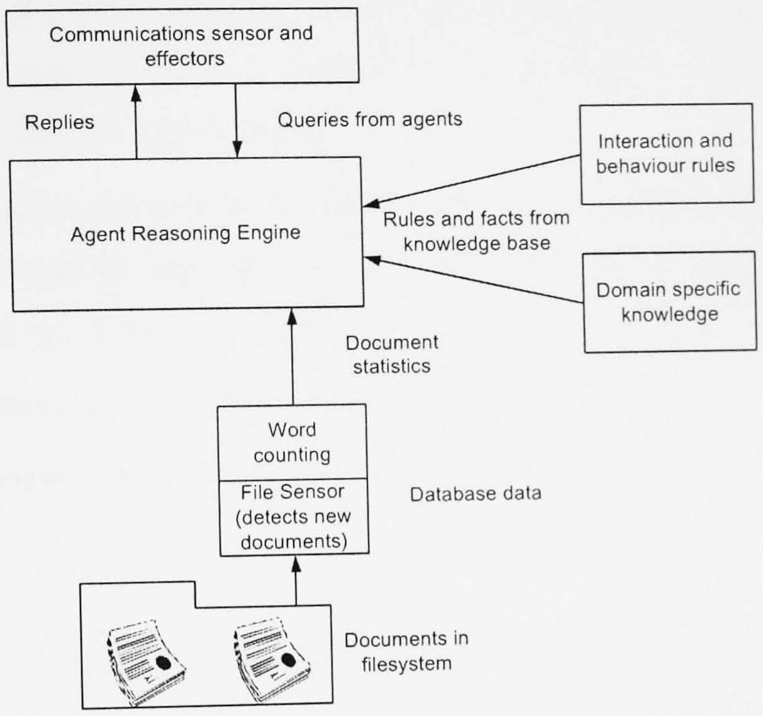

Communications sensor and effectors
Replies
Queries from agents
Interaction and behaviour rules
Agent Reasoning Engine
Rules and facts from knowledge base
Domain specific knowledge
Document statistics
Word counting
File Sensor (detects new documents)
Database data
Documents in filesystem

Figure 3.4: Structure of Document Agent

3.2.2 Agent Specification

The knowledge, capabilities and sensors of the document agent allow it to generate document statistics and retrieve documents relevant to a query. To allow other agents to query for relevant documents, the FIPA Query protocol is used. The FIPA Request protocol allows other agents to request the text of a document.

- Knowledge
  - Document collection statistics: These statistics must provide sufficient information for the agent to perform queries against its doc-

ument collection. For example, the TF/IDF model [104] requires term frequency and document frequency statistics.

- Capabilities
  - Convert document to plain text: In order for document statistics to be generated, the plain text (ASCII/Unicode) of a document must be extracted from the provided representation. This ability (or set of abilities) allows the agent to do this.
  - Generate document statistics: This ability must be able to generate the required document statistics for the particular retrieval methodology used from the plain text of a document.
  - Retrieve list of documents relevant to a query
  - Retrieve document full text on request
- Sensors
  - Detect addition and removal of documents in a specified location. This sensor regularly scans the document repository, and generates an event whenever a new document is added or removed. This event may then be processed by the agent, which should react by generating statistics for the document and updating its document collection statistics.
- Interaction Protocols
  - FIPA Query
  - FIPA Request
- Ontology
  - The agent converses with other agents using the *information management* ontology. This ontology contains terms to describe documents and other information sources and the relevancy of an information source to a query.

3.2.3 Document Agent Issues

The main problem with the document agent is that methodologies for integrating the results of searches on multiple document collections, particularly when statistical methods such as TF-IDF are used by the individual collections, are still being researched. For example, [106] discusses different methodologies for distributed information retrieval. The authors state that, since document ranking heuristics such as TF-IDF make use of collection-dependent statistics, the relevance of a document to a query produced by one collection would not be the same as that produced for the same document by a different collection. Therefore, the authors conclude that "central coordination is necessary for aggregation of the results", particularly where the collection sizes are small.

In the architecture described, this issue could be handled by having a central "document broker" agent, which would hold aggregate statistics from the individual collections as in [106]. However, this might compromise the autonomy of the individual document agents. Further research into this area is required to determine the optimal solution.

3.3 Ontology Agents

The ontology agent used in this architecture implements the FIPA Ontology Service described in [79]. In addition, it registers a number of predicates with the DF as a "query service" as described in Section 2.2. Because the FIPA Ontology Service specification specifies the behaviour of this agent, and no autonomous behaviour is required, a simple database agent architecture may be used for this agent. Therefore, the implementation and structure of this agent is the same as that of the database agent described in Section 3.1. However, a set of rules specifying the FIPA Meta Ontology are required to permit it to accurately answer queries. For example, rather than specify separate rules regarding how to extract *superclass-of* and *subclass-of* from the database, it is possible to define *subclass-of* in terms of *superclass-of*:

$$SubclassOf(a, b) \Rightarrow SuperclassOf(b, a)$$

This requires that the agent has a reasoning engine capable of processing either logical statements or IF/THEN rules.

3.4 Device Agents and Node Agents

3.4.1 Description

A device agent is responsible for the management of a specific data acquisition device. The tasks of the device agent include both input/output and device configuration. However, data interpretation is the responsibility of the plant agents.

A node agent is responsible for managing a data acquisition "node". This corresponds to a single computer or IED, having a CPU and containing or controlling a number of data acquisition devices. Each node has a set of interfaces which enable it to communicate with other nodes and with the data acquisition system.

A device agent must maintain up-to-data knowledge of the current status of its device and the values of each input/output channel, which is achieved using the data acquisition sensor. It must also be able to provide this knowledge to other agents, in particular the plant agents, both via queries and subscriptions. Other agents must also be able to write to the output channels of the device using the FIPA-request protocol.

3.4.2 Agent Specification

The knowledge of a device agent relates to the device that it manages, and its capabilities allow it to interact with that device and with other agents. The node agent is provided with information about its node and has the capability to perform configuration tasks.

Device Agent

- Knowledge
  - Type of device (to select data acquisition sensor implementation - however, this might be implicit in the configuration of the agent rather than explicit knowledge)
  - Current status of device
  - Current value of each channel of the device
  - Device configuration
- Capabilities
  - Set the value of any output channel of the device.
  - Change device configuration (e.g. set sampling rate)
  - Start data acquisition
  - Stop data acquisition
- Sensors
  - Data acquisition sensor to permit data to be acquired from the device. This sensor should use an appropriate communications facility to maintain knowledge of the current state of each (input or output) channel of the device. Whenever a channel's value changes, an appropriate event should be generated. While the communications protocol used by the sensor depends on the device, the format of the event generated should be device-independent. Therefore, this sensor performs a mapping from the device's protocol to a standard event format. In addition, all events must be timestamped. If a timestamp is not provided by the data acquisition system, it should be added by the device agent, adding the appropriate level of accuracy to allow for data acquisition delays.

- Interaction Protocols
  - FIPA query - used by other agents to retrieve data from the data acquisition system
  - FIPA subscribe - used by other agents to establish a subscription to be notified whenever the value of a channel changes.
  - FIPA request - used to write new values to channels of the device.
- Ontology
  - The agent describes its device using terms taken from the automation system ontology. This contains terms describing automation devices and channels. It therefore has no knowledge of the application domain (e.g. power systems) and does not use this domain ontology.

Node Agent

- Knowledge
  - Communications interfaces belonging to the node, and their configuration.
  - A list of devices belonging to that node.
- Capabilities
  - Change node configuration (exact changes will depend on the individual node)
- Sensors
  - The node agent is capable of reading the configuration of its node.
- Interaction Protocols

  - FIPA query - used by other agents to retrieve the configuration of the node
  - FIPA request - used to request configuration changes

- Ontology
  - As for the device agent, the automation systems ontology is used.

3.5 Plant Agents

3.5.1 Description

A plant agent is responsible for the monitoring and control of a single item of plant. In a substation, the main items of plant include circuit breakers, disconnectors, transformers, busbars, capacitors and reactors. For each of these categories, a distinct agent must be created, with appropriate knowledge and capabilities to enable it to represent that plant effectively. The following section discusses the possible design of these agents, which is used in the current prototype. However, because an automatic control application has not yet been implemented it has not been possible to validate the automatic control function of these agents. Therefore, refinements to the design may prove necessary in the light of experience from such an implementation. Another further enhancement which is not included in the design is collaboration between control agents of different items of plant.

A plant agent must maintain up-to-date knowledge of the state of its item of plant. The most efficient way to do this for dynamic properties (e.g. voltage, current) is to establish subscriptions with the relevant device agents. In this way the state information will only be transmitted when a value changes. Providing that the value of a property changes less often than the polling interval, subscription-based monitoring of that property is more efficient than regular polling of the device agent using query messages.

3.5.2 Agent Specifications

Here we provide agent specifications for a generic plant agent. For specific plant agents, e.g. transformer agents and circuit breaker agents, the abilities and knowledge would be customized to the specific item of plant. For example, the abilities of a circuit breaker agent would allow it to open and close the circuit breaker.

- Knowledge
  - Mapping from plant properties to channels monitoring these properties (obtained from mapping database on agent startup).
  - Static information regarding plant (obtained from static database on agent startup).
  - Dynamic properties of plant, e.g. voltage, current, etc (obtained from monitoring agents on regular or event-driven basis).
  - Possibly knowledge regarding agents controlling other items of plant (this is an item for further work as it relates to automatic control of the plant).
- Capabilities
  - Specific to the device:
    - Transformer agent: set tap position and activate / deactivate oil cooler.
    - Switchgear agent: open or close item of switchgear.
- Sensors
  - The agent has no direct connection to the plant, but obtains information by communicating with other agents. Therefore, there are no explicit sensors.
- Interaction Protocols

  - FIPA query (responder) - used to allow other agents to query the status of the plant.
  - FIPA query (initiator) - used to obtain information from device agents.
  - FIPA request (responder) - used to allow other agents control the plant.
  - FIPA request (initiator) - used to forward control requests to device agents.
  - FIPA subscribe (responder) - used by other agents to request updates whenever the status of the circuit breaker changes.
  - FIPA subscribe (initiator) - used to obtain regular updates from device agents.

- Ontology
  - For information management purposes, the plant agent uses the *FIPA meta ontology* to determine the class to which its item of plant belongs, and hence its properties. It can then use this information to set up subscriptions with appropriate device agents, using the *value* relationship and *channel* class from the *automation system ontology*.
  - For control purposes, a set of plant control rules must be written using the plant ontology. As far as information management is concerned the agent is largely capable of self-configuration as the mapping rules may be read from the mapping database. However, the agent must currently be explicitly configured to control a specific device as there is no database for control rules. Further work should investigate the possibility of sharing control rules between similar plant agents (e.g. transformer agents), in the context of a specific application.

3.5.3 Data Acquisition System / Plant Mappings

As described in Section 2.2.2, each property of an item of plant may be measured or controlled by one or more channels of the data acquisition system. However, it is not always the case that there is a one-to-one correspondence between plant properties and data acquisition system channels. Several different types of mapping are possible:

Numerical mappings (input or output) are those in which the value of the plant property is determined by a mathematical expression in which the only other variable is the value of a single channel of the data acquisition system. For example, representing the value plant property by l and the value of a corresponding channel by c, a relevant numerical mapping might be:

$$l = 128.9c + 2.5$$

These mappings may be found in traditional SCADA systems (for example, National Instruments' Supervisory Control Toolkit for LabVIEW permits the specification of linear or square root based scaling between raw analogue input data and tag values).

Enumeration mappings (input or output) are those in which a set of values $\{c_1, c_2...c_n\}$of some channel map to a corresponding set of values $\{l_1, l_2...l_m\}$ of a plant property. The mapping may be either 1:1 or many:1. For example, the input values $\{0.0, 0.4\}$ of an input channel on a data acquisition system may map to the status values $\{open, closed\}$of a particular circuit breaker. In a many-to-one mapping, for example, all values between 0 and 0.2 might map to "open", and all values between 0.2 and 0.4 to "closed".

Complex actions (output) represent sequences of actions that must be undertaken in order to change the value of a property. For example, to alter the tap position of a transformer in the substation simulator described in Chapter 5, it is necessary to write a 1 to the "tap up" or "tap down" channel, and then wait for a response on another channel before writing a 0 to "tap up" or "tap down". To alter the tap position by more than 1, it is necessary to repeat this procedure the appropriate number of times. Complex actions may be represented by a state machine or by a procedure in a computer programming language.

Combinations of the above It may also be possible to perform mappings which are a combination of the above types, for example, converting a value using a mathematical formula and then mapping it to an element of an enumeration.

Representation of Mappings in First-Order Logic

It is necessary for the agent knowledge base to contain a representation of the mappings described above. These could be implicitly specified in the programming of a device agent or plant agent. However, it is intended that agents should be able to exchange knowledge about these mappings, for example, to allow a plant agent to retrieve the mappings corresponding to its item of plant from a device agent or database agent. Therefore, a representation of each type of mapping in first-order logic is required. Here, simple atomic names are used for channels and items of plant; the actual representation of these objects may be more complex.

Simple mappings may be represented as a simple formula, for example, the following states that if the value of a channel "Lvc" is v, then the low voltage current of a transformer "Sgt1" is equal to v multiplied by 128.9, added to 2.5.

$$\forall v\, Value(Lvc, v) \Rightarrow LvCurrent(Sgt1, (v * 128.9) + 2.5)$$

assuming that the + and * functions are defined for the particular agent, and that the *value* predicate gives the value of a particular channel. However, it is not possible for one agent to query another and retrieve these formulae, because a variable is only capable of representing a term, not a well-formed formula [78].

Enumeration mappings may be represented as a set of rules, one for each value in the enumeration. For example, the following formulae state that if the value of a channel "Rl1" is 0.4, then the circuit breaker "X110" is closed. If the value of the "Rl1" channel is 0, then the circuit breaker "X110" is open.

$$\begin{aligned} Value(Rl1, 0.4) &\Rightarrow Status(X110, Closed) \\ Value(Rl1, 0) &\Rightarrow Status(X110, Open) \end{aligned}$$

Many to one mappings may be represented by more complex mathematical expressions, for example, the following expressions state that if the value of "Rl1" is more than or equal to 0.2, then "X110" is closed, otherwise, X110 is open.

$$\begin{aligned} \forall x\, Value(Rl1, x) \wedge x \geq 0.2 &\Rightarrow Status(X110, Closed) \\ \forall x\, Value(Rl1, x) \wedge x < 0.2 &\Rightarrow Status(X110, Open) \end{aligned}$$

Complex actions may be represented as a sequence of actions (*ActionExpression* in the FIPA SL specification). For example, consider changing the tap position of a transformer on the substation simulator as described above. This is a four-step process:

1. Write 1 to the "tap up" channel of the correct data acquisition device.
2. When the tap change starts, the "tap change in progress" channel's value will change to 1.
3. When the tap change is completed, the "tap change in progress" channel's value will revert to 0.
4. The agent should then write a 0 to the "tap up" channel.

This process can be represented in logic by the following sequence of actions, assuming that the "write" action writes a value to a channel, and the "wait" action waits for a particular channel to have a particular value. The ";" operator represents the fact that one action is followed by another, as in FIPA SL and the logics described in [85, 107].

$$\begin{gathered} Action(Agent1, Write(tu1, 1)); \\ Action(Agent1, Wait(tc1, 1)); \\ Action(Agent1, Wait(tc1, 0)); \\ Action(Agent1, Write(tu1, 0)) \end{gathered}$$

A BDI agent might wish to use this sequence of actions as a plan body. It would then be necessary to express the preconditions, postconditions, etc, of this action sequence. Alternatively, we might define a primitive action, such as "tap-up" and then state that the complex action sequence "implements" the primitive action. For example:

$$ImplementedBy(TapUp(Sgt1), < action\,sequence >)$$

where <action sequence> is replaced by the sequence of actions specified above.

To express the preconditions of an action we could use (where <set of conditions>) can be substituted by any legal first order logical formula:

$$< set\,of\,conditions > \Rightarrow Feasible(< action\,sequence >)$$

This is legal because both arguments to *implies* may be well-formed formulae, the conditions are represented by the conjunction (*and*) of several well-formed formulae, and an action sequence in SL is a well-formed formula. For example, suppose we wish to state that it is possible to open a relay if it is not already open (The designation $Agent1$ replaces the frame-based agent identifier that is used in FIPA SL):

$$\neg Status(Relay0, Open) \Rightarrow Feasible(Action(Agent1, Open(Relay0))$$

The following rule is a general rule suitable for all relays, which states that for all a and r, if r is a relay and a manages r and the status of r is not "Open", then it is possible for a to carry out the action *Open(r)*. The *manages* predicate states that an agent "manages" or is responsible for a particular item of plant - in the system described this will be the plant agent corresponding to that item of plant.

$$\begin{aligned} \forall a, r \quad & Relay(r) \wedge Manages(a, r) \wedge Status(r, Open) \\ \Rightarrow \quad & Possible(Action(a, Open(r))) \end{aligned}$$

Transmission of Mappings Between Agents

The mapping representations described above using *implies* are capable of being transmitted between agents (they are legal FIPA SL content expressions). However, it is not possible for an agent to query another to retrieve these mappings, because a variable in FIPA SL may only represent a term and not a well-formed formula. There are a number of possible ways to work around this limitation, including:

1. Use quotes to transmit the mapping expression as a string. It would then be possible to send a query, for example (all ?a (mapping-rule ?a)), and receive an answer such as (= (all ?a (mapping-rule ?a))) (set "(implies (value ds1 0) (status x10 open))")). The problem with this is that it is not possible to query mapping rules for an object involved (because the mapping rule is an opaque string), and so it is necessary for the mapping agent to maintain a database relating mapping rules to items of plant. For example, a database table might contain three columns: plant name, property name and mapping rule, in order that mapping rules could be searched.

2. Use a "mapping" structure to hold mapping information. (e.g. (mapping :channel TLV1 :scale-factor 3 :increment 2 :type "linear")). However,

this only works for simple mappings, and it is also difficult to understand without prior knowledge of the mapping structure. Finally, the names of the mapping type (e.g. "linear") must be predefined, so it is not possible to represent arbitrary arithmetic expressions.

3. Use the "implements" / "implemented-by" relation as described above (only applicable to output mappings).

3.6 User Interface Agents

3.6.1 Description

As in other agent systems, the user interface agent in this system provides a link between the agent community and users. In this system, the interface with the user is via a human machine interface (HMI), different versions of which will be available for the substation and for the wide area network. The user agent must be able to translate knowledge from the representation used by the multi-agent system into that used by the human machine interface. It must also be capable of carrying out tasks by using the directory services provided by the agent platform to locate appropriate agents to perform these tasks.

3.6.2 Agent Specification

- Knowledge
  - The agent should have a representation of the current state of the system, obtained at runtime from the other agents. It has no pre-programmed system knowledge.
  - If necessary, a set of mapping rules may be required to convert from the global system ontology to a representation used by the human-machine interface.
- Capabilities

  - Generate configuration files for mobile data analysis and remote control agents, and launch agents.
  - Display information on graphical user interface.

- Sensors
  - Input from graphical user interface.

- Interaction Protocols
  - FIPA subscribe (initiator)
  - FIPA request (initiator)
  - FIPA query (initiator)

- Ontological knowledge
  - FIPA Meta Ontology, *plant* class from plant ontology. If mapping rules are used to convert information for display on the HMI, the power systems ontology will be used in these mapping rules.

3.7 Summary

This chapter has described the static agents used in the architecture presented in Chapter 2, along with an analysis of some of the design decisions involved. The static agents used in the system are database agents, document agents, ontology agents, device agents, plant agents, and user interface agents. All agents are based around a common agent architecture and consist of sensors, effectors (or abilities), and a knowledge base. Agents participate in one or more of the FIPA standard protocols as described previously.

The database agents and document agents are similar in that they both provide access to data sources. However, the methods used to query these agents differ in that a database is a machine-understandable data source which can be translated into first-order logic, while it is not possible to do this with a document. This affects the implementation of these agents, as the document agent must include an information retrieval algorithm for document ranking, while the database agent has the ability to translate between FIPA SL and SQL to query the database.

The plant agents, device agents and node agents make up the data acquisition and control system. The plant agents are capable of controlling and monitoring a single item of plant. using the device agents, it is possible for a plant agent to do this without any knowledge of the control and monitoring hardware that this item of plant is connected to. Meanwhile, a device agent is responsible solely for a monitoring or control device, and has no knowledge of the plant. This separation of functions improves the modularity and flexibility of the system.

Design decisions involved in agent implementation include the representation of mapping rules used in the plant agents to convert from channel/value data to FIPA SL expressions. It was decided that these rules should be capable of representation in FIPA SL, in order that they could be stored in a database and transmitted to the plant agents when these agents were started, allowing dynamic agent configuration. However, it was not possible to query rules directly using first-order logic, and so a "mapping" frame was introduced in which the plant item to which the rule relates was specified, and the rule was provided as a quoted string. This allowed the plant agent to query mappings from the mapping database agent while maintaining compliance of the content of the query and reply with the FIPA specifications.

This chapter has considered the static agents present in the architecture. The next chapter will look at the use of mobile agents, which make up the remainder of the agents used. The chapter will examine whether these agents have the potential to improve the performance of some of the information management and control tasks that an automation system must perform.

Chapter 4

Applications of Mobile Agents

The previous chapters have described the design of the architecture, and the static agents that are used to perform the various tasks of a power system automation system. However, the performance of such a system may be affected by the slow network links between the WAN and substations, and from one substation to another. For example, many of the current data links to National Grid Company substations operate at 64Kbit/s or 128 Kbit/s[1]. From previous research, such as [51, 108, 53], it can be seen that mobile agents are capable of providing a significant performance increase in applications consisting of multiple interactions or large data transfer over a low bandwidth or high latency network. This chapter therefore evaluates the tasks of data analysis and remote control, to determine whether their performance can be improved by the use of mobile agents.

4.1 Mobile Agent Performance

The model of mobile agent performance used in this chapter is based on that of Straßer and Schwehm [53]. When considering the performance of a mobile agent application, a number of factors must be taken into account:

- The amount of time to transfer a mobile agent a from a source host src to

[1]This information is based on discussions with NGC engineers and visits to substations.

a destination host *dest*. We represent this by $T_{\text{transfer}}(a, src, dest)$. This time will depend on the speed and loading of the source and destination computers, the size of the mobile agent, the bandwidth and latency between the source and destination, and the mobile agent platform in use (different platforms employ different protocols for mobile agent transfer, some of which are more efficient than others).

- The amount of time to send a message m (in a message-based or client-server interaction) from a source host *src* to a destination host *dest*. We represent this by $T_{\text{msg}}(m, src, dest)$. This will depend on the speed and loading of the source and destination computers, the message size and the bandwidth and latency between the source and destination. It will also be affected by characteristics of the protocol, and possibly the programming language or other tools, used.

- The amount of time taken to process a set of data or a request at a given server s. This is represented by $T_{\text{proc}}(s)$. It will depend on the speed and loading of the server and the particular processing operation or task being undertaken.

- The amount of time taken by a server program on server s to retrieve a requested data set from the database and prepare it for transmission to the client agent. This is represented by $T_{\text{ret}}(s)$, and depends only on the data set and the characteristics (performance and load) of s.

Time to transfer a message or agent across the network

To obtain values for T_{msg} and T_{transfer}, the model defined by Straßer and Schwehm [53] may be used. In their paper, the time to perform a client-server interaction between two hosts $L_1$and L_2 (involving a request and a reply) is defined to be equal to twice the latency δ between the two hosts, added to the total amount of data B_{RPC} transferred divided by the bandwidth τ between the two hosts, added to an overhead equal to 2μ multiplied by the amount of data transferred. The constant μ is determined by the software and computers in use. Therefore:

$$\begin{aligned} T_{\mathrm{RPC}}(L_1, L_2, B_{\mathrm{req}}, B_{\mathrm{rep}}) &= 2\delta(L_1, L_2) \\ + & \left(\frac{1}{\tau(L_1,L_2)} + 2\mu\right) B_{\mathrm{RPC}}(L_1, L_2, B_{\mathrm{req}}, B_{\mathrm{rep}}) \end{aligned} \qquad (4.1.1)$$

where

$$B_{\mathrm{RPC}}(L_1, L_2, B_{\mathrm{req}}, B_{\mathrm{rep}}) = \begin{cases} 0 & \text{if } L_1 = L_2 \\ B_{\mathrm{req}} + B_{\mathrm{rep}} & \text{otherwise} \end{cases}$$

In this equation, $\tau(L_1, L_2)$ represents the bandwidth between L_1 and L_2, $\delta(L_1, L_2)$ represents the latency between L_1 and L_2 and μ represents the marshalling overhead, which depends only on the size of the object to be sent.

However, several assumptions made by this equation mean that it may not be suitable in all circumstances. The equation does not take account of the speed of the host computer, which affects the time to process a request and also the marshalling time. It can also be observed from results for Java object serialization (which is the form of marshalling used by Java RMI) in [109], that the time to marshal an object is not the same as the time to unmarshal that object.

In order to generalize the equation above, two functions are defined: $O_S(msg, src)$ to denote the sending overhead for message msg of size B_{msg} from host src, and $O_R(msg, dest)$ to denote the receiving overhead for message msg at host $dest$. These are functions of the message (size and possibly content type) and of the particular host. The transmission time uses the simple $time = \frac{size}{bandwidth} + latency$ approximation, as can be observed by omitting the marshalling time from Equation 4.1.1.

$$\begin{aligned} T_{\mathrm{msg}}(m, src, dest) &= \frac{B_{\mathrm{msg}}}{\tau(src, dest)} + \delta(src, dest) \\ &+ O_{\mathrm{S}}(msg, src) + O_{\mathrm{R}}(msg, dest) \end{aligned} \qquad (4.1.2)$$

The use of two overhead functions, one for sending and one for receiving messages, is proposed by Baldi and Picco [52]. However, they use "overhead" to mean the total time taken to send a message, while here the "overhead" represents only the cost of sending or receiving the message and does not include transmission time.

Time to Transfer a Mobile Agent

The actual time taken to transfer an agent over the network should be the same as for a message of the same size[2]. However, additional overheads are incurred by the time taken to register the agent on the destination platform and to restart its execution, etc. If these overheads are denoted by $O_{\text{reg}}(agent, server)$ and $O_{\text{dereg}}(agent, server)$, the resulting migration time equation is:

$$\begin{aligned} T_{\text{transfer}}(a, src, dest) &= O_{\text{reg}}(a, src) \\ &+ O_{\text{dereg}}(a, dest) \\ &+ O_{\text{S}}(a, src) \\ &+ O_{\text{R}}(a, dest) \\ &+ \frac{B_a}{\tau(src, dest)} + \delta(src, dest) \end{aligned} \tag{4.1.3}$$

In this equation, B_a represents the size of the agent, including code, data and state. As for the message, the main problem with this model is that different types of agent content may take differing amounts of time to transfer.

4.2 Mobile Agent for Data Analysis

In the architecture described in this thesis, data stored in a substation database is accessed using a multi-agent system. This data is made available via a database agent, which can be queried using FIPA ACL[72] messages to

[2]In practice this may not be the case if, for example, additional messages must be sent between the agent platforms to co-ordinate the movement of the agent.

retrieve data. However, problems occur when it is necessary to analyse a large quantity of data, such as monitoring data covering a long period of time. This data can be extremely large, especially when converted into a text-based format, and the connections to substations are often slow (sometimes only dial-up links are available). Therefore, mobile agents are to be employed to analyse the data in the substation and remove the need to transmit it across the network. A subset of the generic architecture described in this thesis, consisting of two database agents per substation, an ontology agent, a user agent and the mobile agent, is used. This is shown in Figure 4.1. The monitoring database contains event data, and the static database contains substation topology information.

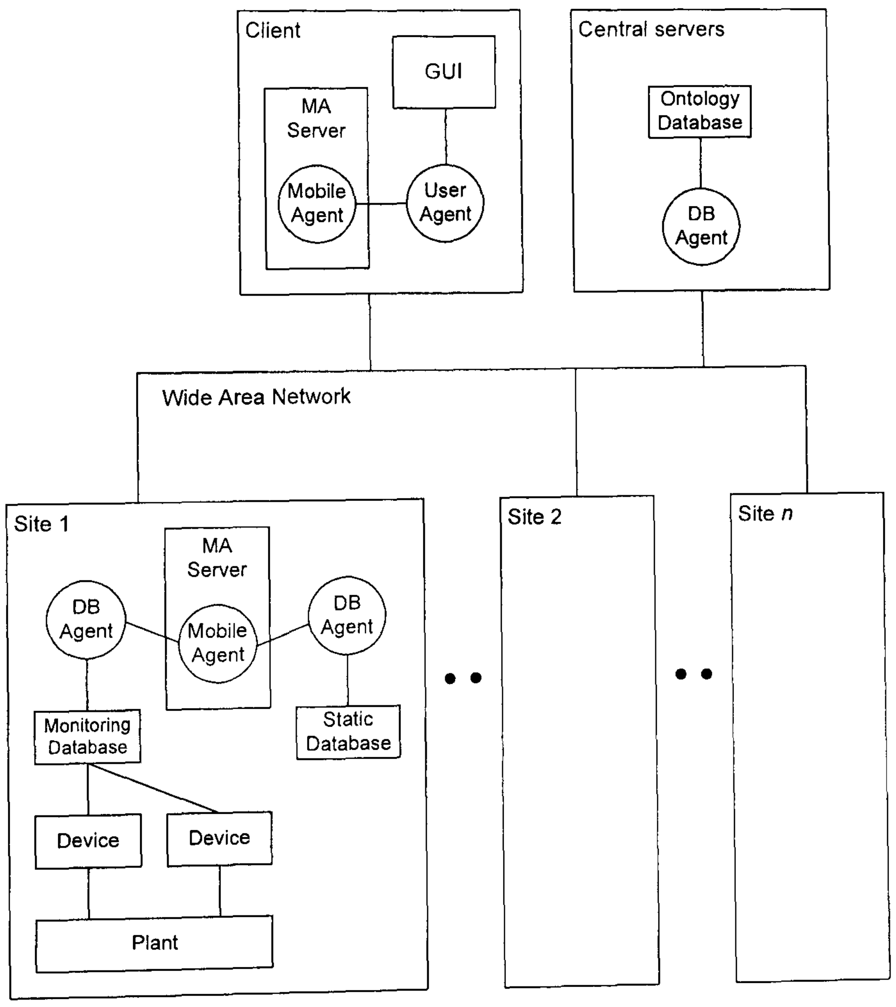


Figure 4.1: Subset of Generic Architecture used for Mobile Agent Based Data Analysis

The basic outline of this application, as implemented in the prototype described in Chapter 5, is as follows:

1. The mobile agent is launched by the user agent and provided with a configuration file, consisting of a series of *data sets* that must be retrieved, a set of *analysis functions* that the agent must perfom on those data sets, and a *report template*, specifying how the results produced by the analysis functions will be displayed in the generated report. Each data set consists of an object (e.g. a transformer), a property (e.g. LV current) and a time range (start time and end time). The process used by the user agent to generate the configuration file is described in Chapter 5.

2. The mobile agent gathers the specified data sets, and performs the required analysis, generating its report. For the multi-hop case, the implemented agent uses a basic planning algorithm to generate its route and the order in which its tasks will be performed.

3. The report is displayed to the user as an HTML page.

It is possible to create both multi-hop and single-hop data analysis agents. However, the single-hop agent is much simpler, as the agent only has to travel to a single location and perform a single analysis, and does not have to perform planning or route optimization.

In the multi-hop case, the agent must analyse monitoring data from a number of substations. Because the data links between substations can be slow, it is desirable that the agent should transfer as little data as possible. Therefore, unless data from two substations must be used as a combined data set, the agent should perform all analysis on a substation's data, and retain only the result, before moving on to the next substation. It is desirable that the agent would be able to generate an optimal (or near-optimal) route through the substations to ensure that the tasks are completed in the shortest amount of time. Variations on this problem have been investigated in a number of papers, including an algorithm which attempts to optimize the number of agents used as well as the total time taken [110]. Moizumi analysed a similar problem, the "travelling agent problem (TAP)" in which an agent must perform a task involving a number of servers [111]. However, in the TAP, the task may be completed without visiting all servers, whereas in the application described here it is necessary for all relevant servers to be visited. Further work should investigate whether these algorithms can be applied to the agent described here.

4.2.1 Agent Algorithms and Implementation

Simple, Single-Hop Analysis Agent

The single-hop data analysis agent can be implemented using a relatively simple algorithm, such as Algorithm 1. The agent travels to the location of the server providing the data set, retrieves the data, performs its analysis and sends the report to the user. The names of items of plant and properties which can be analysed can be obtained by querying the static and ontology databases respectively.

Some of the procedures used in this algorithm are dependent on the agent platform or on a particular implementation and are therefore not described here in detail. The `send` procedure delivers a message to a given agent. The `extract-results` procedure extracts a table of results from the reply to a query. The `location-of` procedure returns the agent platform, or container, on which a particular agent is located. The `kill-self` procedure terminates the agent's execution, freeing system resources. The `generate-report` procedure takes a set of results and a template, and generates a report.

Multi-Hop Agent Without Optimization

The multi-hop agent performs a considerably more complex task than the single-hop agent. As well as retrieving and analysing data, it must be able to generate a plan (or sequence of actions) that ensures that all of the required tasks are completed and attempts to minimize the execution time. For example, if an agent must perform two tasks in substation 1 and one in substation 2, it is normally desirable (assuming that these are independent tasks) to perform either both tasks in substation 1 followed by the single task in substation 2, or

Algorithm 1 Single Hop Analysis Agent

```
Inputs:
    s: server agent
    e: expression defining data set
    template: report template
    F: set of analysis functions
begin
    move(location-of(s))
    results := collect-data(e,s)
    report := generate-report
                (results, template, F)
    display-report(report)
    kill-self()
end

procedure collect-data(e,s)
begin
    message := create-query-message(e)
    send(message,s)
    reply := wait-for-reply(message)
    results := extract-results(reply)
    return results
end
```

the task in substation 2 followed by the tasks in substation 1. Compared to performing one of the tasks in substation 1, followed by the task in substation 2 and then the second task in substation 1, the number of "hops" that the mobile agent must make across the network is less. This problem has been analysed by Xie in [112], who proposed a scheduling algorithm for mobile agent planning based on Distributed Acyclic Graph (DAG) scheduling. However, this algorithm relies on the use of multiple mobile agents, and therefore is not used in the agent described here (a single agent).

In the case of the data analysis agent, there are four types of task which it may carry out: *retrieval*, which involves gathering a single data set from a server, *analysis*, which involves the use of a data set to generate either another data set or a result, *report generation,* which involves the generation of the report for the user, and *display*, which displays the generated report on the screen.

When the multi-hop user agent reads the configuration file, it first generates a task list from the data sets and analysis results defined in the configuration file. This task list will always contain one report generation task and one display task. For each data set a retrieval task is generated, and for each analysis an analysis task is generated. Following Xie's DAG representation, for each task a set of *predecessor* tasks (those which must be completed in order for the task to be performed) and a set of *successor* tasks (those which may only be started once the task is complete) are defined. In the mobile analysis agent, retrieval tasks have no predecessors. An analysis task has as its predecessors all those tasks (either retrieval or analysis tasks) providing input to the analysis. The report generation task can only be performed once all analysis tasks are complete, and the display task can only be performed once the report has been generated.

Once a plan has been generated, the mobile agent will then execute each step of the plan in order. The algorithm of the multi-hop mobile agent is given as Algorithm 2. Each task is assigned a location by the agent, and the *location_of* procedure retrieves the location of a specified task. In the basic agent described here, the location of a retrieval task is always the agent container on which the server agent for that retrieval is located (the agent will always move). The agent will carry out an analysis or report generation task at its current location, and will return home before displaying the report. The basic planning algorithm used in the prototype implementation (not shown) "clusters" tasks so that as soon as a data retrieval task is complete, all possible successor tasks (normally analysis tasks) will be performed. However, it does not currently cluster data retrieval tasks by substation - this optimization will be implemented later.

Algorithm 2 Multi-Hop Analysis Agent

```
Inputs:
    f: configuration file
begin
    config := read_configuration_file(f)
    tasks := generate_task_list(config)
    plan := generate_plan(tasks)

    while not complete(plan) do
        t := first_incomplete_task(plan)
        move(location_of(t))
        execute(t)
    wend
end
```

4.2.2 Data Analysis Agent Performance

Referring to the generic mobile agent performance model described previously, it is assumed assume that T_{msg} is the same for both the mobile agent and client-server cases. The time taken for the data to be retrieved is approximated by $T_{\mathrm{ret}}(S)$, and the time taken for the data to be analysed by a function $T_{\mathrm{proc}}(S)$, where in both cases S represents the server on which the activity takes place. Both T_{proc} and T_{ret} are dependent on the speed and loading of the computers on which the retrieval and analysis are performed, and on the amount of data to be retrieved. However, in order to simplify this analysis it is assumed that both computers are of the same speed and are unloaded, and hence T_{proc} and T_{ret} are the same in both mobile agent and client server cases. The messages used in this interaction are represented by *req* (request for data) and *rep* (reply containing data).

When a mobile agent is used, the time taken to analyse a single database is equal to the time taken to send the mobile agent, added to the time taken to retrieve and analyse the data, added to the time taken to return the report to the user.

$$\begin{aligned} T_{\mathrm{MA}} &= T_{\mathrm{transfer}}(a, client, server) \\ &+ T_{\mathrm{ret}}(server) \\ &+ T_{\mathrm{proc}}(server) \\ &+ T_{\mathrm{msg}}(report, server, client) \end{aligned} \tag{4.2.1}$$

When the client-server system is used, retrieval is performed at the server, but processing is performed at the client. Therefore:

$$\begin{aligned} T_{\mathrm{CS}} &= T_{\mathrm{msg}}(req, client, server) \\ &+ T_{\mathrm{ret}}(server) \\ &+ T_{\mathrm{msg}}(rep, server, client) \\ &+ T_{\mathrm{proc}}(client) \end{aligned} \tag{4.2.2}$$

However, if $T_{\mathrm{proc}}(server) = T_{\mathrm{proc}}(client)$, then the difference between the two methods may be expressed as:

$$\begin{aligned} T_{\mathrm{CS}} - T_{\mathrm{MA}} &= T_{\mathrm{msg}}(req, client, server) \\ &+ T_{\mathrm{msg}}(rep, server, client) \\ &- T_{\mathrm{transfer}}(a, client, server) \\ &- T_{\mathrm{msg}}(report, server, client) \end{aligned} \tag{4.2.3}$$

This will be positive if the mobile agent method is faster, and negative if the client server method is faster. Assuming that T_{msg} and T_{transfer} depend on the size of the message/agent and the bandwidth and latency of the network, whether the client-server method is faster than the mobile agent method depends on the relative sizes of the agent, the request and reply messages and the report, and on the various overheads involved in agent transfer.

4.2.3 Benchmarks

A benchmark evaluation of the single-hop data analysis agent was performed. A data analysis agent was created, which retrieved a data table consisting of two columns (value and time) from a database and performed the following analyses:

1. Calculate the mean value of one column of data (single iteration through data retrieved)
2. Calculate the maximum value of one column of data (single iteration through data retrieved)
3. Plot a graph (two iterations through data retrieved).

Using a configuration parameter of the agent, it could be set to operate either as a static or mobile agent. This enabled a comparison to be performed. A second comparison was also performed between the use of wrapper agents, located at the server, to retrieve data, and an implementation which was granted direct access to the database, and used Java Database Connectivity (JDBC) for database access. In the first case, the database agent used the ACL message from the mobile agent to query the database, returning the results as an ACL message with FIPA Semantic Language (SL) content. In the second, to maintain the independence of the mobile agent from the data source, the agent sent an ACL request to the wrapper agent, which provided it with database connection details and the SQL query that could be used to retrieve the data. (For this experiment, the query and connection information were hard-coded into the database agent's configuration, as a full translation procedure to generate SQL, rather than directly query the database, has not yet been implemented).

The client and server computers were connected to separate Ethernet switches and a computer running the Dummynet program[113] was used as a bridge to control the bandwidth and latency between client and server, as shown in Figure 4.2. A number of different settings of bandwidth and latency were used. The time taken for the agent to complete its analysis and display the results to the user was measured. This was repeated six times for each bandwidth/latency combination, three times in the mobile agent case and three times in the client-server case. While the use of Dummynet means that the network delay is deterministic, it is considered that this is appropriate for a power company network in which the wide area network consists of private wire circuits, which do not suffer from the random delays due to high traffic as exhibited by the Internet.

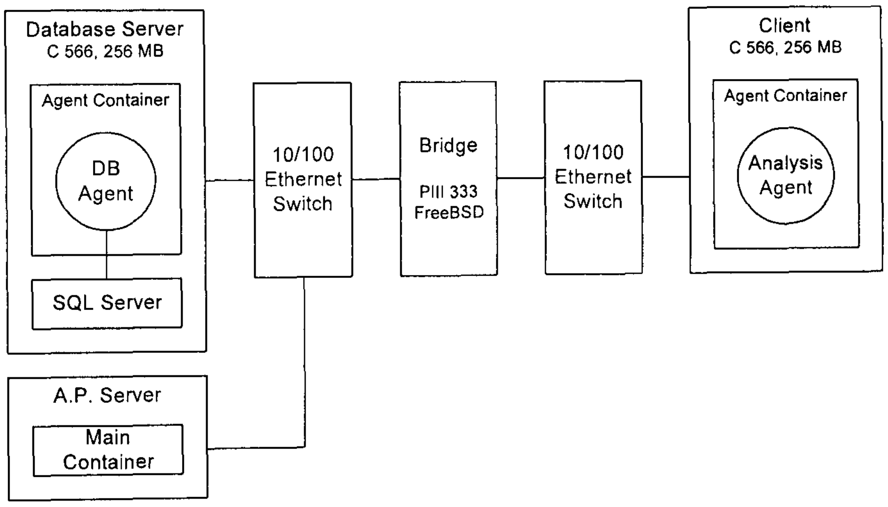


Figure 4.2: Experimental Setup

Based on the performance model discussed in Section 4.2.2, it is expected that the mobile agent-based data retrieval agent will perform significantly better than the client-server agent. This performance advantage should increase with decreasing bandwidth and with increasing latency.

Results

Rather than taking the mean of the three measurements, it was decided to use the first measurement for each set of latency/bandwidth values, as this includes the cost (in the mobile agent case) of transferring the agent's class files, which is not included in later measurements as these files are already transferred. Table 4.1 shows the fixed parameters for the experiment (Object sizes were obtained by serializing the object into a byte array and then recording the length of the array).

Parameter	Value
Number of rows of data retrieved	18,915
Number of columns of data retrieved	2
Object size of wrapper-based mobile agent (outgoing)(B_{data})	7963 bytes
Object size of wrapper-based mobile agent (return)	8183 bytes
Report size of wrapper-based mobile agent (return)	4268 bytes
Size of reply message from wrapper (containing data) (B_{rep})	1053525 bytes
Size of request message (wrapper-based case) (B_{req})	2249 bytes
Total size of wrapper-based agent class files (B_{code})	42950 bytes

Table 4.1: Fixed Parameters for Data Retrieval Experiment

Tables 4.2, 4.3 and 4.4 give the first run times in seconds for each of the six agents, for bandwidth from 100Kbit/s to 100Mbit/s, and one-way latency between 0 and 50 milliseconds. All times are to 3 significant figures. Full results for these experiments are given in Appendix A.

Figures 4.3 and 4.4 show the results obtained by varying the one-way latency of the connection from 0 to 50 milliseconds, while leaving the bandwidth unchanged at 10Mbit/s. Figure 4.4 shows in more detail the results from the four faster scenarios, which are obscured on Figure 4.3 by the long times taken by the static agents with direct database access.

Figure 4.5 shows the effect of bandwidth on the experimental database analysis, using all of the different agents and with no added latency.

τ(bit/s)	1-way latency (ms)			1-way latency(ms)		
	0	25	50	0	25	50
100M	109	105	107	33.4	35.9	38.1
10M	102	104	104	33.1	36.1	38.2
1M	103	105	108	34.1	36.2	38.6
100K	110	112	114	41.8	44.3	46.0
	No caching			Caching		

Table 4.2: Time Taken (secs) By Mobile Direct Access Agents (times to 3 sf)

τ(bit/s)	1-way latency(ms)			1-way latency(ms)		
	0	25	50	0	25	50
100M	96.7	3802	7590	37.6	958	1910
10M	103	3820	7610	32.6	961	1910
1M	158	3940	7730	46.0	992	1940
100K	1390	5180	8970	354	1300	2250
	No caching			Caching		

Table 4.3: Time Taken (secs) By Static Direct Access Agents (times to 3 sf)

τ(bit/s)	1-way latency(ms)			1-way latency(ms)		
	0	25	50	0	25	50
100M	113	113	117	110	116	123
10M	112	113	117	111	119	122
1M	115	114	116	118	123	126
100K	119	120	123	202	203	208
	Mobile			Static		

Table 4.4: Time Taken (secs) By Wrapper-Based Agents

Analysis

The experiment described here is relatively limited. However, there is a significant body of existing work in the field of mobile agent performance, some of which is discussed in Section 4.2.4. Therefore, the intention of the experiment described here is to confirm that the advantages found by existing work are applicable in the context of the power system automation architecture

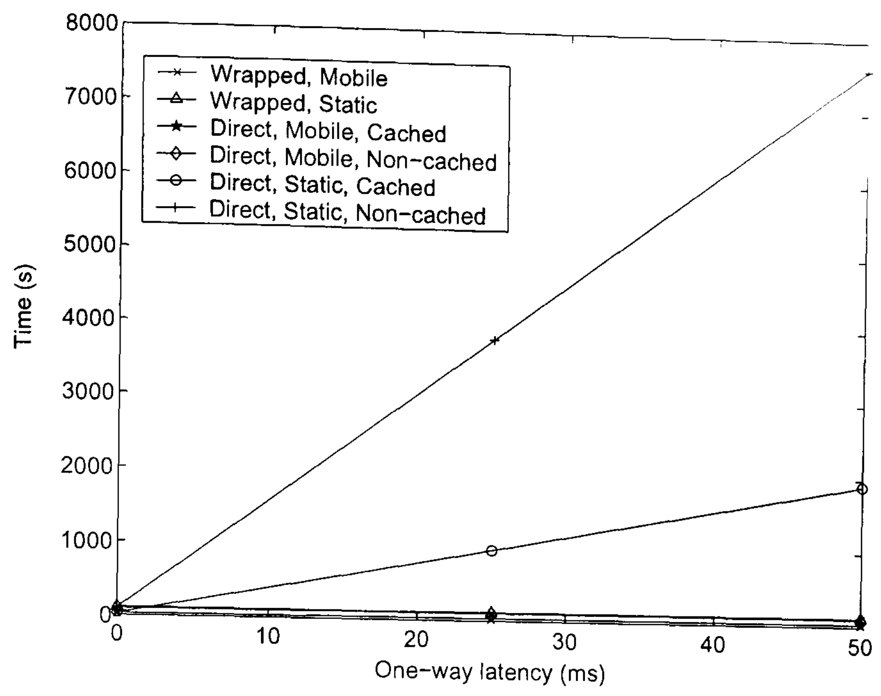


Figure 4.3: All Agents at 10Mbit/s

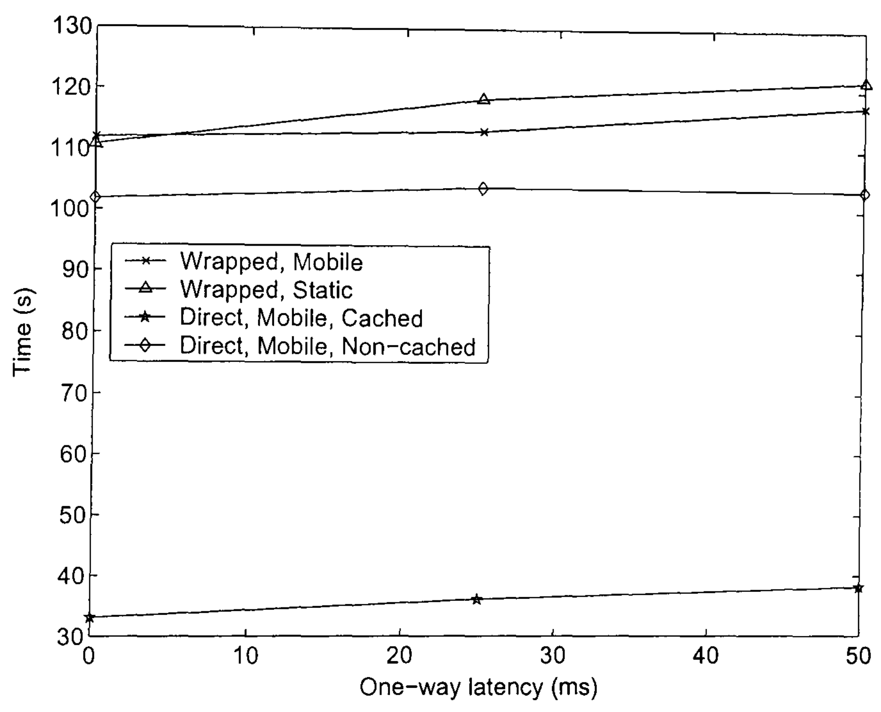


Figure 4.4: Agents at 10Mbit/s with Static Direct Access Agents Removed

developed in this thesis, and to demonstrate that via a specific application used in the prototype. The work here does not attempt to develop a complete model of mobile agent performance.

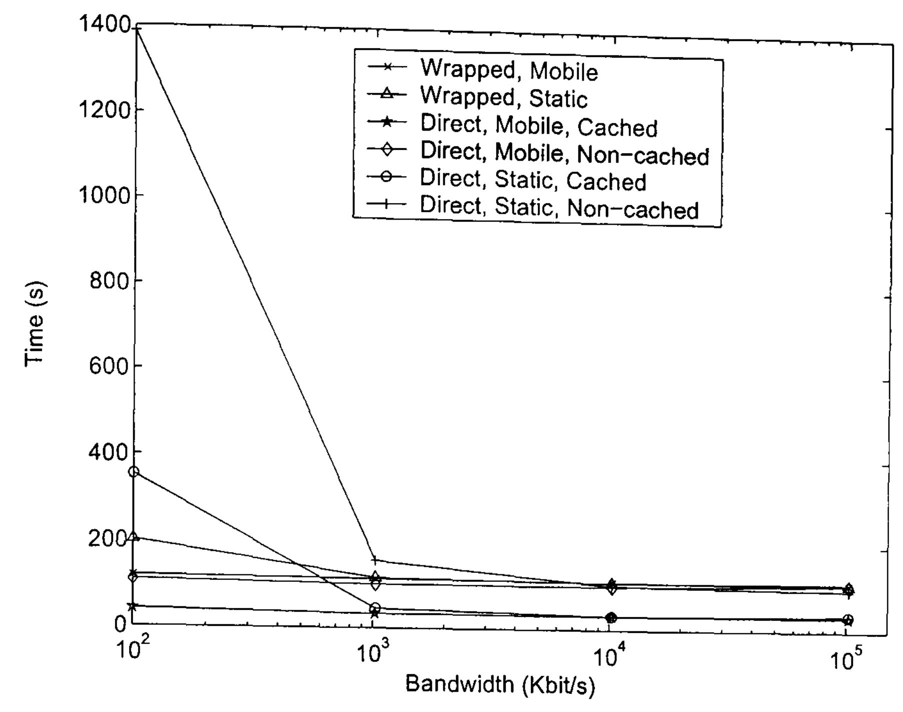


Figure 4.5: Time Taken to Analyse Database at Various Bandwidths, no Added Delay

Effect of Latency

Using a linear approximation to the data shown on Figure 4.3, the equations shown in Table 4.5 are obtained for the time, t, in seconds taken by the agent to perform data analysis at 10 Mbit/s bandwidth, with respect to the latency, l, in seconds of the connection.

Agent	Equation of line (to 3 sf)
Wrapped, mobile	$t = 110l + 111$
Wrapped, static	$t = 219l + 112$
Direct, Mobile, Cached	$t = 101l + 33.3$
Direct, Mobile, Non-cached	$t = 33.4l + 102$
Direct, Static, Cached	$t = 37.6 \times 10^3 l + 28.8$
Direct, Static, Non-cached	$t = 150 \times 10^3 l + 90$

Table 4.5: Linear Approximations to Time Taken

A number of points can be observed from these approximations:

- Both wrapper-based agents had approximately the same fixed term.

- Of the direct access agents, the mobile and static cached agents had similar fixed terms, as did the mobile and static non-cached agents.
- The direct access static agents are affected much more by latency than any of the other agents.
- Neither wrapper-based agent is affected greatly by latency.
- The gradient of the static, cached line is approximately equal to 18915 (number of rows) multiplied by 2. This might suggest that for each row, a request and reply message were sent, meaning that the time to retrieve the data would be approximately equal to the number of rows multiplied by the latency, added to the fixed processing cost. At 10Mbit/s, the effect of bandwidth limitations would be negligible as the amount of data for each row would be very small. However, more experiments are required to establish whether or not this is the case.
- The time taken by the direct access, static, cached agent is between 3 and 4 times the time taken by the direct, static, uncached agent. It is likely that this is because the data was iterated 4 times. Therefore, 4 times as much data was transferred in the uncached case. However, some of the time taken was due to processing rather than data transfer, so the ratio is less than 4.

Referring to Equations 4.2.1 and 4.2.2, the majority of the fixed term appears to be made up of the time to retrieve and process the data ($T_{\text{ret}} + T_{\text{proc}}$), as there is only a minimal difference between the fixed term for the mobile and static agents. This difference, if not an experimental error, might represent the setup and movement overheads of the mobile agent, or the fact that having the agent on the same server as the database increases the server load.

Effect of Bandwidth

Equations 4.2.1 and 4.2.2 may be applied to the wrapper-based agents only, as the direct access agents access the database using a large number of client-server messages, and the model was based on a single request and a single reply. The equations predict that the time taken should be inversely proportional to the bandwidth and proportional to the amount of data transferred. However, while Figure 4.5 does show that the time taken increases as the bandwidth decreases in all cases, and that this effect is much greater for the static agents (due to the greater amount of data transferred), there is insufficient data available to confirm or refute the proportionality assumption.

Discussion

The results show that the performance of the client-server data analysis application is highly dependent on the bandwidth of the network, while that of the mobile agent is relatively unaffected. Therefore, by using a mobile agent it is possible to greatly improve the benchmark performance in low bandwidth cases. These results agree with the predictions made by the model of this application in Section 4.2.2. Equation 4.2.3 states that the difference between the time taken by a client-server program to perform the task and the time taken by a mobile agent is equal to the difference between the amount of time taken for the client-server program to send and receive the query and reply messages and the amount of time to transfer the agent. Because the reply message is much larger than the agent, as the bandwidth increases the time taken by the client-server program should increase more quickly than the time taken by the mobile agent program. This agrees with the results of the benchmark. However, because of the significant overheads (platform overheads and data retrieval time) involved in the benchmark, the actual time taken does not match that predicted by the equations, which do not include the data retrieval time, as discussed previously.

When using an ACL-based data analysis agent, because only a small number of messages are sent, performance is not affected as much by high latencies as by low bandwidth, at least for realistic latencies (a typical "ping" time when using a dial-up modem is around 200 milliseconds, which gives 100 milliseconds one-way latency). As only a small number of messages (two in the optimal case) are sent and received, latency should only slightly affects the agent performance. The fact that altering the latency produces a greater change in time taken than this suggests that a greater number of messages are sent across the network when transmitting a mobile agent or an ACL message.

When using direct database access, the performance of the client-server agent was heavily affected by latency. This might indicate that the agent had to send a request for each row of data. This lack of optimization of the client-server program may exaggerate the performance advantage of using a mobile agent in this application. By retrieving more data in each call, it might be possible to improve the performance of the direct access static agent to more closely match that of the wrapper-based static agent. However, this would mean that it would still be slower than the mobile agent in many cases.

It would be possible to equal the performance of the mobile agent for data analysis in a client-server system by building a server containing all of the analysis functions, and calling these functions remotely. However, as discussed in [54], this would greatly reduce the flexibility of the system, and make it more difficult to add new analysis functions as this would require the server program to be modified and restarted.

If the task is modified, this may alter the results of this benchmark. For example, changing the size of data retrieved will affect both the mobile agent and client-server systems. However, following the results of Johansen [114], mobile agents would be expected to perform better in relation to client-server systems as the data size increased, and vice versa. Changing the number of iterations required through the retrieved data would particularly affect the non-caching agents, as they must retrieve all of the data on each iteration. The effect on the caching agents would be limited.

4.2.4 Related Work

Johansen [114] demonstrated the performance advantages of mobile agents for the analysis of large data sets, using a mobile agent to retrieve weather satellite data. Johansen provides a comparison of the performance of client server and mobile code for different data sizes, which shows a number of performance improvements from using mobile agents. Johansen's work provided some of the motivation for the work described here, as it showed that it was possible to improve the performance of a data analysis application using mobile agents. The work in this thesis extends Johansen's work by applying mobile agent data analysis in the context of the specific automation systems architecture developed in Chapters 2 and 3, and by providing benchmarks at different bandwidths and latencies.

Tsukui *et al* [115] used mobile agents to retrieve data from power system protection devices. As in our work, they utilised a mobile agent for analysis. In their application it was used to analyse fault records from devices. They also used mobile agents to gather status information from devices and to alter device settings. However, their work does not include a detailed analysis of application performance, or describe the agent implementation in detail.

Gray *et al* [51] describe two applications of mobile agents, one of which is an information retrieval application. They test the scalability of a mobile agent system for performing document retrieval and filtering as the number of clients increases, and find that client-server solutions perform poorly with large numbers of clients because the network becomes overloaded. In contrast, mobile agent solutions may perform poorly if a server machine becomes overloaded. Therefore, they conclude that if the bandwidth is high it is usually preferable to use a client-server system, and when the bandwidth is low or the number of clients is high it is usually preferable to use a mobile agent system. These conclusions are similar to the ones obtained from the experiment described here. However, the experiment that they performed used multiple mobile agents and was for document retrieval, whereas the experiment described here is for a single mobile agent performing database analysis. Therefore, the additional results provided here are useful to confirm that the conclusions drawn by Gray *et al* are still valid for this application.

4.2.5 Conclusions

This section has describe the use of mobile agents for performing data analysis tasks within the architecture described in this thesis. A performance model has been derived to allow the quantitative analysis of this mobile agent application. Performance benchmarks have shown that where the bandwidth of the wide area network is low or its latency is high, the use of mobile agents can provide significant performance improvements compared to a client-server or static agent implementation. Results have also demonstrated that it is possible to improve the performance of the mobile agent by providing direct, rather than wrapper-based, access to a database and by caching the data retrieved. However, these results may relate only to the specific wrapper-based implementation used here. It may be possible to improve the efficiency of the wrapper-based agent using an alternative implementation method, although it is likely that, because the wrapper-based implementation would still involve packaging the retrieved data as ACL and then parsing the ACL data, the direct access agent would still have a performance advantage.

When building the system it was desired to adhere as closely as possible to the FIPA standards for multiagent systems. This motivated the choice of the JADE platform for system implementation, as at the time it was the only publically available FIPA platform to support mobile agents. However, JADE's focus is not on mobile agents, and so it lacks some of the mobile agent related features of other systems (e.g. strong mobility, support for multiple versions of a Java class in the same container, inter-platform mobility). For the benchmark, this was irrelevant as intra-platform mobility could be used to simulate inter-platform mobility, and the versioning problem was not encountered as only one version of the benchmark agent was used at a time. For a full deployment of the system it might be necessary to extend the platform or to re-examine the available alternatives.

Further work on this system should involve the full evaluation of multi-hop data analysis agents. In addition to this, it is also intended to add document retrieval functionality.

4.3 Mobile Agent for Remote Control of Power Systems

Many substations in an electricity network are unmanned, and must therefore be controlled remotely. Currently this is normally done from a control centre, using dedicated network links to the substation. Here we investigate the possibility of using mobile agents as a control mechanism to allow users to remotely control the substation plant over a standard IP network.

For example, suppose that a substation contains a transformer, with a circuit breaker and earth breaker connected to the high voltage windings, and a circuit breaker and earth breaker connected to the low voltage windings. This partial substation layout is shown in Figure 4.6. The substation provides both a mobile agent server and a client-server interface which allow commands to be sent to substation devices. A user at a remote site intends that the following actions should be performed in the substation, as illustrated by the numbers on Figure 4.6:

- Open two circuit breakers to isolate the transformer (1 and 2)
- Measure the voltage across the transformer to ensure that it has been isolated (3)
- If the transformer is isolated, operate two earth switches (one on either side of the transformer) to earth it (4 and 5)
- When all of these operations are completed, display a message on the user's machine.

This could be achieved in one of two ways. Firstly, it would be possible to use the client-server interface to execute all of the commands remotely. Suppose that the protocol is simple, and for each command only a single command message and a reply or acknowledgment message are required. In this case, 10 messages across the wide area network would be necessary (two for each of the two circuit breakers, two for the voltage measurement and two for each

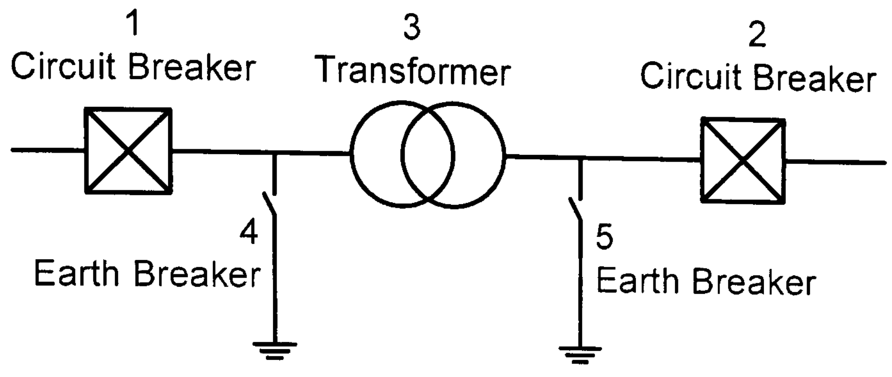


Figure 4.6: Example Substation

of the two earth switches). Alternatively, a mobile agent could travel to the substation and execute the same commands locally over the substation local area network. The 10 messages required in the client-server case would then be sent and received over the LAN. Only two messages would need to be sent across the WAN: the mobile agent and the final message informing the user that the sequence of actions was completed. Depending on the characteristics of the network involved, this might mean that the operation would complete more quickly if the mobile agent was used.

The major anticipated drawbacks to implementing this application are in the areas of security and reliability. Much has been written about the security concerns of mobile agents (e.g. [58]). However, these security problems mostly apply to open systems, in which it is possible for anyone to send a mobile agent to a server, and the servers are operated by different authorities. In a power system, which is relatively closed, these problems could, at least to a certain extent, be avoided. Also, the advantages of mobile agents for remote control might apply equally to less critical applications such as remote configuration management, in which, instead of instructing devices to operate substation plant, the mobile agent was used to alter device configurations. The performance model for this application is the same as that for remote control, as the agent is still performing a sequence of interactions with a device or devices, except that the interactions result in changes to the device configuration rather than changes to the status of the substation plant.

There is another problem when an agent is allowed to interact directly with IEDs. If multiple mobile agents are permitted to be present in the substation simultaneously, particularly when they are controlled by different users, it is possible that the commands transmitted by these agents might be inconsistent with one another. Also, it is not possible simply to "lock" individual items of plant being controlled, as commands executed on one part of the substation can affect other parts of the substation. For example, suppose a substation has two transformers. At the same time, user A sends a command to isolate transformer 1, and user B sends a command to isolate transformer 2. The result of this is that both transformers are isolated, and the substation then provides no output current.

It is proposed to solve this problem by using the plant agents as an intermediate layer between mobile agents and IEDs. When a mobile agent wishes to perform an action, it must request that the appropriate plant agent perform this action. By negotiation with other plant agents, that agent must then determine whether or not that action is possible, given any constraints which have been placed on the actions of the plant agents by other mobile agents or by human controllers. One possibility would be to adopt the Joint Intentions methodology, developed by Jennings for the ARCHON industrial control multi-agent system[116]. Alternatives might include the use of a single, static, "supervisor" agent responsible for co-ordinating the actions of all agents in the substation, or the use of a constraint programming [117][118] method.

4.3.1 Agent Algorithms and Implementation

For Experiment 2 (Section 4.3.4), a simple (non-intelligent) mobile agent based on Algorithm 3 was implemented. The agent is given the address of a server agent (located at the closest agent server to the device), the URL of a relay device and a series of actions (either open or close) to be performed. When the agent starts, it will move to the location of the server agent and carry out the sequence of interactions with the device. The agent is capable of interacting with one device only. The preliminary benchmark used a similar algorithm, but without the server agent (the address of the destination agent platform was used instead).

Algorithm 3 Mobile Remote Control Agent

```
Inputs:
    s: name of server agent
    A: set of actions to perform
    d: device
    h: home location
begin
    l = locate(s)
    move(l)
    for each a in A do
        send-request(a, d)
        wait-reply(a, d)
    next a
    move(h)
end
```

Algorithm 4 shows a multi-hop agent, which acts through other agents, and uses the DF to determine which agents are capable of carrying out a particular action in its specified action sequence. The algorithm used in this case is as follows (in which the *location-of* procedure should return the substation in which either the item of plant affected by an action or a particular server agent is located):

The agent is given an ordered sequence of actions, which are represented by FIPA SL action expressions, giving the agent itself as the actor. For each of these actions, the agent first determines whether or not it is capable of performing the action itself. If it is, it will perform the action by directly interacting with the relevant IED. If not, the agent will attempt to perform the action using the capabilities of other agents. It first searches the DF, to find a list of agents (the variable *Servers* in Algorithm 4) capable of performing

Algorithm 4 Improved Remote Control Agent

```
Inputs:
    A: sequence of actions to perform
    h: home location
begin
    for each a in A do
        if (capable-of(a)) then
            move(location-of(a))
            perform(a)
        else
            Servers = find-servers(a)
            for each s in Servers do
                move(location-of(s))
                request(s, a)
                if(success) then break
            next s
            if not complete(a) then fail
        endif
    next a
end
```

the relevant action. It will then request the action from each of these agents in turn, stopping when the action is complete.

In a realistic implementation, it would also be necessary to provide the agent with the ability to handle exceptional cases such as the failure of one of its actions (denoted in Algorithm 4 by `fail`).

4.3.2 Remote Control Agent Performance

In order to determine the conditions under which it is appropriate to use a mobile agent for substation control, a performance analysis is required. This will be carried out for the single-hop agent, which carries out interactions in a single substation only. The agent is "launched" from a client computer and travels to a substation, where it interacts with one or more IEDs, either by moving to the IEDs themselves or by sending messages across the substation local area network, before transmitting a message to the client indicating its success or failure, and any additional information specified, such as the new state of the substation.

It is assumed that the mobile agent interacts with the IEDs over the substation LAN. Therefore, it makes only one hop, from the client (denoted as c) to the substation's mobile agent server (denoted by s). To simplify the model further, it is also assumed that the bandwidth and latency between all points on the LAN are fixed and uniform, and that the bandwidth and latency between the client computer and the substation are fixed. Finally, it is assumed that each interaction requires only a single request message and a single reply message, that only one message is required to transfer the mobile agent, and that only one message is required to transfer the results. This is not the case in some "real-world" protocols, and in fact would be unreliable as the receipt of the mobile agent and results are not acknowledged, but represents the most efficient possible scenario.

Data size The amount of data sent across the network for the j^{th} interaction is then:

$$B_j = B_{\mathrm{req}_j} + B_{\mathrm{rep}_j}$$

Suppose that the mobile agent must perform n_i interactions with each of a set of d devices $D_0 \ldots D_d$, where D_i represents the i^{th}device. Let n then represent the total number of interactions.

The total amount of data transmitted across the network is then:

$$\begin{aligned} B_{\mathrm{interactions}} &= \sum_{j=0}^{n} B_j \\ &= \sum_{j=0}^{n} B_{\mathrm{req}_j} + B_{\mathrm{rep}_j} \end{aligned}$$

In a client server case, $B_{\text{interactions}}$ represents the total amount of data sent over the network. In the mobile agent case, the total amount of data sent over the network is then equal to the size of the mobile agent B_{MA}, which consists of the size of the agent's object along with any code that must be transferred, added to the size of the interaction messages and to the size (B_{results}) of the final acknowledgment message sent to the client when all interactions are complete.

$$\begin{aligned} B_{\text{MA}} &= B_{\text{a}} + B_{\text{interactions}} + B_{\text{results}} \\ B_{\text{CS}} &= B_{\text{interactions}} \end{aligned}$$

However, the mobile agent and results message are sent over the WAN, while the interaction messages are passed across the substation LAN. If we only consider the amount of data sent over the WAN, in the mobile agent case the data transferred is $B_{\text{a}} + B_{\text{results}}$, while in the client-server case the amount of data transferred is $B_{\text{interactions}}$.

Time taken The amount of time taken to perform all of the interactions depends on the data size, along with the bandwidth and latency of the network. A simple model which took into account these factors was used in [119]. However, in order to get a realistic time, it is necessary to include the time taken for the device to carry out the command or to respond to a request for information. This will be denoted, for the j^{th} interaction, by the processing time t_{proc_j}. This time will depend on the nature of the command, along with the processing speed of the device or response time of the plant to which it is attached. However, it should be the same regardless of whether the client-server or mobile agent interaction method is used.

Using the client-server method, and denoting the total time taken by T_{CS}:

$$T_{\text{CS}} = \sum_{j=0}^{n} (T_{\text{msg}}(req_j, c, D_j) + T_{\text{msg}}(rep_j, c, D_j) + T_{\text{proc}_j}) \qquad (4.3.1)$$

Using the mobile agent method, and denoting the total time taken by T_{MA}:

$$\begin{aligned} T_{\mathrm{MA}} &= \sum_{j=0}^{n}(T_{\mathrm{msg}}(req_j, s, D_j) + T_{\mathrm{msg}}(rep_j, s, D_j) + T_{\mathrm{proc}_j}) \qquad (4.3.2) \\ &+ T_{\mathrm{transfer}}(ma, c, s) \\ &+ T_{\mathrm{msg}}(results, s, c) \end{aligned}$$

Extension to multi-hop agents

Under certain circumstances, it might be desirable to perform a series of actions in one substation, and then, depending on the success or failure of these actions, perform another series of actions in a different substation. However, in this application, only a small amount of data would be transferred from one substation to the next. Therefore it should be possible to determine whether mobility or remote interaction was the preferred method for each sequence of actions using the same formula as for a single hop, but substituting values of bandwidth and latency of the network between the agent's current location and its next location for those in the formula representing the bandwidth and latency between the client and the substation. A similar approach is taken by [53]. Due to time constraints, a full analysis of multi-hop agent performance is not attempted here.

4.3.3 Experiment 1

A preliminary experiment was performed to provide an initial evaluation of the performance benefits of mobile agents for a remote control scenario. In this experiment, three computers were used. These were a client computer (733MHz Pentium III), a mobile agent server (333MHz Pentium II) and a routing server. Also, a relay unit having an Ethernet connection was used to provide a device to be controlled.

The experiment consisted of a series of interactions between the client computer and the relay. For each interaction cycle, the client connected to the relay, read its current status (open or closed) and disconnected. The client then reconnected to the relay, operated it, and disconnected. Each cycle therefore consisted of a read interaction and a write interaction. The protocol used for these interactions was a proprietary protocol developed by the manufacturers of the relay, running on top of UDP/IP. Because the experiment was intended to simulate interaction with a number of relays, rather than the single relay actually used, the client or agent disconnected from the relay after each interaction and then reconnected. This required the sending of a small number of additional messages for each interaction.

A number of interaction cycles were performed using both mobile agent and client server methods. Both methods were identical except that when the mobile agent was used it travelled to the mobile agent server before performing the interactions. The mobile agent and client server were implemented as Java programs, and the Java Native Interface (JNI) was used to access an ActiveX protocol library for communications with the relay.

The experiment was performed for two scenarios. In the first, all of the computers were connected to the same 100BaseT LAN. In the second, the router, mobile agent host and relay were connected to the LAN, and the client was connected to the router by a 19.2 kbps serial link, in order to simulate interactions over a wide area network. The one-way latency of the 100BaseT LAN, as measured by the Windows 'ping' program, was under 10ms, while that of the WAN was measured as 30ms. The 'ping' program works by sending a message to a remote computer and measuring the amount of time for a reply to arrive. However, the version provided with Windows can measure only to the nearest 10 milliseconds. Therefore, while inaccurate, these measurements are sufficient to provide a general indication of the relative latencies of the two networks. The results of these experiments are shown in Table 4.6, and on Figure 4.7, which relates to the slow network, and Figure 4.8, which relates to the fast network. Each interaction cycle in this data represents a connect-read-disconnect-connect-write-disconnect interaction.

The results of these experiments demonstrate that, for this application, mobile agents are faster than a client-server approach if the following two

No. Cycles	MA, 19.2kbps	CS, 19.2kbps	MA, 100Mbps	CS, 100Mbps
0	1073.6	0	86.2	0
1	1828.6	1696.4	714.8	528.8
2	2375.4	3553.2	1309.8	967.6
3	2962.2	5149.4	1768.4	1470
4	3232.6	6948.2	2311.4	1900.4
5	3179.7	8758.8	2834.2	2489.4
10	7181.6	17240.8	5874.8	4668.8

Table 4.6: Time Taken (seconds) for a Mobile Agent (MA) and Client-Server System (CS) to Perform a Number of Interaction Cycles Using Either a 19.2kbps or 100Mbps Network. Times are the mean of three runs.

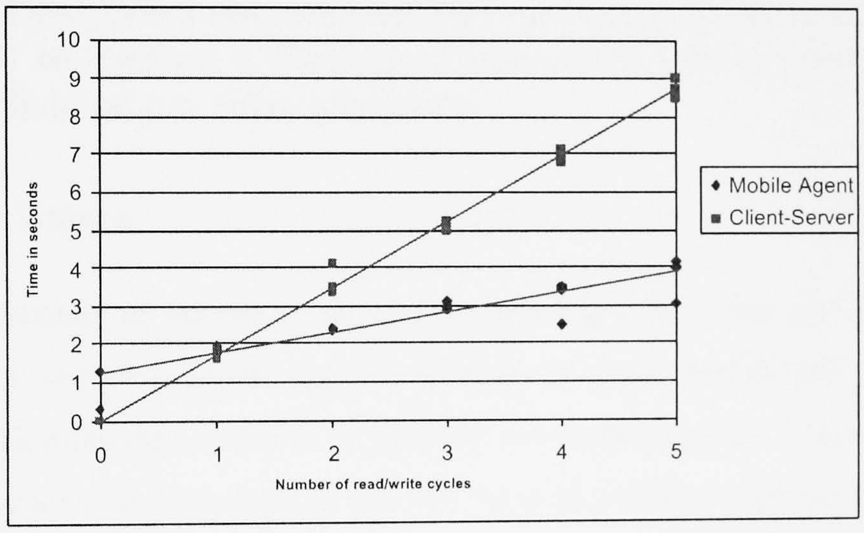


Figure 4.7: Time Taken (seconds) for a Mobile Agent (MA) and Client-Server System (CS) to Perform a Number of Interaction Cycles Using a 19.2kbps WAN. Individual run times are shown.

conditions are met:

- The network layout is such that, by moving, the mobile agent can reach a location that provides a connection of lower latency and higher bandwidth to the computer it wishes to communicate with. In a power system, this would be true when a mobile agent was being sent to a substation from another location over a dial-up link, as the internal network of the substation would have a bandwidth of 10 or 100Mbps and a low latency, while the dial-up link could have a bandwidth of 64 or 128 kbps and a

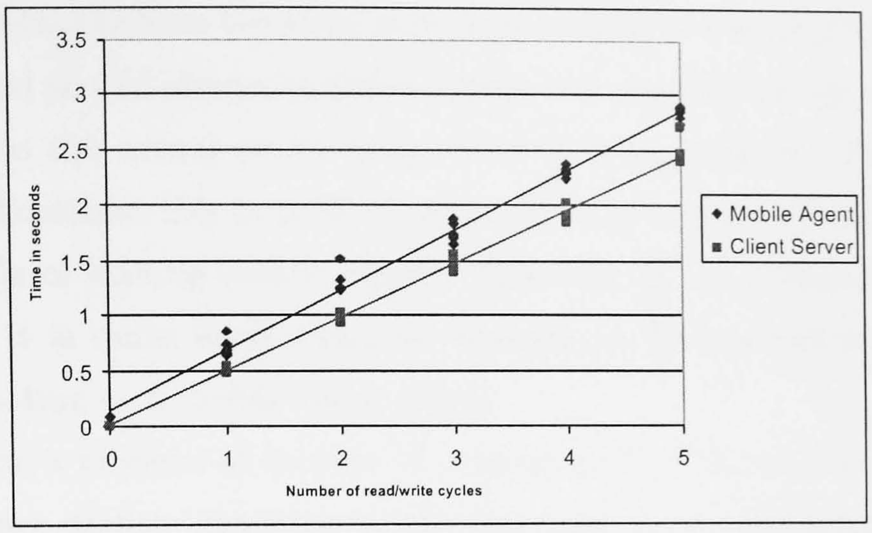


Figure 4.8: Time Taken (seconds) for a Mobile Agent (MA) and Client-Server System (CS) to Perform a Number of Interaction Cycles Using a 100Mbps network. Individual run times are shown.

higher latency.

- The number of interactions to be performed between the mobile agent and the server is such that the savings in data transferred over a slower network outweigh the cost of sending the mobile agent. The exact number of interactions will depend on the relative network speeds and the sizes of the mobile agent and the client-server messages. In this experiment, both were of small size, and so avoiding network latency became most important. Therefore the mobile agent was faster than client-server as long as more than 1 cycle of interactions was performed.

This is because to carry out a sequence of interactions remotely using the client-server system requires at least four messages per interaction (send read request, get value, send write request, receive acknowledgement of write). The delay incurred is the number of messages multiplied by the network latency. To move a mobile agent and send a reply after the mobile agent has interacted with a system incurs a delay equal to twice the latency. Therefore, the mobile agent will incur less delay due to network latency, and the difference between the two methods will increase with the number of interactions.

If both networks have the same performance characteristics, then the client-server method should always be faster, unless it is possible for the mobile agent to move onto the actual server with which it is to interact. For industrial control applications, this is unlikely, as most monitoring and control devices are incapable of hosting mobile agents. However, in an information retrieval application it is quite likely that, for example, a Web server might also be capable of acting as a mobile agent server.

There are a number of sources of inaccuracy in this experiment. Firstly, the results are slightly distorted by the fact that the mobile agent server was slower than the client computer. This gave a slight advantage to the client-server method, which is most visible in the case where the 100Mbps network is used, although part of the difference between the two methods in this case is due to the delay in transferring the mobile agent. In order to quantify the performance difference, 10 client-server interactions were performed between the mobile agent server and relay, and between the client computer and relay. The average time taken for 5 sets of these interactions was 4779ms for the client computer, and 6289ms for the mobile agent server. Therefore, the mobile agent server took approximately 31% longer on average. Other factors that may have distorted the results include just-in-time compilation by the Java Virtual Machine.

4.3.4 Experiment 2

Because of the limiting factors of the previous benchmark, and the need to gather more data, a second experiment was performed. In this experiment, an agent was used to remotely control a network-attached relay for a number of read/write interactions. Because of the (proprietary) protocol used by the relay, each of these interactions resulted in a number of messages being exchanged between the agent and the relay. As in the data analysis experiment, the bandwidth and latency were varied using a bridge running the Dummynet program[113]. The experimental setup is shown in Figure 4.9.

The experiment was designed only to evaluate the use of mobile code in this application. Therefore the interactions were performed using a software library, rather than via static agents as described earlier in this thesis.

Some problems were encountered with the Java library that connected to the relay. In order to prevent these problems, a call to Thread.sleep(500) was added after each operation, causing the agent to sleep for 500 milliseconds. The agent also slept for 300 milliseconds after connecting to the relay. To obtain the final results, the total sleep time ($0.5n + 0.3$ seconds) was calculated and subtracted from the measured time. This might introduce a small error due to the inaccuracy of the Thread.sleep function. However, this should be reduced by the number of interactions used.

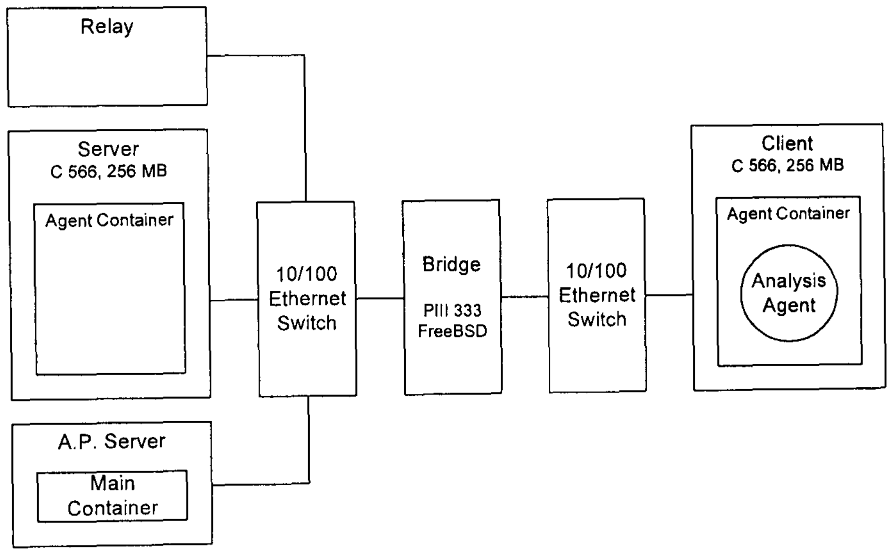


Figure 4.9: Experimental Setup for Remote Control Experiment

Theoretical Results

Using the equations 4.3.1 and 4.3.2, it should be possible to predict the point at which the mobile agent method becomes more efficient than the client server method. The processing time (T_{proc}) is unknown. However, the processing time is the same for both the mobile agent and client-server methods, and so can be omitted from the comparative analysis. Also, the equation will be simplified further by setting $T_{\text{msg}} = \frac{\text{size}}{\text{bandwidth}} + \text{latency}$, and by supposing that all request messages have the same size B_{req} and all reply messages the same size B_{rep}. While this ignores marshalling overheads, accurate measurements of these are not available, and in any case, the marshalling overhead should be small compared to the actual message / mobile agent transfer time.

For the client - server method, the equation is simplified to:

$$T_{\text{CS}} = \frac{n(B_{\text{req}} + B_{\text{rep}})}{\tau_{\text{wan}}} + 2n\delta_{\text{wan}}$$

For the mobile agent method, the simplified equation is:

$$\begin{aligned} T_{\text{MA}} &= \frac{n(B_{\text{req}} + B_{\text{rep}})}{\tau_{\text{lan}}} + 2n\delta_{\text{lan}} \\ &+ \frac{2B_{\text{MA}}}{\tau_{\text{wan}}} + 2\delta_{\text{wan}} \end{aligned}$$

In these equations, τ_{wan}represents the bandwidth of the wide area network, δ_{wan}represents the latency of the wide area network, and τ_{lan} and δ_{lan} represent the bandwidth and latency of the local area network. Because in the experiment the mobile agent returns to the client to display its results, the amount of data transferred in the mobile agent case is twice the size of the agent, rather than the size of the agent added to the size of an acknowledgment message.

The "crossover point" at which using a mobile agent becomes more efficient than a client-server system then depends on the bandwidth τ, latency δ, number of interactions n and the size of the mobile agent and of the replies. The sizes of the request and reply messages are unknown. As an approximation, it is assumed that each message has size 100 bytes. The size of the mobile agent is 6353 bytes (object) + 10796 bytes (class files). For these calculations, it is assumed that the class files are already present, and the object size only is used.

The three graphs, Figures 4.10, 4.11 and 4.12, show the theoretical time taken against the number of interactions, for several different settings of WAN bandwidth and latency. It can be seen from Figures 4.10 and 4.11that decreasing the bandwidth of the WAN has the following effects:

- The gradient of the client/server line is increased

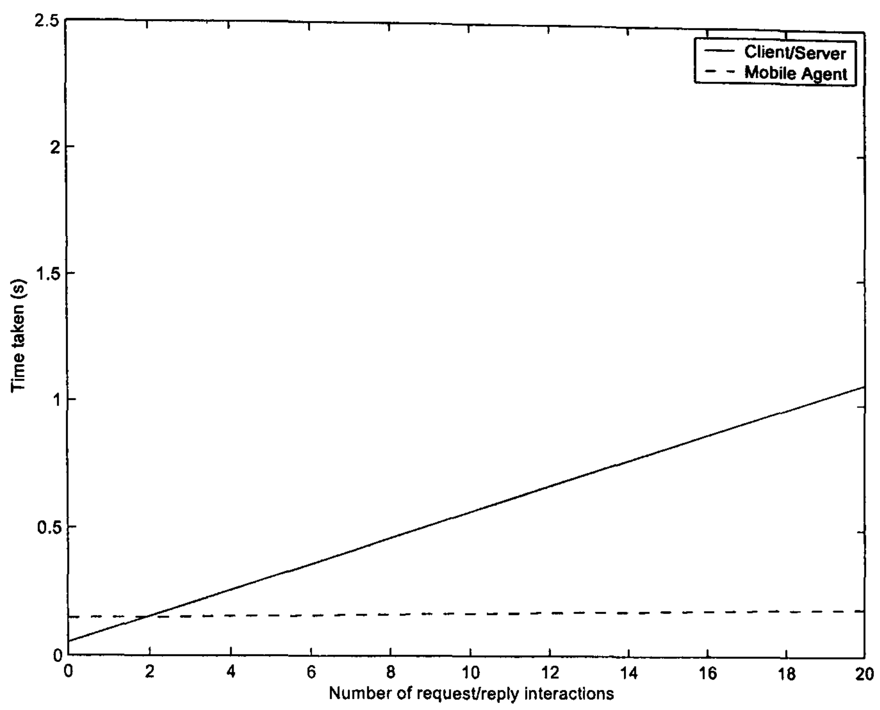


Figure 4.10: Theoretical Timing (Excluding Marshalling and Operation Time) for 1Mbit/s, 25ms WAN, 100Mbit/s, 1ms LAN

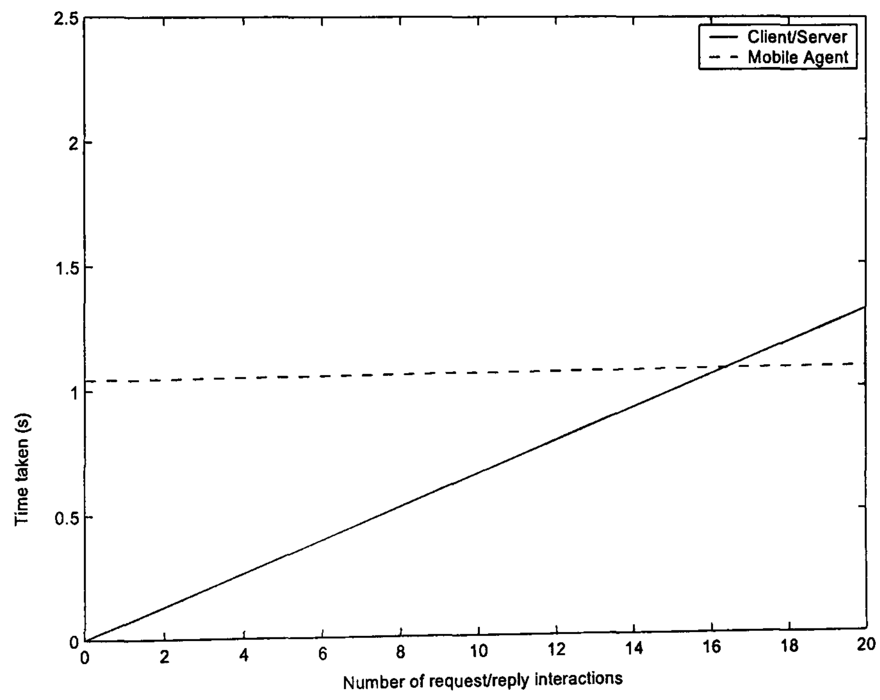


Figure 4.11: Theoretical Timing (Excluding Marshalling and Operation Time) for 100 kbit/s, 25ms WAN, 100Mbit/s, 1ms LAN

- The intercept of the mobile agent line is increased

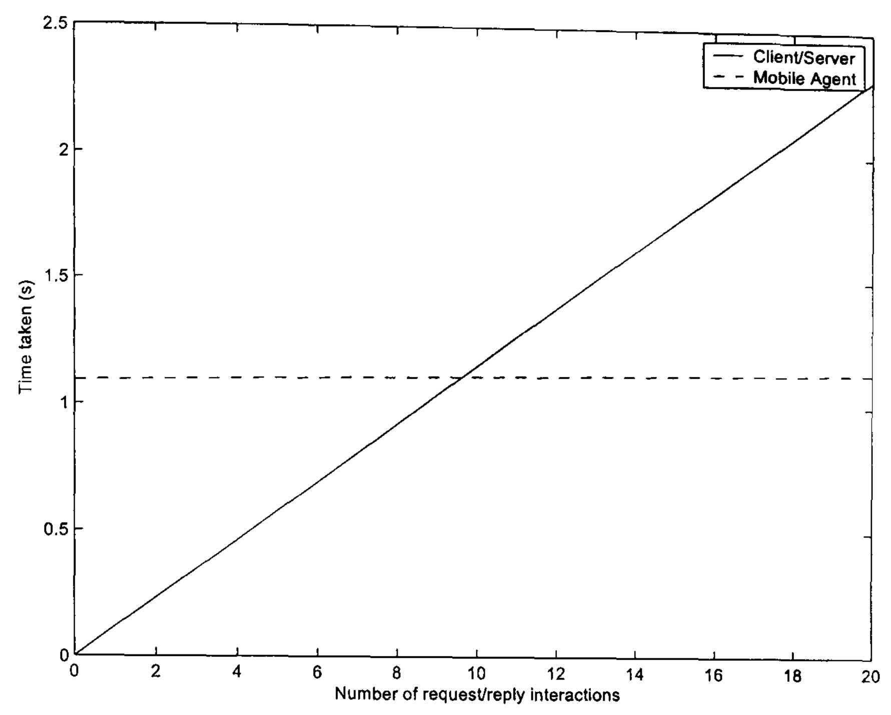


Figure 4.12: Theoretical Timing (Excluding Marshalling and Operation Time) for 100 kbit/s, 50ms WAN, 100Mbit/s, 1ms LAN

Due to the relative magnitudes of these changes (the client/server messages are much smaller than the mobile agent) the overall effect is to significantly increase the number of interactions required before the mobile agent is the most efficient interaction method.

It can be observed from Figures 4.11 and 4.12 that increasing the latency of the WAN (by 25 milliseconds) has the following effects:

- The gradient of the client/server line is increased
- There is little effect on the mobile agent line (actually, the intercept will be increased by 50 milliseconds, but this is not noticeable on the graph).

Therefore, the effect of increased latency is to decrease the number of interactions required before the mobile agent is the most efficient interaction method. However, the number of interactions will never fall below 1, as two messages (equivalent to a single interaction) must always be sent across the WAN when the mobile agent is used. Therefore, for a single request/reply interaction, the mobile agent will always be less efficient (as long as the combined size of the request and reply messages is smaller than that of the mobile agent), due to a larger *size/bandwidth* term.

Problems with the estimated crossover point The estimated "crossover point" will be optimistic (in favour of the mobile agent) because the mobile agent transfer overheads have been omitted. Also, the JADE platform is not as efficient as is theoretically possible, as a number of messages, rather than a single message, are sent in order to send or retrieve an agent. Therefore, it is expected that in the experiment the overall trends should be the same as the theoretical results described, but that the crossover point and time taken will occur at a slightly greater number of interactions. The time taken in all experiments will also be greater than the theoretical time due to the relay operation time and the time taken to connect to the relay.

Results

Table 4.7 shows the fixed parameters for the experiment. The latency between the mobile agent server and the relay and that between the client and relay at 0 latency were measured using a free "ping" implementation[3] and are the average of 10 pings. The size of the mobile agent is slightly inaccurate as the agent had to be modified in order to print out its size. However, this should only make a small difference to the object size. The class file sizes are those from the actual agent used in the experiments.

Object	Value
$\delta(s,d)$ (server-device latency)	1.13ms
$\delta(c,s)$ when set $\delta(c,s)=0$ (client-server latency)	0.82ms
$\delta(c,d)$ when set $\delta(c,s)=0$(client-device latency)	1.68ms
Size of agent object	6353 bytes
Total size of agent .class files (3 files)	10796 bytes

Table 4.7: Fixed Experiment Parameters

[3]www.cfos.de/ping/ping.htm

An initial experiment (not described here) varied the number of actions between agent and relay from 0 to 5. However, it was found that with this number of interactions the client-server method always outperformed the mobile agent. Because the results of the first experiment did not show any point at which mobile agents became more efficient than the client-server method (the client-server method always completed the task more quickly than the mobile agent), a second experiment was performed. It was decided for this experiment to focus on the parameters of latency and number of interactions. Therefore, the bandwidth was fixed at 1Mbit/s, the one-way latency was varied between 0 and 100 milliseconds and the number of interactions was varied between 0 and 100. The results of this experiment (as the mean time of four runs) are summarized in Tables 4.8 and 4.9. Full results are given in Appendix A. The sleep time between interactions and after connecting to the device has been subtracted from both mobile agent and client server results, as this time was the same for both agents and should not be present in a full implementation.

		Number of interactions					
		0	20	40	60	80	100
Latency(ms)	0	2.36	2.79	3.36	3.64	4.07	4.57
	25	3.87	4.22	4.75	5.23	5.63	6.12
	50	5.32	5.71	6.32	6.83	7.31	7.61
	75	6.72	7.21	7.79	8.48	9.05	9.39
	100	8.22	8.92	9.36	10.02	10.64	11.00

Table 4.8: Time (secs) Taken By Mobile Remote Control Agent (Mean Time - Sleep Time)

Because of the implementation of the mobile agent, it was necessary for it to locate the server before moving. This was done by giving it the name of an agent on the server (representing a device agent) and then having the mobile agent ask the AMS for the location of this agent, which could then be used to request a move. This interaction used the FIPA request protocol, which involves three messages, a request message from the agent to the AMS, followed by an agree message and an inform message from the AMS to the

		Number of interactions					
		0	20	40	60	80	100
Latency(ms)	0	1.63	2.09	2.47	2.92	3.31	3.75
	25	1.98	3.30	4.78	6.07	7.50	9.00
	50	2.53	4.84	7.31	9.64	12.05	14.47
	75	2.95	6.21	9.57	13.01	16.39	19.79
	100	3.30	7.53	11.88	16.29	20.69	25.12

Table 4.9: Time (secs) Taken By Static Remote Control Agent (Mean Time - Sleep Time)

agent.

In order to quantify the cost of this operation, another series of experiments were performed. The mobile agent was modified to print out both the total time taken, and the time taken excluding the time to locate the server. The time taken to locate the server is then the difference between these two times. The mean time to locate the server for four interactions, at different latencies, is shown in Table 4.10.

Latency(ms)	0	25	50	75	100
Time to locate server	0.13	0.34	0.53	0.74	0.94

Table 4.10: Cost of Locating Server

Analysis

Figure shows how the crossover point at which the time taken by a mobile agent became less than that taken for a static agent varied as the bandwidth and latency were altered. The area above the line is that for which a mobile agent provided superior performance.

It can be seen that for small numbers of interactions, or where the latency was low, the client-server, or static agent, method was superior. However, when the number of interactions and the latency were large, the time taken by the mobile agent was less than that for the static agent.

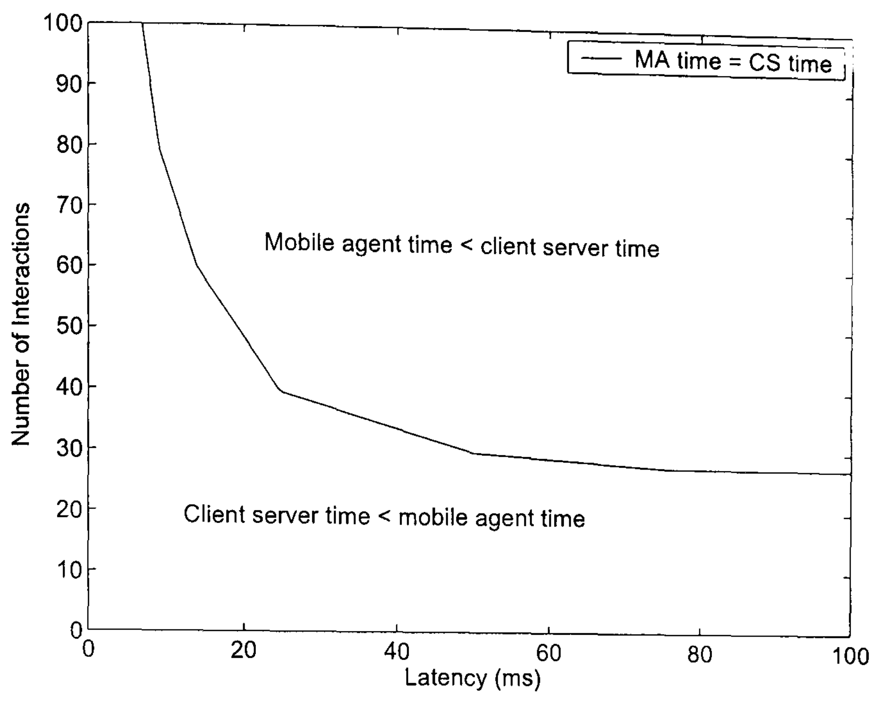


Figure 4.13: Crossover Point

Comparison to theoretical model

Using the model discussed earlier, the experimental results are now compared with theory, first for a scenario in which the latency is altered, and then for a scenario in which the number of interactions is altered. While it is not expected that the theoretical results will match the experimental results completely, due to the simplifications made in the formula and also due to the startup cost (starting the agent, loading a library and connecting to the relay) incurred in the experiment, it is expected that there should be a strong correlation between the two sets of results. In the theoretical results presented here, it is assumed that the latency of the LAN is 0 and the latency of the WAN is equal to the set latency. Using the actual values should not make a significant difference, as the latency of the LAN is actually less than 1 millisecond, giving an error for 100 interactions of 100 milliseconds or 0.1 seconds.

Effect of altering latency By taking the client-server results for $n = 40$, the graph shown in Figure 4.14 is obtained. This shows that the theoretical and experimental results do not agree completely. Firstly, there is a fixed cost in the experimental results (shown by the point at which $\delta = 0$). This consists of the time to load the library used to communicate with the relay, the processing time, or response time, of the relay, and the per-interaction overheads O_S and O_R. The difference in the gradients of the two lines (overhead varying with latency) is probably due to the time taken to exchange the messages required to establish a connection to the relay.

Using a linear approximation, the equation of the measured line is $t = 94\delta + 2.5$. The equation of the theoretical line is $t = 80\delta + 6.1 \times 10^{-5}$.[4]

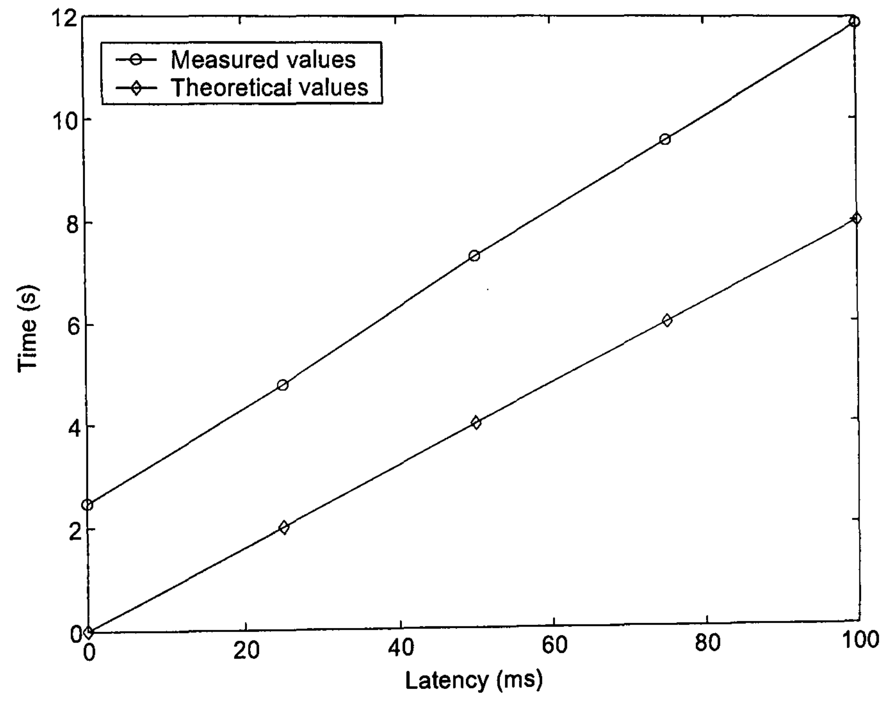


Figure 4.14: Comparison of Theoretical and Actual Results for Static Agent with $n = 40$

Performing the same procedure for the mobile agent, and taking the mobile agent size to be 6353 bytes (its size excluding class files) the graph of Figure 4.15 is obtained.

This graph shows a significant difference between the predicted and experimental results for the mobile agent method. Even when allowing for processing

[4]Because $\frac{B_{\text{req}}+B_{\text{rep}}}{\tau(d,c)}$ is insignificant compared to the rest of the equation (the bandwidth is 1Mbit/s, which approximately equals 128Kbyte/s, changing the (estimated) values of $B_{\text{rep}}$and B_{req}makes little difference to the theoretical results.

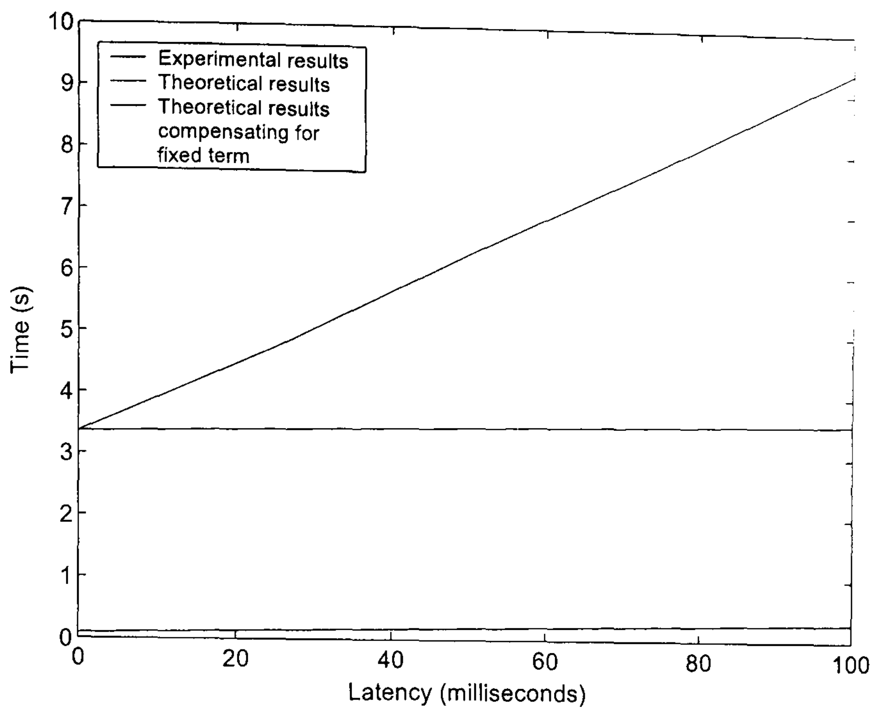


Figure 4.15: Comparison of Theoretical and Actual Results for Mobile Agent with $n = 40$

time, mobile agent marshalling and registration overheads and the actual latency of the local area network by shifting the mobile agent line so that the values at $\delta = 0$ are identical, there is a marked difference in the gradient of the two lines. The model predicts that, because there should be only one message sent from client to server and one from server to client (to transfer the mobile agent), the latency of the wide area network should have little effect on the time taken (If the latency is 100 milliseconds, then this should add only 200 milliseconds to the total time). If the recipient of a mobile agent must send a message to acknowledge the mobile agent, the effect of latency should double, which would double the gradient of the line. However, this would still not be close to the experimental results. In the experiment, there is a difference of approximately 6 seconds between the time taken when $\delta = 0$ and the time taken when $\delta = 100$, when even allowing for a request and a reply between the mobile agent sender and recipient the difference in theory would be 400 milliseconds.

Part of this difference is undoubtedly due to the need to locate the server before the mobile agent moves. As shown by Table 4.10, the time to do this ranges from 0.13 seconds at 0 latency to 0.94 seconds at 100 milliseconds one-way latency. However, this is insufficient to account for the whole difference. Another possible reason might be the inefficiency of the Java RMI protocol, used by JADE for message and mobile agent transfer. According to [120], it is possible for an interaction using RMI to require as many as six round-trip interactions for a single request and reply. There may also be other unknown overheads in mobile agent transfer, which result in message exchange.

Changing the number of interactions Figure 4.16 shows the results for a set latency of 50 milliseconds, varying the number of interactions. As for when the number of interactions was fixed, there is a large fixed overhead due to the time taken to start the agent and connect to the relay. There is also a variable overhead which increases with the number of interactions. We hypothesize that this is due to the marshalling overheads and to the relay's operation time. For the measured results, a linear approximation is $t = 0.12n + 2.5$. For the theoretical results, a linear approximation is $t = 0.1n$. Therefore, for this case, the variable overhead is equivalent to $0.02n$, or 20 milliseconds/interaction, which is a realistic value for T_{proc}. However, as the value is so small it is also possible that it is affected by experimental error.

For the mobile agent, the graph obtained is shown in Figure 4.17. The discrepancy between the fixed term in theory and in practice can be explained by the increased cost (compared to the theoretical value) of moving the mobile agent and to the cost of establishing a connection to the relay, as observed in previous results. The difference in gradient can be attributed to the processing time T_{proc} and to the fact that the actual latency of the LAN was 1.13 milliseconds, and not 0, which would create an additional time cost of approximately 2.2 milliseconds per interaction. A linear approximation to the plot for the measured data is $t = 0.024\delta + 5.3$, whereas that for the theoretical data is $t = 1 \times 10^{-5}\delta + 0.19$.. However, the gradient of the measured data plot is only slightly larger than the suggested value for $T_{\text{proc}} = 0.02$seconds per interac-

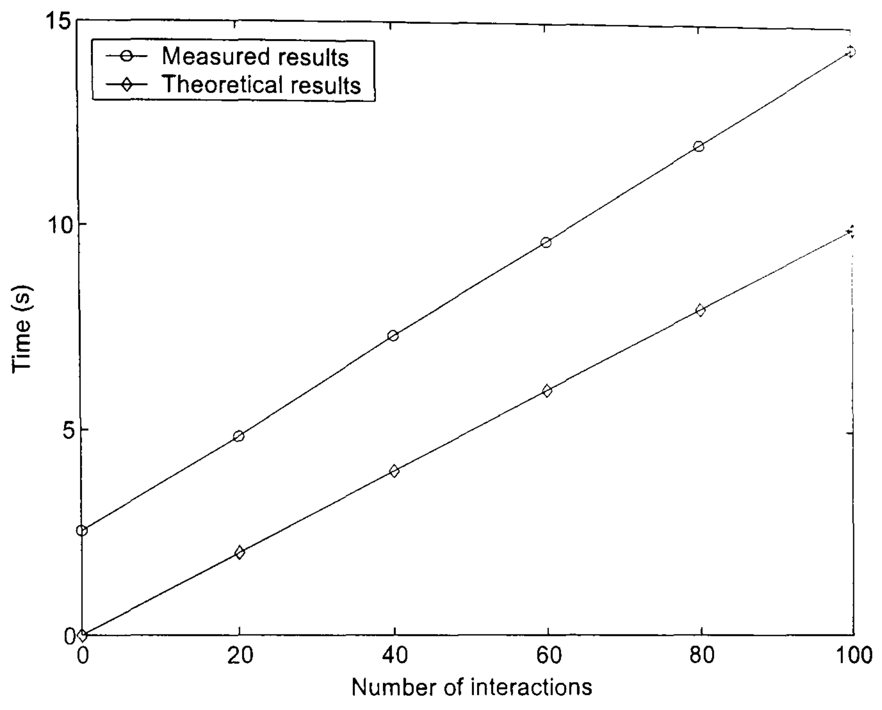


Figure 4.16: Comparison of Theoretical and Actual Results for Static Agent with One-Way Latency = 50ms

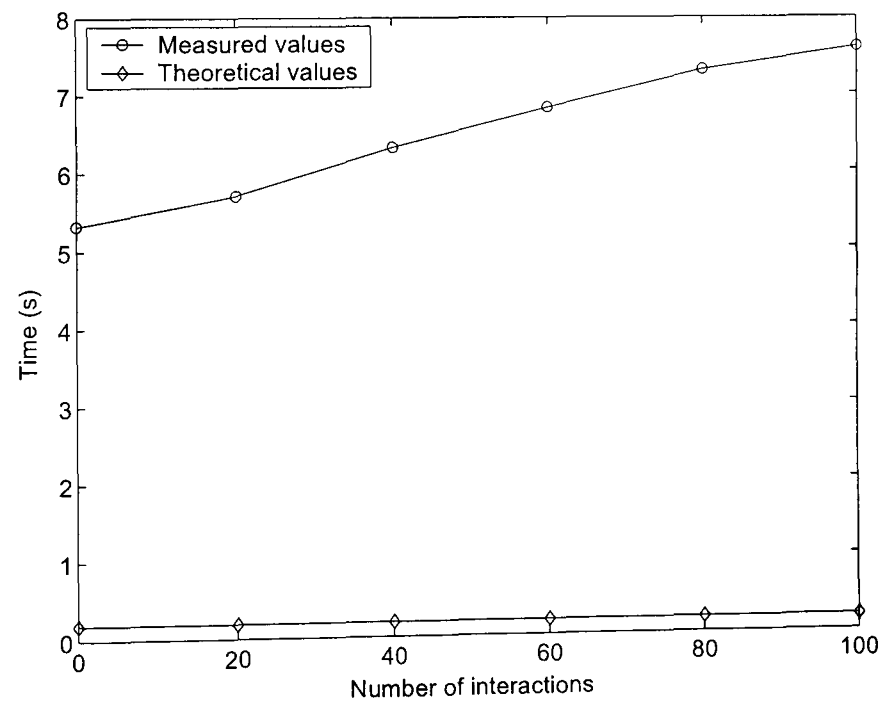


Figure 4.17: Comparison of Theoretical and Actual Results for Mobile Agent with One-Way Latency = 50 ms

tion obtained from the plot for the static agent. This suggests that the main problem with the theoretical model for the mobile agent based method is that, as previously discussed, it significantly underestimates the time to move the mobile agent.

Discussion

One drawback of using a mobile agent for remote control is the fact that the actions to be performed must be known to the operator at the time the agent is launched. If an agent is only to perform a single action, its performance is no better than the client-server case. This means that the agent cannot be used, for example, for cases where the operator must select an action to perform based on the results of the previous action. However, in circumstances when a sequence of actions can be determined prior to launching the agent, such as the example given at the start of this section, the mobile agent does provide appreciable performance improvements.

In the experiment described in this thesis, the "crossover point" between mobile agent and static agent implementations came at a relatively high number of interactions (around 30-40 interactions for 25ms<latency<100ms). This is probably because the time taken by the mobile agent to locate the server, move and connect to the relay (shown by the results for $n = 0$) represents a high proportion of the total time taken in all of the interaction sequences. In contrast to this, the "startup time" for the static agent was much less, as it had only to connect to the relay. In order to make mobile agents more suitable for small numbers of interactions, effort should be made to reduce this overhead. This problem is similar to one highlighted in other work [54].

4.3.5 Summary and Conclusions

While the results obtained in the two experiments (Sections 4.3.3 and 4.3.4) display the same overall trend, with mobile agents having better performance at high latencies, the number of interactions required before mobile agents performed better was much lower in the preliminary benchmark than in the detailed benchmark. The reasons for this include:

- In the first benchmark, each interaction consisted of a sequence *connect-read-disconnect-connect-write-disconnect.* In the second benchmark, the agent connected to the relay at the start, and each interaction consisted only of a *read-write* pair. Therefore, the number of messages per interaction was significantly lower in the first benchmark. With hindsight, a more realistic scenario, even when simulating interactions with multiple relays, would be for each interaction to be either a *connect-read-write-disconnect* interaction, which should produce results somewhere between those of the two benchmarks, or the *read-write* interaction used by the second benchmark. This is probably the most significant difference between the two benchmarks.
- The mobile agent system used for the first benchmark was simpler than the one used for the second benchmark, and fewer messages had to be exchanged in order to move an agent from one server to another.
- The agent size of the agent in the first benchmark was less than that of the agent in the second benchmark.

Taken together, the two sets of experimental results permit the conclusion to be drawn that mobile agents can improve the performance of a sequence of remote control operations over a high latency network. However, the exact sequence length and latency required before mobile agent performance is better than client-server performance depend on the particular implementations of the agent, agent platform and control devices.

When building the system, which has a multi-agent component as well as a mobile agent component, it was desired to adhere as closely as possible to the FIPA standards. This resulted in the choice of the JADE platform for system implementation. However, JADE's focus is not on mobile agents, and so it lacks some of the mobile agent related features of other systems (strong mobility, support for multiple versions of the same agent class in the same container, inter-platform mobility), and its agents are also more "heavyweight" (around 30 kilobytes for a basic serialized agent, excluding class files which, as part of the system, were already present). For a full deployment of the system it might be necessary to extend the platform or to re-examine the available alternatives.

Another problem encountered in constructing the system was that much of the industrial automation software and hardware that was to be integrated with the mobile or static agents is Windows-based and communicates only via ActiveX or via DLL libraries. Therefore considerable effort had to be undertaken, using the Java Native Interface, to allow agents to access these applications. Further work in this area would benefit from the development of simpler methods of accessing industrial automation devices using Java, or from the use of alternative technologies for the implementation of multi-agent systems (for example, [121] discusses the porting of a mobile agent system to Microsoft .NET).

Related Work

Tsukui et al [115] have used mobile agents to retrieve data from power system protection devices. As in our work, they utilized a mobile agent for analysis. In their application it was used to analyse fault records from devices. They also used mobile agents to gather status information from devices and to alter device settings. This application is similar to the control application described here, as the alteration of a setting on a device can be considered, from the point of view of an agent, as a control operation (the agent sets the setting and receives a reply to confirm that it has been altered). However, their work does not include a detailed analysis of application performance, or describe the agent implementation in detail.

Harrison, Chess and Kershenbaum [55] suggested the use of mobile agents for remote control applications, and suggest that agents facilitate "remote real-time control when the network latency prevents real-time constraints being met by remote command sequences". However, they do not provide a performance model or any experimental results. The results of the experiment described here suggest that the statement made by Harrison *et al* may be correct.

4.4 Summary

This chapter has examined the application of mobile agents within the architecture of Chapter 2, concentrating on the applications of data analysis and remote control. In the data analysis application, a mobile agent was used to travel to a database and perform some analysis on the data in that database, before returning a report to the user. An extension of this agent was capable of analysing multiple data sets, and was implemented in such a manner that new analysis functions could be programmed by the user. In the remote control task, an agent was used to carry out a sequence of actions in a substation, via the control devices. It was demonstrated that in the data analysis task, the use of a mobile agent provided superior performance to the use of static agents, provided that the amount of data transferred was large or the bandwidth of the network was low. In addition to this, it was demonstrated that when using a mobile agent, a performance improvement could be achieved by providing direct JDBC access to the database rather than by using a wrapper agent to perform translation. However, data source independence could still be achieved by allowing a wrapper agent to be used to generate the SQL query, which was then passed to the mobile agent. This technique is particularly appropriate due to the simple nature of the data sets used by the analysis agent (flat tables consisting of two columns), and may not work in more complex data retrieval tasks. For control tasks, it was demonstrated that the mobile agent had higher performance than a client-server control system when the number of interactions to be performed was large and the latency of the network was high. However, because control messages are typically small, reducing the bandwidth of the network, at least for smaller numbers of interactions, decreased the relative performance of the mobile agent as the agent size exceeded the total size of the control messages.

The next chapter presents a prototype implementation of the full architecture, including both mobile and static agents, and provides examples of its usage. Based on a real-time simulator provided by the National Grid Company, the prototype provides a realistic testbed for the concepts described in the thesis so far.

Chapter 5

Implementation of Substation Information Management System Based on Proposed Architecture

5.1 Introduction

This chapter describes a distributed substation automation system prototype developed using the architecture and models described in this thesis. To provide a realistic test environment for the system without the difficulty involved in an implementation on an operational site, a real-time substation simulator provided by the National Grid Company (Figure 5.1) was used. This simulator consists of an industrial computer with a large number of analogue input and output channels, which are identical to the I/O facilities available to substation controllers in a current substation. Because the simulator was designed to test substation control equipment, any system which operates correctly with the simulator should be capable of doing so in an actual substation environment.

The tasks of the prototype system are to gather data from the substation

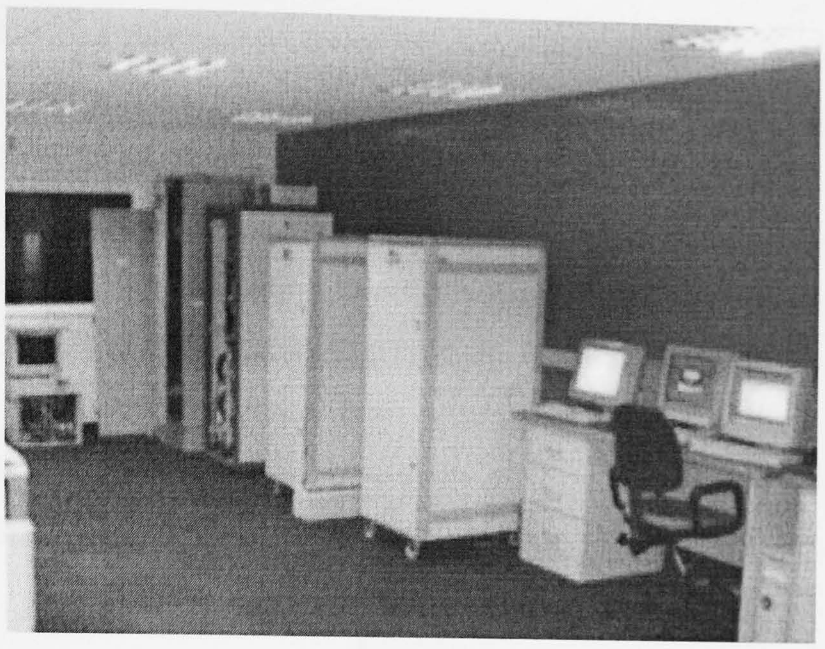


Figure 5.1: National Grid Company Substation Simulator

simulator via a data acquisition system, store this data in the National Grid Information Management Unit (IMU) database, and provide online display of data, historical data querying, data analysis and documentation management services to users via a human-machine interface.

The prototype is implemented using the JADE multi-agent systems toolkit[95]. The reasoning engines of the agents are implemented using a Prolog interpreter[1] with a Java interface. FIPA SL expressions (queries and requests) may be converted into Prolog expressions in a relatively simple manner, as both are based on first-order logic, and the main difference lies in the syntax (although FIPA SL has modal expressions and frames which are not implemented by Prolog).

5.2 System Architecture and Agents

The information management system, shown in Figure 5.2, uses the generic architecture described elsewhere in this thesis. However, the generic data logging database has been replaced by the Information Management Unit (IMU),

[1]AMZI Prolog (http://www.amzi.com).

which performs the same functions but represents the actual system to be installed in National Grid Company substations. The IMU is based on a Microsoft SQL Server database, and has a Web service interface which allows other programs to use the Simple Object Access Protocol (SOAP), a World Wide Web Consortium standard, over Hypertext Transfer Protocol (HTTP) to insert data into the IMU or to perform queries. The component of the IMU used to acquire data is the Data Recording Service (DRS), and that used to query the IMU is known as the Query Service. There is also a second interface to the IMU via the Microsoft .NET Remoting protocol.

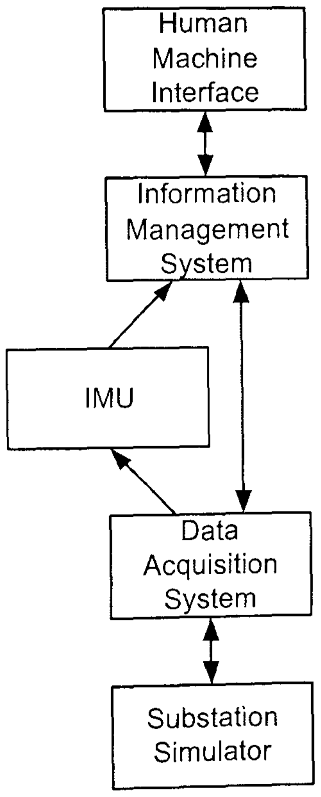


Figure 5.2: Overall Structure of System

Figure 5.3 shows the architecture of the information management system, based around the IMU system. Data is gathered from the substation simulator by a data acquisition PC, and stored into the IMU. The IMU is managed by its own agent, the IMU agent, which is based on the database agent described in Section 3.1. The other agents shown on the diagram are as given on the architecture diagram of Figure 2.7 and described in Chapters 2 and 3. How-

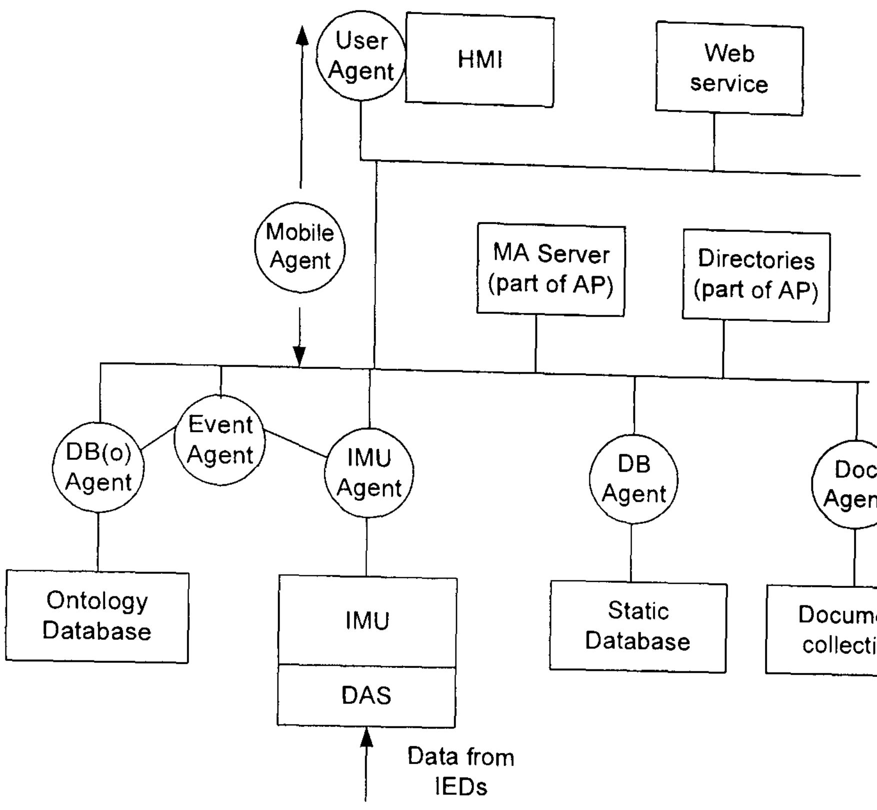


Figure 5.3: Information Management Agent System Architecture

ever, the mobile server (MS) agent has been omitted, as no mobile servers are available in the prototype implementation. Also, there are currently no task agents present in the prototype.

Because the IMU is only a database, it is not possible to control the system via the IMU. Also, it does not provide any means to automatically notify a subscriber when data in the database is updated. This means that it is difficult to support event-driven updating, for example, the FIPA subscribe protocol, using the IMU. Therefore, the agent-based data acquisition and control system shown in Figure 5.4, which uses the architecture described in Section 2.2, is used for control and event-based data updates.

The system contains only a single IED, the data acquisition PC system. Although this PC contains several I/O cards, it communicates with the multi-agent system over a TCP/IP connection, and it is simpler to treat the PC as a single device. Therefore only one device agent (represented in Figure 5.4 by "DAQ Agent") is present in the system. There are multiple plant agents

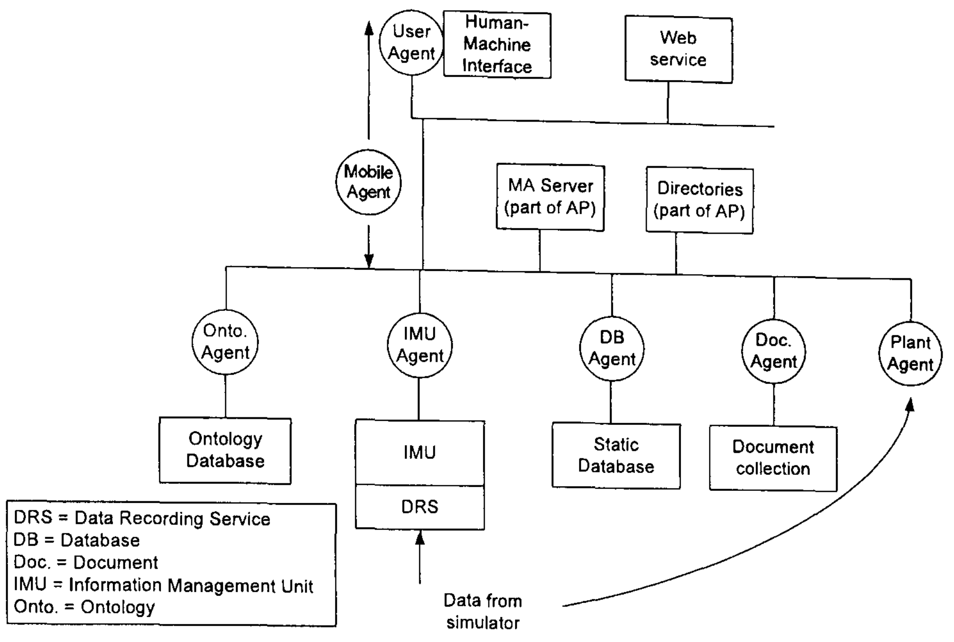


Figure 5.4: Data Acquisition Agent System Architecture

(in a complete system one for each item of plant, however, in the current prototype only three have been implemented), co-located with the device agent and acquiring data by communicating with it. These agents are capable of passing information to the user interface agent, and of carrying out control commands sent to them.

5.2.1 Information Management System Agents

The information management system consists of database agents (as described in Section 3.1), an IMU agent for communication with the IMU, user interface agents and mobile agents. In addition, broker or task-oriented agents may be used to provide specific services. Further detail on the implementation of these agents is provided by Appendix B, and the format of the configuration files used is described in Appendix C.

Database Agents There are three database agents present in the system. The *static database agent* manages a database containing static configuration information regarding the substation plant and the data acquisition system. The *mapping agent* (part of the data acquisition system) manages the mapping rules allowing agents to perform input data interpretation. The *ontology database agent* manages a database containing the system ontologies. The behaviour of these agents is as described in Section 3.1.

The IMU Agent While the National Grid Company IMU is a database, it is not accessed via an SQL-based interface, but via a Web service. Therefore, an additional agent, the *IMU agent*, is required for this system. The reasoning engine of this agent has been modified to add procedures which allow the Prolog interpreter to query the IMU via the Web service interface. However, to external agents, this agent will appear the same as it would if it were a database wrapper agent, as it still uses the FIPA Query protocol and the global ontology. The mapping rules of the IMU agent, written in Prolog, translate between the global system ontology and the schema used by the IMU's Web service interface.

The Alarm/Event Agent The alarm/event agent is a *broker* agent that takes events and alarms generated by plant agents and forwards them to user agents as appropriate.

It operates by locating agents providing a subscription service and then advertising the subscriptions provided by those agents as if it provided them itself. When an agent subscribes for a particular event notification, the alarm and event agent establishes a subscription with the provider agent and forwards any event notifications received.

The use of a broker agent provides the possibility to add new features, for example, to generate alarm conditions by combining information from multiple sources. This functionality has not yet been implemented in the prototype and is a topic for further work.

User Interface Agent and HMI The user interface agent is implemented in Visual Basic (Microsoft Corporation, Redmond, WA) and provides a link between the HMI interface (implemented in LabVIEW (National Instruments, Austin, TX)) and the multi-agent system. It also provides its own graphical user interface for mobile data analysis agent generation and database querying.

5.3 System Ontology

As described in Section 2.4.3, the system's ontology consists of several components: the *automation ontology*, which provides a (partial) generic ontology for data acquisition and control systems, the *substation plant ontology*, which describes the different items of plant found in substations, and the *information management ontology*, which describes documents, other information resources and querying. The instantiations of those ontologies used in the development of the prototype are described here. The ontologies are modelled using a UML class diagram notation, with operations representing actions that can be performed on an instance of a particular class.

Automation Ontology The automation ontology describes industrial automation devices and systems. In the prototype system, the ontology used is that shown in Figure 2.5, which was used to design the data acquisition multi-agent system and is described in Section 2.2.

Plant Ontology The plant ontology describes substation plant and its properties, along with the operations that may be performed on it. It is not intended to represent all of the properties available in a full substation automation system would require a substantially larger ontology, but only to provide an example and a simple ontology for the prototype implementation. This ontology is based on the properties provided by the substation simulator, and is shown in Figure 5.5. Additional properties of the transformer object have been taken from the database schema of the transformer monitoring system described in [92]. However, in this schema, there are quantities representing both actual and predicted quantities, for example OIL_IN_ORG (actual oil in) and OIL_IN_OUT (predicted oil in temperature). For the purposes of the ontology, it is considered that the predicted temperature is not actually a separate property, but that it represents what some agent believes that the value of that property will be at a certain time (see Section 2.4.2). The ontology represents only actual properties of the transformer. However, this is not the only way to represent the transformer. An alternative would be to represent the transformer as a collection of components (e.g. tap changer, windings, tap change miniature circuit breaker), and then represent the properties of these components.

All properties of the disconnector and circuit breaker are inherited from the parent class, "switchgear". This is because the difference between these types of plant is that it is not possible to open a disconnector while it is live. This cannot be shown in the object model and should instead be encoded into the behaviour of the relevant plant agents.

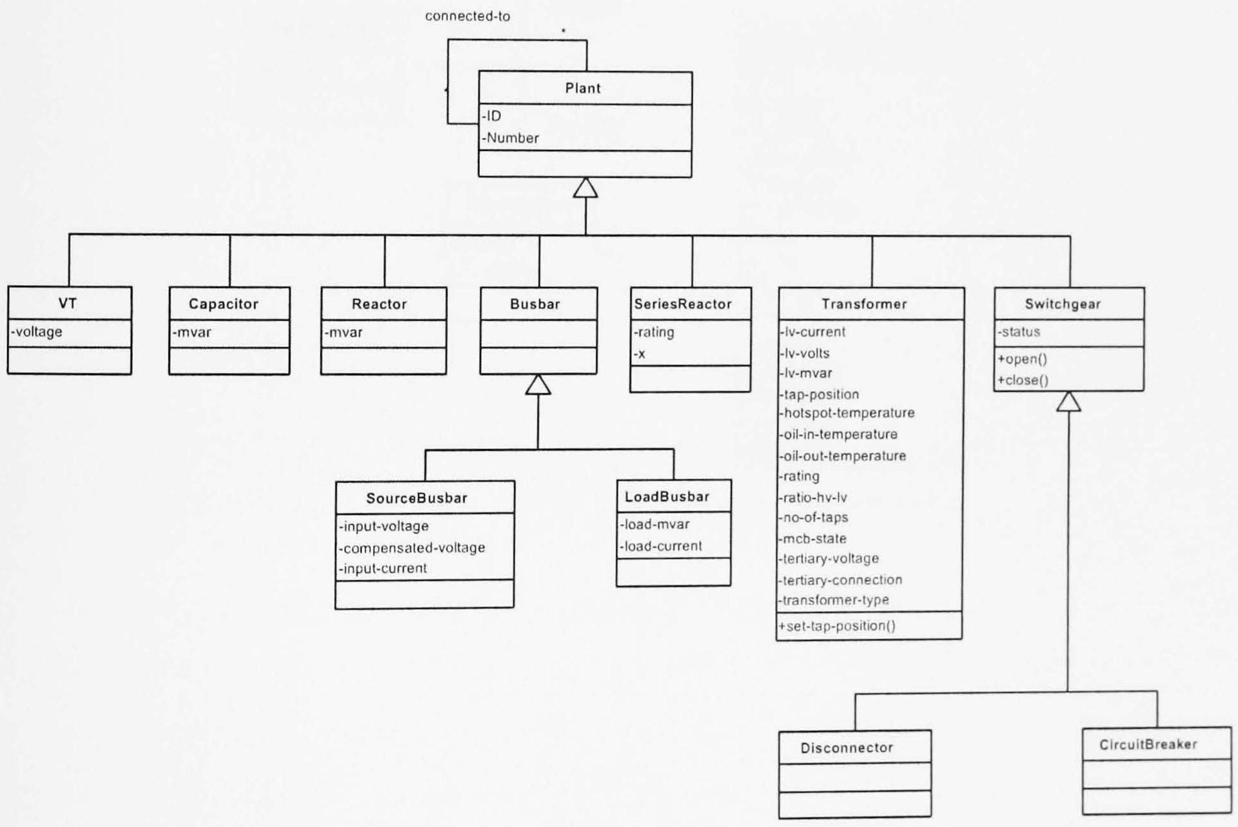


Figure 5.5: Ontology of Substation Plant (UML class diagram), Based on Substation Simulator Data and Transformer Monitoring System

Information Management Ontology The information management ontology, shown in Figure 5.6, provides the predicates used for document retrieval and querying. The properties of the *resource-description* class are based on the Dublin Core metadata standard[91].

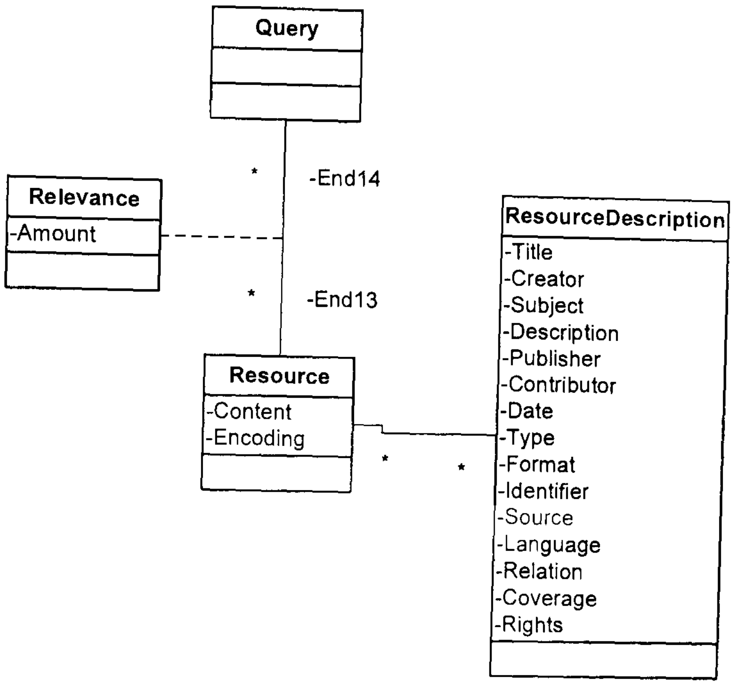


Figure 5.6: Basic Information Systems Ontology (UML class diagram)

This is a very basic ontology, which would not be suitable for all information retrieval applications, but fits with the needs of this system. Each document is considered to be a *resource*, which is described by a *resource description*. Also, a query has a particular *relevance* to each resource, which is represented by a real number between 0 and 1. Relevance is defined as a three-place relation:

$$relevance(Document, Query, Relevance)$$

Therefore, it is possible to query an information management agent, such as a document agent, using a query such as:

```
(all
    (sequence ?doc ?rel)
    (relevance
        ?doc
        (query :keywords
            (set ''transformer'' ''maintenance''))
        ?rel))
```

The above example means "Find all sets of a document *?doc* and relevancy *?rel* such that the relevance of *?doc* to the query "transformer maintenance" is *?rel*". In the example above, the query is represented as a frame containing a set of keywords. In the current implementation, this is abbreviated to just the set of keywords, as no other type of query is permitted.

5.4 Examples of Usage

This section provides several examples of how the system may be used to perform various tasks, including data querying, mobile agent data analysis and remote operation. Example FIPA ACL messages are provided to demonstrate the use of the FIPA standard protocols. Each ACL message has a sender, receiver, content (which is written in the FIPA Semantic Language (SL)) and protocol. In these examples, the agent name `user@pc2214:1099/JADE` represents the user agent. Certain message parameters (e.g. conversation-id, ontology) have been omitted from the messages for brevity. Also, searches of the directory facilitator (DF) are not shown. Each time an agent wishes to carry out an action or to submit a query, it will first search the DF to find other agents capable of processing that request or answering that query, unless an appropriate agent is already known and is still available (has not disconnected). This means that it is possible to substitute different agents providing the same information, and permits agents to be added and removed at runtime. In these examples, code appears in typewriter font (e.g. `code`) and comments in roman font.

5.4.1 Querying the IMU for a Data Set

In this system, a *data set* represents a series of events, each specifying the value of a property of an item of plant at a certain time. The user agent provides the user with the ability to define a data set to be retrieved using the name of the property and item of plant, a start time and an end time. In the future, additional criteria to define the data set could be added by modifying the user agent and graphical user interface.

Step 1: Select Plant Class

In order to assist the user in generating a query, the user agent provides a series of steps for query generation. The agent first retrieves the names of all plant classes (those that are a subclass of *plant*) from the ontology agent. This is done using the following message (assuming that the name of the ontology agent is `onto@pc2214:1099/JADE`):

```
(query-ref                                              A query message
    :sender
        (agent-identifier                               Sent by user agent
            :name user@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                           To ontology agent
                :name onto@pc2214:1099/JADE))
    :content                                            Query to find all plant
        ``((all ?a (subclass-of ?a plant)))''
        :protocol fipa-query)                      FIPA Query protocol
```

The ontology agent should then reply, providing the names of these classes. For example, suppose that the available classes are *transformer, circuit-breaker and switchgear*:

```
    (inform                                        Information message
        :sender
            (agent-identifier                      From ontology agent
                :name onto@pc2214:1099/JADE)
        :receiver
            (set
                (agent-identifier                  To user agent
                    :name user@pc2214:1099/JADE))
        :content
            ''(= (all ?a (subclass-of ?a plant))   Query results
                 (set transformer circuit-breaker switchgear))''
        :protocol fipa-query)                      FIPA Query protocol
```

The user agent then presents these classes to the user as a list, and the user selects a class in which they are interested. The system may then proceed to the next step.

Step 2: Select Individual Item of Plant

Once the user has selected a class of plant, the user agent will then retrieve the instances of that class from the static database agent. For example, suppose that the *transformer* class has been selected:

```
    (query-ref                                     Query message
        :sender
            (agent-identifier                      From user agent
                :name user@pc2214:1099/JADE)
        :receiver
            (set
                (agent-identifier                  To ontology agent
                    :name onto@pc2214:1099/JADE))
    :content                                        Find all transformers
        ``((all ?a (instance-of ?a transformer)))''
    :protocol fipa-query)                           FIPA Query protocol
```

Now suppose that the available instances of *transformer* are *sgt1, sgt2* and *sgt3*:

```
(inform                                             Information message
    :sender
        (agent-identifier                           From ontology agent
            :name onto@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                       To user agent
                :name user@pc2214:1099/JADE))
    :content                                        Query results
        ``((= (all ?a (instance-of ?a transformer))
                   (set sgt1 sgt2 sgt3)))''
    :protocol fipa-query)                           FIPA Query protocol
```

Step 3: Select Property

The user agent must now determine the properties of the selected object. This is a more complex operation than the previous two. The ontology agent holds information about the properties of classes, while the static database agent holds information about which classes a particular object belongs to. It is not sufficient in all cases simply to query the ontology agent for the properties of the class selected previously, because if the object is actually an instance of a subclass of the selected class, there may be properties of the object that are not properties of the selected class. Therefore, this step involves the integration of information from the two databases. This can be done either by the user agent or by the database agents; which of these alternatives is best is discussed in Section 6.2.2. Here we suppose that the integration is done by the database agents, and that the user agent only queries the static database agent[2].

[2]This is how this is currently accomplished in the prototype system.

Supposing that the object selected by the user in the previous step was *sgt1*, the following query will be sent to the static database agent (static@pc2214:1099/JADE) (note that a *slot* is equivalent to a property, but is the term used by the FIPA Ontology service):

```
(query-ref                                            Query message
    :sender
        (agent-identifier                             From user agent
            :name user@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                         To static DB agent
                :name static@pc2214:1099/JADE))
    :content ''((all ?a (slot-of ?a sgt1)))''         Find all slots of SGT1
    :protocol fipa-query)                             FIPA Query protocol
```

The static database agent must then do two things: retrieve the class of sgt1 from its own database, and then retrieve the properties of that class from the ontology agent. Having determined that sgt1 is an instance of *transformer*, the static database agent sends the following message to the ontology agent [3] :

```
(query-ref                                            Query message
    :sender
        (agent-identifier                             From static DB agent
            :name static@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                         To ontology agent
                :name onto@pc2214:1099/JADE))
    :content
            ''((all ?a (template-slot-of ?a transformer)))''
    :protocol fipa-query)                             FIPA Query protocol
```

The ontology agent will then reply with the properties of the *transformer* class. Suppose that these are *lv-current, lv-mvar* and *lv-volts:*

[3] In an optimal implementation, this is the message that will be sent. In practice, because of the *ad hoc* implementation of distributed backtracking in the prototype which does not propagate the *any* or *all* quantifier from the originating query, a series of query-ref messages will be used, containing *any* queries rather than *all* queries.

```
(inform                                         Information message
    :sender
        (agent-identifier                       From ontology agent
            :name onto@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                   To static DB agent
                :name static@pc2214:1099/JADE))
    :content                                    Query results
        ``((= (all ?a (template-slot-of ?a transformer))
                    (set lv-current lv-mvar lv-volts)))''
    :protocol fipa-query)[4]                    FIPA Query protocol
```

Finally, the static database agent will forward the slot names to the user agent:

```
(inform                                         Information message
    :sender
        (agent-identifier                       From static DB agent
            :name static@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                   To user agent
                :name user@pc2214:1099/JADE))
    :content
        ``((= (all ?a (slot-of ?a sgt1))        Query results
                    (set lv-current lv-mvar lv-volts)))''
    :protocol fipa-query)                       FIPA Query protocol
```

Step 4: Retrieve Data Set

Once the user has selected the object, property, start time and end time of the data set, this information is converted into a FIPA ACL query and forwarded to the IMU agent. For example, suppose that the user has selected *sgt1* as the object, *lv-current* as the property, 13/3/03 12:00:00 as the start time and 15/3/03 12:00:00 as the end time. The user agent will then query the IMU agent with the following message:

[4]As for the query message, in the current implementation there will be many of these messages. It is hoped that this issue may be resolved in a later implementation.

```
(query-ref                                          Query message
    :sender
        (agent-identifier                           From user agent
            :name user@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                       To ontology agent
                :name onto@pc2214:1099/JADE))
    :content ''((all (set ?a ?t)   Query for current values be-
tween specified times
                    (and
                        (t (lv-current sgt1 ?a) ?t)
                        (and
                            (?t > 13032003T120000000)
                            (?t < 15032003T120000000))))''
    :protocol fipa-query)                           FIPA Query protocol
```

The IMU agent will then (using the Prolog rules of its reasoning engine) convert this into a call to the IMU web service to retrieve the relevant data, which will be passed back to the user interface agent. The user agent then displays the data on the screen as a graph and table.

Discussion

The database querying procedure functions correctly in the prototype implementation, and is relatively simple to use. The main problem lies in the performance of this process when handling large data sets. This is due to both the method of retrieving data from the database using backward chaining, which means that only one row of data at a time is retrieved, and the use of the string-based SL language, which generates large messages in comparison to binary encodings and requires computing time to be expended in constructing and parsing the messages. It can be seen from the mobile agent data analysis benchmark in Section 4.2.3 that the wrapper-based mobile agent (which used the database agent described here) performed approximately 3.4 times worse than a mobile agent with direct database access and data caching (probably close to the optimal solution). However, this does not show how much of this performance difference is due to data retrieval and how much to the use of string-based messaging. The problem of slow data retrieval might be alleviated by using a different implementation of the server agent, which could retrieve multiple rows at a time. However, all string-based data representations, including the commonly used XML format, encounter performance problems, and so these would be more difficult to solve. For example, [122] describes problems caused by the XML format generating large data files and creating overheads in terms of parsing and transformation. The only solution to this problem would appear to be the use of binary messages, which would remove the advantages of using an explicit, standardised and implementation-independent knowledge representation.

5.4.2 Mobile Agent Based Analysis of Data

The mobile agent based data analysis proceeds in three steps. Firstly, the data set or data sets to be analysed must be defined. The configuration file for the mobile agent is then generated by the user interface agent, and the mobile agent is launched. Finally, the mobile agent carries out the analysis and displays the results to the user.

Step 1: Define Data Sets and Analysis Report

For a mobile agent based data analysis, the steps used to define a data set are the same as those for a database query, apart from that a mobile agent is capable of analysing multiple data sets. Therefore, the graphical user interface used is the same except for buttons allowing the user to move between data sets. Each data set is defined as described in Section 5.4.1, Steps 1-3.

Once the data sets have been selected, the user must specify the analyses to perform. This is done by selecting an analysis function, and then specifying the data set on which this function will operate, and the element of the data set that will be used (either the plant property or the time). Some analysis functions may operate on multiple arguments, which may be from the same or different data sets. The user then specifies the report by entering any text which should appear with the analysis results.

Step 2: Launch Mobile Agent

Once the user has entered all of the details required to perform the analysis procedure, the user agent generates a configuration file and launches the mobile agent described in Section 4.2. When started, the agent reads the configuration file. It then retrieves each data set in turn, carrying out each analysis as soon as all of its required data sets have been retrieved. The messages used to retrieve the data set are the same as those described in Section 5.4.1, Step 4, except that the role played by the user agent is played by the mobile agent. Once all analyses have been performed, the report is generated.

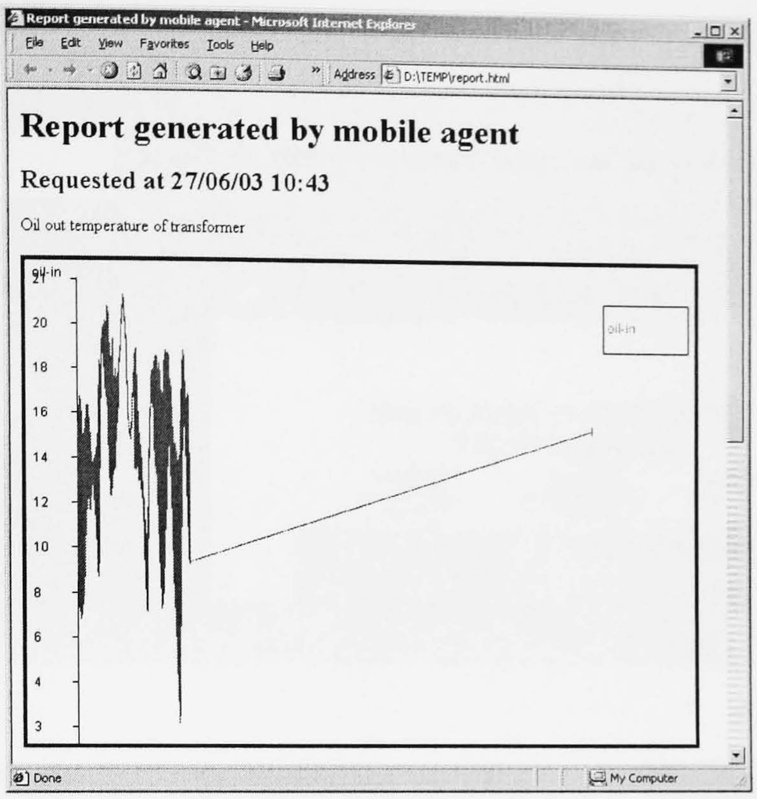
Step 3: Display Results

The results of the analysis are displayed by the mobile agent using the system's default Web browser, as shown in Figure 5.7. The large gap between the sequence of data points to the left of the graph and the single point to the right is due to a discontinuity in the example data being used.

5.4.3 Searching for Documents

As only a single document agent is used in the prototype system, there is no need to attempt the integration of results from multiple sources, which is a difficult problem into which research is still ongoing [106][123][124]. To search for documents, the user first inputs a set of keywords into the HMI (the text entry box marked by "A" on Figure 5.8)[5]. These are then transmitted to the user agent via the DataSocket connection. The user agent then sends a message to the document agent, requesting it to inform the user agent of all documents relevant to that query. For example, suppose that the query chosen

[5]The HMI interface shown was developed by Jun Qiu Feng, Intelligence and Automation Research Group, Department of Electrical Engineering and Electronics, University of Liverpool. The user agent was developed by the author.


Report generated by mobile agent - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address D:\TEMP\report.html
oil-in
21
20
18
16
14
12
10
8
6
4
3
oil-in
Done
My Computer

Report generated by mobile agent

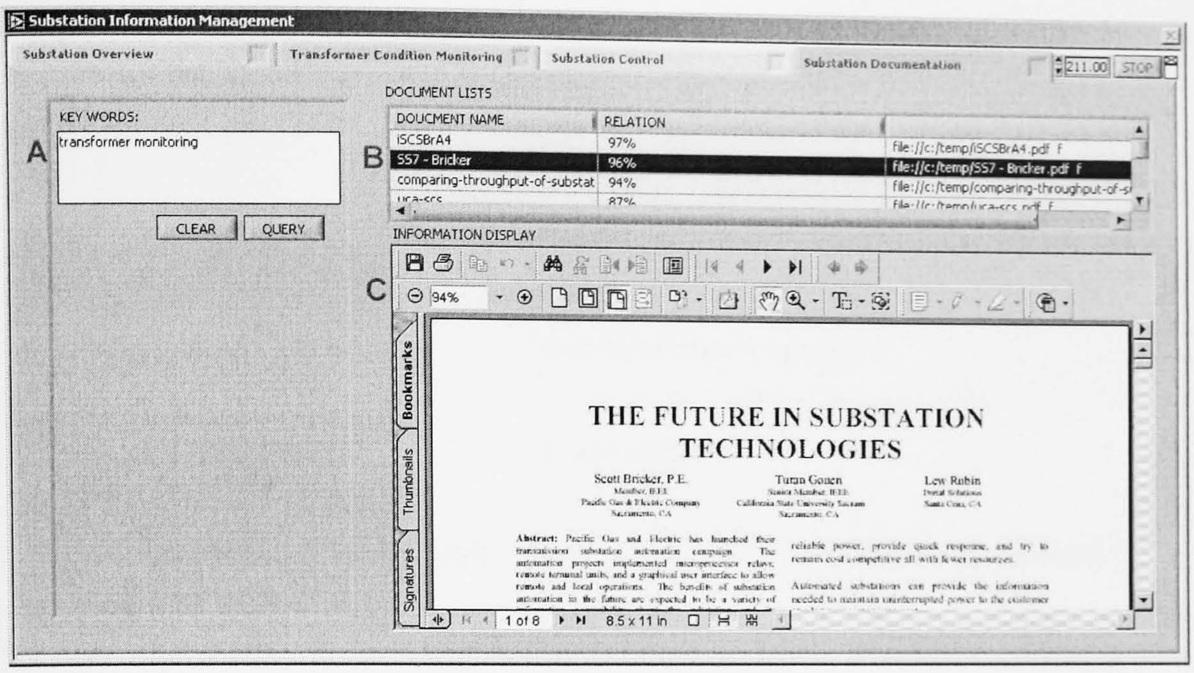
Requested at 27/06/03 10:43

Oil out temperature of transformer

Figure 5.7: Report Generated by Mobile Agent

by the user was "transformer monitoring". The following message might then be sent:

```
(request                                      Request message
    :sender
        (agent-identifier                     From user agent
            :name user@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                 To document agent
                :name doc@pc2214:1099/JADE))
    :content ''((all                          Query for documents
                   (sequence ?d ?r)           relevant to
                   (relevance                 transformer monitoring
                       ?d
                       (set ''transformer''
                            ''monitoring'')
                       ?r)))''
    :protocol fipa-query)                     FIPA Query protocol
```



KEY WORDS: transformer monitoring

DOUCMENT NAME	RELATION	
iSCSBrA4	97%	file://c:/temp/iSCSBrA4.pdf f
SS7 - Bricker	96%	file://c:/temp/SS7 - Bricker.pdf f
comparing-throughput-of-substat	94%	file://c:/temp/comparing-throughput-of-s
uca-scs	87%	file://c:/temp/uca-scs.pdf f

Figure 5.8: User Interface for Document Querying

This would be followed by a reply from the document agent, giving the resource descriptions of any relevant documents. The names and relevancies of these documents are passed from the user agent to the user interface and displayed in the list box marked “B” on Figure 5.8.

```
(inform                                             Information message
    :sender
        (agent-identifier                           From document agent
            :name doc@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                       To user agent
                :name user@pc2214:1099/JADE))
    :content ''((=
                    (all                            Query
                        (sequence ?d ?r)
                        (relevance
                            ?d
                            (set ''transformer''
                                 ''monitoring'')
                            ?r)))
                    (set                                Query results
                        (sequence
                            (resource-description
                                :title iSCSBrA4)
                            0.9705195)
                        (sequence
                            (resource-description
                                :title ''SS7 - Bricker'')
                            0.95818204))
                    ))''
        :protocol fipa-query)                           FIPA Query protocol
```

The user agent may then request the document agent to transmit the contents of a document to it. These will be encoded using the Base64 encoding. The protocol used is *fipa-request*, and the message format is [6]:

```
(request                                        Request message
    :sender <user agent>                        From user agent
    :receiver (set <document agent>)            To document agent
    :content ''((action <document agent>        Retrieve documents
                (retrieve <resource description>)))''
    :protocol fipa-query)                       FIPA Query protocol
```

The document agent then sends an *agree* message to the user agent, as specified by the fipa-request protocol, and following that encodes the document and transmits it to the user agent as an *inform* message[7]:

```
(inform                                         Information message
    :sender <document agent>                    From document agent
    :receiver (set <user agent>)                To user agent
    :content ''((result                         Set of documents
                <action>
                (resource
```

[6]where <user agent> is replaced by the agent identifier of the user agent, <document agent> by the agent identifier of the document agent and <resource description> by the resource description of the document required

[7]<action> is the action expression contained in the previous message. <document> represents the encoded document content. <format> is the format of the file, expressed as a MIME type (e.g. "application/pdf").

```
                              :content <document>
                              :encoding base64
                              :format <format>
                              :title <title>
                          )))''
```

5.4.4 Performing an Action Using the Data Acquisition Agents

To set the value of a property of an item of plant, the *fipa-request* protocol is used. The user agent must first send a *request* message to the appropriate plant agent (if the appropriate agent is unknown, it can be located using the DF). For example, suppose that the user wishes to open a circuit breaker "x10", and that the name of the plant agent is `x10@pc2214:1099/JADE`. The user first sends a request to the user agent via the graphical user interface shown in Figure 5.9[8].

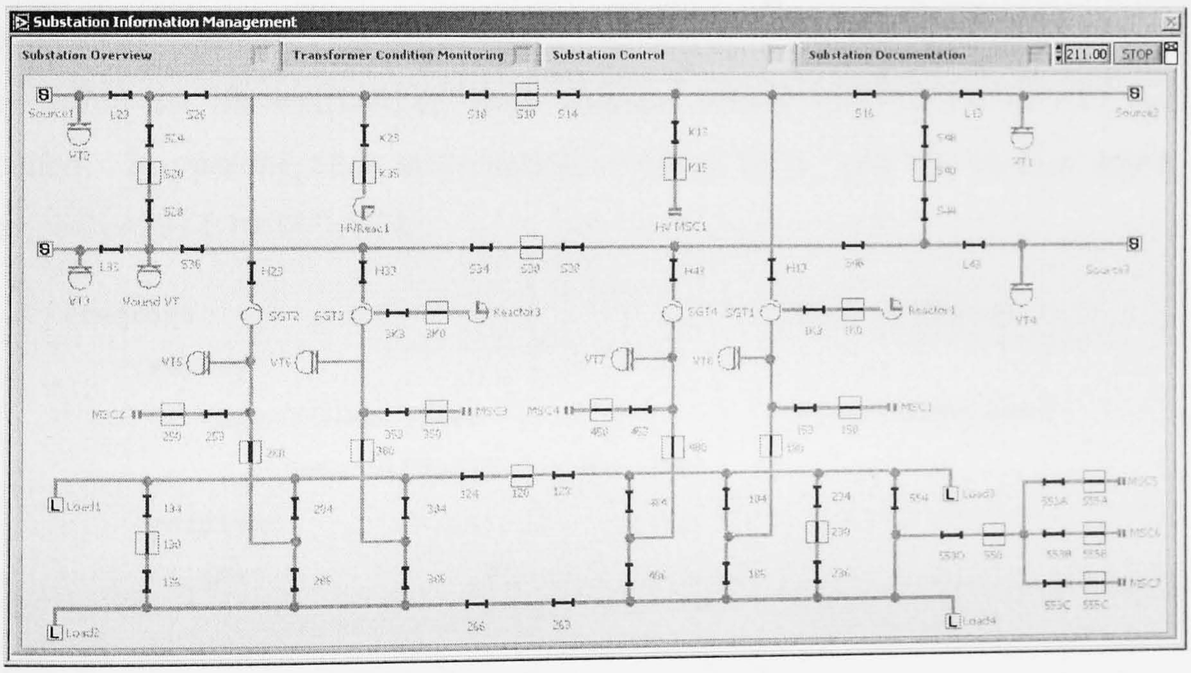


Figure 5.9: Graphical User Interface for Substation Control

The user agent then sends the following message to the plant agent (there

[8]User interface implemented by Jun Qiu Feng.

may also be other message parameters used such as *conversation-id* or *reply-with* if the user agent wishes to track the conversation using these parameters):

```
(request                                          Request message
    :sender
        (agent-identifier                         From user agent
            :name user@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                     To breaker agent
                :name x10@pc2214:1099/JADE))
    :content
        ''((action
                (agent-identifier                 Open breaker
                    :name x10@pc2214:1099/JADE)
                (open x10)))''
    :protocol fipa-request)                       FIPA Request protocol
```

The plant agent will then consult its mapping rules to determine the data acquisition node, device and channel responsible for the control of x10, and the value to be written to that channel which will result in x10 being opened. Supposing that the channel used is DC1, and the device agent is device@pc2214:1099/JADE:

```
(request                                          Request message
    :sender
        (agent-identifier                         From breaker agent
            :name x10@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                     To device agent
                :name device@pc2214:1099/JADE))
    :content
        ''((action
                (agent-identifier                 Write value to channel
                    :name device@pc2214:1099/JADE)
                (write-value dc1 0)))''
    :protocol fipa-request)                       FIPA Request protocol
```

The device agent will then carry out the request, and send an *inform* message to the plant agent to notify it that the action is complete. The plant agent will then notify the user agent in the same way. To do this, the *fipa-request* protocol is used. The message format is:

5.4.5 Reading a Plant Property Using the Data Acquisition Agents

To read a property, the *fipa-query* protocol is used. The user agent first locates the appropriate plant agent. Then the user agent sends a *query-if* or *query-ref* message to that agent (depending on whether the user wishes to confirm the value of a property or to find out what the value of that property is). For example, suppose that the user wishes to determine the *lv-current* (Low voltage current) of transformer *sgt1*, and that the name of the plant agent is `sgt1@pc2214:1099/JADE`. The following message would then be used:

```
(query-ref                                           Query message
    :sender
        (agent-identifier                            From user agent
            :name user@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                        To transformer agent
                :name sgt1@pc2214:1099/JADE))
    :content ''((iota ?a (lv-current sgt1 ?a)))''  Get cur-
rent value
    :protocol fipa-query)                            FIPA Query protocol
```

The plant agent must then use its mapping rules to locate the appropriate device agent and channel. Suppose that the channel name is SL1, and the device agent is device@pc2214:1099/JADE. The plant agent will then send the following message:

```
(query-ref                                           Query message
    :sender
```

```
        (agent-identifier                       From transformer agent
            :name sgt1@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                   To device agent
                :name device@pc2214:1099/JADE))
    :content ''((iota ?a (value sl1 ?a)))''     Get value of channel
    :protocol fipa-query)                       FIPA Query protocol
```

The device agent will then read the value of the SL1 channel, and send a reply to the plant agent. Supposing that the value is 12.5:

```
(inform                                         Information message
    :sender
        (agent-identifier                       From device agent
            :name device@pc2214:1099/JADE)
    :receiver
        (set
            (agent-identifier                   To transformer agent
                :name sgt1@pc2214:1099/JADE))
    :content                                    Value of channel
        ''((= (iota ?a (value sl1 ?a)) 12.5))''
    :protocol fipa-query)                       FIPA Query protocol
```

Finally, the plant agent sends a similar *inform* message to the user interface agent.

5.5 Implementation Issues

Initially it was intended to use the Information Management Unit (IMU) to provide the sole interface between the information management system and the substation simulator. However, this proved impossible because of the fact that theIMU did not provide "subscription" functionality - that is, it was not possible to have the IMU automatically update its agent when new information arrived. This meant that event updates had to be handled by periodically "polling" the IMU to retrieve new data, which is a relatively inefficient mechanism.

Introducing the data acquisition multi-agent system corrected this problem by providing direct access between the multi-agent system and the data acquisition device. However, this might cause problems in an operational deployment if, for security or other reasons, it was decided not to allow agents to access devices. This would mean that event functionality would either have to be implemented by polling or omitted from the deployed system.

Agents that require to be updated when new information becomes available in the system (for example, the plant agent needs to know what information is available from device agents regarding its item of plant) should do so by establishing a subscription with the DF to be informed of agents joining or leaving the system. This is not possible with the current system implementation but is available in newer versions of the JADE toolkit.

To allow agents not based on JADE (such as the user agent) to locate the agent platform, a basic broadcast discovery mechanism based on UDP is used, as none is provided by FIPA. However, as the FIPA work on ad-hoc platforms progresses[9], it is possible that such a mechanism will be provided by FIPA platforms.

5.6 Summary

This chapter has described a prototype system based on the architecture of Chapter architecture-chapter. The various agents described in Chapter 3 are implemented using the JADE platform, and a reasoning engine and knowledge base based on Prolog. Use of the Prolog language provides relatively simple conversion to and from FIPA SL, as both are based on first-order logic. The prototype demonstrates the feasibility of the architecture and shows how it may be implemented. It also reveals some of the problems involved in this implementation. For example, integrating the multi-agent system with the

[9]www.fipa.org/activities/ad_hoc.html

HMI platform written in LabVIEW was a particularly time-consuming task, involving the implementation of a user agent in Visual Basic (in order to access ActiveX controls) and linking this agent to the JADE platform using a TCP/IP connection.

The next chapter presents an evaluation of the architecture, using both theory and the experience provided by the implementation of this prototype. It examines whether the architecture provides the required functionality, and considers its ability to be modified easily in response to changes in the substation plant or in the data acquisition system.

Chapter 6

Evaluation and Analysis

This chapter presents a brief evaluation of the architecture described in Chapters 2 and 3. Where applicable, experience from the prototype described in Chapter 5 is used. The software engineering quality attributes of performance, modifiability, availability and security [125] are considered with reference to the described architecture. Different authors in the software engineering literature use different variations on these quality attributes, for example, Sjeko [126] uses a more complex quality model consisting of dependability (including safety and security), satisfaction, functionality, flexibility (including modifiability) and performance. The criteria used here are drawn from both of these sources.

It is very difficult to evaluate the information management functionality of the system. The system's document retrieval function is based on standard algorithms, and therefore the use of information retrieval metrics such as precision and recall would test only these algorithms and not the architecture itself. It can be shown that the database retrieval function is capable of retrieving data from a database. However, the data integration functions are largely based on previous work in other domains such as Infomaster [100] and RETSINA [59], and evaluating data retrieval performance would largely test the implementation, which is not of production quality, and not the architecture. The preferred method of evaluation would be to install the system in a substation and request that the substation engineers compare it with other systems and existing technology. However, the time that would be required to implement the system to a standard suitable for such a test meant that this was not possible. Therefore, this chapter presents a largely theoretical evaluation based on software engineering principles. This is sufficient to draw certain limited conclusions about the flexibility and modifiability of the architecture. Further research should involve comparison to other systems and user-oriented testing.

6.1 Functionality

The functionality provided by a system implemented using the proposed architecture should, at least, be capable of matching that provided by a traditional, object-oriented or Web-based HMI/SCADA substation automation system. Various criteria taken from the literature are now used to describe the desired functionality and compare it to that provided by the prototype system. However, it is important to note that the work described in this thesis has developed an architecture for the construction of industrial automation systems, rather than a specific automation system. The prototype is intended only to demonstrate certain functionalities of the architecture, and not to include every feature of a full substation automation system. Therefore, many of the features described in these criteria have not been implemented. Where this is the case, the way in which a feature might be implemented using the multi-agent system is discussed.

6.1.1 National Grid Company Requirements for Substation Control Systems

The National Grid Company requires a substation control system to be capable of fifteen functions[83]. For each of these functions, Table 6.1 considers how they might be implemented using the multi-agent architecture described in this thesis.

Therefore, the current system provides only a subset of the functionality

Requirement	Implementation using multi-agent system
Local control	Performed using the operator intervention process through HMI, user interface agent, plant agents, device agents and IEDs.
Telecontrol	Performed using the same process as local control. Mobile agents may also be used for predetermined sequences of interactions
Alarm annunciation	May be performed by a task agent using data supplied by plant agents to generate alarms
Data archiving	Uses data logging database or Information Management Unit
Synchronisation Plant Performance Monitoring Delayed Auto Reclose Switching Automatic Tap Change Control Automatic Reactive Switching Fault Recording Primary System Monitoring Sequential Isolation Switching Interlocking	These functions are not implemented in the prototype. In a full system, they would be implemented either using task agents (one agent for each task) or by modifications to the control rules of the plant agents.
Database Creation and Amendment	Performed by data storage agents
Diagnostic Facilities	This function is not currently implemented and should be considered in further work

Table 6.1: National Grid Company Requirements

provided by a substation control system. However, the focus of this research has been on the design of a generic architecture, rather than the implementation of a complete system. Therefore, a number of functions are abstracted in the architecture as generic "task-specific agents".

6.1.2 Haacke "Opportunity Matrix"

Haacke et al [127] provide a list of "candidate functions" for a substation automation system which aim to address the needs of a power operating company, and are not (in their view) currently being met. These are as follows:

1. Equipment condition monitoring

2. Automatic load restoration
3. Dynamic transformer ratings
4. Adaptive relay settings for distribution circuits
5. Power system disturbance and power quality data
6. Feeder automation support
7. Expert alarm processing
8. Access to substation metering data
9. Access to power company documents and systems
10. Corporate data repository
11. Additional SCADA quantities
12. Adding SCADA to non-SCADA substations
13. Training simulator

Of these suggested needs, the multi-agent architecture may contribute to three:

- Access to documents and systems (9): The use of a multi-agent architecture and IP networks allows access to external systems, via the use of wrappers. For example, the document agent provides access to documents. However, it would also be possible to implement this function using a client-server or distributed object approach.
- Data repository (10): The multi-agent system maintains individual data repositories in each substation. However, the use of mobile agents provides a capability to integrate data from different databases, hence providing a type of "distributed data repository".
- Additional SCADA quantities (11): By adding additional provider agents to the system, additional quantities may be made available at runtime.

Additionally, equipment condition monitoring (1) and expert alarm processing (7) might be implemented using a multi-agent methodology, but are not currently part of the system or architecture described in this thesis.

6.1.3 Summary of Functionality Results

Overall, the multi-agent system, extended with additional agents, should prove capable of providing all of the functionality provided by a traditional automation system. However, at least without the addition of artificial intelligence, it does not offer any significant additional functionality that cannot be implemented by other means, although, as stated by [54] for mobile agents, it does provide a consistent framework for the implementation of different functions and tasks, such as control and information management. In order to be a useful technology, the multi-agent system should also provide advantages in other areas, such as performance, modifiability or ease of system development.

6.2 Performance

It has been shown in Section 4 that, in certain circumstances, mobile agents provide increased performance over client-server alternatives. Therefore the use of mobile agents is not discussed here. However, other aspects of the architecture's performance are considered. Firstly, we consider the performance of the architecture for data acquisition and control tasks. We then consider the response time of the system, when performing information management tasks, to user queries.

6.2.1 Data Acquisition Performance

Because of the two-layered structure of the data acquisition system, shown in Figure 6.1, when new data arrives it must be first acquired by the device agent, and then passed to the relevant control agent. This means that, for control applications in which a control agent must respond to events, it is necessary for the data to pass through at least one intermediate agent (the device agent) before reaching the control agent. This could be a problem if it is necessary to achieve very fast control.

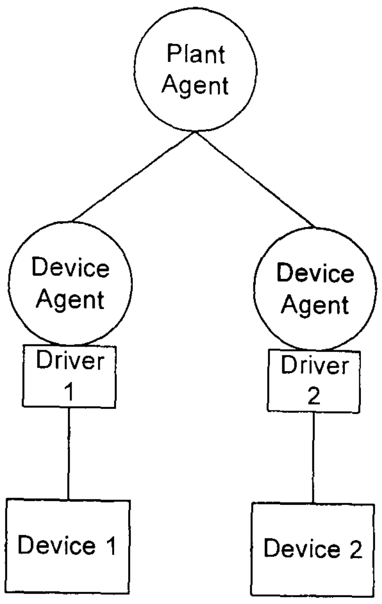


Figure 6.1: Plant Agent and Device Agents

An alternative structure, shown in Figure 6.2, would be for the control agents to retrieve information directly from the data acquisition devices, using device-specific communications protocols or APIs. However, this compromises the modularity and information abstraction provided by the separate device agents and control agents, as it requires a control agent to have knowledge regarding a number of (possibly heterogeneous) data acquisition devices, and a number of separate capabilities and protocol drivers for accessing these devices.

The performance of different configurations of device agent and control agent is now analysed in greater detail.

Message Passing

Suppose that a plant agent a wishes to send a command *cmd* to a device d over a network of bandwidth τ and latency δ, and receive a reply *ack* when the command is complete. The procedure taken is as follows:

1. Plant agent sends message to device

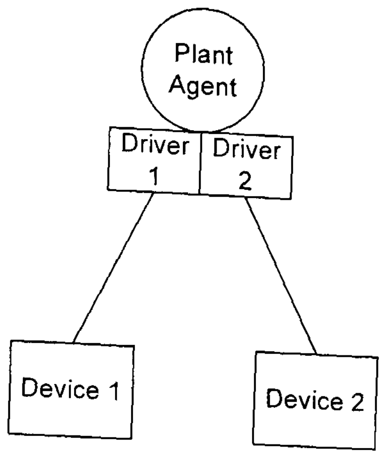


Figure 6.2: Plant Agent with Direct Access to Devices

2. Device carries out action

3. Device sends acknowledgment to plant agent

Now suppose that instead of transmitting the command directly to the device, the plant agent communicates first with an intermediate device agent *da*. In this case, the following procedure must be carried out:

1. Plant agent sends message to device agent

2. Device agent processes message

3. Device agent sends message to device

4. Device carries out action

5. Device sends acknowledgment to device agent

6. Device agent processes acknowledgment

7. Device agent sends acknowledgment to plant agent

In the first scenario, two messages are sent (a to d, d to a). In the second scenario, four messages are sent (a to da, da to d, d to da, da to a). For each message, there is a cost to encode the message in a particular protocol (either a device protocol or an ACL) and a cost to interpret the message, along with the cost of transmitting it across the network. However, it is not necessarily the case that all messages must be sent across the network. Whether this is the case for different configurations of plant agents and device agents will now be examined.

Co-location Scenarios

In order to determine the number of messages transmitted across the network, it is necessary to determine whether any of the three objects involved (device agent, control agent and device) are co-located. In the general case, if we accept that each item of plant may be managed by multiple devices and each device may manage multiple items of plant, it is not possible for all three to be co-located. Suppose agent a manages item of plant p_1, which is managed by devices d_1 and d_2. If these two devices are not at the same node ($location(d_1) \neq location(d_2)$), it is impossible for a to be co-located with both.

Although, conceptually, it would be preferable to locate the device agent on the device, it would be possible to co-locate the device agent with the control agent, although this would only be the case if either each device agent corresponds only to one control agent, or all control agents corresponding to a particular device agent may be located with that device agent (Figure 6.3).

In this case, only the messages between the device agents and their respective devices are transmitted across the network.

If, instead, da and d are co-located (Figure 6.4), the messages between the plant agent and the device agents are transmitted across the network, whereas the messages between the device agents and the devices are transmitted locally. The number of messages transmitted across the network for a given interaction is the same in both cases. Therefore, any performance difference is due to the relative efficiency of the agent-agent and agent-device protocols. Also, it may not always be possible to locate an agent on a device.

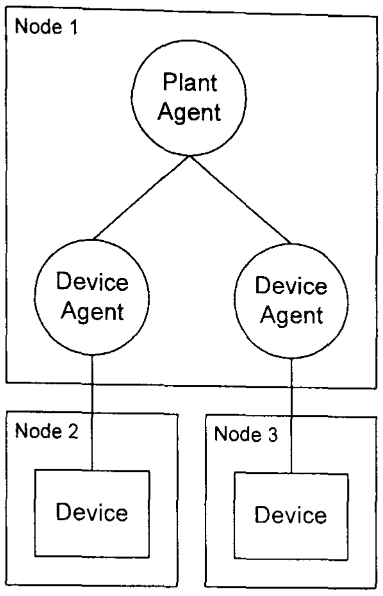


Figure 6.3: Plant Agent Co-Located with Device Agents

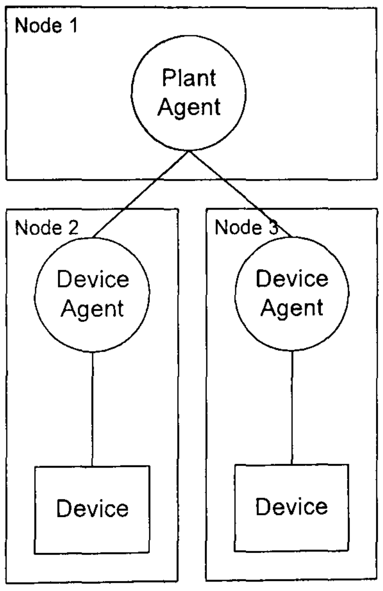


Figure 6.4: Device Agents Co-Located with Devices

Other Factors

Other factors affecting the performance of the data acquisition system include the relative inefficiency of general agent communication languages compared to specific protocols designed for data acquisition and control. This results in increased message size and in longer parsing times.

6.2.2 Responding to User Queries

The time taken to respond to a query from the user varies widely depending on the type and complexity of the query, and also on whether it relates only to one substation or to multiple substations. It also depends on the mechanism (mobile agent or static agent) used to answer the query, and whether information from multiple agents is required to produce an answer.

There are a number of possible bottlenecks involved in this procedure. The database wrapper agents might become overloaded if multiple queries were submitted simultaneously. Also, the directory facilitator might become overwhelmed by a large number of requests, although it might be possible to avoid this by introducing multiple federated DFs.

It has been observed that providing the database agents with the ability to interact as described in Section 3.1 produces a significant delay in query answering, due to the number of interactions between database agents involved in a typical backtracking procedure. It would be possible to reduce the load on the database agents by removing the ability for them to interact with each other, and having another agent (possibly the user agent or a broker) which integrated the information from multiple databases. However, the total number of agent interactions would not be reduced, and may in fact increase as the broker agent must interact with multiple database agents. An alternative strategy would be a modification of the inter-agent backtracking process, for example, to retrieve all the relevant results at once and then backtrack locally through the retrieved list. However, this would only reduce, rather than eliminate, the extra workload on the database agents.

6.2.3 Data Display

The National Grid specification for substation control systems [83] provides stringent performance requirements for the display of data in a substation control system, for example that data should be displayed on the VDU screen within a mean time of 2 seconds of a digital input changing state, with a standard deviation of 0.5 seconds. While it would not be possible to completely ensure that the system meets these requirements with a basic prototype implementation, it is necessary for the architecture to be designed in such a way that a more developed implementation would be able to do so.

The time to display data on the HMI of the system is affected by two things: the data acquisition speed, discussed in Section 6.2.1, and the amount of time taken for the user agent to retrieve data from the plant agents. The number of ACL messages involved in data display (at minimum) would be two:

1. Message from device agent to plant agent.
2. Message from plant agent to user agent.

This assumes that subscriptions are set up between user agent and plant agent and between plant agent and device agent. Providing that all agents are located in the substation, it will be possible in an otherwise unloaded for these messages to be transmitted within the 2 second limit (in fact in a much shorter timescale). For example, the time taken to transmit a 9905 byte ACL message between two containers on the same computer in the experiment documented in Section 4.5 was 8.05 milliseconds (although longer time than this would be required to create the message and for the client to parse its contents). However, the effects of loading on the system have not been tested in the prototype, and remain an issue for further work.

6.3 Modifiability

One of the important claimed advantages of agent-based systems is their flexibility and ease of modification. For example, Ferber [8] states that multi-agent architectures are "especially suitable" for adapting to changes in the system in which it executes and to changing requirements.

6.3.1 Modifying the Substation

Replacement of Substation Plant

If an item of plant is replaced with a new one of the same type, then the details of the replacement item must be entered into the static database. If any of the monitoring devices related to the item of plant are changed, then the procedure described under "adding an IED" must also be carried out.

Because the plant agent obtains its plant configuration from the substation databases, it is not necessary to re-program this agent, but it must be restarted in order to force it to re-read the configuration. This could be done remotely using the administrative interface of the agent platform. However, it might also be possible for the plant agent to provide a command which could be sent in an ACL message and would cause it to restart itself and re-read its configuration.

Addition or Removal of Plant

If a new item of plant is added of an existing category (already in the ontology) then:

- The details of the item of plant must be added to the static database
- A plant agent must be created for that item of plant. It is likely that only minor changes to the configuration of the control agent would be required, as it could be modified from an existing plant agent for a similar item of plant.
- The mapping rules relating to the item of plant must be added to the mapping database.
- The user interface must be modified to display the new item of plant.

If an item of plant is added which is of a category not already present in the substation, then additionally the details of that category of plant must be added to the ontology.

Furthermore, when an item of plant is added it is necessary to add a number of IEDs to monitor that plant. Therefore, for each of these IEDs the procedure described in "Adding a new IED" must be carried out.

Adding a New IED

When a new IED is added, the following steps must be taken:

- Program a device agent for that IED, or modify an existing agent.
- Add the IED configuration to the static database.
- If the IED monitors a property of an item of plant which is not present in the ontology, then the ontology database will need to be updated. It may also be necessary to update the user interface to show the new property.
- Add the mapping rules for the IED to the mapping database.
- Restart the relevant plant agent to reload its mapping rules.

6.3.2 Modifying the Data Sources

Adding a New Database

To add a new database to the system, a wrapper agent for that database must be implemented. At minimum, this involves creating a set of rules to provide a mapping from the schema of the database into the global system ontology[1]. If the database is to act not only as a source of data, but is also to store data gathered by other agents in the system, it is also necessary for the database agent to establish subscriptions for appropriate data with the agents providing that data[2].

[1]This is not true in the case where the database uses an identical schema to another database already present in the system, in which case the mapping defined for that database can be re-used.

[2]It might also be possible to use two database agents - one reading and one writing. This might reduce the problems caused by load on either of the agents.

Adding a Document Repository

Providing that the documents in a document repository are of a format which can be handled by an existing document agent, the addition of a document repository to the system should involve only the instantiation of an agent to manage that document repository. No programming or compilation should be required.

However, supposing that a future incarnation of the system provides not only "search engine" functionality, but also information extraction, it would be necessary to develop wrappers for the documents in the document repository to allow this extraction to be performed.

User and mobile agents will be able to locate the document repository automatically, without restarting, as the document agent will be registered with the directory facilitator[3].

Adding a New Type of Service

If a new service is to be made available of a type not already present, then the amount of work to be performed might be more substantial.

- If the service provided only query functionality, and the terms which could be queried were already available in the global ontology, then a wrapper agent would have to be written for the service. That wrapper agent would register with the directory facilitator, providing the list of available queries, and other agents would then automatically discover it and be able to query the service.
- If the service provides query functionality but additions to the global system ontology are required, the process is more complex. Firstly, the global ontology must be updated to include these new terms. Any agents which need to use these terms may also have to be updated, unless the

[3]This assumes that agents do not cache DF entries, and either search the DF each time they perform a document search, or subscribe to the DF for notification of new registrations (this is possible in certain versions of JADE and other platforms, but is not yet in the FIPA specifications).

agent automatically discovers them from the ontology database, as is the case with the user interface agent (for mobile agent generation) and the plant agent (for information management only - if the information was to be used in control the control algorithms/rules of the plant agent would need to be altered as they would not take account of this information).

- If the service provides functionality other than simple data querying (for example prediction) then any client agents which need to make use of this service should be updated, as they will not be aware of the functionality that this service is capable of providing. The most likely agent to require alteration is the user agent.

6.3.3 Modifying the User Interface

Adding a New User

Adding a new user to the system would not currently require any changes to be made. However, in a system in which security was implemented it would be necessary to add any required details of the user (e.g. username and password) to the authentication service.

Adding New Capabilities to the User Interface

To add capabilities to the user interface requires modifications to the user interface and possibly the user agent only, unless a new service is required that is not currently provided by the multi-agent system. For example, adding the ability to control the substation from the user interface in the prototype system required the addition of new input buttons, and the modification of the user agent (to read commands and transmit them to other agents) and the interface between the user agent and user interface.

6.3.4 Summary

As described above, the addition of a new item of plant or service requires relatively few changes in the multi-agent system. For example, alterations to an item of plant require only the device agent (and the configuration databases) to be modified. This is an advantage compared to a centralized system, in which modifications affect the entire system. However, the major drawback of the system described here is that many changes require alterations to the user interface or user agent (in order, for example, to display new items of plant on the one-line diagram) rather than just the agents directly responsible for an item of plant or database.

6.4 Security, Reliability and Availability

It is difficult to evaluate the security, reliability and availability of a prototype application. Therefore, it is necessary to consider similar results from other agent-based applications and architectures.

As far as security is concerned, the use of a static agent-based approach should not introduce additional problems beyond those introduced by a client-server model, providing that all agents in the system are developed and owned by a single entity (the power company). There are security and trust issues associated with *open* multi-agent systems, in which agents are developed by different entities; these issues are mentioned in [128].

Those security problems posed by the use of mobile agents should be relatively limited in the power systems domain, provided that all mobile agents originate within the company itself (the origin of a mobile agent could be verified using digital signatures as described in [58]). If it is not permissible for external users to submit mobile agents to the system, the problem of malicious mobile agents should be reduced. The problem could be reduced even further by preventing users from implementing their own mobile agents, and allowing them to use only the pre-implemented mobile analysis agent and mobile remote control agent. However, this would require that a large function library was available for the analysis agent, to provide all of the analysis functionality required by the different users. This process could be helped by extending the capability of the analysis agent to permit analysis functions to operate on the results of other analysis functions, rather than only on retrieved data sets. This would reduce the number of different functions required.

Fedoruk and Deters [129] suggest that the "brittleness" of multi-agent systems is a main contributor to their lack of deployment in industry, and that this brittleness is caused by the fact that there is no centralized control of a multi-agent system, and that therefore it is "difficult to detect and treat failures of individual agents". They found that introducing replicated agents significantly reduced the failure rate of a multi-agent system, but that this was at the price of increased system load. In a system such as a power system, in which reliability is important, it would probably be necessary to introduce replicated agents, possibly in addition to other techniques such as "watchdog" agents or agents which monitor each other's behaviour during task execution [130].

Failure of an Agent

One possible cause of failure in the system is the failure of one or more individual agents. If one agent establishes a subscription with another for some item of information, the subscriber will only receive messages from the provider when that item of information changes. Therefore, if the provider agent fails, the subscriber will not have any means to detect this, and will probably be unaware of any further changes to the item of information in question.

Another possible cause of failure is that the DF entries of agents remain after they have failed. This means that an agent may attempt to contact an agent that no longer exists, but which still has a DF entry. This problem is solved by the use of leasing (similar to that implemented by Jini) in the latest version of the FIPA specifications.

6.5 Integration Into Existing Substations

One measure of the practical usefulness of the architecture described in this thesis would be the ability to integrate it into existing substations which already have a substation control system or automation system.

The complete multi-agent system relies on access to the substation IEDs for monitoring and control. This means that for each IED, a protocol driver must be available. There would also be problems involving the development of task-oriented agents to replace functions of the substation control system, and the development of algorithms for control agent cooperation. Finally, there might be security concerns involved with the open architecture and the use of mobile agents.

However, it is possible to make more limited use of the multi-agent system to carry out only the task of monitoring and information management.

6.6 Possible Applicability to Other Industries

In the initial requirements for the architecture described in this thesis, it was desired that it should be as generic as possible, and able to be applied to a number of different industries. As a possible example, we consider the hospital ward described in [131]. In the hospital, there are seven wards, each with up to six babies. Each baby is connected to a monitoring system which continuously monitors a number of numerical parameters such as the electrocardiogram (ECG) waveform. The monitoring system also generates derived knowledge, such as the heart rate in beats per minute which is derived from the ECG. Each measured parameter is archived once per minute by an archival system [131, p2].

To adapt the architecture to this scenario, it is suggested that the plant agent be used to represent the baby, with a new configuration, set of control rules and ontology being defined for this purpose. The device agent would be used to represent the monitoring and control device.

The user interface agent should be able to be used without any modification, because the only ontological commitment it has to the substation domain is the use of the "plant" class, which could be kept as the root of the new domain ontology (covering humans). However, the human-machine interface itself would have to be rewritten.

For the archival system, a database agent would be used and appropriate mapping rules would have to be created. This supposes that the archival system uses a standard database. If not, a special "archival agent" would have to be created (as for the IMU in Chapter 5), but it would perform similar functionality to a database agent.

One problem with the portability of the architecture in this scenario is that the term "plant" does not easily apply to a human, and, in order for the system to function correctly using its built-in ontology, it would be necessary to define "baby" as a subclass of "plant". However, the difficulties posed by this should be mostly cosmetic, and would possibly be hidden from the user by the human-machine interface.

Another, more serious, problem, is that the measurements taken in the hospital scenario involve a high degree of uncertainty, which is not present in a power substation.

The conclusions drawn from this exercise have not been tested in practice. Therefore, during an actual implementation a number of difficulties might emerge which would necessitate changes to the architecture or to the individual agent implementations.

6.7 Discussion

From the evaluation in this chapter, a number of points can be made regarding the advantages and disadvantages of multi-agent systems in comparison to existing technologies for power system automation.

6.7.1 Advantages

Flexibility As demonstrated in Section 6.3, the use of a multi-agent system, and in particular the directory facilities provided, enhances the flexibility of the system by permitting new devices and items of plant to be added without changing the software of the rest of the system. However, this advantage is reduced in the power systems domain by the fact that the system does not change rapidly (items of plant are rarely added to substations). Also, in the current prototype it is necessary to restart several of the agents (plant agents and device agents) in order to change their configuration. It would be useful to add a feature enabling the agent to read updated configuration rules at runtime.

Autonomy The use of a multi-agent system provides a basis for the introduction of distributed control systems in which agents (perhaps representing items of plant or other entities) act in an autonomous manner without outside intervention. However, in the prototype system described here, agents do not exhibit significant autonomous behaviour, as it would be most useful in the automatic control task, which has not yet been implemented.

Inherent Distribution The inherently distributed nature of the power system means that a multi-agent system, which provides autonomy to its constituent components, is well-suited to this application domain. For example, the use of agents to represent objects such as transformers and circuit breakers is a natural "fit" to the system being controlled. However, a similar structure might also be obtained using a distributed object system.

Integration Using a multi-agent system provides a convenient framework to represent different tasks and to integrate different data sources. Rather than a number of separate software programs, all tasks are performed through the multi-agent system, enabling data to be shared between tasks. The use of a standard agent communication language provides a fixed communications mechanism which can be used by heterogeneous agents. However, there are other methods (such as distributed object systems) which might be able to achieve the same goals, although distributed object communications do not have the high-level semantic content of agent communication languages.

6.7.2 Disadvantages

There are several disadvantages of multi-agent systems when compared to other power system automation systems.

Management of Large Agent Societies In a substation containing a large number of items of plant, there is a correspondingly large number of plant agents (the same is true for devices). This creates a difficult task of managing these agents.

Difficulty of Integration with Devices Although the device agents provide a convenient interface to other agents in the system, the implementation of a device agent is still performed in a similar way to that which would be used for a component of a traditional industrial automation system, and it is necessary to write a specific device agent for each model of device. Therefore, a multi-agent system may not represent a significant improvement in this area.

Inflexibility of User Interface The major modifiability problems of the architecture that have been identified concern the user interface. It is not possible to automatically modify the user interface for a substation if the layout of the substation changes or new plant or data acquisition devices are added. Also, if new software services are added, the user interface and its agent must be modified to make use of these services.

It might be possible to minimize these problems in two ways: developing a method of automatically generating the one-line diagram and user interface for a substation from a logical description of that substation, and providing a more modular user interface to which new services could be added as they became available. The first of these might be similar to that developed by Qui and Gooi[4], who were able to generate one-line diagrams from a model of a substation. However, further work might be necessary to provide a means to automatically generate or modify the other elements of the user interface such as the menus, dialog boxes etc.

Possible Performance Problems The use of ACL messages may degrade the performance of the multi-agent system due to parsing and message passing overheads.

6.8 Summary

This chapter has presented an evaluation demonstrating both advantages and disadvantages to the use of a multi-agent system for power system automation. The major advantages of the multi-agent architecture derive from the use of directories. Because agents locate each other at runtime using a directory service based only on their capabilities, it is possible to add and replace components during the operation of the system. This means that, for example, the user interface agent is capable of obtaining data either from a database or from a plant agent without modification. Another important facet of the multi-agent system is the standard agent communication language, whose defined semantics permit the integration of data from multiple sources. It is also considered that agent autonomy will prove useful in implementing distributed control schemes. However, these are not covered in the prototype. The main disadvantages arise from the complexity of the multi-agent implementation, which consists of a large number of agents and can prove difficult to administer. There are also performance problems, both specific to the implementation described here and generic problems relating to, for example, agent communication. Also, the use of a distributed system can result in increased network traffic and communications overhead compared to a centralized implementation.

Further work is required to fully complete this evaluation. Firstly, it is necessary to install the system in a substation or substations in order to perform a user-focused evaluation involving substation engineers. From the evaluation performed, it is also unclear how the reliability of a multi-agent system compares to that of a traditional system. It would be necessary to examine this question more fully once reliabilty mechanisms such as redundancy and fault recovery have been developed and incorporated into the prototype.

The next chapter will conclude the thesis by examining the results that have been obtained and suggest opportunities for further work on this architecture and on individual applications.

Chapter 7

Summary, Conclusions and Future Research

7.1 Summary

This thesis has described a distributed architecture, based on Internet Protocol networks, for substation information management and control systems. The basis of the architecture is a multi-agent system, consisting of a standard agent platform based on the Foundation for Intelligent Physical Agents (FIPA) standards and a number of different types of agent to perform various control, condition monitoring and information management tasks. The architecture is split into Wide Area Network (WAN) components and Local Area Network (LAN) components, of which the LAN components are located in substations, and the WAN components at any other locations on the power company's computer network.

Although initially targeted at power systems, the architecture described in this thesis is sufficiently generic that it should be able to be applied to a range of industrial situations. For example, generic types of agent, such as plant agents and device agents are described, which may be instantiated into application-specific agents such as transformer agents and agents representing a specific model of data acquisition device. This generic nature significantly enhances the reusability of the work described. However, there are a number of problems and opportunities for further work, which are discussed in Section 7.3 below.

In summary, the contribution of this thesis has been to develop a generic multi-agent architecture for the implementation of automation systems for distributed industrial systems. The design of the individual agents and the collaboration between agents has been investigated in detail. The use of mobile agents has been evaluated, and it has been demonstrated that for certain applications and network characteristics mobile agents provide a performance improvement over client-server systems. Finally, the whole architecture has been evaluated. This evaluation suggests that the use of a multi-agent system for distributed industrial automation should provide increased flexibility over traditional systems. It also provides a framework for integrating different data sources and managing a variety of tasks. However, there may be a price to pay in the form of slightly reduced application performance and increased complexity of administration.

7.2 Conclusions

The following conclusions may be drawn from the work presented in this thesis:

1. Using the multi-agent systems methodology, it is possible to design a software architecture capable of performing all of the tasks required in a power system automation system. This is described in Chapters 2 and 3.

2. The architecture uses generic agents (plant agents, device agents, database agents), which are applicable to a wide range of distributed industrial automation systems. The architecture may also be extended by adding agents to perform specific tasks. Therefore, it should be possible to transfer the architecture to applications in other industries, as discussed in Section 6.6.

3. The use of separate device agents, representing the physical data acquisition system and plant agents, representing the substation plant, permits either the plant or the data acquisition system to be altered while affecting the minimum number of agents. It also increases the modularity of the system by reducing the amount of knowledge required by each agent.

4. Using mobile agents can reduce the amount of time taken to perform common data analysis and remote control tasks, as shown in Chapter 4. For the data analysis task, mobile agents are most useful on low bandwidth networks. For the remote control task, mobile agents should be used when the latency of the network is high.

5. The prototype system demonstrates that the architecture described in Chapters 2 and 3 is capable of performing remote control, data analysis, database querying and display of status and condition information. It also demonstrates the feasibility of implementing the architecture.

6. By using the directory facilities provided by a Foundation for Intelligent Physical Agents (FIPA) compliant agent platform, it is possible to permit changes to be made to the system at runtime. This permits, for example, an item of plant or a device to be replaced while the system is operational, while changing only the agent associated with that particular object.

The above points demonstrate the advantages of the multi-agent techniques used in this system, in comparison to traditional systems such as SCADA, for the automation of large, distributed industrial systems. The use of a multi-agent system permits the integration of the different tasks of control, information management and condition monitoring in a single architecture. The resulting architecture provides increased flexibility compared to a traditional system, although there may be some decrease in performance.

7.3 Future Research

Due to the broad scope of the research described in this thesis, insufficient time has been available to investigate all of the possibilities of an agent-based substation automation system. This section describes some of the remaining problems and possible improvements to the architecture and the prototype system. In some cases, the method by which these improvements might be achieved has been considered, but has not been implemented or tested in practice.

7.3.1 Learning and Intelligence

The agents described in this thesis are relatively limited, in fact, some authors might dispute whether or not certain agents (such as the database agents) deserved the "agent" title. However, there is scope within the architecture for adding intelligence to various agents to improve their performance or enable them to perform a wider variety of tasks. For example, machine learning in the user interface agent might enable it to learn about its user's information needs, as has been done by, for example [42], or the control agents could be provided with learning behaviour to allow them to optimize the control of an item of plant.

7.3.2 Control Agents

Only limited work has so far been performed on the implementation of the control agents, and the current control agents basically act as servers, providing querying and user control functions. However, a full implementation would also include automatic control functions.

Another important issue regarding the control agents is how to implement collaborative control between control agents, in particular to allow the control agents to prevent mobile agents or agents operated by multiple users from performing conflicting actions on the substation plant. It is suggested that either all of the control agents might collaborate with each other to achieve this (possibly with agents having a "neighbourhood" consisting of those agents of plant that is directly connected to the plant of the agent in question), or that a hierarchical structure might be employed, in which the transformer agents are responsible for the actions of the agents of switchgear linked to their transformers, and transformer agents cooperate with each other.

Agent collaboration might be performed using a number of methods, including joint intentions [116], which provide a mechanism for controlling agent collaboration using joint goals and jointly agreed plans for achieving these goals. An alternative would be the use of distributed constraint programming [118], which uses a multi-agent system to solve a problem consisting of a set of mathematical constraints distributed over a number of agents.

7.3.3 Real Time Control and Monitoring

So far, no real time functions are provided by the prototype implementation. However, real time considerations are an important factor in the design of a power system automation system. Therefore, the architecture should be enhanced to take account of these considerations, and an implementation including real-time behaviour should be created and evaluated.

The major modifications required for real-time behaviour would be to the device agents and plant agents, as real-time constraints are not an issue for much of the information management system. However, it might also be necessary to include the substation user interface agent, due to the real-time constraints on the display of information on the substation HMI given in [83].

A number of challenges associated with the use of artificial intelligence in real-time domains are described in [132].

7.3.4 Multi-Hop Mobile Agents and Mobile Agent Planning

The current agents, though capable of performing multi-hop information retrieval and multi-hop control, do not attempt to optimize the route taken when performing such tasks. Therefore the performance obtained is not optimal. Further investigation should implement a planning algorithm to attempt to optimize the route of these mobile agents.

Another optimization that could be used, particularly for the data analysis agents, is the use of multiple agents. This should also be investigated in more detail.

7.3.5 Document Retrieval and Integrated Document and Data Retrieval Using Mobile Agents

A further possible application of mobile agents would be to provide document retrieval functionality. This could be implemented either as an independent function or in combination with the data retrieval functionality. The task of the document agent would be to retrieve a set number of documents (e.g. 10) in response to a query issued by a user. The agent would visit the various available document repositories, and collect matching documents. The following questions should be addressed during the development of this agent:

1. If the agent returns to the user after collecting $n$relevant documents (where $n$is the number specified by the user), it is possible that these are not the most relevant n,and that much more relevant documents exist at other servers. However, if the agent continues to other servers, it will use more network resources. Therefore, the agent's termination criteria must be investigated.

2. The question of how to integrate results from multiple document repositories applies equally to the mobile agent as to the client-server document management agent (as discussed in Section 3.2.3).

3. If possible, a planning methodology should be developed to allow the agent to estimate the best route through the available servers, in order to maximize the likelihood of obtaining the most relevant documents while minimizing the time taken.

7.3.6 Further Evaluation

The evaluation of the system presented in this thesis is quite basic, and few quantitative results are presented, particularly for the static agents. In order to fully characterize the architecture described, further experiments would be required.

Appendix A

Data Tables and Experimental Results

This section provides full results tables for the experiments described in Chapter 4.

A.1 Mobile Agent Control Experiment 1

These tables give the total time in milliseconds to perform N connect-read-disconnect-connect-write-disconnect interactions between an agent and relay.

N	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
0	1271	1262	1262	301	1272	1073.6
1	1812	1863	1762	1773	1933	1828.6
2	2364	2373	2384	2383	2373	2375.4
3	2874	2974	2934	2944	3085	2962.2
4	2474	3354	3485	3455	3395	3232.6
5	4136	3956	3034	3956	3996	3815.6
10	8552	7040	6769	6770	6779	7182

Table A.1: Mobile Agent, Serial Link.

N	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
0	0	0	0	0	0	0
1	1853	1642	1633	1692	1662	1696.4
2	4096	3375	3495	3485	3315	3553.2
3	4977	5177	5228	5207	5158	5149.4
4	6980	6930	6830	7051	7151	6948.2
5	9003	8462	8993	8713	8623	8758.8
10	17566	17355	17585	17134	16564	17240.8

Table A.2: Client/Server, Serial LINK.

N	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
0	90	91	80	80	90	86.2
1	681	651	741	861	640	714.8
2	1232	1522	1231	1322	1242	1309.8
3	1882	1842	1713	1662	1743	1768.4
4	2293	2334	2313	2244	2373	2311.4
5	2884	2734	2854	2895	2804	2834.2
10	5949	5899	5658	5919	5949	5874.8

Table A.3: Mobile Agent, 100Mbps Ethernet.

N	Run 1	Run 2	Run 3	Run 4	Run 5	Mean
0	0	0	0	0	0	0
1	530	541	551	481	541	528.8
2	962	931	1012	961	972	967.6
3	1532	1422	1442	1392	1562	1470
4	1853	1852	2033	1852	1912	1900.4
5	2423	2714	2403	2423	2484	2489.4
10	4627	4737	4617	4807	4556	4668.8

Table A.4: Client/Server, 100Mbps Ethernet.

A.2 Mobile Agent Control Experiment 2

Lat.	N	Run 1	Run 2	Run 3	Run 4
0	0	1.93	1.93	1.93	1.93
0	20	12.35	12.42	12.36	12.43
0	40	22.77	22.76	22.78	22.77
0	60	33.22	33.21	33.20	33.23
0	80	43.61	43.62	43.62	43.59
0	100	54.04	54.05	54.05	54.06
25	0	2.23	2.23	2.23	2.41
25	20	13.62	13.59	13.56	13.59
25	40	25.05	25.17	25.04	25.06
25	60	36.36	36.38	36.37	36.36
25	80	47.83	47.80	47.78	47.80
25	100	59.22	59.39	59.22	59.35
50	0	2.83	2.83	2.83	2.84
50	20	15.15	15.11	15.16	15.13
50	40	27.90	27.49	27.52	27.52
50	60	39.91	39.94	39.96	39.93
50	80	52.36	52.33	52.35	52.36
50	100	64.70	64.77	64.77	64.83
75	0	3.25	3.25	3.25	3.24
75	20	16.47	16.53	16.49	16.54
75	40	29.88	29.85	29.89	29.87
75	60	43.28	43.32	43.33	43.32
75	80	56.66	56.64	56.68	56.77
75	100	70.11	70.13	70.02	70.10
100	0	3.58	3.58	3.59	3.64
100	20	17.82	17.79	17.86	17.56
100	40	32.20	32.13	32.21	32.16
100	60	46.65	46.57	46.56	46.57
100	80	60.97	61.00	60.98	61.01
100	100	75.52	75.39	75.35	75.40

Table A.5: Static Agent

Lat.	N	Run 1	Run 2	Run 3	Run 4
0	0	2.78	2.57	2.63	2.65
0	20	13.24	13.04	13.04	13.04
0	40	23.61	23.59	23.61	23.84
0	60	34.03	33.97	33.90	33.89
0	80	44.46	44.41	44.30	44.29
0	100	54.91	54.73	54.92	54.92
25	0	4.23	4.07	4.25	4.11
25	20	14.69	14.49	14.43	14.46
25	40	25.06	25.9	25.08	24.98
25	60	35.64	35.47	35.41	35.60
25	80	46.04	45.89	45.87	45.92
25	100	56.52	56.28	56.25	56.63
50	0	5.66	5.51	5.71	5.61
50	20	16.11	15.90	15.99	16.02
50	40	26.50	26.66	26.63	26.74
50	60	37.21	37.02	37.12	37.16
50	80	47.72	47.54	47.57	47.64
50	100	58.06	57.84	57.83	57.89
75	0	7.14	6.96	6.95	7.04
75	20	17.53	17.61	17.39	17.41
75	40	28.12	28.09	28.12	28.01
75	60	38.82	38.60	38.74	38.97
75	80	49.37	49.34	49.37	49.31
75	100	60.24	59.41	59.45	59.64
100	0	8.56	8.60	8.36	8.55
100	20	18.78	19.04	19.19	18.86
100	40	29.61	29.41	29.84	29.79
100	60	40.52	40.22	40.25	40.29
100	80	50.93	50.87	51.01	50.94
100	100	61.41	61.06	61.18	61.56

Table A.6: Mobile Agent

A.3 MA Analysis Experiment

These tables give the time taken to perform the experiment described in Section 4.2 for each agent type. In the tables, M represents Megabits/second, K represents Kilobits/second and bandwidth is abbreviated as B/w.

B/w	1st run	2nd	3rd
100M	112.64	106.63	106.20
10M	111.97	104.32	107.67
1M	115.21	105.14	108.88
500K	111.33	107.45	106.60
100K	119.38	112.80	111.53
10K	231.88	221.28	224.27

Latency = 0ms

B/w	1st run	2nd	3rd
100M	112.53	112.02	111.74
10M	113.20	108.08	111.02
1M	113.71	105.87	108.49
500K	114.92	106.60	110.77
100K	120.49	113.64	112.29
10K	230.19	231.38	223.45

Latency=25ms

B/w	1st run	2nd	3rd
100M	116.93	109.66	108.87
10M	117.48	110.13	110.67
1M	116.11	110.45	110.29
500K	118.41	110.04	109.98
100K	122.97	113.98	114.29
10K	238.09	225.96	227.57

Latency = 50ms

B/w	1st run	2nd	3rd
100M	117.11	115.66	112.68
10M	118.38	111.93	117.90
1M	118.37	114.94	110.14
500K	120.98	114.40	115.14
100K	123.54	116.51	122.90
10K	234.55	223.12	226.39

Latency=75ms

B/w	1st run	2nd	3rd
100M	119.88	116.89	113.66
10M	119.30	111.57	113.27
1M	122.43	117.07	116.77
500K	120.71	114.72	119.41
100K	128.15	122.27	122.00
10K	233.21	226.07	222.89

Latency = 100ms

Table A.7: Wrapper-Based Agent, Mobile

The result marked with a * was a re-run after the original result was affected by network load.

B/w	1st run	2nd	3rd
100M	109.89	107.23	107.20
10M	110.78	107.45	107.20
1M	117.81	114.00	113.95
500K	126.67	124.14	123.35
100K	202.39	202.10	198.17
10K	1141.44	1118.52	1129.47

Latency = 0ms

B/w	1st run	2nd	3rd
100M	115.93	112.91	112.69
10M	118.50	114.24	118.66
1M	122.94	121.15	120.22
500K	133.21	128.43	129.46
100K	202.97	199.76	200.01
10K	1132.78	1100.77	1158.64

Latency=25ms

B/w	1st run	2nd	3rd
100M	123.24	120.40	119.96
10M	121.71	118.42	118.16
1M	126.47	123.72	122.16
500K	135.16	132.34	132.41
100K	207.90	204.62	203.88
10K	1138.87	1119.30	1120.29

Latency = 50ms

B/w	1st run	2nd	3rd
100M	129.51	126.01	126.64
10M	133.94	127.68	126.59
1M	131.50	129.00	128.67
500K	138.04	135.81	135.05
100K	209.34	206.97	206.09
10K	1137.08	1122.46	1119.27

Latency=75ms

B/w	1st run	2nd	3rd
100M	137.83	134.35	133.88
10M	134.84	132.22	130.90
1M	139.37	135.55	135.15
500K	144.28	140.40	138.67
100K	209.34	212.54	211.46
10K	1112.98	1110.51	1121.70

Latency = 100ms

Table A.8: Wrapper-Based Agent, Static

B/w	1st run	2nd	3rd
100M	33.45	33.90	33.29
10M	33.11	32.64	36.46
1M	34.09	35.01	40.31
100K	41.81	38.88	39.55

Latency = 0ms

B/w	1st run	2nd	3rd
100M	35.92	32.76	32.79
10M	36.07	32.13	32.81
1M	36.20	33.13	33.40
100K	44.26	46.24	47.52

Latency=25ms

B/w	1st run	2nd	3rd
100M	38.11	44.05*	35.18
10M	38.15	35.20	34.39
1M	38.55	38.69	35.24
100K	45.97	48.72	45.22

Latency = 50ms

Table A.9: Direct Access Agent, Mobile, Cached

B/w	1st run	2nd	3rd
100M	108.64	106.52	98.72
10M	101.84	105.17	105.91
1M	102.75	100.13	101.95
100K	110.48	106.73	110.40

Latency = 0ms

B/w	1st run	2nd	3rd
100M	105.23	101.64	106.35
10M	103.87	105.20	104.48
1M	104.87	108.18	102.58
100K	111.62	109.36	113.02

Latency=25ms

B/w	1st run	2nd	3rd
100M	107.15	107.47	111.04
10M	103.51	109.98	104.00
1M	107.53	107.30	109.73
100K	114.29	110.87	117.36

Latency = 50ms

Table A.10: Direct Access Agent, Mobile, Uncached

B/w	1st run	2nd	3rd
100M	37.60	32.77	32.18
10M	32.60	33.38	33.39
1M	46.03	46.94	46.88
100K	354.04	354.46	354.62

Latency = 0ms

B/w	1st run	2nd	3rd
100M	958.14	959.68	959.70
10M	961.01	962.66	962.68
1M	992.24	993.31	993.68
100K	1303.65	1305.58	1304.92

Latency=25ms

B/w	1st run	2nd	3rd
100M	1906.72	1908.17	1908.20
10M	1911.97	1912.17	1911.86
1M	1941.62	1942.71	1942.81
100K	2253.65	2254.84	2254.35

Latency = 50ms

Table A.11: Direct Access Agent, Static, Cached

B/w	1st run	2nd	3rd
100M	96.67	97.99	98.05
10M	103.40	104.42	104.57
1M	157.93	159.17	159.35
100K	1387.70	1387.75	1388.09

Latency = 0ms

B/w	1st run	2nd	3rd
100M	3802.99	3804.81	3804.90
10M	3815.83	3817.96	3818.42
1M	3939.50	3941.48	3941.55
100K	5184.07	5185.72	5185.38

Latency=25ms

B/w	1st run	2nd	3rd
100M	7593.96	7593.89	7595.24
10M	7608.65	7613.69	7611.18
1M	7730.43	7733.47	7732.64
100K	8974.44	8978.56	8978.06

Latency = 50ms

Table A.12: Direct Access Agent, Static, Uncached

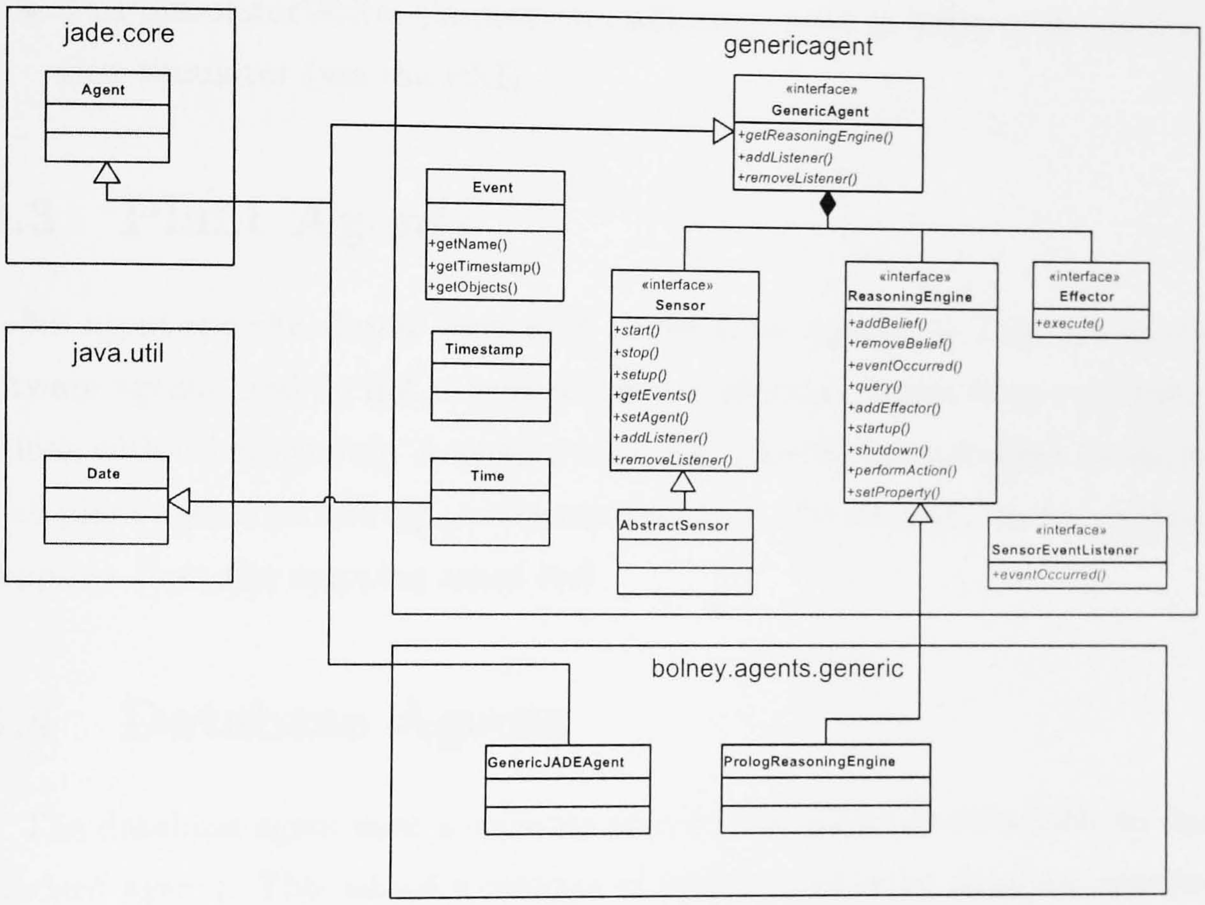
Appendix B

Implementation Details of Prototype System

This chapter describes in more detail the implementation of the multi-agent system for substation information management and control. Three programming languages are used in this implementation. The main agent components are implemented in Java using the JADE toolkit. Prolog is used to implement the reasoning and querying of the server agents, and the user interface agent is implemented as a set of ActiveX components using Microsoft Visual Basic.

B.1 Class Hierarchy

Figure B.1 shows the upper class hierarchy of the substation information management system, which implements the architecture described in Chapter 3. Other classes, such as task-specific sensors, effectors and custom reasoning engines, inherit from the classes of this hierarchy.



Note: the methods *addListener()* and *removeListener()* of the genericagent.Sensor and genericagent.GenericAgent classes are actually *addSensorEventListener()* and *removeSensorEventListener()* . For readability the names have been abbreviated on this diagram

Figure B.1: Upper Class Hierarchy of Substation Information Management System

B.2 Substation Simulator Agent

B.2.1 Agent-Specific Classes Used

- TCPSimulatorSensor (extends AbstractSensor): used to acquire data from the substation simulator via the data acquisition PXI system.
- TCPSimulatorWriter (implements Effector): used to write to the substation simulator (via the PXI).

B.3 Plant Agents

No agent-specific classes were used in the plant agents, as they are solely software agents, and do not require sensors or effectors (apart from communications with other agents). A single prolog file[1] provides generic rules common to all plant agents (acquiring plant properties from the ontology agent, reading mappings from the mapping agent etc).

B.4 Database Agents

The database agent used a separate kernel (DatabaseAgentKernel2) to the standard agents. This added a number of methods allowing database queries to be executed from Prolog.

B.5 Document Agents

The *ps2ascii* tool (part of GhostScript) is used to generate the plain text from PDF and PostScript files. To generate plain text from a Word document, a program written in Visual Basic invokes Microsoft Word to convert the file. Therefore, in order for the document agent to work with Word documents, Microsoft Word must be installed on the system.

[1]plant.pro

B.6 User Interface Agent

The user interface agent is implemented in Visual Basic. It communicates with the other agents (resident on the JADE platform) via a TCP-based message transport system added to the JADE main container.

The protocol used by the message transport system and user agent is as follows:

1. Discovery

   (a) Client broadcasts UDP packet with content "SEND-AP-DESCRIPTION" to address 224.0.0.1 and port 7878. Alternatively, an address may be specified by the user.

   (b) On receipt of this message the MTS sends an *inform* message to the client. The content of this message is ((result <action> <ap-description>)) where <ap-description> is the agent platform description (as defined in [76]) of the platform on which the MTP is situated. The client can use this to extract the TCP address of the platform, which will be given as "stcp://<address>:<port>", for example "stcp://192.168.1.5:7879".

2. Message Transport

   (a) Client establishes TCP connection to the MTS.

   (b) To send a message, the client first writes "BEGIN", at the start of a line and followed by a line feed or carriage return and line feed. The client then writes the message, using the FIPA ACL string encoding. To terminate a message, the client writes a line "END".

   (c) To close the socket, the client writes a line "CLOSE". The server will then close its connection to the client.

   (d) Messages from the other agents to the client (via the MTS) are transmitted on the same socket connection, using the same protocol.

Appendix C

Data File Formats

The prototype system is configured using several data files. The format of these files is described here.

C.1 Generic Agent Configuration

The format of the configuration file for the generic agent is as a FIPA SL functional expression with parameters (frame). There are 5 possible parameters:

- reasoning-engine: Specifies information regarding the agent's reasoning engine.
- abilities: Set of *ability* frames. Specifies the abilities / effectors of the agent
- sensors: Set of *sensor* frames. Specifies the sensors of the agent
- include-beliefs: Set of strings (the filenames). Specifies additional files containing the agent's beliefs.
- interaction-protocols: Set of strings (interaction protocol names). Specifies interaction protocols which the agent will participate in. Allowable values are `fipa-subscribe`, `subscribe`, `fipa-request`, `fipa-query`.[1]

[1]There is a bug in the agent implementation with regards to the `fipa-subscribe`

The Ability frame

The ability frame has the classname/functor "ability" and the following possible parameters:

- implemented-by: String. Specifies the Java classname of the ability's implementing class
- name: String. Currently ignored.

The Sensor frame

The sensor frame has the classname/functor "sensor" and the following possible parameters:

- implemented-by: String. Specifies the Java classname of the sensor's implementing class
- name: String. Currently ignored.
- parameters: Set of *parameter* frames, each having the parameters *name* and *value*. The possible parameters depend on the sensor class in use.

The Reasoning Engine Frame

The reasoning engine frame has the classname/functor "reasoning-engine" and the following possible parameters:

- name: String. Currently ignored
- implemented-by: Java classname of the class implementing the reasoning engine
- properties: Set of *property* frame, each having the parameters *name* and *value*. The possible parameters depend on the chosen reasoning engine.

protocol, which was formerly referred to as `subscribe` during the period of time that `fipa-subscribe` was deprecated by FIPA. To ensure that an agent is capable of handling subscriptions, put both `subscribe` and `fipa-subscribe` in the interaction protocol set.

C.1.1 Example

This example is an abbreviated version of that used by the substation simulator agent of the prototype system (some of the filenames have been shortened).

```
(agent-definition
  :reasoning-engine
    (reasoning-engine
      :name ``PrologRE''
      :implemented-by
        bolney.agents.generic.PrologReasoningEngine
      :properties
        (set
          (property
            :name xplfile
            :value d:/work/prolog/test/test.xpl)
          (property
            :name profiles
            :value
            (set
              d:/java/bolney/config/simulator/simulator.pro
              d:/java/bolney/config/create_action_sd.pro
              d:/java/bolney/config/create_query_sd.pro))))
    :abilities
      (set
        (ability
          :implemented-by
            bolney.agents.generic.effector.RegisterWithDF2))
    :sensors
      (set)
    :include-beliefs
      (set
        d:/work/java/bolney/config/simulator/simulator.sl
        d:/work/java/bolney/agents/monitoring/device_ontology.sl)
    :interaction-protocols
      (set fipa-query subscribe fipa-subscribe fipa-request))
```

C.2 Mobile Analysis Agent Configuration

The mobile analysis agent uses a somewhat simpler configuration file than the generic agent. The format is similar, with an SL frame being used as the main configuration object. The main frame has classname *analysis-agent*. It has the following parameters:

- data-sets : Set of *data-set* frames. Specifies the data sets to be retrieved.
- functions : Set of *function* frames. Specifies the analysis functions used by the agent.
- report : *report* frame. Specifies how the report is to be generated.
- mobile : boolean. Specifies whether the agent is to be mobile or static.

The analysis agent's configuration file may be automatically generated by the user interface agent.

The Data Set Frame

The data-set frame defines a data set. Properties:

- number : Integer. A number used to identify the data set.
- selection-string : String. A FIPA SL IdentifyingExpression used to specify the data to retrieve.

The Function Frame

The function frame defines an analysis function, specifying the name to be used and the Java class which implements the function. Properties:

- name : String
- implemented-by : String. Name of Java class.

The Report Frame

The report frame defines the report in terms of how the analysis functions are to be applied, and any additional text to be inserted around the results.

C.2.1 Example

This example is for an agent which retrieves a single data set containing *current* and *time*, calculates the maximum current and displays it in a report entitled "Analysis Report".

```
(analysis-agent
  :data-sets
    (set
      (data-set
        :number 0
        :selection-string
          ''((all
              (set ?current ?time)
              (t (lv-current sgt1 ?current) ?time)))''))
  :functions
    (set
      (function
        :name max
        :implemented-by
          bolney.agents.analysis.functions.Max))
    :mobile true
    :report
      (report
        :title ''Analysis Report''
        :items
          (set
            (report-item
              :results (max ds0:current ds0:time)
              :text-before ''Maximum LV Current =''
              :text-after ''''))))
```

C.3 Mobile Remote Control Agent Configuration

The remote control agent's configuration consists of a set of actions, and a configuration setting to determine whether or not the agent is mobile. The main frame of the configuration is the *control-agent* frame. Parameters:

- actions: FIPA SL action sequence
- mobile: boolean. Specifies whether the agent is mobile or static.

C.3.1 Example

```
(control-agent
    :actions
        (action
            (agent-identifier :name ca@pc2214:1099/JADE)
            (open h13))
    :mobile true)
```

References

[1] The National Grid Company plc. Seven year statement. `http://www.nationalgrid.com/uk/library/documents/sys_03/default.asp`, March 2003.

[2] J. D. McDonald. Substation automation: IED integration and availability of information. *IEEE Power and Energy Magazine*, 1(2):22–31, March/April 2003.

[3] V. Lohmann. New strategies for substation control, protection and access to information. In *Proceedings of the sixteenth international conference and exhibition on electricity distribution (CIRED 2001)*, volume 3, pages 211–215. IEE Publishing, June 2001.

[4] B. Qui and H. B. Gooi. Web-based SCADA display systems for access via internet. *IEEE Transactions on Power Systems*, 15(2):681–686, May 2000.

[5] S. Bricker, T. Gonen, and L. Rubin. Substation automation technologies and advantages. *IEEE Computer Applications in Power*, 14(3):31–37, July 2001.

[6] J. V. Hughes, J. E. Fitch, and R. W. Silversides. Substation information project- field experience with internet technologies. In *Proceedings of the Seventh International Conference on Developments in Power System Protection*, pages 122–125, Amsterdam, Netherlands, 2001.

[7] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[8] Jacques Ferber. *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, Harlow, England, 1999. ISBN 0201360489.

[9] Douglas Maxwell Considine and Glenn P Considine. *Standard Handbook of Industrial Automation.* Chapman and Hall, New York, 1986. ISBN 0412008319.

[10] John W. Bernard. *CIM in the process industries.* Instrument Society of America, Research Triangle Park (N.C.), 1989. ISBN 1556171676.

[11] Theodore J. Williams. *A Reference Model for Computer Integrated Manufacturing.* Instrument Society of America, Research Triangle Park (N.C.), 1989. ISBN 1556172257.

[12] B.M. Weedy and B.J. Cory. *Electric Power Systems.* Wiley, Chichester, 1998. ISBN 0471976776.

[13] Otto Preiss and Alain Wegmann. Towards a composition model problem based on IEC61850. In *Proceedings of the 4th ICSE Workshop on Component-Based Software Engineering*, Toronto, Canada, May 2001. Available online: `http://www.sei.cmu.edu/pacc/CBSE4_papers/PreissWegmann-CBSE4-4.pdf`.

[14] Q. Zhao, H. In, X. Wu, and G. Huang. Transforming legacy energy management system (EMS) modules into reuseable components: A case study. In *Proceedings of IEEE International Computer Software and Applications Conference (COMPSAC 2000)*, pages 105–110. IEEE Computer Society Press, 2000.

[15] J. W. Evans. Energy management system survey of architectures. *IEEE Computer Applications in Power*, 2(1):11–16, 1989.

[16] S. Humphreys. Substation automation systems in review. *IEEE Computer Applications in Power*, 7(2):24–30, April 1998.

[17] K. Caird. Integrating substation automation. *IEEE Spectrum*, 34(8): 64–69, August 1997.

[18] Darold Woodward and David Tao. Comparing throughput of substation networks. Technical report, Schweitzer Engineering Laboratories Inc., Pullmann, WA, USA, 2000. Available online: `http://www.selinc.com/techpprs/6116.pdf`.

[19] Mark Adamiak and William Premerlani. The role of utility communications in a deregulated environment. In *Proceedings of the Hawaii International Conference on System Sciences*, volume 3, Maui, HI, USA, January 1999. CDROM.

[20] T. Skeie, S. Johannessen, and C. Brunner. Ethernet in substation automation. *IEEE Control Systems Magazine*, 22(3):43–51, June 2002.

[21] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison Wesley, Reading, Massachusetts, 1999. ISBN 0201379279.

[22] Robert Orfali and Dan Harkey. *Client/Server Programming with Java and Corba*. Wiley Computer Publishing, New York, 2nd edition, 1998. ISBN 047124578X.

[23] Ian Somerville. *Software Engineering*. Addison-Wesley, Harlow, England, 6th edition, 2001. ISBN 020139815X.

[24] International Electrotechnical Commission. Communication networks and systems in substations. IEC Standard 61850, International Electrotechnical Commission, Geneva, CH, 2002.

[25] K. Clinard. GOMSFE (generic object models for substation and feeder equipment) models of multifunctional microprocessor relays. In *Power*

Engineering Society Summer Meeting, 1999, volume 1, pages 36–38. IEEE, July 1999.

[26] G. Brunello, R. Smith, and C. B. Campbell. An application of a protective relaying scheme over ethernet LAN/WAN. In *Transmission and Distribution Conference and Exposition*, volume 1, pages 522–526. IEEE/PES, 2001.

[27] DNP Users Group. DNP users group website. `http://www.dnp.org`.

[28] IEEE Computer Applications in Power Tutorial. Fundamentals of utilities communication architecture. *IEEE Computer Applications in Power*, 14(3):15–21, July 2001.

[29] John Downes, Jack Goody, Keith Walker, Brian Baker, and Dave Cooper. A strategy for substation information, control and protection. Technical Report TR(E)312, National Grid Company, 1998.

[30] Brian Baker. Substation information control and protection local area network. Draft NGTS 3.24.1, National Grid Company, February 2000.

[31] Michael Woolridge and Nicholas Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[32] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)*, pages 312–319, San Francisco, USA, June 1995.

[33] Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6): 34–44, December 1992.

[34] Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff. Formal Methods in DAI: Logic-Based Representation and Reasoning. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed*

Artificial Intelligence, chapter 8, pages 331–376. The MIT Press, Cambridge, Massachusetts, 1999. ISBN 0262731312.

[35] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[36] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–77. The MIT Press, Cambridge, Massachusetts, 1999. ISBN 0262731312.

[37] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[38] John R. Koza, Forrest H Bennett III, David Andre, and Martin A. Keane. Genetic programming: Biologically inspired computation that creatively solves non-trivial problems. In Laura Landweber, Erik Winfree, Richard Lipton, and Stephen Freeland, editors, *Proceedings of Discrete Mathematics and Theoretical Computer Science (DIMACS) Workshop on Evolution as Computation*, pages 15–44, Princeton University, 11-12 1999. Springer-Verlag. ISBN 3540667091.

[39] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[40] C. Heinze, S. Goss, I. Lloyd, and A. Pearce. Plan recognition in military simulation: Incorporating machine learning with intelligent agents. In *Proceedings of IJCAI-99 Workshop on Team Behaviour and Plan Recognition*, pages 53–64, 1999.

[41] Cindy Olivia, Chee-Fon Chang, Carlos F. Enguix, and Aditya K. Ghose. Case-based BDI agents: an effective approach for intelligent search on

the world wide web. In *AAAI Spring Symposium on Intelligent Agents*. Stanford University, USA, March 1999.

[42] Pattie Maes. Agents that reduce work and information overload. In Jeffrey M. Bradshaw, editor, *Software Agents*, chapter 8. MIT Press, Cambridge, Massachusetts, 1997. ISBN 0262622349.

[43] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, pages 671–678, Cambridge, Massachusetts, 1993. MIT Press. ISBN 0262201070. Also available as: `http://www.cs.duke.edu/~mlittman/papers/routing-nips.ps`.

[44] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robotics*, 8(3), July 2000.

[45] I. A. Ferguson. TouringMachines: Autonomous agents with attitudes. *IEEE Computer*, 25(5):51–55, 1992.

[46] Klaus Fischer, Jorg P. Muller, and Markus Pischel. A pragmatic BDI architecture. In *Proceedings of ATAL 95*, number LNAI 1037 in Lecture Notes in Artificial Intelligence, pages 203–218. Springer Verlag, 1995.

[47] Onn Shehory. Architectural properties of multi-agent systems. Technical Report CMU-RI-TR-98-28, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, December 1998.

[48] D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[49] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[50] Roch H. Glitho, Edgar Olougouna, and Samuel Pierre. Mobile agents and their use for information retrieval: A brief overview and an elaborate case study. *IEEE Network*, 16(1):34–41, January 2002.

[51] Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, and Daniela Rus. D'Agents: Applications and performance of a mobile-agent system. *Software— Practice and Experience*, 32(6):543–573, May 2002.

[52] Mario Baldi and Gian Pietro Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In *Proceedings of the 20th international conference on Software engineering*, pages 146–155. IEEE Computer Society, 1998. ISBN 0-8186-8368-6.

[53] M. Straßer and E Schwehm. A performance model for mobile agent systems. In *Proceedings of Parallel and Distributed Processing Techniques and Applications 1997*, volume 2, pages 1132–1140, Las Vegas, Nevada, USA, 1997.

[54] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. Mobile agents for distributed information retrieval. In Matthias Klusch, editor, *Intelligent Information Agents*, chapter 15, pages 355–395. Springer-Verlag, 1999.

[55] Colin G Harrison, David M. Chess, and Aaron Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM T. J. Watson Research Center, 1996.

[56] Dejan Milojicic. Trend wars: Mobile agent applications. *IEEE Concurrency*, pages 80–90, July-September 1999.

[57] Wayne Jansen and Tom Karygiannis. Mobile agent security. Special Publication 800-19, National Institute of Standards and Technology, 1999.

[58] David Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, number 1419 in Lecture Notes in Computer Science, pages 159–187. Springer-Verlag.

[59] Katia Sycara, Keith Decker, Anandeep Pannu, Mike Williamson, and Dajun Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.

[60] Michael N. Huhns and Munindar P. Singh. All agents are not created equal. *IEEE Internet Computing*, 2(3):94–96, May-June 1998.

[61] J M Corera, I Laresgoiti, and N Jennings. Using Archon, part 2: Electricity transportation management. *IEEE Expert*, 11(6):71–79, December 1996.

[62] N. R. Jennings and D. Cockburn. ARCHON: A distributed artificial intelligence system for industrial applications. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 319–344. Wiley, 1996.

[63] The PABADIS Consortium. Pabadis white paper. `http://www.pabadis.org/downloads/pabadis_white_paper.pdf`, 2002.

[64] Steffen Deter, Ralf Blume, and Klemm Eckehardt. Generic machine representation in the PABADIS community. In *Proceedings of e-2002 Conference*, Prague, October 2002. `http://www.mathematik.uni-marburg.de/~pabadis/publics/e2002.doc`.

[65] S. Bussmann and K. Schild. Self-organizing manufacturing control: An industrial application of agent technology. In *Proceedings of the Fourth International Conference on Multi-Agent Systems*, pages 87 – 94, 2000.

[66] N. R. Jennings and S. Bussmann. Agent-based control systems: Why are they suited to engineering complex systems. *IEEE Control Systems Magazine*, 23(3):61–73, June 2003.

[67] P. Leitao and F. Restivo. An agile and cooperative architecture for distributed manufacturing systems. In *Proceedings of the IASTED International Conference on Robotics and Manufacturing*, pages 188–193, Cancun, Mexico, May 2001.

[68] E. E. Mangina, S. D. J. McArthur, and J. R. McDonald. Reasoning with modal logic for power plant condition monitoring. *IEEE Power Engineering Review*, 21(7):58–59, July 2001.

[69] E.E. Mangina, S.D.J. McArthur, and J.R. McDonald. Commas (condition monitoring multi-agent system. *Autonomous Agents and Multi-Agent Systems*, 4(3):279–282, 2001.

[70] C. Lucas, M. A. Zia, M. R. A. Shirazi, and A. Alishahi. Development of a multi-agent information management system for Iran power industry: A case study. In *Proceedings of 2001 IEEE Porto Power Tech Conference*, Porto, Portugal, 2001.

[71] Gustaf Neumann and Uwe Zdun. High-level design and architecture of an HTTP-based infrastructure for web applications. *World Wide Web*, 3(1):13–26, 2000.

[72] Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. `http://www.fipa.org`, August 2000.

[73] Foundation for Intelligent Physical Agents. FIPA communicative act library specification. `http://www.fipa.org/specs/fipa00037/`, October 2000.

[74] H. Van Dyke Parunak. Industrial and Practical Applications of DAI. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 9, pages 377–421. The MIT Press, Cambridge, Massachusetts, 1999. ISBN 0262731312.

[75] Weiming Shen and Douglas H. Norrie. Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems*, 1(2):129–156, 1999. Also available at: `http://imsg.enme.ucalgary.ca/publication/abm.htm`.

[76] Foundation for Intelligent Physical Agents. FIPA agent management specification. `http://www.fipa.org/specs/fipa00023/`, August 2000.

[77] D. Martin, H. Oohama, D. Moran, and A. Cheyer. Information brokering in an agent architecture. In *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 467–486, Blackpool, UK, 1997.

[78] Foundation for Intelligent Physical Agents. FIPA SL content language specification. `http://www.fipa.org/specs/fipa00008/`, August 2000.

[79] Foundation for Intelligent Physical Agents. FIPA Ontology Service Specification. `http://www.fipa.org/specs/fipa00086/`, August 2001.

[80] Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

[81] Foundation for Intelligent Physical Agents. FIPA interaction protocol library specification. `http://www.fipa.org/specs/fipa00025/`, August 2001.

[82] H. Van Dyke Parunak. 'Go to the ant': Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[83] The National Grid Company. Specification for substation control systems. NGTS 2.7, The National Grid Company, National Grid House, Kirby Corner Road, Coventry CV4 8JY, April 1998.

[84] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, USA, 1995. ISBN 0131038052.

[85] W. van der Hoek and M. Wooldridge. Towards a logic of rational agency. *Logic Journal of the IGPL*, 11(2):135–159, 2003.

[86] James F. Allen. Time and time again: the many ways to represent time. *International Journal of Intelligent Systems*, 6:341–355, 1991.

[87] Yoav Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33(1):89–104, 1987.

[88] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[89] J.T. Tengdin. Development of an IEEE standard for integrated substation automation commmunication(p1525): specifications for substation applications and key communication performance drivers. In *Power Engineering Society Summer Meeting, 2000*, volume 1, pages 136–137. IEEE, 2000.

[90] K.E. Martin, G. Benmouyal, M. G. Adamiak, M. Begovic, R. O. Burnett, K. R. Carr, A. Cobb, J. A. Kusters, S. H. Horowitz, G. R. Jensen, G. L. Michel, R. J. Murphy, A. G. Phadke, M. S. Sachdev, and J. S. Thorp. IEEE standard for synchrophasors for power systems. *IEEE Transactions on Power Delivery*, 13(1):73–77, 1998.

[91] Dublin Core Metadata Initiative. The Dublin Core Element Set version 1.1. `http://www.dublincore.org/documents/dces/`, February 2003.

[92] J.Q. Feng, P. Sun, W.H. Tang, D.P. Buse, Q.H. Wu, Z. Richardson, and J. Fitch. Implementation of a power transformer temperature monitoring system. In *Proceeding of 2002 International Conference on Power System Technology*, volume 2002 (3), pages 1980–1983. IEEE, 2002.

[93] N. Guarino. Formal ontology and information systems. In *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems*, pages 3–15, Trento, Italy, June 1998. IOS Press.

[94] Foundation for Intelligent Physical Agents. FIPA homepage. `http://www.fipa.org`.

[95] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience*, 31(2):103–128, 2001.

[96] Foundation for Intelligent Physical Agents. Publically available implementations of FIPA specifications. `http://www.fipa.org/resources/livesystems.html`.

[97] Harry L. Chen. Developing a dynamic distributed intelligent agent framework based on the Jini architecture. MSc thesis, University of Maryland Baltimore County, 1999.

[98] W. Keith Edwards. *Core Jini.* Prentice Hall PTR, Upper Saddle River, NJ 07458, 2001. ISBN 0130894087.

[99] Federico Bergenti and Agostino Poggi. LEAP: A FIPA platform for handheld and mobile devices. In *Intelligent agents VIII, LNAI 2333*, pages 436–445. Springer-Verlag, Berlin Heidelberg, 2002.

[100] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: an information integration system. In *Proceedings of 1997 Association for Computing Machinery Special Interest Group on Management of Data (ACM SIGMOD) Conference*, pages 539–542, May 1997.

[101] Norman W. Paton and Oscar Daz. Active database systems. *ACM Computing Surveys (CSUR)*, 31(1):63–103, 1999.

[102] Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, Mats Carlsson, and Manuel V. Hermenegildo. Parallel execution of prolog programs: a survey. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(4):472–602, 2001.

[103] Charles J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6):24–29, December 1996.

[104] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24 (5):513–523, 1988.

[105] Gerald Kowalski. *Information Retrieval Systems: Theory and Implementation.* Kluwer Academic Publishers, Boston, MA, USA, 1997. ISBN 0792399269.

[106] O. de Kretser, A. Moffat, T. Shimmin, and J. Zobel. Methodologies for distributed information retrieval. In *Proceedings of the Eighteenth International Conference on Distributed Computing Systems*, pages 66–73, Amsterdam, The Netherlands, May 1998.

[107] Cohen P and Levesque H. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[108] Robert S. Gray, David Kotz, Joyce Barton Ronald A. Peterson, Daria Chacn, Peter Gerken, Martin Hofmann, Jeffrey Bradshaw, Maggie Breedy, Renia Jeffers, and Niranjan Suri. Mobile-agent versus client/server performance: Scalability in an information-retrieval task. In *Mobile Agents: 5th International Conference, MA 2001*, number 2240 in Lecture Notes in Computer Science, pages 229–243, Atlanta, Georgia, 2001.

[109] Fabian Breg and Constantine D. Polychronopoulos. Java virtual machine support for object serialization. In *ISCOPE Conference on ACM 2001 Java Grande*, pages 173–180. ACM Press, 2001. ISBN 1-58113-359-6.

[110] Jin-Wook Baek, Jae-Heung Yeo, Gyu-Tae Kim, and Heon Y. Yeom. Cost effective mobile agent planning for distributed information retrieval. In *International Conference on Distributed Computing Systems(ICDCS)*, pages 65–72, Mesa, Arizona, USA, April 2001.

[111] Katsuhiro Moizumi and George Cybenko. The traveling agent problem. *Mathematics of Control, Signals and Systems*, 14(3):213–232, 2001.

[112] Rong Xie, Daniela Rus, and Cliff Stein. Scheduling multi-task agents. In *Mobile Agents: 5th International Conference, MA 2001*, number 2240 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[113] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1): 31–41, 1997.

[114] Dag Johansen. Mobile agent applicability. In *Proceedings of Mobile Agents (MA) 1998*, number 1477 in Lecture Notes in Computer Science, pages 80–98. Springer Verlag, 1998.

[115] Ryo Tsukui, Phil Beaumont, Tatsuji Tanaka, and Katsuhiko Sekiguchi. Power system protection and control using intranet technology. *IEE Power Engineering Journal*, pages 249–255, October 2001.

[116] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2): 195–240, 1995.

[117] Kim Marriott and Peter J. Stuckey. *Programming with constraints: an introduction.* MIT Press, Cambridge, Mass., 1998. ISBN 0262133415.

[118] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.

[119] D. P. Buse, P. Sun, Q. H. Wu, and J. Fitch. Mobile agents for industrial information management, monitoring and control. In *Proceedings of Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2003)*, Vienna, Austria, 2003. ISBN 1740880692. CDROM.

[120] Stefano Campadello, Heikki Helin, Oskari Koskimies, and Kimmo Raatikainen. Wireless Java RMI. In *Proceedings of the 4th International Enterprise Distributed Object Computing Conference*, pages 114–123, Makuhari, Japan, September 2000. IEEE Computer Society.

[121] Marcio Delamaro and Gian Pietro Picco. Mobile code in .NET: A porting experience. In *Mobile Agents: 6th International Conference, MA 2002*, number 2535 in Lecture Notes in Computer Science, 2002.

[122] Steven J. Vaughan-Nichols. XML raises concerns as it gains prominence. *IEEE Computer*, 36(5):14–16, May 2003.

[123] Brendon Cahoon and Kathryn S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In *Proceedings of the 19th annual international Association for Computing Machinery Special Interest Group on Information Retrieval (ACM SIGIR) conference on Research and development in information retrieval*, pages 110–118. ACM Press, 1996. ISBN 0-89791-792-8.

[124] Norbert Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems (TOIS)*, 17 (3):229–249, 1999.

[125] R. Kazman, M. Klein, and P. Clements. ATAM: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Carnegie Mellon University, 2000.

[126] Stanislaw Szejko. An exercise in evaluating significance of software quality criteria. *ACM SIGCSE Bulletin*, 31(3):199, 1999.

[127] Steve Haacke, Sam Border, Dehn Stevens, and Bob Uluski. Plan ahead for substation automation. *IEEE Computer Applications in Power*, 1(2): 32–41, March/April 2003.

[128] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems.*

[129] Alan Fedoruk and Ralph Deters. Improving fault-tolerance by replicating agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 737–744. ACM Press, 2002. ISBN 1-58113-480-0.

[130] G.A. Kaminka. *Execution Monitoring in Multi-Agent Systems.* PhD thesis, University of Southern California Computer Science Department, 2000.

[131] N. H. Huxley. *Intelligent Monitoring of Pre-Term Babies During the First Weeks of Life.* PhD thesis, The University of Liverpool, 2001.

[132] David John Musliner, James Hendler, Ashok K. Agrawala, Edmund H. Durfee, Jay K. Strosnider, and C. J. Paul. The challenges of real-time AI. Technical Report CS-TR-3290, University of Maryland, 1994.