

Chapter 2

Agent-based System for Knowledge Acquisition and Management Within a Networked Enterprise

A.J. Soroka

Abstract The examination of tasks involved in the gathering and processing of fault information with enterprises has shown that there are several stages where human errors can occur and has also revealed inefficient and time-consuming operations, resulting in bottlenecks that can reduce the potential benefits of automatic rule generation. Therefore, these various tasks could themselves be automated through the use of agent-based systems and machine learning techniques. This chapter shows that it is possible to automate the gathering and manipulation of fault reports using an agent-based system. Such a system can remove the need for manual processing of fault reports and the problems that may result from this. It also shows that a finite state automata (FSA)-based agent architecture is suitable for application in this particular problem domain, due to the reactive nature of an FSA. An FSA and state-table approach coupled with the modularity of the system should also enable it to be modified readily for different applications.

Keywords Multi-based system, machine learning, knowledge acquisition, networked enterprise

2.1 Agent-based and Related Systems

“Agent” has become a very popular term in the fields of computer science and artificial intelligence (AI) (Nwana and Ndumu, 1996). Agent-based systems are being heralded as the next significant breakthrough in software development and the new revolution in software (Wooldridge and Jennings, 1995). In comparison to other topics of AI research, agent research is comparatively young. There is no unanimously agreed date as to when agent research began, for example in Maes

A.J. Soroka (✉)

Manufacturing Engineering Centre, Cardiff University, Queen’s Buildings, Cardiff CF24 3AA, UK

e-mail: SorokaAJ@cardiff.ac.uk

(1995) 1985 is cited, yet in Wooldridge and Jennings (1995) and Nwana and Ndumu (1996) 1977 is cited. Therefore, it could be considered that the majority view is that the start date is 1977, pertaining to Hewitts work on actors (Hewitt, 1977). However, the term intelligent or autonomous agent is not explicitly mentioned in Hewitt (1977) and this is the reason for some of the confusion relating to when agent research began.

2.1.1 Origins of Agent Research

Agent research as it currently stands can be considered to have had perhaps two inter-related starting points, namely blackboard (BB) systems and distributed artificial intelligence (DAI). Therefore these two areas are briefly discussed before agents are considered in more detail.

Blackboard Systems

The BB system approach to problem-solving is based upon the concept of several knowledge sources (KSs) working on a problem via a globally accessible data structure known as a BB, which contains all solution elements generated during problem solving (Englemore and Morgan, 1988; Craig, 1995). The only communication these KSs have between one another is by adding or modifying entries on a BB. The operation of the KSs is normally governed by some form of control system, known as a scheduler, which implements the chosen problem-solving strategy (Craig, 1995). The KSs within a BB system are essentially expert systems that are dedicated to solving problems in a specific area. A typical structure of a BB system is shown in (Figure 2.1). Even though the basic ideas of BB and agent systems are similar, they differ from each other as it can be considered that there is no autonomy within a BB system. This is because of the scheduler, which reviews the KSs and selects the most appropriate KS. Therefore, the KSs do not decide of their own accord whether to act or not.

BB systems have typically been applied to tasks such as speech understanding (Hearsay II (Erman *et al.*, 1988)) and image understanding (Dreper *et al.*, 1988), where multiple KSs are required to perform the overall task. The technique of using a common BB is still being employed today, along with other related technologies, but in the domain of agent-based systems where agents replace KSs. Several examples exist of such applications. One is Guardian (Hayes-Roth *et al.*, 1994), which is an agent-based patient-monitoring system for an intensive care unit that employs a BB-based control architecture. Another example is described in Holt and Rodd (1994), which discusses how a dynamic BB is used to facilitate inter-agent communication within a manufacturing system. Concepts from BB systems have also been applied in the area of DAI and agent systems for conflict

resolution and co-operative working within an intelligent design system (Lander *et al.*, 1990).

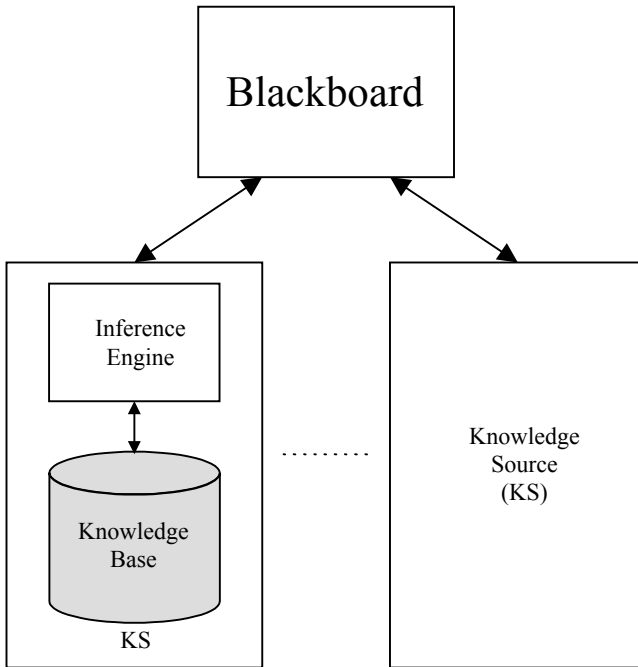


Fig. 2.1 Structure of a typical BB system

Distributed Artificial Intelligence

Distributed AI is based upon the principle of co-operative problem solving by a decentralised group of agents. These agents can take the form of a simple processing element or a complex entity (intelligent agents) (Huhns, 1987). The term distributed is used because in DAI the processing elements (or agents) are distributed amongst many computers, thereby creating a problem-solving system that is very robust.

DAI research can be split into three different areas, distributed problem solving, multi-agent systems (MAS), and parallel AI (Bond and Gasser, 1988). The MAS research conducted within the area of DAI tended to concentrate upon *macro* phenomena (the social level) (Wooldridge and Jennings, 1995). This resulted in a great deal of work having been carried out on collaborative and co-operative agents and their interaction for joint problem solving. Examples of research conducted in this area include work by Georgeff concentrating on methodical and theoretical aspects (Georgeff, 1988). A significant amount of work was performed on inter-agent negotiation and collaboration using various approaches

(Sycara, 1989; Jennings, 1992, 1995; Osawa and Tokoro, 1992; Khedro and Genesereth, 1994; Sen *et al.*, 1994). Some research in this area looked specifically at applying game-theory techniques (Axelrod, 1984) to multi-agent systems (Zlotkin and Rosenschein, 1989; Ito and Yano, 1995), in an attempt to foster and facilitate co-operation between agents. From this work relating to co-operative behaviour, several frameworks and systems were proposed for the collaborative operation of agent systems (Lander *et al.*, 1990; Jennings, 1992, 1995; Jennings *et al.*, 1992).

A natural extension of the work pertaining to agent collaboration and co-operation can be considered to be the creation of agent communication and knowledge interchange languages and formats. Examples include the Knowledge Interchange Format, which deals with message content (Genesereth and Fikes, 1992), Knowledge Query and Manipulation Language, which is concerned with the performance of actions (Mayfield *et al.*, 1996; Labrou and Finin, 1997) and also the FIPA (Foundation for Intelligent Physical Agents) agent communication language specification (FIPA, 1997). Such languages were aimed at enabling communications between homogenous agents operating within a system and also heterogeneous agents existing within different systems created by different people, thus attempting to achieve the goal of interactivity between all agents.

It can perhaps be considered that DAI and agent system research have become synonymous with each other in recent times, with researchers in one domain conducting research in the other. As such there exist agent-based systems whose origins are within the area of DAI.

2.1.2 Definition of an Agent

Currently there exist many different “definitions” of what an agent is and names for “agents”, for example, intelligent agents, autonomous agents, robots or hardware agents, and softbots or software robots. This has resulted in a degree of confusion such that the question “What is an agent?” is regarded to be as embarrassing to the agent-based computing community as the question “What is intelligence?” is to the AI community (Wooldridge and Jennings, 1995). In (Wooldridge, 1996ab) it is said that obtaining a definition for an agent is like the Rorschach test, because every person has his own answer. Whilst at first glance this may appear to be a cynical opinion, the next section illustrates that this is very much the case.

This confusion can also be partially attributed to the fact that there were many uses of the term agent prior to its adoption by the agent-based computing community. For example, the term refers to people or organisations that perform services for others such as estate or travel agents. There are also uses in the areas of biology and chemistry, biological and chemical agents respectively, which are either organisms or chemicals having an effect on something else be it animal, mineral or vegetable.

Agent definitions in general can be grouped together according to the sets of characteristics that are specified for an agent. Several definitions specify the characteristics dependent upon the operation of the agents. For instance, agents act within an environment and can sense or receive stimuli from the environment and can act upon and affect that environment to achieve a set of goals (Hayes-Roth, 1993; Gibbins *et al.*, 1994; Maes, 1995; Franklin and Graesser, 1996; Kearney, 1996). Other definitions are based upon features that characterise an agent, such as the ability to act autonomously, react and plan (Jennings and Wooldridge, 1995; Ekdahl *et al.*, 1995; Moffat and Frijda, 1995; Alty, 1997; Wooldridge, 1996ab). Others such as Struthers (1996) define an agent in terms of both structural and functional aspects. Some examples are very specific, such as Genesereth and Ketchpel (1994) and Singh and Huhns (1999) which give the definition that an agent is only an agent if it communicates with another agent.

From this it can be appreciated that it may not necessarily be feasible or practical to give an exact definition of an agent. Indeed it is considered that the only agreement concerning definitions is that there is no one single definition (Singh and Huhns, 1999).

There is, however, a need to give a generic description of ideas that agents encompass. A simple specification is given in Foner and Crabtree (1996). If it were not for the mention of performing personalised tasks, which may or may not be applicable to all agent systems, that description would be generic and encompass the majority of agent definitions. A simple generic definition of an agent adapted from Foner and Crabtree (1996) could be “A piece of software, or hardware, that acts in an autonomous manner to complete a task or tasks”. This definition does not specify how an agent operates or what it features but rather can be applied to most agents.

2.1.3 Agent Architectures

Agent architectures are mainly concerned with the implementation of a proposed agent system and the implementation of theories of agency, where the theories deal with issues such as how an agent should behave, its properties, and how to conceptualise these ideas. There are several agent architectures that are commonly utilised: deliberative, reactive, and hybrid.

Deliberative

Deliberative agents are agents that depend upon models of the environment to decide what course of action to take. It is stated in Wooldridge and Jennings (1995) that the term deliberative agent stemmed from Genesereth and Nilsson (1988) who stated that “The key in defining an agent in this class is the use of an automated

inference method like resolution in deriving a sentence that indicates the required action on each cycle”.

Several examples exist of agents based implicitly on deliberative architectures. A significant proportion of these architectures possess models of beliefs, desires and intentions (BDI) (Bratman *et al.*, 1988; Jennings *et al.*, 1992; Rao and Georgeff, 1995). Others possess similar properties such as beliefs, actions and plans (Kumar and Shapiro, 1991). Beliefs are an expression of what the agent currently knows about its environment, desires are an expression of what the agent wishes the future states of itself or other agents to take and intentions are goals that the agent would like to fulfil. These features enable the agent to make decisions about how to perform a task that are dependent upon BDI such that what an agent will do will be affected by the current state of its model. This attempts to mimic the behaviour of humans, where a decision is typically made based upon the knowledge the person possesses.

Examples of practical applications of deliberative agents include GRATE (generic rules and agent model testbed environment) and its derivations (Jennings *et al.*, 1992, 1993).

However it is considered that deliberative agents, which have to perform theory proving and symbolic manipulation, are hard to implement due to problems with theory proving and the computational complexity of symbolic manipulation (Wooldridge and Jennings, 1995). In Chapman (1987) it was shown that even refined planning techniques such as hierarchical and non-linear planning would prove unusable in time-constrained systems (Wooldridge and Jennings, 1995).

Reactive

The main aim of reactive architectures is to overcome some of the issues and perceived weaknesses of deliberative architectures, thereby creating a robust, flexible and fault-tolerant system (Nwana and Ndumu, 1996).

Reactive agents or agent architectures are also known as reflex, behaviour-based or situated. Reactive agents differ from deliberative agents as they do not possess any models of their environment but rather respond to stimuli. Therefore the agent's reaction depends upon the current state of its environment. In Maes, (1991) three concepts are given that underpin reactive agents: emergent functionality—reactive agents are relatively simple and interact in basic ways and yet complex behavioural patterns can emerge; task decomposition—an agent is viewed as a collection of autonomous modules; reactive agents tend to operate on representations that are close to raw data, in contrast to the high-level symbolic representations that occur in other types of agents.

One potential drawback to reactive systems was considered to be that there was no guarantee that a reactive agent would provide an efficient solution (Knight, 1993). It was, however, discovered that the problem could be solved by increasing the number of reactive agents that are simultaneously attacking a problem. Re-

search has also shown that this is better with regard to improving the efficiency of a reactive system than increasing the amount of deliberative reasoning used.

Most reactive architectures can be considered to be developed around various forms of rule-based architecture (Agre and Chapman, 1987; Kowalski and Sadri, 1996; Meghini, 1997). This is primarily due to agents very much being governed by logical formulas or action rules that specify how they should behave.

It is considered by some that relatively few actual examples of applications of reactive agent architectures exist (Nwana and Ndumu, 1996), and as such there exists no standard mode of operation. However, several can be found, mainly within the robotics or hardware agent domain, which seem to be ideally suited to reactive architectures.

Perhaps the most cited agent system based upon a reactive architecture is PENG1, a game-playing program (Agre and Chapman, 1987). The authors stated that most everyday activities can be considered routine and as such requires little new learning. The PENG1 agent decides what to do depending on the current state of the environment.

Other examples include the subsumption and behaviour-based architectures presented in Brooks (1986, 1991). This architecture employs a layer-based control system. These layers control the various behaviours of a mobile robot. In the subsumption architecture these various layers compete with one another to be activated and therefore to control the robot. The lower layers within the system represent relatively primitive operations and can be activated more easily, therefore having precedence over the higher levels. However, these higher levels are able to subsume the roles of the lower ones by suppressing their outputs.

Work similar to that of Brooks has been conducted into agent network architectures (Maes, 1989, 1991). Maes' network comprises types behaviour-based modules or competence modules. These modules will perform certain tasks, but as with the subsumption architecture they must compete to become active. This is because every competence module has an activation level that must be reached before it becomes active. These modules have links connecting them that can be dynamically changed as the system gains more experience, thereby achieving flexible operation.

Another application in the area of robotics is that of situated automata (Kaelbling, 1991; Rosenschein and Kaelbling, 1995). The idea of situated automata is related to that of synchronous sequential digital circuits. These automata are then represented using formal logic. This is then encoded into a program, which is compiled to produce the agent. This approach enables the robot to operate in a time-bounded manner.

Hybrid

Hybrid architectures are a compromise between the deliberative and reactive paradigms, in that they combine the desirable properties of the two approaches. Researchers working upon hybrid agent architectures argue that neither of the two

paradigms offer the most effective solution, but rather a combination would (Wooldridge and Jennings, 1995; Fisher, 1995). An analogy for hybrid architectures could be drawn here with the human body in that some actions are triggered by reflex (for example when a hot object is touched) whilst others are planned (for instance, lifting up a cup).

Most hybrid architectures have a tendency to be composed of layers (Ferguson, 1995; Muller *et al.*, 1995; Muller, 1996; Schroeder *et al.*, 1996). Within these layer-based architectures, one set of layers will perform reactive tasks whilst the others will perform deliberative tasks.

Examples of agents based upon hybrid architectures include the Procedural Reasoning System (Georgeff and Lansky, 1987), which has the BDI attitudes of deliberative agents and has been constructed for mobile robots. Another system constructed for robots is Interrap (Muller *et al.*, 1995; Muller, 1996) which simulates robots operating within a loading bay.

Other examples of hybrid architectures include Touring Machines (Ferguson, 1995), where a MAS test-bed is developed and Diagnostic Agent (Schroeder *et al.*, 1996), which also possesses BDI models.

2.1.4 Agent Types and Applications

There are many different types of agents mentioned in the literature, and attempts have been made to categorise them into groups. However, the boundaries between different types of agents are fuzzy as an agent can in general be placed in several different types. As a result of this the type is closely aligned to the application domain within which the agent is being used. The different agent types found in the literature (Wooldridge and Jennings, 1995; Nwana and Ndumu, 1996) include collaborative/co-operative, interface, information/Internet and mobile agents.

Collaborative/Co-operative Agents

The distinction between collaborative and co-operative agents is not necessarily clear as the two terms are often used in an interchangeable manner. However, if separate definitions were to be given, the following could be said. Collaborative agents are agents that interact with other agents in an attempt to accomplish a common set of tasks or achieve a common goal. Co-operative agents, however, interact with each other on rare occasions, to assist each other in completing their own separate tasks, as opposed to achieving an overall goal. Collaborative and co-operative agent research is the most mature research area within the agent field due to the large quantities of work performed by the DAI community. However, despite this it is considered by some that there have been relatively few industrial applications (Nwana and Ndumu, 1996).

In Jennings *et al.* (1993) a collaborative agent system for the control of particle accelerators is proposed. Other applications are in business process management (Alty *et al.*, 1994; Jennings *et al.*, 1996, O'Brien and Wiegand, 1997). Here agent-based systems assist in providing access to documentation relevant to a decision-making process, identifying parties that may be interested in results and informing decision makers about changes that might impinge on the decision-making process. Services within the system are provided by a community of collaborative agents that have had particular tasks assigned to them. Other applications of collaborative agents include a distributed system for the management of patient care (Huang *et al.*, 1995).

Interface Agents

“Interface agent” could be considered a somewhat misleading term as most interface agents do not, as the name suggests, act as an interface between a user and the computer. In reality most successful interface agents are not of this type (Maes, 1994). An interface agent essentially performs the task of a personal assistant which works in collaboration with the user to relieve the need to perform repetitive and time consuming tasks. As such, interface agents can be regarded as assistant agents, which defines their purpose more clearly.

There exist interface agents that perform mainly assisting tasks such as helping the user of a kiosk-based information delivery system (Mamdani and Charlton, 1996). There are agents which assist in the scheduling of meetings (Kozierok and Maes, 1993; Maes, 1994) where an agent learns the user's meeting scheduling patterns and then suggests suitable meeting times. A further progression of this idea is given in Franklin *et al.* (1996) where a clerical agent that creates university seminar schedules is proposed. The agent receives e-mail from seminar organisers, extracts the necessary data, and then checks for any time/room conflicts.

Other tasks performed by interface agents include filtering of newsgroups (Maes, 1994; Sheth, 1994) and e-mail messages (Lashkari *et al.*, 1994; Maes, 1994), where agents search for interesting news, according to criteria specified by the user, and filter important and unimportant e-mail, based upon behaviour patterns, respectively. An application similar to this is performed by the web browsing assistant Letizia proposed in Lieberman (1995, 1997, 1999). This agent system builds a series of heuristics regarding a user's web-browsing habits and searches links within a page to find potentially interesting websites, and then presents a summary of potentially interesting web pages to the user.

Internet/Information Agents

Information agents tend to work in the areas of collecting, manipulating, and processing information from different sources. These sources can take many forms, for example, databases and document repositories. They could be within one system

or distributed. The distributed sources could either be distributed throughout one or many organisations, which leads to Internet agents.

Internet agents perform what can be considered to be essentially the same role as information agents. However, they are a specific form of information agents targeted at using the Internet as the source of knowledge or information. This involves them performing similar collection, manipulation and processing functions where the information is distributed amongst many different sites on the Internet.

There are several examples of applications of both information and Internet agents. For example, the RETSINA infrastructure (Sycara, 1999) is a multi-agent system for assisting users in decision-making processes and information-management tasks such as information gathering, filtering and integration.

Several examples exist of agents and their application to business processes. In Papazoglou and van den Heuvel (2000) an agent-based co-operative information system is proposed, and there is also the ADEPT project (Alty *et al.*, 1994; Jennings *et al.*, 1996, O'Brien and Wiegand, 1997), although it is a collaborative agent system, which performs business information-processing tasks on behalf of its users.

Some other examples of information agents include Carnot (Huhns *et al.*, 1992), which is a system that comprises several databases and enables queries to be answered that are not within the scope of any of the databases and the information retrieval agent (Voorhees, 1994), that facilitates the searching of articles held with document repositories.

The distinction between Internet and other agents sometimes is not necessarily clear. For example, some forms of interface agent such as Letizia (Lieberman, 1995) could in their own right be considered as a form of Internet agent, as they collect and process data from a website and provide a concise set of links to websites that the user may wish to follow or be interested in. Internet agents can also theoretically be mobile agents. This means that there is a great deal of overlap between Internet/information and other forms of agent.

An agent that could be termed an Internet-based information agent is Jasper (Davies *et al.*, 1997), which is an agent that attempts to summarise and visualise information stored on the Internet.

Mobile Agents

In Gray *et al.* (1996), the following definition of a mobile agent is given: "Mobile agents are programs that can move through a network under their own control, migrating from host to host and interacting with other agents and resources on each". It has been reported that Maes, in an interview, argued that there is no satisfactory answer to the question "what can you do with mobile agents that you cannot do with stationary agents?". This may be an overstatement but there are many applications that could be achieved effectively using a population of stationary agents.

However, applications of mobile agents exist including those in e-commerce (White, 1994; Andreoli *et al.*, 1997) where agents perform such tasks as finding cheapest prices or negotiating services. Mobile computing is another area where mobile agents have been applied (Gray *et al.*, 1996). Here tasks are performed remotely from mobile computers therefore conserving resources for users of personal digital assistants. Mobile agent-based Internet search engines have also been proposed (Kato *et al.*, 1999).

2.1.5 Machine Learning for Generation of Knowledge Bases

Inductive learning algorithms are algorithms that convert specific data into general rules (Quinlan, 1988; Forsyth, 1989; Hancox *et al.*, 1990). The purpose of inductive learning can be considered to be two-fold, firstly to perform a synthesis of new knowledge, which is independent of the form of the original input data (Kodratoff, 1988) and secondly to ease the knowledge acquisition bottleneck that occurs when knowledge-based systems are developed.

An inductive learning algorithm will typically either produce a rule set of IF...THEN rules or a decision tree (Figures 2.2, 2.3). However, the decision tree generated by the algorithm can be easily converted into a set of production rules (Al-Attar, 1991; Quinlan, 1987). This is because every branch of a decision tree can be considered as an IF...THEN rule. The rule set derived from the decision tree illustrated in Figure 2.2 is shown in Figure 2.3. It should be noted that it may not necessarily be possible to convert a rule set into a decision tree.

Inductive learning algorithms can take two different forms, non-incremental, or batch, and incremental. Non-incremental algorithms use the entire set of training examples to generate a rule set. If a new example is added, the entire historical set of examples plus the new example must be used to generate the new rule set. With incremental learning algorithms, only the new example need be presented to the algorithm. There are several advantages in the use of incremental learning algorithms including being able to update stored knowledge rapidly and to perform on-line learning (Schlimmer and Fisher, 1986). However, incremental algorithms only makeup a minority of the inductive learning algorithms in use (Pham and Dimov, 1997b).

There exist several well-known families of inductive learning algorithms, which contain both the incremental and non-incremental types. Some of these families of algorithms are examined and described below.

AQ family

The AQ family includes AQ11 (Michalski, 1983), CN2 (Clark and Niblett, 1989; Clark and Boswell, 1991), and the incremental AQ15 (Michalski *et al.*, 1986) learning algorithms.

Members of the AQ family generate sets of decision rules from a series of instances or examples presented to the algorithm. The AQ family is based upon the “extension-against” approach. This means a hypothesis rule space is generated based upon an example set that is refined to find more general rules.

The AQ algorithm performs a heuristic search through a space of logical expressions or descriptions that cover only examples of the one selected type of class (positive examples). Then, according to a set of criteria, the final rule is chosen. AQ15 (Michalski *et al.*, 1986) when being employed for incremental learning performs this operation with what is termed a “perfect memory” in that both rules and examples used to create rules are recorded. This also enables the user to specify rules *a priori* then the algorithm can build upon these rules if required.

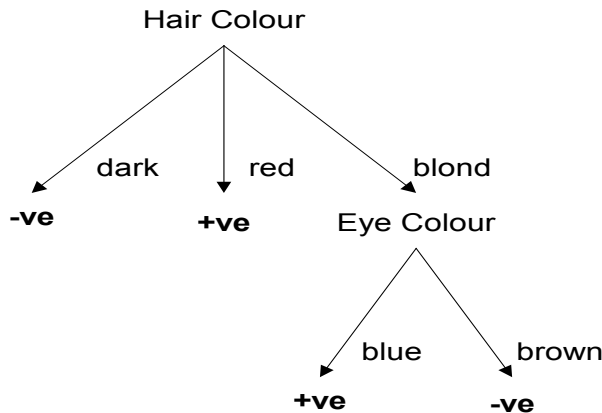


Fig. 2.2 Example decision tree

IF Hair Colour = dark THEN – ve
 IF Hair Colour = red THEN + ve
 IF Hair Colour = blond AND Eye Colour = blue THEN + ve
 IF Hair Colour = blond AND Eye Colour = brown THEN – ve

Fig. 2.3 Rule set for example decision tree

The rule generation process of the AQ family can basically be summarised as follows (Aksoy, 1993):

1. Select an unclassified example. The class of this instance is considered to be positive whilst the others are negative.
2. Apply the “extend against” approach to produce a complete set of descriptions for the current instance.
3. From the set of descriptions produced in the 2nd step, a description is selected based upon its consistency (related to a description and negative classes) and completeness (related to a description and positive classes).
4. If the description is complete, i.e. it covers all positive classes, then go to the 6th step.

5. If not, then reduce the size of the positive set to include only those instances which are not covered by the description. Go to the 1st step.
6. Convert all descriptions into a rule that can cover all examples in the positive class. If no more unclassified examples exist then stop. Otherwise go to the 1st step again.

ID3 Family

The ID3 family of learning algorithms is based upon the ID3 learning algorithm (Quinlan, 1986, 1987, 1988), which itself is based upon the concept learning system (Hunt *et al.*, 1966). The family includes the ID4 (Schlimmer and Fisher, 1986) and ID5 (Utgoff, 1988) incremental learning algorithms. The ID3 family also includes the C4.5 and C5.0 learning algorithms.

The ID3 family of algorithms generate multi-branched decision trees from the set of examples that they are presented with. ID3 uses an information theory approach to decision tree generation in an attempt to minimise the number of tests required to classify an object (Quinlan, 1986). The ID3 family follow what is known as the top-down approach, which refers to the breaking down of a set of examples into several subsets until there remains a unique class in each of these subsets. The rule generation method of ID3 can be expressed as a basic number of steps that describe the process involved.

1. Select an attribute to divide the set of examples into subsets.
2. If there are subsets containing examples in different classes, return to step 1.
3. Stop when, within each subset, there are examples in one class only.

A modified version of ID3, ID3-IV (Quinlan, 1986), named in Cheng *et al.* (1988), which includes a function for calculating information gain was produced, as ID3 has a tendency to choose condition attributes that have more values, thus causing generic ones to be missed. Another version for solving the irrelevant condition problem (gives rise to rules testing unnecessary attributes), called GID3 or generalised ID3 (Cheng *et al.*, 1988), also uses the idea of the information gain ratio.

The ID4 (Schlimmer and Fisher, 1986) approach of generating decision trees differs from the approach adopted by ID3. As opposed to using a set of examples, ID4 generates and updates the tree based upon each example within the data set. At each node in an ID4 decision tree there is located a table, within which are contained values of non-test attributes and the numbers of positive and negative examples. When the information measure is calculated for a non-test attribute and it transpires that it is lower than the value for the current test attribute, the sub-trees below this node are deleted and the node itself is modified.

ID5 (Utgoff, 1988) differs not only from ID3, but also from ID4. ID5 builds on the concept introduced with ID4 of keeping counts of positive and negative examples of each attribute that could be used to create a test attribute (Utgoff, 1988). The principal difference between ID4 and ID5 relates to the methods used for the

replacement of test attributes. Instead of the method of discarding the sub-trees below the test attribute, the decision tree is reorganised by adding or changing the positions of current nodes.

The C4.5 algorithm has essentially the same structure as the ID3-IV algorithm and is considered to be the “industrial” version of the algorithm. However, there have been several modifications. For example, C4.5 has decision tree pruning features and, once a tree has been pruned, it is converted into rule sets. Also, C4.5 is able to deal with continuous numbers, noise, and missing values. C5.0 is a modified version of C4.5, with new features added including variable misclassification costs and additional data types.

RULES Family

The RULES family of algorithms consists of RULES-1, RULES-2, RULES-3 (Pham and Aksoy, 1993, 1995), RULES-3+ (Pham and Dimov, 1997a) and RULES-4 (Pham and Dimov, 1997b), which is the incremental version of the algorithm. The RULES family was developed in the author's laboratory.

The rule-forming procedures of the different variants of the RULES family all work in slightly different ways, except for RULES-3+ and RULES-4 where the processes are essentially the same. Basically the algorithm takes an example from a set of unclassified instances, then attempts to form a set of objects (proto-rules) for it using different combinations of conditions based upon the attributes and the values in the example. Objects which do not belong to the class of the example are removed. From the remaining objects the best rule is selected. Then all examples that are classified by this rule are removed from the list of unclassified examples.

The following gives an overview of the rule generation process of the RULES-3 learning algorithm:

1. Define the ranges for continuous attributes (a user action).
2. Specify the minimum number of conditions for each rule (a user action).
3. $\text{Number_Conditions} = \text{minimum number of conditions} - 1$.
4. Take an unclassified example.
5. If Number_Conditions is less than Number_Attributes , then increment Number_Conditions by one.
6. Get all values and attribute labels for example.
7. Form objects that consist of Number_Conditions , conditions, using values and labels in the example.
8. If one or more objects belong to a unique class then form rules with them, ELSE go to the 5th step.
9. Select the rule that classifies the most examples.
10. Remove these examples from the list of unclassified examples.
11. If there are no more unclassified examples then stop. Otherwise go to the 3rd step.

The rule-forming procedure of RULES-3+ and RULES-4 differs from that for RULES-3. This derives mainly from the fact that the information content, generality and accuracy of rules are used in the algorithms (Pham and Dimov, 1997b).

The incremental learning procedure in RULES-4 uses two forms of memory, the long-term memory (LTM) and short-term memory (STM), which contain the generated rules and a selection of the examples used to form the rules. Firstly when an example is chosen a check is performed as to whether the rules in the LTM classify the example. At this stage the accuracy and information content measures are updated and the LTM is pruned if the accuracy falls below a given threshold. Then, depending on whether there are free spaces in the STM and if the example is not covered by the LTM, the example is added to the STM. If an example in the STM is not covered by the LTM then the rule-forming procedure is applied, and the new rule is added to the LTM, until there are no examples uncovered by the LTM.

2.2 Product Fault Knowledge Acquisition and Management

2.2.1 Automating Knowledge Base Management

A suitable means for the automation of the knowledge base management process is the combined use of an agent-based system together with machine learning techniques for knowledge base generation. Agent-based systems as shown in the previous section are ideally suited to information-processing tasks and have been employed in many different information-processing applications (Alty *et al.*, 1994; Etzoni and Weld, 1994; Maes, 1994; Sheth, 1994; Jennings *et al.*, 1996; Davies *et al.*, 1997; O'Brien and Wiegand, 1997; Papazoglou and van den Heuvel, 2000; Leiberman, 1999; Pham *et al.*, 2001). Agent systems are also ideally suited to operating within distributed environments (Huang *et al.*, 1995; Jennings and Wooldridge, 1998). Machine learning techniques have a long history of being applied to the generation of rule sets for use in knowledge-based systems as a means of reducing the bottlenecks which occur when creating such systems (Pham and Dimov, 1997b; Quinlan, 1988).

The agent-based system proposed within this chapter performs both user assistance and information gathering and processing and could therefore be perceived as being an interface agent.

The proposed agent-based system could be developed to adopt either an approach that uses the ISLE rule extraction algorithm, or to utilise an approach involving a combination IFCLEAR and RULES-4 (or ISLE without its unclassified example abilities). As the intention is also to provide a generic framework to allow any learning algorithm to be employed (such as ID3 or C5.0), it was decided to implement an agent-based system that encompassed both approaches thus increasing the potential applicability of the proposed architecture.

2.2.2 Analysis of Knowledge Base Management Process

To be able to develop an agent-based system for the management of expert system knowledge bases, the manual processes involved in the knowledge gathering and processing procedures should be known. To do this, tasks and steps that are carried out in real life are analysed. From this analysis, the actual tasks that need to be performed by the different agents within the system can be determined and specified.

The decision-making and physical processes involved should therefore be understood before the structure and functionality of the agents can be specified. As such, the workflow processes for gathering and processing fault data are generated from an analysis of the processes involved at these two stages. It is essential to separate these processes even though the latter is dependent upon the former. This is because they take place at different locations and involve separate organisations.

Therefore the following two sections analyse and define business processes for the collection and processing of fault data respectively, taking into consideration the people, actions and decision-making processes involved at the various stages.

Collection of Fault Data

The process for a service technician compiling the fault report illustrated in Figure 2.4 can be considered relatively straightforward. However, there is the problem of report forms not being completed due to procrastination. An equally serious problem arises during the subsequent processing of the forms if they have not been completed correctly.

The first task a service technician must perform is to determine what type of fault report he has to provide: a new account of an existing class of fault, a description of a new type of fault or a report on a fault relating to attributes that are not included within the present knowledge base.

If the report contains a new fault or provides a new account of an existing fault then the service technician will need to specify the fault and the conditions that cause it. If a new attribute has to be defined because the current knowledge base does not contain it, details of the new attribute along with the relevant information would normally need to be specified.

Once the process of actually gathering fault information has been completed the report has to be forwarded to the manufacturer. This is another situation where problems can arise, namely that the report documents can be mislaid (either at the maintenance or manufacturing site) or perhaps not forwarded to the manufacturer. Therefore, potentially valuable information regarding product faults might be lost through elementary mistakes.

Generation and Management of Knowledge Bases

The procedure involved in the processing and management of fault data has been simplified with the development of a parser to convert a spreadsheet table containing both training and attribute data into the data and format files typically required by learning algorithms. This removes the need to perform the conversion manually. The files contain the data used to create the final rule set and information concerning what this data describes. The action of this parser is illustrated in Figure 2.5.

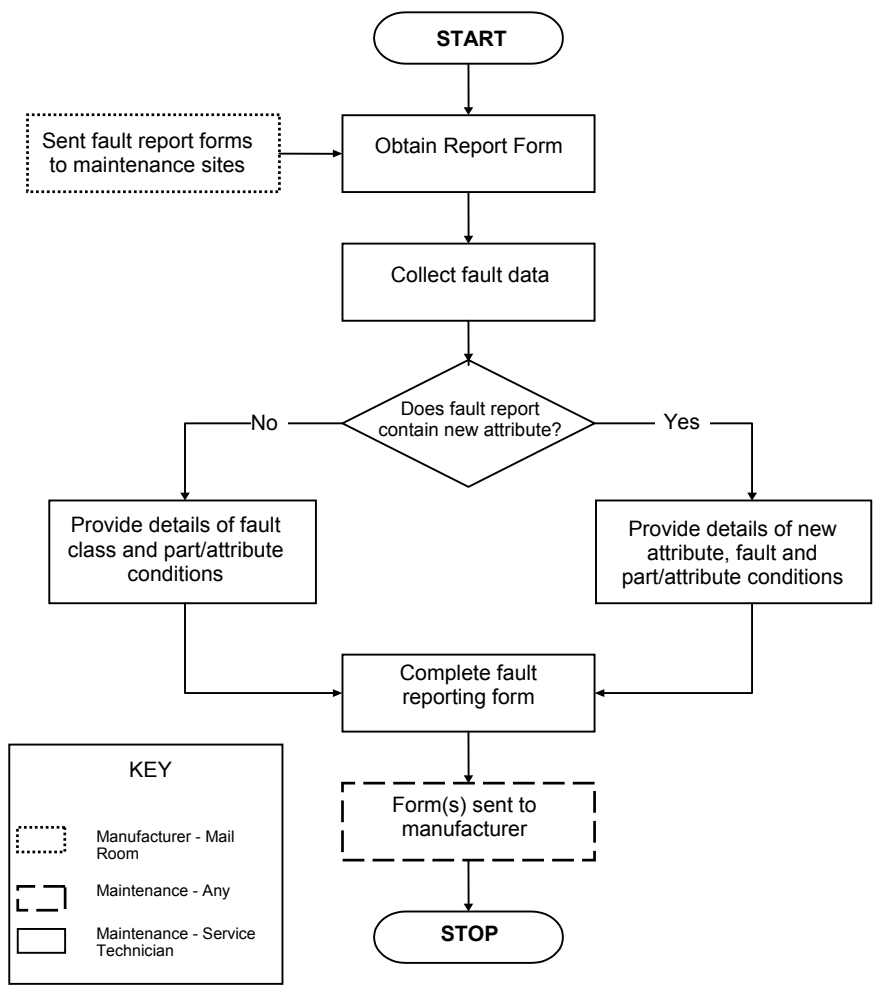


Fig. 2.4 Tasks involved in providing fault data

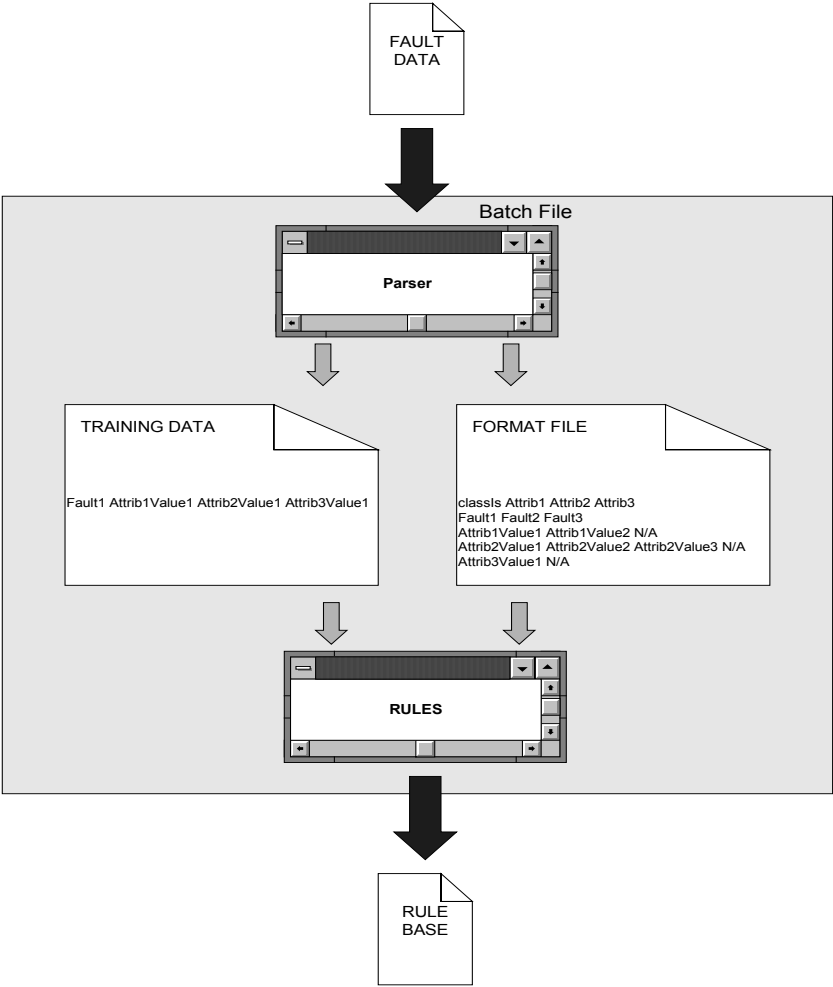


Fig. 2.5 Generation of rules from fault data file

Once a manufacturer has received a fault report, a more complex procedure and business process must be followed. The workflow for these tasks is illustrated in Figure 2.6. First, the fault report would be received in the mailroom (or its equivalent) within the organisation. The report then ideally would be passed to an administrative assistant who would ascertain whether it is legible and therefore useable. If not, depending upon its procedures, the company may wish to request that the form be resubmitted.

An acceptable fault report is then passed on to a suitably qualified and experienced engineer for processing. The data must then be checked to discover whether all attributes have been instantiated. If this is not the case, then a resubmission request is made as, without these items of data, the new scenario described in the fault report may be of little or no value. If the data is complete then a check is performed as to whether this fault is already covered by the knowledge base or instance history. If so, there is no need for further work as this fault is already known about. Otherwise the procedure continues.

The fault report is then checked to see if the fault class has been specified. If not, then it must be found. This task would typically be done by the engineer and might utilise several possible factors including his expert knowledge of the product and faults that have occurred with similar products or be based on attempting to replicate the fault. If the fault class was specified or discovered then the data pertaining to the new fault report is entered into a spreadsheet. At this stage, consistency checks have to be performed to ensure that the attributes and their values are correctly spelt and that the spreadsheet table is correctly formatted. This is particularly important as otherwise problems may arise, for example, an inaccurate rule set is generated or errors occur during the running of the machine learning algorithm itself. The data is then saved in a format appropriate for the parser. As illustrated in Figure 2.5 the parser is executed, which takes the fault data and generates the training and format data required by the machine learning algorithm. The latter in turn generates the knowledge base. The old knowledge base must be deleted and replaced with the newly created knowledge base.

Knowledge Base Management Issues

The above discussion has highlighted the work involved in the various tasks relating to knowledge base management. From this analysis, it can be determined that there are several problems inherent in the manual process that can potentially be overcome or mitigated by automation. The problems relate both to inaccuracies in the data and inefficiencies of certain processes resulting in bottlenecks within the overall knowledge management procedure. These problems, which have been already mentioned in different places, are summarised below.

- Failure to fill-in fault report forms: this might be caused by forgetfulness, procrastination, or an unwillingness to carry out the task.
- Non-receipt of report forms by manufacturer: this might arise from the forms not being sent or having gone astray during posting.
- Illegible fault report forms: this might be caused by untidy handwriting or oil, grease and dirt marks; consequently the data either is unusable or requires a large amount of time to interpret.
- Incomplete data: this might be due to carelessness during form completion.

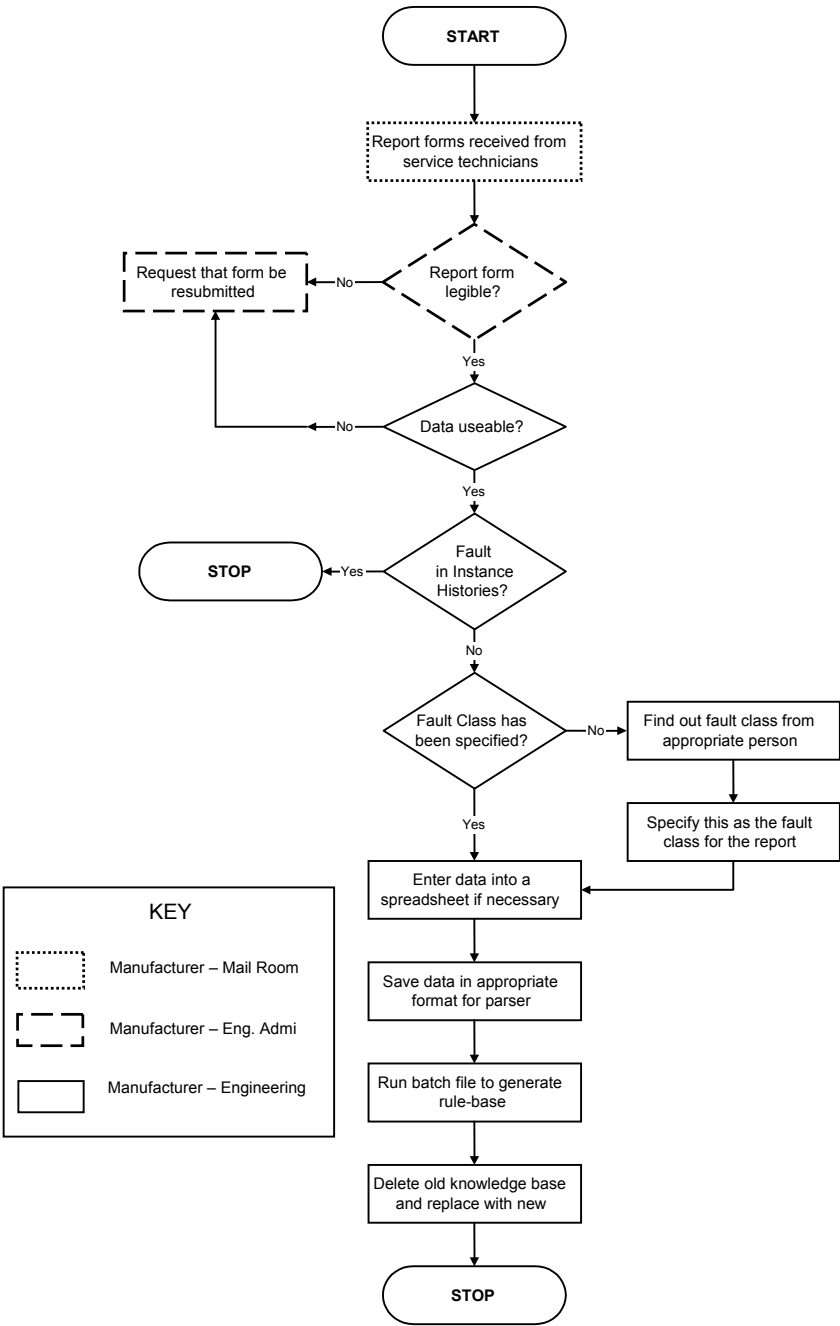


Fig. 2.6 Process – manufacturer’s side

- Lengthy movement of reports: the fault reports would typically pass through the hands of several people before reaching the relevant engineer, thereby wasting time and also increasing the risk of documents being delayed or misplaced.
- No guarantee as to when a report will be processed: the engineer responsible may be unavailable.
- Long and tedious data entry: entering data from a received report into a spreadsheet requires translation of data into an electronic form and is potentially very time consuming.
- Need to perform consistency checks and corrections: the data entered has to be checked to ensure that attributes and their values are consistent throughout the spreadsheet. This takes time particularly when data sets are large.
- Need to ensure the data is formatted correctly: this is similar to above but involves checking and arranging that attributes and their values are in the correct places within a document.

2.3 Agent System for Knowledge Acquisition and Management

The proposed agent-based system for collecting and processing new fault data consists of two agents, the server agent (SA) and user agent (UA), as illustrated in Figure 2.7. These agents perform two separate but interdependent roles, much as in the business processes discussed in the previous section.

From the illustration in Figure 2.7, it can be envisaged that UA operates within the end-user's computer and interacts with him. A "virtual form-filling" session collects and sends new fault reports to SA via a BB. SA then processes this new fault data and incorporates it into the knowledge base. The system is structured such that it allows SAs that manage the knowledge bases for different product variants to communicate with one another if necessary, in order to facilitate the retrieval of fault classes.

In the implementation of this proposed application, there is a requirement for communication between agents that operate at geographically distributed locations. This communication is envisaged to take place via local-area or wide-area networks, such as an Intranet and the Internet respectively.

To meet these requirements, this system is implemented using Sun Microsystems Java 1.1 programming language. Java was chosen as it is very network/Internet oriented and is platform independent, which is very important since the intelligent product manual (IPM) is intended as a platform-independent solution. Of particular use are the remote method invocation (RMI) features of Java, which facilitate the construction of agent-based systems. RMI allows Java-based programs to communicate with one another across a local-area network or a wide-area network. Java provides a useful means to develop small applications (applets), which are capable of being executed from within web pages viewed in a web browser such as Internet Explorer or Netscape. Since the IPM uses Internet-

based delivery (Pham *et al.*, 1998), this ability is of particular importance as it allows UA to be embedded within an HTML page in the user's computer.

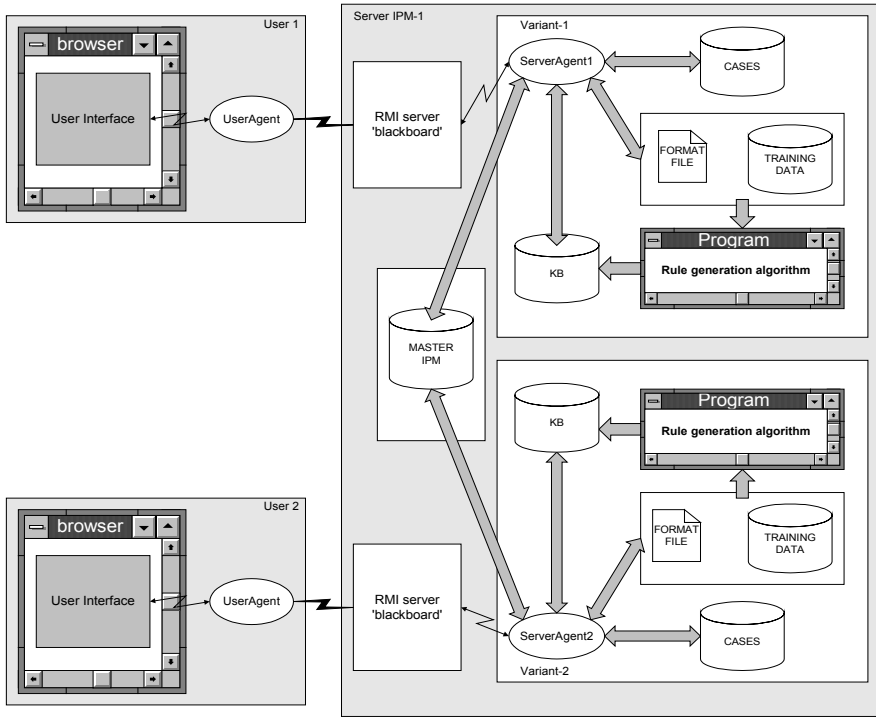


Fig. 2.7 Agent system

The system as a whole is not purely agent based but can be considered to be a hybrid of an agent-based system and a BB system in a manner similar to Lander *et al.* (1990), Hayes-Roth *et al.* (1994) and Holt and Rodd (1994). This is because there are certain BB features and concepts that are particularly appropriate to this system. Also RMI makes the system lean towards being BB based. The feature of BB systems that is of particular relevance is the globally accessible data structure known as a BB as mentioned in Englemore and Morgan (1988) and Craig (1995).

The utilisation of concepts from BB systems manifests itself in the BB server that is used as a channel or medium for all communications. The BB server receives and queues all messages sent to a particular SA in a first-in-first-Out (FIFO) manner. The FIFO strategy is the most logical in this application as it helps ensure that no request will wait a disproportionately long period before it is processed. The use of the BB server also removes the need for SA to perform any message management and thus frees SA to concentrate upon performing information-processing-related tasks. Once SA has finished a task, it can request that the

next message in the queue be forwarded. If there are no messages then SA goes into a wait state until it is given a new task to perform.

An additional benefit of this method is that it allows SA to be located on a machine remote from the BB server. For example, the BB server can be placed on an Intranet/Internet server and SA can be running on another machine that communicates with the BB server via the local area network or the Internet. Indeed, the IPM itself could be placed on a third machine if necessary. This allows a distributed system architecture to be used if deemed necessary, thus spreading processing burdens across several machines. Theoretically, if such a distributed approach is used the degree of fault tolerance and robustness with respect to problems that might occur with SA can be improved as the BB server can keep operating whilst SA is unavailable.

2.3.1 User Agent

The purpose of UA is to facilitate the acquisition of new fault data from service technicians. This is achieved by UA providing a user interface for the entry of new fault examples where the user partakes in a “virtual form-filling” session. The functionality of UA derives from the business process for the collection of fault reports analysed previously.

Task Analysis

The majority of the tasks involved in UA collecting new fault data from the user (Figure 2.8) are very similar to those in the process illustrated in (Figure 2.4). There are, however, additional actions taken by UA that are necessary to ensure the complete collection of data from the user. In particular, UA determines when all attributes have been instantiated.

The user will activate UA, which initially will obtain attribute and legal value information from the BB server. Once this has been done UA will need to know whether the end user is specifying a new attribute and fault as opposed to just a new fault scenario. If a new attribute and fault scenario are being specified then additional information such as part (or attribute) name, current condition and other known part condition values must be entered.

The fault class and attribute values are then collected from the end-user (i.e. the service technician). Because it is necessary for all attributes to have a value, the interface needs to be designed in such a manner that all attributes used are assigned a value (even it is N/A for not applicable/available). UA gathers the information until all attributes have been instantiated. Once this has been done, the information regarding the fault is passed to the BB server and then the connection to the BB server closed.

Architecture

As a result of the highly procedural nature of the process shown in Figure 2.8), the agent is essentially rule-driven. For example, the process can be specified in terms of such statements as “IF new fault scenario THEN obtain only attribute values” and “IF attribute instantiated THEN proceed to next attribute”. However, instead of adopting an explicit rule-based representation where UA uses a rule base to express the functions that it will perform, the rules are actually embodied in the code for UA. As such the operation of UA is essentially hardwired into the code of UA.

Whilst this is perhaps not the most elegant or flexible solution, there are several factors which necessitate it. This is very much due to a need to have a trade-off between the complexity of the operation and implementation and the size of the UA Java applet. The latter issue is particularly important as the potential range of network connection speeds has to be taken into consideration.

It is highly feasible that the link between the site where a service technician is operating and the manufacturer may be via an Internet service provider and hence may employ the standard telecommunications network. Therefore, modems with operating speeds no higher than 56 kbps (kilobits per second) will probably be used. This means that ideally the applet should be engineered to be small, resulting in a minimal amount of transmission taking place to reduce any delays caused by the speed of the Internet link. It is therefore highly advantageous that the agent does not have to load any more information than is absolutely necessary. The size of the applet is sufficiently lightweight at only 7 kB, or 57,344 bits, therefore achieving download times that could be considered to be acceptable.

The nature of UA and its implementation mean that its architecture does not necessarily prescribe to one of the many different agent architecture types given in earlier in the chapter. However, it could be considered to be of a reactive nature as no planning can be said to occur and thus it could perhaps reasonably be described as a procedural agent. Despite the fact that UA performs what can be considered very elementary tasks, it does exhibit some characteristics of an intelligent or autonomous agent.

Operation

UA is downloaded as a Java applet embedded within an HTML web page residing on a web server. Once this applet has been downloaded and initialised, it opens a connection and registers itself with the BB server. Upon the completion of the registration process, UA automatically receives a custom Java object of type “Legal-Values” that contains a list of the current attributes and their legal values. The user interface of UA, shown in (Figures 2.9 and 2.10), is then displayed within the Internet browser.

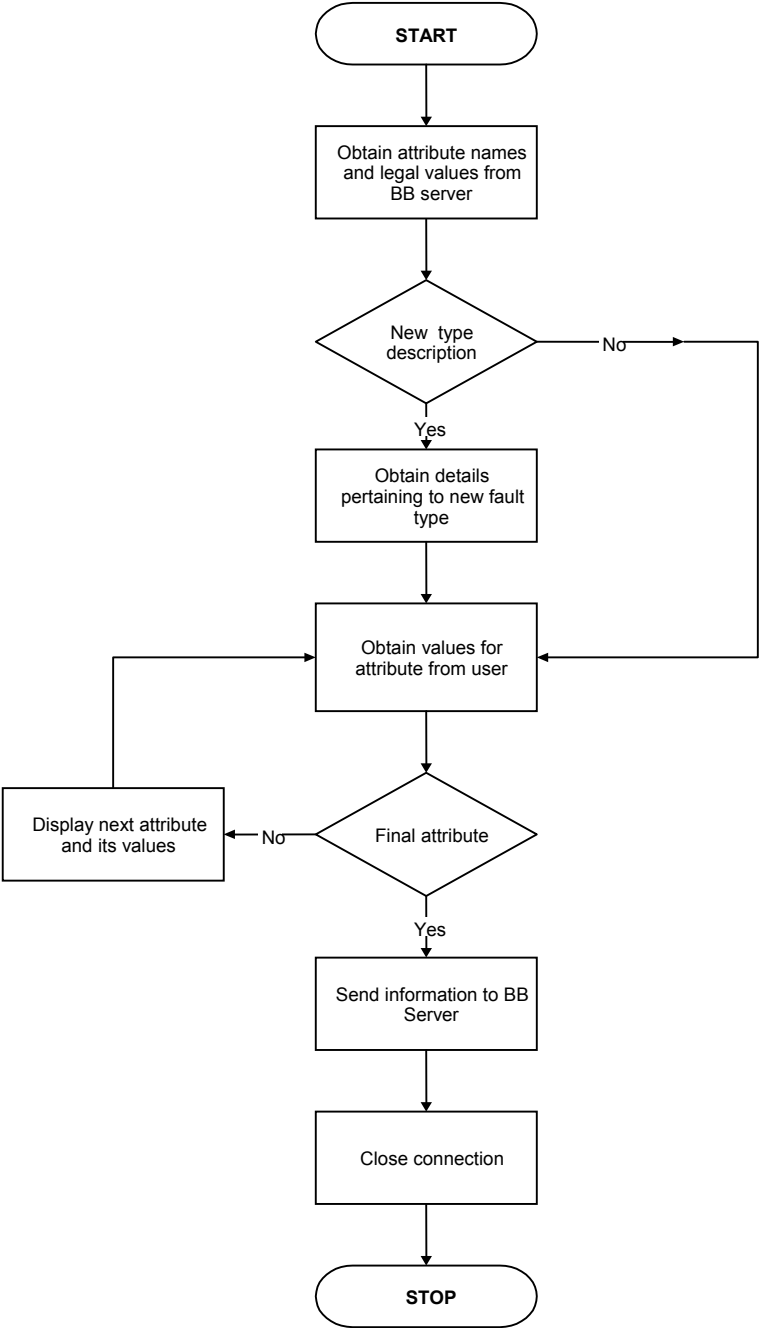


Fig. 2.8 Task analysis for UA

The user interface, consists of essentially three panels:

- *Options (left-hand side)*: this panel allows users to select whether they wish to enter a description of a new fault that contains new attributes (Figure 2.9) or to input a new fault scenario, which is a new description of an existing fault type (Figure 2.10). This is achieved by activating the appropriate button on the panel, either the Fault + Attrib or New Scenario button respectively.
- *New Attribute Details (middle)*: this panel allows the user to define a new attribute and its values if required. It would be employed when a fault involves a part or attribute that is not included in the current knowledge base. Otherwise this panel is not visible as can be noted in (Figure 2.10).
- *Fault Details (right-hand side)*: this panel provides a standardised means for entering the details of a new fault. Initially, UA asks for the fault class. It then proceeds to request all of the part or attribute conditions. The end-user can either choose these from the list shown or input a new value in the text-entry area below the list of values.

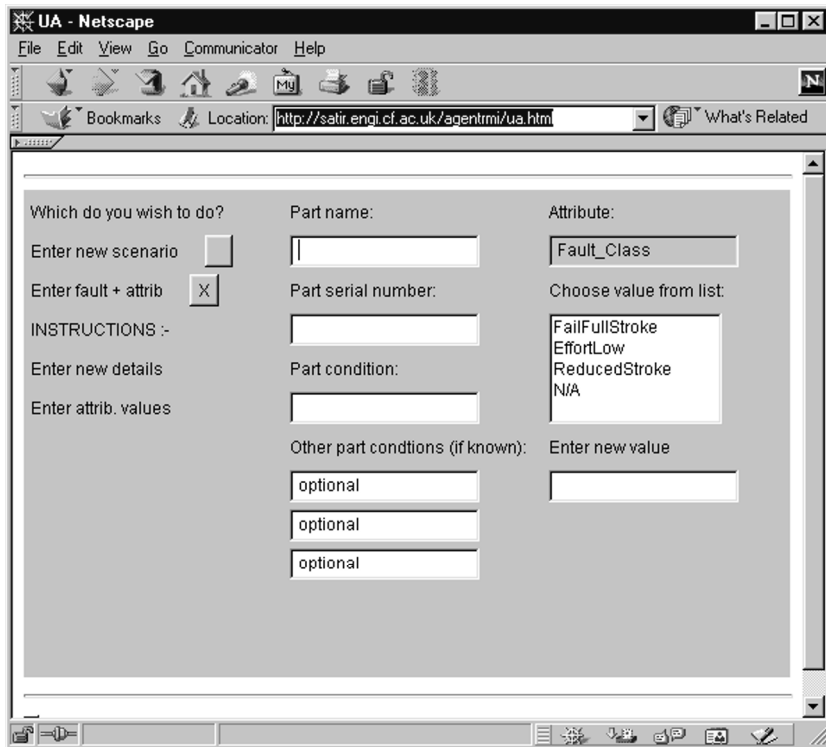


Fig. 2.9 Initial screen/entering fault containing new attribute

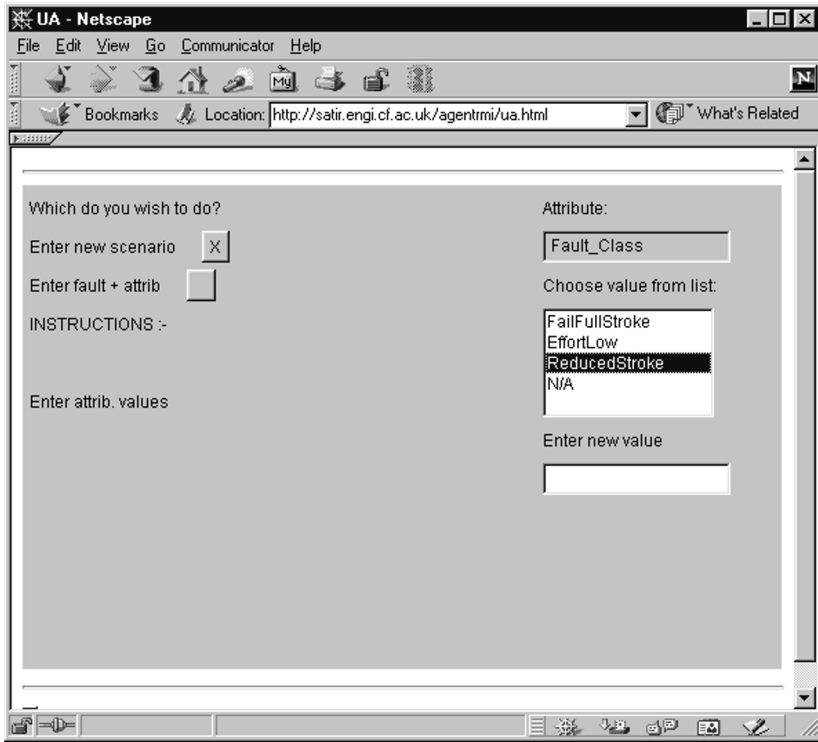


Fig. 2.10 Entering new scenario screen

As can be seen from both (Figures 2.9 and 2.10), the user interface of UA does not contain the usual OK button. This omission may make the interface less intuitive than it could be, but its purpose is to help ensure that values are specified for all attributes. With such an arrangement, the user would have to either double-click on an item in the list or enter a value manually and then press the “return” button before the value would be accepted. This forces him to instantiate all of the attributes used.

From this description of the user interface and its operation, it is apparent that UA has two modes of data entry available:

- *Description of new fault and attribute:* this involves using the middle panel to specify the details of a new attribute (system, sub-system, part, etc.) that is to be included within the knowledge base. The right-hand panel is employed to enter or select the fault class and conditions of attributes within the system. This process of entry is illustrated in Figure 2.11. In the first screen, the end-user provides information relating to the new attribute (part) that is to be entered into the knowledge base by specifying its name, part number and conditions. Next, the fault class is entered. This can pertain to either a new fault or an

existing fault. The end-user then proceeds to specify the values of the other attributes. These can be either selected or manually entered. Once the final attribute has been instantiated, the left-hand panel is not needed and thus no longer appears on the user interface.

- *Entering a new fault scenario*: this procedure is similar to that adopted for new faults and attributes. The primary exception is that no new attributes (parts) are specified and only values for the fault and attribute are selected or entered. This process is illustrated in Figure 2.12. Initially, the fault class is selected or entered, followed by the attribute values that describe this fault. As before the attribute values can either be chosen from a list or manually specified. At the end of the fault-reporting session, the left-hand panel of the user interface is removed.

After the service technician has completed the “virtual form-filling” process UA returns a custom Java object of type “Fault” to the BB server. Here, the fault report awaits collection by SA so that it can be processed. The network links between UA and the BB server are then disconnected.

2.3.2 Server Agent

The purpose of SA is to manage the generation and modification of the knowledge bases within an IPM. The new fault details are passed to SA from UA. SA then processes the fault data and, if appropriate, generates a new knowledge base using an inductive learning algorithm.

Task Analysis

The task analysis produced for SA is understandably more involved than that for UA, due to the greater complexity of the relevant business process (Figure 2.6). However, as mentioned previously, for this task analysis there are potentially two differing approaches that can be followed. A conventional learning algorithm such as RULES-4 (or ISLE but without its ability to handle unlabelled instances) combined with the IFCLEAR algorithm detailed in Pham and Soroka (2006) could be used. Alternatively, ISLE or RULES-IS (Pham and Soroka, 2005ab, 2006) with all its features could be employed. Despite this, the basic processes involved will be nearly identical for both approaches. Therefore, only one task analysis, that for the pure ISLE-based approach, is presented with the non-common areas highlighted.

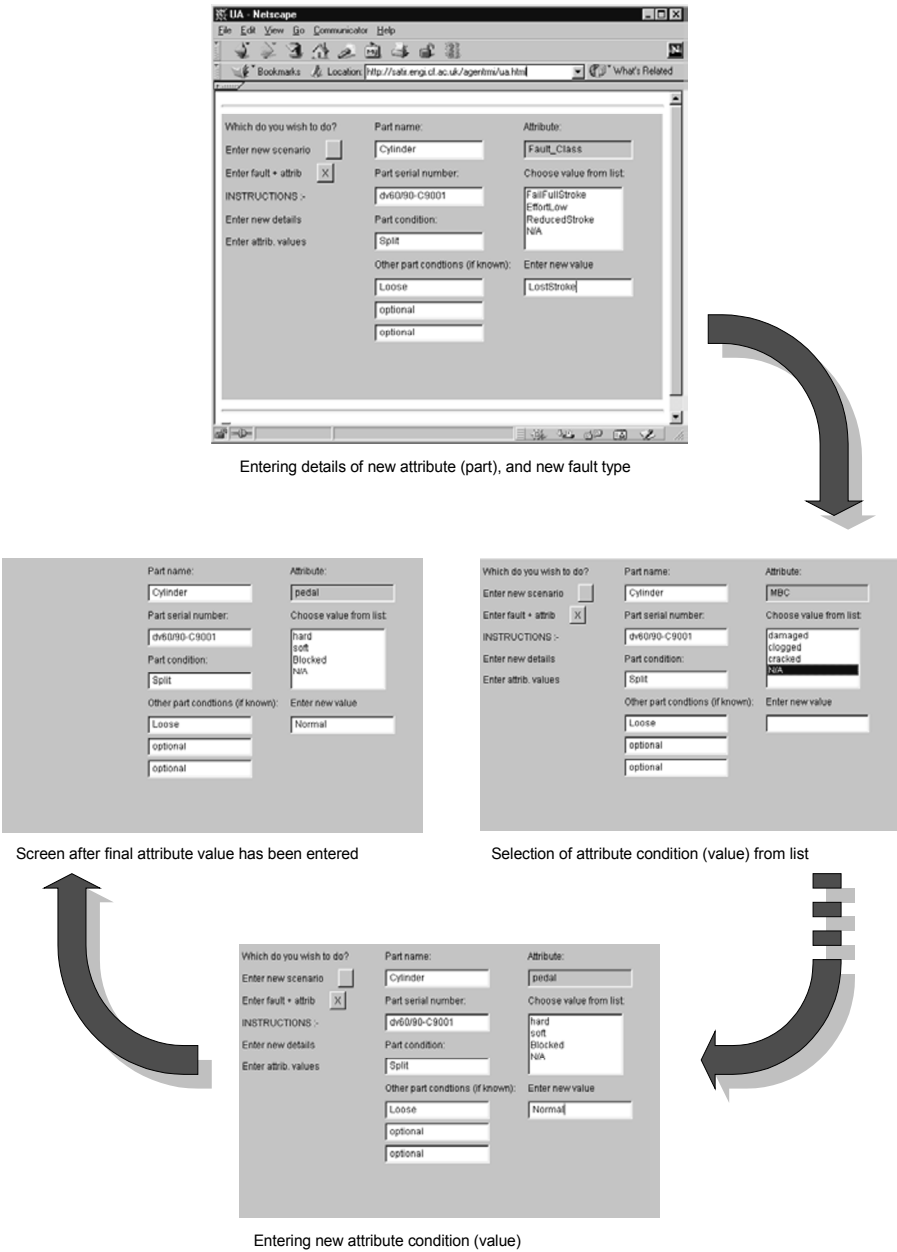


Fig. 2.11 Entering a fault description that contains a new attribute

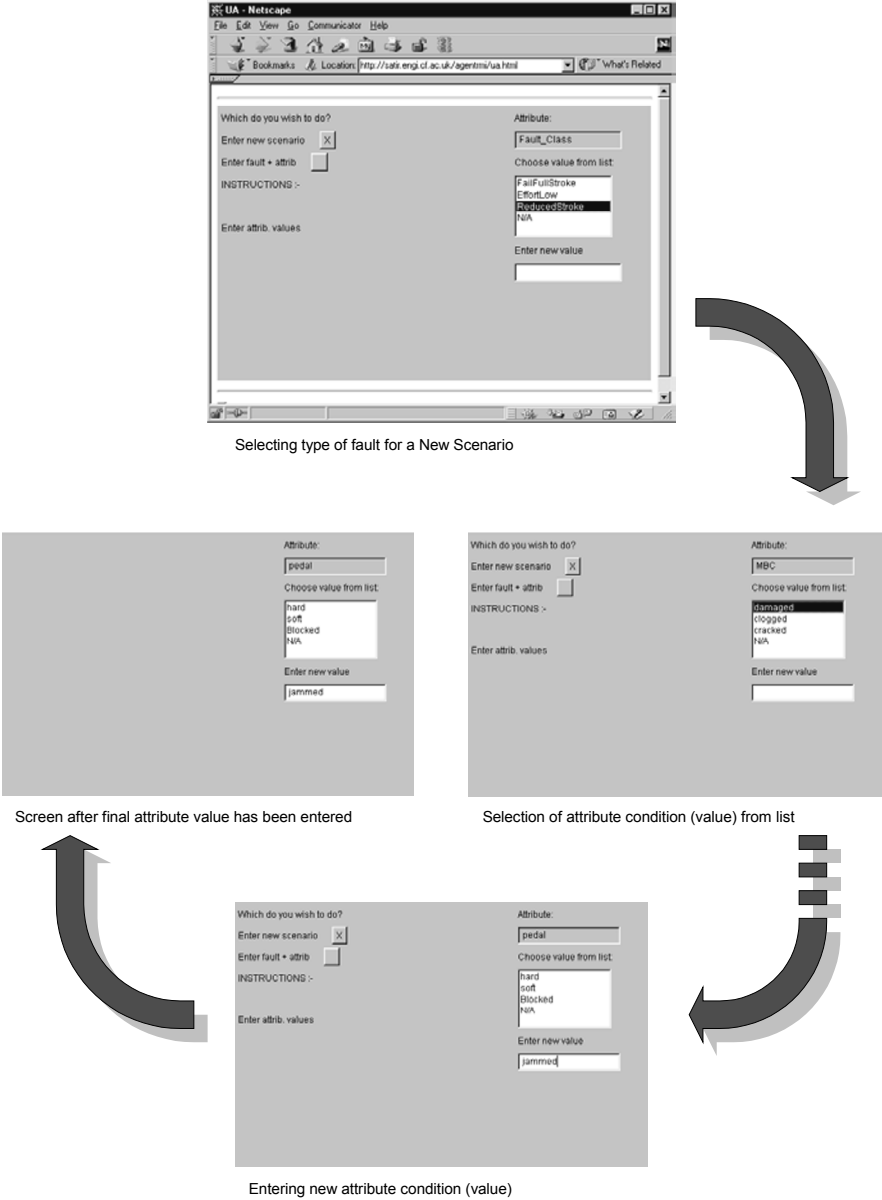


Fig. 2.12 Process for entering a new fault scenario

The task analysis illustrated in Figure 2.13 shows that the first procedure to be performed is the initialisation of the agent when various parameters are set. The agent would then go into a wait state where it looks out for messages to arrive at the BB server. Once a message has been received, the agent checks its origin. If

the message is from another SA then it is considered to be a request to find a fault class. Therefore, SA will query the instance history using the IFCLEAR algorithm. If the message is from UA, then a check is performed to ascertain whether the same fault description is already contained within the instance history. If so, then the knowledge base is regenerated using the entire instance history as the reported fault has not been recognised by the expert system. This is because with some incremental algorithms the proportion of the instances in the instance history covered by the rule set might decrease as new instances are presented. In experiments, this has shown to be an issue with the RULES-4 algorithm when various user-defined parameters, such as the size of the short-term memory, were set inappropriately. This has led to several training instances subsequently failing to be covered by the generated rule set.

What happens next depends upon whether ISLE is being used. If it is not, a check is performed to see if the fault class has been provided. In case the class has not been specified, another agent or an engineer is asked to supply it.

If the fault class has been provided or ISLE is being used, then the format and training data files need to be generated. This depends upon the fault data. When there are new attributes, faults, or attribute values, then the format file for the algorithm needs to be regenerated, and the legal value file is also updated to take these changes into consideration. Once this has been done, a rule generation algorithm would be activated and a new rule set generated. The algorithm could be either RULES-4 or ISLE, or with appropriate modifications an algorithm such as ID3 could be used. The old knowledge base is deleted and replaced by the newly generated one. A blank HTML template file is created if necessary, as for example when new fault types have been added to the knowledge base. The agent then returns to the wait-state awaiting any new messages.

Architecture

SA is more like a conventional agent than UA, in that it is required to perform more complex tasks, such as reasoning about and processing the fault data and also sending and handling requests. This results in the need to pay greater attention to architectural details.

Architectural Requirements

Of the various architectural approaches available, a semi-reactive or partially hybrid approach was chosen. Several factors influenced this decision. Deliberative architectures are typically based upon BDI models of the agent and its environment that directly influence the decision-making processes of the agent. These models contain *inter alia* an expression of the beliefs of the agent with respect to the environment.

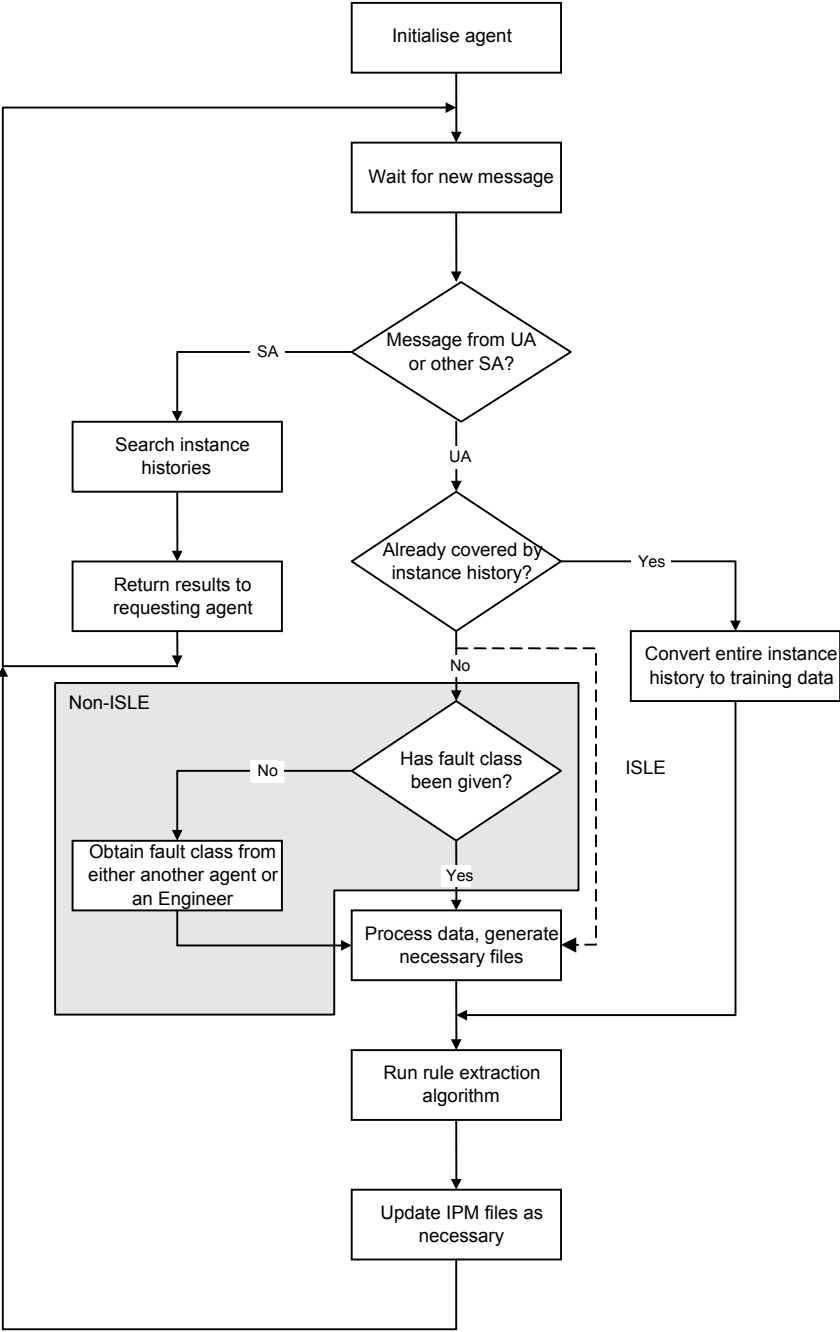


Fig. 2.13 Task analysis for server agent

In this type of application, such models may be inappropriate. Within the proposed system, there is a large and potentially continually expanding population of UAs for which models might have to be maintained. To do so, symbolic manipulation would have to be performed. The interactions with the group of UAs as a whole will be frequent. However, for a single UA, it is probable that interaction would be relatively infrequent. This would raise issues regarding how to maintain accurate models of agents with which there is minimal interaction. As a deliberative architecture is likely to decide what actions to take based upon some model, an empirically constructed model may not be suitable for this application. For example, if, on the few occasions it has interacted with SA, a given UA has failed to give a fault class, this could potentially lead SA to assume, incorrectly, that this would hold true for the next interaction.

There are also situations where no *a priori* knowledge exists of what will occur. For example, the result of a help request to another SA cannot be anticipated. Therefore, there would be no real value in the agent attempting to create plans depending on possible outcomes of processes that have yet to occur. It is more logical for SA to base its decision-making mainly upon the state of the environment at the current instant. The agent should take decisions according to the results of its own previous actions or those of others. This leads to the conclusion that a reactive architecture might be the most appropriate, as reactive agents can respond directly to stimuli.

Despite the assertion that a reactive architecture appears to be the most suitable, there is still a need to maintain some form of database of other SAs. Contained within this database are the lists of other agents within the system and the scores relating to the similarity of the instance histories maintained. This ensures that an SA knows what assistance it may be able to obtain from other SAs. This database is not a model due to its being fairly static and not including any representation of how useful an agent has been in the past.

Specification of Architecture

There are different ways in which such a semi-reactive architecture could be implemented. It could be modelled on a rule-based system or a neural network. However, it was decided to investigate the application of the concepts of FSA as the basis for the architecture. An FSA is a model of a dynamic system that switches from one state to another depending on the current state and the result of the actions taken in that state.

The architecture presented here can be considered to have two main sources of inspiration and influence, namely situated automata (Kaelbling, 1991; Rosenschein and Kaelbling, 1995) and subsumption and behaviour-based architectures (Brooks, 1986, 1991; Maes, 1989).

Work has been conducted relating to situated automata in the domain of robotics where such systems are expressed as fixed sequential circuits (Kaelbling, 1991; Rosenschein and Kaelbling, 1995). A situated-automata-based system uses a synchronous approach, much in the way a digital circuit operates. This approach is

concerned with the agent and its sensing and effecting and the formal logical specification of actions in an environment to perform a task (Kaelbling, 1991; Rosenschein and Kaelbling, 1995). This approach is not the most appropriate for the implementation of SA. Instead, the proposed FSA architecture is more concerned with the overall performance of a task and with the two environments that exist, the internal environment of the agent and the external world within which it operates.

The architectural approach presented here has some similarities with the subsumption and behaviour-based architectures in Brooks (1986, 1991) and also with the agent network architecture (Maes, 1989). Both of these architectures consist of modules that have certain behaviours prescribed to them. The modules then compete against one another to be activated and allowed to perform their action. The states within the FSA could be considered to be analogous to these various modules although they do operate on a much more basic level.

The FSA-based reactive agent architecture can in fact be considered analogous to a rule-based system. This is because theoretically the structure of the FSA could be represented as a series of rules. However, the use of an FSA simplifies the implementation in that it removes the requirement for a rule base. This is because a simple state table can be used to represent the FSA. Therefore there is no requirement for any inference engine. Instead the system queries the state table and activates the appropriate process. The use of a state table offers a benefit in that the table can be readily altered to allow the functionality and behaviour of the agent to be modified as required. The state table of the FSA depicted in Figure 2.14 is given in Table 2.1. The table was directly obtained from the FSA.

To illustrate the ease of changing the behaviour of SA, consider a hypothetical scenario where a report it just received is already covered in the instance history (see Figure 2.14). Instead of switching to generating a new knowledge base, suppose SA is required to change to the “check fault class” state. Only one entry in the state table need be altered, i.e. entry 6 for which the new state would now be “check fault class” instead of “generate KB”. If required, this table could be altered dynamically, much in the way that links in the agent network in Maes (1991) can be changed.

In summary, a FSA lends itself to being used to implement SA in that it provides a system with reactive features while operating within a strictly bounded framework. It can also exhibit a complex functionality arising from states that perform simple tasks.

The architecture shown in Figure 2.14 illustrates the FSA at a high level of abstraction. As such, it contains several multiple process states. Within these are what could be considered as sub-FSAs that together constitute the entire FSA.

The concise representation of Figure 2.14 hides a great deal of the complexity of the system. By comparing the task analysis in Figure 2.13 and the FSA in Figure 2.14, it can be seen that the task and FSA are closely related. If the task analysis and full FSA were to be compared, it would become apparent that the FSA breaks down the sections of the task analysis into much smaller pieces. These separate states can be considered to represent very simple modules or elements.

However, through combining these simple elements a complex information-processing architecture can be obtained, achieving an outcome that is similar in parts to the concepts for reactive architectures outlined in Maes (1991).

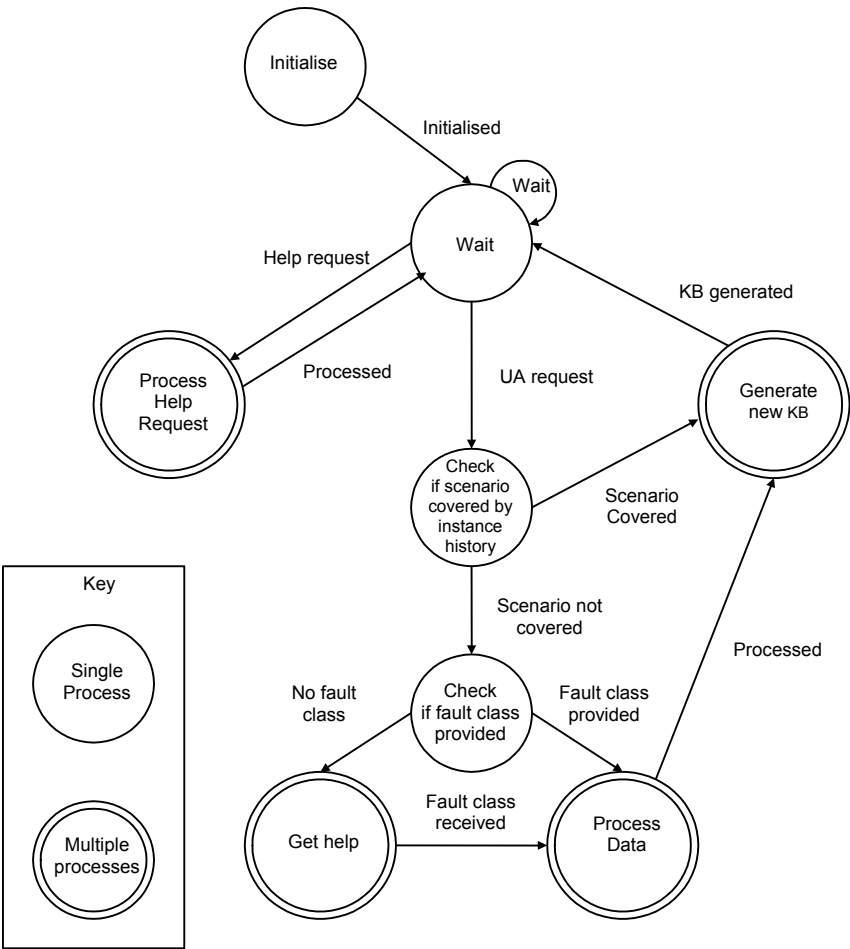


Fig. 2.14 FSA representing architecture of SA (KB = knowledge base)

Table 2.1 State table for FSA

	Current state	Result	New state		Current	Result	New
1	Initialise	Initialised	Wait		1	1	2
2	Wait	Wait	Wait		2	2	2
3	Wait	Help request	Process help	⇒	2	3	7
4	Wait	UA request	Check scenario		2	4	4
5	Process help	Processed	Wait		3	5	2

Table 2.1 (continued)

	Current state	Result	New state	Current	Result	New
6	Check scenario	Covered	Generate KB	4	6	8
7	Check scenario	Not covered	Check fault class	4	7	5
8	Check fault class	No fault class	Get help \Rightarrow	5	8	6
9	Check fault class	Fault class	Process data	5	9	7
10	Get help	Received	Process data	6	10	7
11	Process data	Processed	Generate KB	7	11	8
12	Generate KB	Generated	Wait	8	12	2

Operation

For a server-based system to operate, several software applications must be running at all times. First, Java's RMI registry must continually perform basic housekeeping tasks such as TCP/IP socket management for the Java virtual machines, which are the environments within which the applications run. Second, the BB server must be functioning so that UA is able to communicate with SA. Finally, SA must be running as it performs the various information processing tasks.

The user interface of SA is minimal as the user only interacts with SA, when a fault class has to be provided. Therefore the only output provided by SA is related to what task it is currently performing, so that the engineer is aware of its state. Figure 2.15 does not show the process of setting up the state table. To do this, the table is loaded into a custom Java object called *stateTable*. This object then stores the state table and provides methods for the SA to examine it as required.

The following list describes the purpose of and tasks performed by each normal and multiprocess state depicted within the FSA.

- *Initialise*: the agent is initialised. The list of legal fault types, attributes and values is loaded and other basic housekeeping tasks are performed.
- *Wait*: SA will wait in this state until a fault report is received by the BB server from a UA or until a help request is intercepted from another agent. A few other minor tasks are performed in this state. For example, if it is not already empty, the training data file will be cleared. Also if deemed appropriate the new fault can be checked by an engineer to ensure it is a genuine fault.
- *Process help request*: this is a multiple process state. SA will check the instance history using IFCLEAR and then, depending upon the outcome, it will either post a "can't help" message to the requesting agent or extract and send the fault class to the latter.
- *Check scenario*: SA checks if the new fault report is already contained inside the instance history maintained within it.
- *Check fault class*: SA will check whether the fault class for the new fault report has been specified. Note that if a pure ISLE-based approach is used, this step would immediately return the result that there is a fault class.

- *Get help* (a multiple process state): SA will check if there are any agents that can assist it. It does this by accessing its list of agents. If there are suitable agents then a help request will be sent and subsequently the reply will be processed. If there are no agents that can help or no other agents exist, then assistance is sought from the user.
- *Process data* (a multiple process state): the new fault report is processed and, if necessary, the format, legal value and instance history files are updated. If new attribute values or fault types have been added, then the legal value and format files are updated to take them into account. When a report contains a new attribute and its associated values, then the current instance history is modified to include this new attribute. The value N/A (not available) is given for that attribute for all of the existing instances. The instance history in that situation will be copied into the training data file. The format and legal value files will also be modified accordingly.
- *Generate new KB* (a multiple process state): the new fault report is appended to the instance history and the training data file. In situations where a new fault is same as one in the instance history (as mentioned previously) the contents of the instance history are copied into the training data file first. The rule extraction algorithm is executed and the IPM is updated as necessary. If a new fault has been specified then a template HTML file is generated so that the appropriate repair procedure can be inserted within it, and any files that require updating are duly modified.

2.3.3 Testing

The agent-based system proposed in this chapter consists of several distributed agents and applications that are interdependent. To assess the operation of the system, different issues have to be examined concerning:

- the correct transfer of data from UA to SA;
- the appropriate updating of files in SA;
- the correctness of the files; and
- the accuracy of the rule set generated.

An example is shown in Figure 2.15 where a new description for a known fault is given.

Which do you wish to do?
Enter new scenario ☐
Enter fault + attrib ☒
INSTRUCTIONS -
Enter attrib. values

Attribute:

Choose value from list
Fault1
Fault2
Fault3
Enter new value

Attribute:

Choose value from list
Attrib1Value1
Attrib1Value2
N/A
Enter new value

Attribute:

Choose value from list
Attrib2Value1
Attrib2Value2
Attrib2Value3
N/A
Enter new value

Attribute:

Choose value from list
Attrib3Value1
N/A
Enter new value

a) Entry of Fault Details

Fault details sent to SA via BB Server

```
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>
C:\InetPub\wwwroot\AgentRMI>java serverApplication
Connecting to BBServer
waitForInput
serverApp> classis Fault1
serverApp> Attrib1 Attrib1Value1
serverApp> Attrib2 Attrib2Value1
serverApp> Attrib3 Attrib3Value1
class Given
No New Attrib
Attrib1Value1=Attrib1Value1:
Attrib2Value1=Attrib2Value1:
Attrib3Value1=Attrib3Value1:
Run Rules
Update
^C
C:\InetPub\wwwroot\AgentRMI>
```

b) SA Processing Fault Details

Files for rule generation updated/created

classis Fault1 Fault2 Fault3
Attrib1 Attrib1Value1 Attrib1Value2 N/A
Attrib2 Attrib2Value1 Attrib2Value2 Attrib2Value3 N/A
Attrib3 Attrib3Value1 N/A

Legal.txt

Fault1 Attrib1Value1 Attrib2Value1 Attrib3Value1

Training.txt

classis Attrib1 Attrib2 Attrib3
Fault1 Fault2 Fault3
Attrib1Value1 Attrib1Value2 N/A
Attrib2Value1 Attrib2Value2 Attrib2Value3 N/A
Attrib3Value1 N/A

Format.txt

Fault1 Attrib1Value1 Attrib2Value2 Attrib3Value1
Fault2 Attrib1Value2 Attrib2Value2 Attrib3Value1
Fault3 Attrib1Value2 Attrib2Value3 Attrib3Value1
Fault1 Attrib1Value1 Attrib2Value1 Attrib3Value1

History.txt

IF Attrib1=Attrib1Value1 THEN classis=Fault1
IF Attrib2=Attrib2Value3 THEN classis=Fault3
IF Attrib1=Attrib1Value2 AND IF Attrib2=Attrib2Value2 THEN classis=Fault1

rules.txt

c) Generated Files

Fig. 2.15 Illustrated example of entry and processing of new fault details with generated files

To provide an overview of how the agents within the system operate the stages of fault entry and processing are depicted. Initially, the new fault report is entered via the user interface of UA. Once this has been completed, the information is

forwarded to the BB server, which then passes it to SA for processing. SA then generates the various files illustrated in Figure 2.15.

Testing showed that the system was able to cope with a variety of situations regarding data. Including scenarios where: new description of an existing fault is provided that also contains a new value for an existing attribute; a new type of fault is entered that contains existing attributes and utilises the existing values for those attributes; a fault occurs that requires both a new fault type and new attributes; an existing fault is given a new attribute and new attribute values.

2.4 Conclusions

The system described here has shown itself to be capable of accepting various types of fault reports and of processing these reports so that new knowledge bases can be generated. The system has also demonstrated a degree of robustness in being able to handle asynchronously submitted reports. This is due to its distributed nature, in particular, the use of the BB server to act as a depot for messages. This shows that agent-based systems together with Internet-based technologies can prove to be a vital tool for an enterprise when dealing with volumes of information that require processing.

References

- Agre PE and Chapman D (1987) Pengi: an implementation of a theory of activity. In: Proceedings of the 6th national conference on artificial intelligence AAAI'87 Seattle, WA, USA, 1:268–272
- Aksoy MS (1993) New algorithms for machine learning. PhD. thesis, Cardiff University, UK
- Al-Attar A (1991) Rule induction from mythology to methodology. Research and developments in expert systems VIII, London, UK, 85–103
- Alty JL (1997) Agents and intelligent user interfaces. In: Proceedings of UNICOM tutorial on agents and intelligent user interfaces, Heathrow, London, UK
- Alty JL, Griffiths D, Jennings NR et al. (1994) ADEPT-advanced decision environment for process tasks: overview and architecture. In: Proceedings of the 14th annual BCS technical conference on expert systems, 359–371, Cambridge, UK
- Andreoli JM, Pacull F, Pareschi R (1997) XPECT: a framework for electronic commerce. IEEE Internet Comput., 1(4):40–48
- Axelrod R (1984) The evolution of cooperation. New York: Basic Books.
- Bond AH, Gasser L (1988) An analysis of problems and research in DAI. In: Bond AH and Gasser L (eds) Readings in distributed artificial intelligence, Morgan Kaufmann, San Mateo
- Bratman ME, Israel DJ, Pollack ME (1988) Plans and resource-bounded practical reasoning. Comput. Intell., 4:349–355
- Brooks R (1986) A robust layered control system for a mobile robot. IEEE Trans. Rob. Auto., 2:14–23
- Brooks R (1991) Integrated systems based on behaviours. ACM SIGART Bulletin, 2(4):46–49
- Chapman D (1987) Planning for conjunctive goals. Artif. Intell., 32:333–378

- Cheng J, Fayyad UM, Irani KB et al. (1988) Improved decision trees: a generalised version of ID3. In: *Proceedings of the 5th international conference on machine learning*, Ann Arbor, USA, 100–106
- Clark P, Boswell R (1991) Rule induction with CN2: some recent improvement. In: *Proceedings of the 5th European conference on machine learning EWSL-91*, 151–163
- Clark P, Niblett T (1989) The CN2 inductive algorithm. *Machine Learning*, 3:261–283
- Craig I (1995) *Blackboard systems*. Ablex Publishing Corporation, Norwood
- Davies NJ, Weeks R, Revett MC (1997) Information agents for the world wide web. In: Nwana HS, Azarmi (eds) *Software agents and soft computing: concepts and applications*, Springer, Berlin, 81–89
- Dreper BA, Collins RT, Brolio J et al. (1988) Issues in the development of a blackboard-based schema for image understanding. In Englemore RS, Morgan AJ (eds) *Blackboard systems*, Addison-Wesley, Wokingham, 189–218
- Ekdahl B, Astor E, Davidsson P (1995) Toward anticipatory agents. In: Wooldridge M, Jennings NR (eds), *Lect. Notes in Artif. Intell.*, 890:191–202
- Englemore RS, Morgan AJ (1988) *Blackboard systems*. Addison-Wesley, Wokingham
- Ermann LD, Hayes-Roth F, Lesser VR et al. (1988) The hearsay II speech understanding system. In: Englemore RS, Morgan AJ (eds) *Blackboard systems*. Addison-Wesley, Wokingham, 31–86
- Etzioni O, Weld D (1994) A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76
- Ferguson IA (1995) Integrated control and coordinated behaviour a case for agent models. In : Wooldridge M, Jennings NR (eds) *Lect. Notes in Artif. Intell.*, 890:203–218
- FIPA (1997) Foundation for intelligent physical agents. Agent communication language specification. [<http://www.fipa.org/spec/f8a22.zip>] FIPA Secretariat c/o S.I.A.–C.P. 3176, Strada Antica di Collegno 253, I-10146, Torino, Italy
- Fisher M (1995) Representing and executing agent-based systems. In: Wooldridge M, Jennings NR. (eds) *Lect. Notes in Artif. Intell.*, 890:203–218
- Foner L, Crabtree IB (1996) Multiagent matchmaking. *B.T. Technol. J.* 14(4):115–123
- Forsyth R. (1989) *Machine Learning: Principles and Techniques*. Chapman and Hall, London
- Franklin S, Graesser L (1996) Is it an agent, or just a program?: A taxonomy for autonomous agents. In: Muller JP, Wooldridge MJ, Jennings NR, (eds) *Lect. Notes in Artif. Intell.*, 1193: 21–35
- Franklin S, Graesser A, Olde B et al. (1996) Virtual mattie—an intelligent clerical agent. [<http://www.msci.memphis.edu/~franklin/Vmattie.html>] Department of Mathematical Sciences, the University of Memphis, Memphis, Tennessee 38152, USA
- Genesereth MR, Nilsson NJ (1988) *Logical foundations of artificial intelligence*. Morgan Kaufmann, San Mateo
- Gensereth MR, Fikes RE (1992) *Knowledge interchange format, Version 3.0 Reference Manual*. Stanford University Report Logic-92-1, Stanford University, Stanford, CA 94305-9025, USA
- Genesereth MR, Ketchpel SP (1994) Software agents. *Communications of the ACM* 37(7):48–53
- Georgeff M (1988) Communication and interaction in multiagent planning. In: Bond AH, Gasser L (eds) *Readings in distributed artificial intelligence*, Morgan Kaufmann, San Mateo, 200–204
- Georgeff MP, Lansky AL (1987) Reactive reasoning and planning. In: *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, pp. 677–682, Seattle, WA, USA
- Gibbins PF, Kearney P, Sehmi A et al. (1994) Intelligent agents in multimedia systems. *Sharp Tech. J.*, 60:11–14
- Gray R, Kotz D, Nog S et al. (1996) Mobile agents for mobile computing. Technical Report PCS-TR96-285, Dartmouth College [<ftp.cs.dartmouth.edu/TR/TR96-285.ps.Z>], Department of Computer Science, Dartmouth College, 6211 Sudikoff Laboratory, Hanover, NH 03755-3510, USA
- Hancox PJ, Mills WJ, Reid BJ (1990) Keyguide to information sources in artificial intelligence/expert. In: Lawrence KS: *Ergosyst*

- Hayes-Roth B (1993) An architecture for adaptive aystems. Stanford University report number STAN-CS-TR-93-1496 (also available as KSL 93-19), Computer Science Dept, 353 Serra Mall, Stanford University, Stanford CA 94305-9025, USA
- Hayes-Roth B, Gaba D, Uckun S et al. (1994) Guardian: a prototype intelligent agent for intensive care monitoring. In: Proceedings of the 12th national conference on artificial intelligence, 2:1503
- Hewitt C (1977) Viewing control structures as patterns of passing messages. *Artif., Intell.* 8(3):323-364
- Holt J, Rodd MG (1994) An architecture for real-time distributed artificial intelligent systems. . *Real-Time Systems*, 6:263-288
- Huang J, Jennings NR, Fox J (1995) An architecture for distributed medical care. In: Wooldridge M, Jennings N.R., (eds) *Lect. Notes in Artif. Intell.*, 890:219-232
- Huhns M.N. (1987) (ed) *Distributed artificial intelligence*. Pitman, London
- Huhns MN, Jacobs N, Ksiezyk T et al. (1992) Integrating enterprise information models in carnot. In: Proceedings of the international conference on intelligent and cooperative information systems, Rotterdam, Netherlands, 32-42
- Hunt EB, Marin J, Stone PJ (1966) *Experiments in induction*. Academic press, New York, USA
- Ito A, Yano H (1995) Can machines ever learn to cooperate? The emergence of cooperation in a society of autonomous agents. *J. Comm. Res. Lab.*, 42(3):243-249
- Jennings NR (1992) Using GRATE to build cooperating agents for industrial control. IFAC symposium on artificial intelligence in real-time control, Delft, Netherlands, 1-6
- Jennings NR (1995) Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artif. Intell.*, 75:195-240
- Jennings NR, Faratin P, Norman TJ et al. (1996) ADEPT: Managing business processes using intelligent agents. In: Proceedings BCS expert systems 96 conference, Cambridge, 5-23
- Jennings NR, Mamdani EH, Laresgoiti I et al. (1992) GRATE: a general framework for co-operative problem solving. *IEE-BCS J. Intell. Sys. Eng.*, 1(2):102-114
- Jennings NR, Varga LZ, Aarnts RP et al. (1993) Transforming standalone expert systems into a community of co-operating agents. *Eng. App. Artif. Intell.*, 6(4):317-331
- Jennings NR, Wooldridge M (1995) Applying agent technology. *App. Artif. Intell.*, 9(4):357-369
- Jennings NR, Wooldridge M (1998) Application of intelligent agents. In: Jennings NR, Wooldridge M (eds) *Agent technology: foundations, applications and markets*, Springer-Verlag, Berlin, 3-28
- Kaelbling LP (1991) A situated-automata approach to the design of embedded agents. *ACM SIGART Bulletin*, 2(4):85-88
- Kato K, Someya Y, Matsubara K et al. (1999) An approach to mobile software robots for the www. *IEEE Trans. Know. Data Eng.*, 11(4):51-548
- Kearney P (1996) Personal electronics: personal agents. In: Proceedings 1st IEE colloquium on intelligent agents 7/1-7/6
- Khedro T, Genesereth MR (1994) Progressive negotiation for resolving conflicts among distributed heterogeneous co-operating agents. In: Proceedings of the 12th national conference on artificial intelligence, Seattle, USA, 1:381-386
- Knight K (1993) Are many reactive agents better than a few deliberative ones? In: Proceedings of the 13th international joint conference on artificial intelligence (IJCAI'93), Chambéry, France, 432-437
- Kodratoff Y (1988) *Introduction to machine learning*. Pitman Publishing, London
- Kowalski R, Sadri R (1996) Towards a unified agent architecture that combines rationality with reactivity. In: Proceedings of the international workshop logic in databases LID, San Miniato, 137-149
- Kozierok R, Maes P (1993) A learning agent for scheduling meetings. In: Proceedings ACM-SIGCHI international workshop, Orlando, USA, 81-88
- Kumar D, Shapiro SC (1991) Architecture of an intelligent agent in SNePS. *ACM SIGART Bulletin*, 2(4):89-92

- Labrou Y, Finin T (1997) A proposal for a new KQML specification. The University of Maryland, Baltimore County report TR CS-97-03, The University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD21250, USA
- Lander SE, Lesser VR, Connell ME (1990) Knowledge-based conflict resolution for co-operation among expert agents. In: Proceedings of MIJ-JSME workshop, 253-268
- Lashkari Y, Metral M, Maes P (1994) Collaborative interface agents. In: Proceedings of the 12th national conference on artificial intelligence AAAI'94, Seattle, USA, 1:444-449
- Lieberman H (1995) Letizia: an agent that assists web browsing. In: Proceedings of the 14th international conference on artificial intelligence IJCAI'95, Montreal, Canada, pp. 924-929
- Lieberman H (1997) Autonomous interface agents. In: Proceedings of the ACM conference on computers and human interfaces CHI-97, Atlanta, USA
- Lieberman H (1999) Personal assistants for the web: A MIT perspective. In Klusch M (ed): Intelligent information agents: agent-based information discovery and management on the Internet, Springer, Berlin, Germany, pp. 279-292
- Maes P (1989) The dynamics of action selection. In: Proceedings of the 11th international conference on artificial intelligence IJCAI-89, Detroit, USA, 2:991-997
- Maes P (1991) Designing autonomous agents: theory and practice, from biology to engineering and back. MIT Press, Cambridge, USA
- Maes P (1994) Agents that reduce work and information overload. Communications of the ACM, 37(7):30-40
- Maes P (1995) Modelling adaptive autonomous agents. In: Langton CG (ed) Artificial life, MIT Press, Cambridge, USA
- Mamdani EH, Charlton PM (1996) Agent-based support for public-information kiosks. In: Proceedings of the 1st IEE colloquium on intelligent agents, London, UK, 4/1-4/3
- Mayfield J, Labrou Y, Finin T (1996) Evaluation of KQML as an agent communication language. IJCAI'95 workshop, Montreal, Canada, 347-360
- Meghini C (1997) A reactive logical agent. In: Proceedings of the 1st international workshop on cooperative information agents CIA'97, Kiel, 148-158
- Michalski RS (1983) A theory and methodology of inductive learning. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning—an artificial intelligence approach, Morgan Kaufmann, Los Altos, 83-134
- Michalski RS, Mozetic I, Hong J et al. (1986) The multi-purpose incremental learning algorithm AQ15 and its testing, application in three medical domains. In: Proceedings of the 5th national conference on artificial intelligence, Philadelphia, USA, 1041-1047
- Moffat D, Frijda NH (1995) Where there's a will there's an agent. In: Wooldridge M, Jennings NR (eds) Lect. Notes in Artif. Intell., 890:245-260
- Muller JP (1996) The design of intelligent agents: A layered approach. Lect. Notes in Artif. Intell., 1177
- Muller JP, Pischel M, Thiel M (1995) Modelling reactive behaviour in vertically layered agent architectures. In: Wooldridge M, Jennings NR (eds) Lect. Notes Artif. Intell., 890: 261-276
- Nwana HS, Ndumu DT (1996) An introduction to agent technology. British Telecom Technology J., 14(2):55-67
- O'Brien P, Wiegand M (1997) Agents of change in business process management. In: Nwana HS, Azarmi N (eds) Software agents and soft computing, concepts and applications, Springer-Verlag, Berlin, Germany, pp. 132-245
- Osawa E-I, Tokoro M (1992) Collaborative plan construction for multi-agent mutual planning. In: Werener E, Demazeau Y (eds), Decentralized AI3, Elsevier, Oxford, USA, pp. 169-185
- Papazoglou M, van den Heuvel WJAM (2000) Configurable business objects for building evolving enterprise models and applications. In: van der Aalst WMP, Desel J, Oberweis A (eds), Business process management: models, techniques and empirical studies, Lect. Notes Comp. Sci. N. 1806, pp. 328-344
- Pham DT, Aksoy MS (1993) An algorithm for automatic rule induction. Artif. Intell. Eng., 8:277-282

- Pham DT, Aksoy MS (1995) RULES: a simple rule extraction algorithm. *Expert Sys. App.* 8(1):59–65
- Pham DT, Dimov SS (1997a) An efficient algorithm for automatic knowledge acquisition. *Pattern Recognition* 30(7):1137–1143
- Pham DT, Dimov SS (1997b) An algorithm for incremental inductive learning. In: *Proceedings of the Institution of Mechanical Engineers Part B: J. Eng. Manuf.*, 211:239–249
- Pham DT, Dimov SS, Setchi RM (1998) Intelligent product manuals. In: *Proceedings of the Institution of Mechanical Engineers Part B: J. Eng. Manuf.*, 213:65–76
- Pham DT, Dimov SS, Soroka AJ (2001) Knowledge acquisition for intelligent product manuals. In: *Proceedings of the Institution of Mechanical Engineers Part B: J. Eng. Manuf.*, 215:97–103
- Pham DT, Soroka AJ (2005a) ISLE – A novel immune-system inspired rule extraction algorithm. In: *Proceedings of the 16th IFAC World Congress, Prague*
- Pham DT, Soroka AJ (2005b) RULES_IS immune-network inspired machine learning algorithm. In: *Proceedings of the 1st I*PROMS virtual conference on intelligent production machines and systems*
- Pham DT, Soroka AJ (2006) An algorithm based on the immune system for extracting fault classes from instance histories. In: *Proceedings of the 2nd I*PROMS virtual conference on intelligent production machines and systems*, pp. 313–318
- Quinlan JR (1986) Induction of decision trees. *Machine learning Vol. 1*. Kluwer Academic Publishers, Boston, USA, 81–106
- Quinlan JR (1987) Generating production rules from decision trees. In: *Proceedings of the 10th international joint conference on artificial intelligence, Milan, Italy*, pp. 304–307
- Quinlan JR (1988) Induction knowledge and expert systems. In: Gero JS, Stanton R (eds) *Artificial intelligence developments and applications*, North Holland, Amsterdam, 253–271
- Rao A, Georgeff MP (1995) BDI agent from theory to practise. In: *Proceedings of the 1st international conference on multi-agent systems, San Francisco, USA*, 312–319
- Rosenschein SJ, Kaelbling LP (1995) A situated view of representation and control. *Artif. Intell.* 73(1-2):149–173
- Schlimmer JC, Fisher D (1986) A case study of incremental concept induction. In: *Proceedings of the 5th national conference on artificial intelligence, Philadelphia, USA*, pp. 495–501
- Schroeder M, Mora IA, Pereira LM (1996) A deliberative and reactive diagnosis agent-based on logic programming. In: Muller JP, Wooldridge M, Jennings NR (eds) *Lect. Notes in Artif. Intell.*, 1193:293–307
- Sen S, Sekaran M, Hale J (1994) Learning to co-ordinate without sharing information. In: *Proceedings of the 12th national conference on artificial intelligence, Seattle, USA*, 1:426–431
- Sheth BD (1994) A learning approach to personalized information filtering. MSc Thesis: MIT, media laboratory communications and sponsor relations, 77 Massachusetts Avenue, Cambridge, Massachusetts, USA, pp. 2139–4307
- Singh MP, Huhns MN (1999) Social abstractions for information agents. In: Klusch M (ed) *Intelligent information agents: agent-based information discovery and management on the internet*, Springer, Berlin, Germany, pp. 37–52
- Struthers A (1996) Agent-based developments: a discussion of opportunities and hurdles, intelligent agents and their applications. In: *Proceedings the 1st IEE colloquium on intelligent agents, London, UK*, 6/1–6/3
- Sycara KP (1989) Argumentation: planning other agents' plans. In: *Proceedings of the 11th international joint conference on artificial intelligence, Detroit, USA*, 2:517–523
- Sycara KP (1999) In-context information management through adaptive collaboration of intelligent agents. In: Klusch M (ed) *Intelligent information agents: agent-based information discovery and management on the internet*, Springer, Berlin, Germany, pp. 78–99
- Utgoff PE (1988) ID5: an incremental ID3. In: *Proceedings of the 5th international conference on machine learning, Ann Arbor, Michigan, USA*, pp. 107–120
- Voorhees EM (1994) Software agents for information retrieval. In: Etzioni O (ed) *Software, agents – spring symposium*, AAAI Press, pp. 126–129

- White JE (1994) Telescript technology: the foundations for the electronic market place, <http://www.genmagic.com/telescript> General Magic, 420 North Mary Ave., Sunnyvale, CA 94086, USA
- Wooldridge MJ, Jennings NR (1995) Agent theories, architectures and languages: a survey. In: Wooldridge MJ, Jennings N. (eds) *Lect. Notes in Artif. Intell.*, 890:1–39
- Wooldridge MJ (1996a) Agents as a Rorschach test: a response to Franklin and Graesser. In: Muller JP, Wooldridge MJ, Jennings NR (eds) *Lect. Notes in Artif. Intell.*, 1193: 47–48
- Wooldridge MJ (1996b) What agents aren't: A discussion paper. In: *Proceedings of the 1st IEE colloquium on intelligent agents*, London, UK, 2/1–2/2
- Zlotkin G, Rosenschein JS (1989) Negotiation and task sharing amongst autonomous agents in cooperative domains. In: *Proceedings of the 11th international joint conference on artificial intelligence IJCAI'89*, Detroit, USA, 2:912–917

Artificial Intelligence Techniques for Networked
Manufacturing Enterprises Management

Benyoucef, L.; Grabot, B. (Eds.)

2010, XXII, 508 p. 228 illus., Hardcover

ISBN: 978-1-84996-118-9