

CHAPTER 3 APPROACHES TO INTELLIGENT ASSISTANCE

3.1 Introduction

This chapter describes several commonly used approaches to intelligent assistance. Following this discussion, a strategy for implementing intelligent assistance within the scope of a software project planning tool is proposed.

3.2 Intelligent Assistance Approaches

Many solutions have been proposed to the notion of intelligent assistance (in different domains) over the years. These fall under several main categories: Decision Support Systems, Expert Systems, Expert Critiquing Systems, Intelligent Tutoring Systems, Blackboard Systems and Intelligent Agents.

The following sections review each of the above approaches and attempt to assess the potential application of each to the development of more powerful and useful software project management tools. Finally, in section 3.3 a proposal for the incorporation of intelligent assistance within a software project planning tool will be presented.

3.2.1 Decision Support Systems

Decision Support Systems are interactive computer based systems which help decision makers utilise data and models to identify and solve problems and make decisions. [Bonczek et al., 81] offers the following description of a DSS:

“The system must aid a decision maker in solving unprogrammed, unstructured (or semistructured) problems”.

Essentially DSS are computer-based systems that bring together information from a variety of sources, assist in the organisation and analysis of this information and facilitate the evaluation of underlying assumptions [Mallach, 94]. They help managers/decision makers use and manipulate data; apply checklists and heuristics; and build and use mathematical models. The availability of DSS provides the opportunity to improve data collection and analysis processes associated with decision making, and further, DSS provide opportunities to improve the quality and responsiveness of decision making and hence the opportunity to improve the management of projects. According to Turban [Turban, 95], a DSS has four major characteristics: DSS incorporate both data and models; they are designed to assist managers in their decision processes in semistructured (or unstructured) tasks; they support, rather than replace, managerial judgment; their objective is to improve the effectiveness of the decisions, not the efficiency with which decisions are being made.

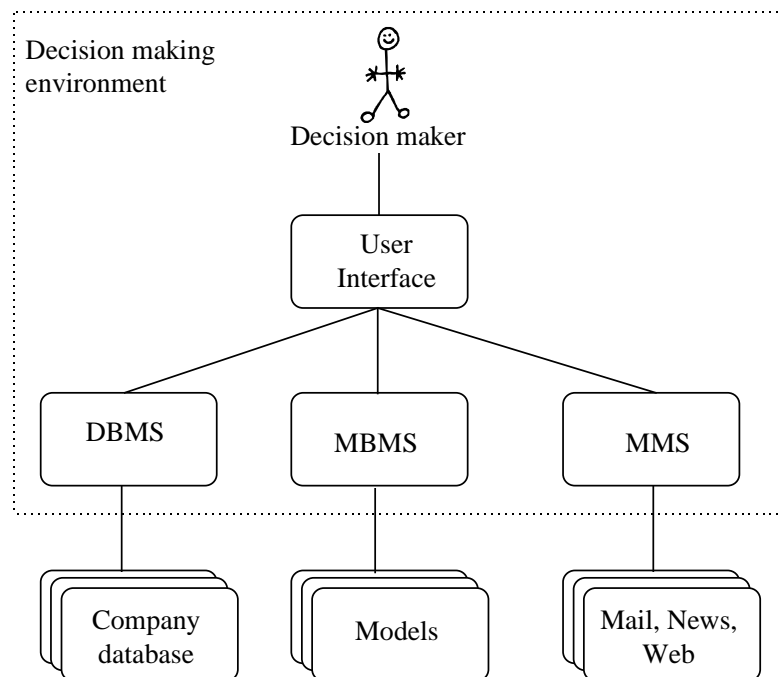


Figure 3.1 - Components of a Decision Support System

A DSS is typically composed of four components (figure 3.1); DBMS (Database Management System) - providing access to data and control programs to get the data into appropriate forms for analysis; MBMS (Model-Base Management System) keeps track of all models running during an analysis and provides the user with a facility to question the assumptions of models; A user interface provides the mechanism

whereby information is presented to the user; and recently a new component - the MMS (Mail Management System) has emerged which incorporates mail and other on-line data into the decision support models.

DSS have been successfully applied in a number of fields. For example the DESSERT project [Tierney and Davison, 95], a European Commission funded RACE II project which examined the use of DSS technology to help users (suppliers of telecommunications products) cope with increasingly complex engineering problems associated with telecommunications service provision. The project implemented the GSAC (Generation and Selection of Alternative Configurations) prototype DSS which took customers technical requirements as input and generating all feasible telecommunications access configurations and established the best (optimal) solution that meets both the customer and service providers requirements. The group's conclusions, supported by prototype trials by British Telecom and SEMA Group Telecom, suggested that DSS technology was an important element in making key parts of service management more efficient.

3.2.2 Expert Systems

An Expert System (ES) is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice [Jackson, 90]. An ES may completely fill a function that normally requires human expertise, or it may play the role of assistant to a human decision maker. The symbolic reasoning of an ES enables it not only to draw conclusions, through a process similar to the one used by human experts, but also enables them to provide explanations concerning their estimations. ES technology is based on the domain knowledge of the problem being addressed. A problem domain defines the objects, properties, tasks and events within which a human expert works and also the heuristics that trained professionals have learned to use in order to perform better [Klein and Methlie, 95].

The DENDRAL project [Lederberg, 87] initiated by Ed Feigenbaum and others at Stanford University in the mid 1960s was the first system to demonstrate that it was possible for a computer program to rival domain experts in a specialised field. In the case of DENDRAL the domain was the identification of complex compounds in a molecular structure.

DENDRAL was an early example of the basic structure of an expert system: problem solving behavior and formalized domain specific knowledge (in the form of a rule-based system). It had the ability to explore and abandon potential goal paths and is considered one of the earliest successes in expert systems. Many systems were spawned from DENDRAL. Two of the most noteworthy are Meta-DENDRAL and GENOA [Buchanan and Feigenbaum, 78].

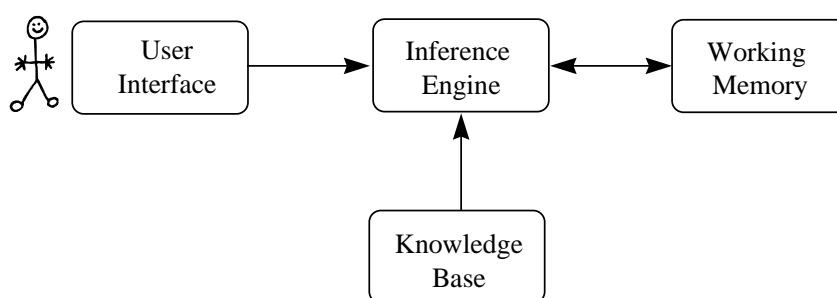


Figure 3.2 - Components of a Expert System

The heart of every expert system consists of two principal parts (see figure 3.2): the knowledge base; and the reasoning, or inference, engine. The knowledge base of expert systems contains both factual and heuristic knowledge. Factual knowledge is that knowledge of the task domain that is widely shared, typically found in textbooks or journals, and commonly agreed upon by those knowledgeable in the particular field. Heuristic knowledge is the less rigorous, more experiential, more judgmental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic. It is the knowledge of good practice, good judgment, and plausible reasoning in the field. It is the knowledge that underlies the “art of good guessing”. Other system components include: Working memory, which is used to store the current ‘facts’ or state of the domain - and a user interface to handle user input and output.

Primarily, the benefits of ES's to end users include [Engelmore , 93]:

- Improved quality of decision making.
- A speed-up of human professional or semi-professional work.
- Preservation of scarce expertise. ESs are used to preserve scarce know-how in organizations, to capture the expertise of individuals who are retiring, and to preserve corporate know-how so that it can be widely distributed to other factories, offices or plants of the company.
- Introduction of new products. For example, a pathology expert advisor sold to clinical pathologists to assist in the diagnosis of diseased tissue.

A more recent example of ES technology applied to a technical domain would be the EXPIDER system [Shen et al., 97] which was applied to solving channel routing problems in VLSI (Very Large Scale Integration) design. The system was built using CLIPS (C Language Integrated Production System) [Giarratano and Riley, 94].

3.2.3 Expert Critiquing Systems

Expert Critiquing Systems (ECS) [Silverman, 92] are a class of programs that receive as input the statement of the problem and the user-proposed solution. They produce as output a critique of the users judgement in terms of what the program considers is wrong with the user-proposed solution. Simple, yet widely used examples of critiquing system are spell checkers.

Critiquing programs may be user invoked (passive) critics or active critics. They may work in 'batch mode' - process the entire solution after the user has finished it; or incremental mode - interrupt the user during their problem-solving task. They do not necessarily solve problems for the user. Their core task is to recognise and communicate debatable issues concerning a product. Critiquing systems point out errors and suboptimal conditions that might otherwise remain undetected. They also advise users on how to improve the product and explain their reasoning, thus helping users avoid problems and learn different views and opinions.

An appealing characteristic of ECS is that they can be employed in a wide variety of situations with a broad range of supportive functions. They can, for example, complement problem-solving systems with or without full (i.e. global) knowledge of the problem space by being able to comment on particular sections of the problem/solution at hand. They can also function in situations where no near-optimal or optimal solution can be given at all and still be able to provide useful comments.

The critiquing process is illustrated in figure 3.3. The user initiates the task using some task support software. For a given task, the users input to the system consists of:

1. A description of the problem (e.g. design requirements). The problem may also be one the computer is displaying to the user, as perhaps in a process control application.
2. The proposed solution to the problem, such as the final design or completed diagnosis.

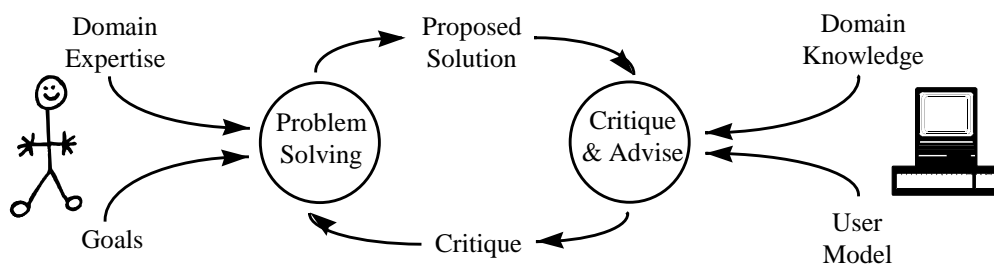


Figure 3.3 - The Critiquing Process

The critic applies its domain knowledge to the user problem and attempts an analysis of the user supplied solutions. It produces a critique and advises on possible flaws in the solution and on possible unforeseen implications of the proposed solution. This information is then used by the user to apply in the next iteration of this process.

Critiquing systems have appeared in fields such as LISP programming environments [Fischer, 87] and CAD (Computer Aided Design) [Gupta et al., 96]. They have also been applied in the field of software project support systems - the IMPW project (Integrated Management Process Workbench) [Jenkins et al., 87] a European Commission funded ESPRIT project, designed a workbench of tools to aid project

managers organise and control projects. One of the tools developed was RISKMAN [Verbruggen et al., 89] which enabled project managers 'walk around' a proposed project and help him/her anticipate any major risks. This project was taken further with the development of the RISKMAN2 tool [Moynihan et al., 94].

3.2.4 Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) [Polson and Richardson, 88] allow the emulation of a human teacher in the sense that an ITS can know what to teach (domain content), how to teach it (instructional strategies), and learn certain teaching-relevant information about the student being taught. This requires the representation of a domain expert's knowledge (Expert Model), an instructor's knowledge (Instructional Model) and the particular student that is being taught (Student Model).

The expert model represents information specific to the subject being taught, the student model portrays the current student understanding or misunderstanding of the subject and the instructor model contains knowledge required of teachers to select meaningful lessons for their students. Through the interaction of these models, intelligent tutoring systems are able to make judgments about what the student knows and how well the student is progressing. Instruction can then be automatically tailored by the Instructional Model to the student's needs, without the intervention of a human instructor. The ITS acts as the student's private tutor, while the human trainer or teacher is then free to focus on more complex and individualised student needs.

The design of most ITS is closely linked to how the ITS presents the domain to a student: knowledge from the simulation, knowledge packaged as rules, frames, or scripts, or intuitive interfaces. Figure 3.4 illustrates the dimensions of device or operational simulation, the domain expert and the interface.

A large number of ITS's have been constructed, although few are in widespread use [Frasson and Gauthier, 86]. ITS have been applied in many domains such as: IDE (Instructional Design Environments) [Russell et al., 89] which has been used in

foreign language instruction; ISD Expert [Merrill, 87] for instructional developers; teaching concepts in debugging electronic circuits [Brown et al., 82] and EDUC [Cox, 94] in the domain of engineering education.

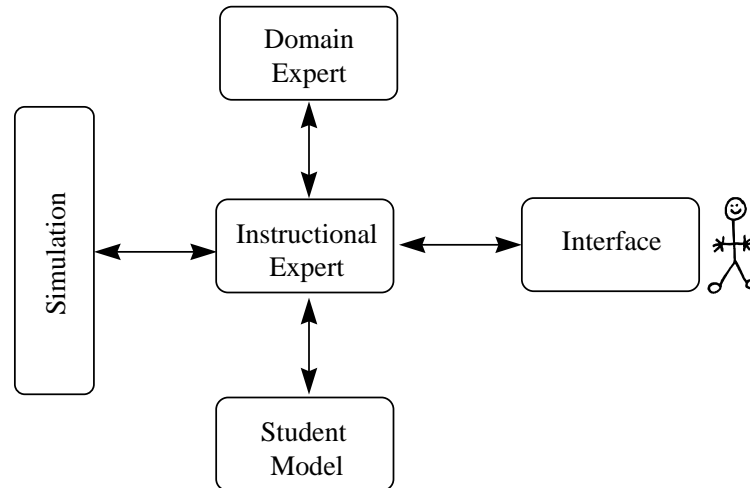


Figure 3.4 - Domain Knowledge Architecture

3.2.5 Blackboard Systems

The Blackboard [Hayes-Roth, 83] is a problem solving model prescribing the organisation of knowledge, data and the problem solving behavior within the overall organisation. The blackboard paradigm is best described using the following analogy [Englemore and Morgan, 88]:

“Imagine a number of people in a room trying to solve a jigsaw puzzle together. The room has a large blackboard and around it there is a group of people each holding pieces of the puzzle. Some volunteer to put their most ‘promising’ pieces on the blackboard and the others look at the pieces in place on the blackboard and then examine their own pieces to see if any fit. Those who have pieces that fit then go to the blackboard and fit their pieces. This process continues until all the pieces have been placed on the blackboard and the problem is solved.”

Essentially blackboard systems use multiple independent knowledge sources to analyse different aspects of a problem. Each knowledge source contributes its information to the common working memory (the blackboard). Blackboards compose solutions from component subsolutions, each of which may be generated or modified by its own knowledge sources.

Blackboards consist of three major components (figure 3.5):

1. **Knowledge sources** - (KSs). Like the human experts, each KS provides specific expertise needed by the application.
2. **The blackboard** - a shared repository of problems, goals, partial solutions, suggestions and contributed information. The blackboard can be viewed as a dynamic library of requests and contributions that have been recently published by other KSs.
3. **The control shell** - which controls the flow of activity in the application. Just as eager human specialists sometimes need a moderator to prevent them from trampling in a mad dash to grab the chalk, KSs need a mechanism to organize their use in the most effective and coherent fashion.

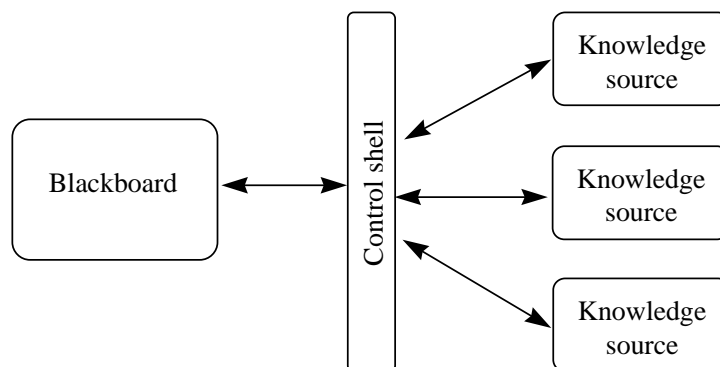


Figure 3.5 - Blackboard Components

The blackboard paradigm offers a number of important advantages, including:

- *Modularity* - KSs can be developed independently (for example, by programming teams) or even developed long before the application itself is developed.

- *Integration* - KSs can be implemented using widely differing approaches and programming languages. They may run remotely, on diverse hardware.
- *Extensibility* - New KSs can be added easily, and existing KSs can be replaced with enhanced versions.
- *Reusability* - KSs that provide expertise to one application can be redeployed in new applications.
- *Strategic Control* - Strategic control knowledge can be used to determine where the application expends its computational resources. Effective control is important when the number of KSs grows, when KSs have overlapping capabilities and when the best approach to solving today's problem differs from the approach used in yesterday's problem.

Blackboard technology has been successfully applied in a number of problem domains. The RISKMAN2 expert critiquing system project discussed in section 3.2.3 used a blackboard approach to pool knowledge from multiple knowledge sources. More recently USA based BBTech have developed Generic Blackboard Builder [Corkhill, 97] a generic blackboard product which has been used by the Ford Motor Company in their engineering design process and the US Army for logistics planning.

3.2.6 Intelligent Agents

The area of Intelligent Agents has emerged in recent times as a 'hot topic' in several branches of computing research from artificial intelligence to information systems. Among the agent community there is no commonly agreed definition of a software agent, indeed Carl Hewitt remarked¹:

"...the question what is an agent? is embarrassing for the agent-based computing community in just the same way that the question what is intelligence? is embarrassing for the mainstream AI community".

¹ At the 13th International Workshop on Distributed Artificial Intelligence, Washington, USA, 1994.

The problem is that although the term is widely used by many people working in closely related areas, it defies attempts to produce a single universally accepted definition. With this in mind it is possible to distinguish two general usages of the term 'agent': the first is weak, and relatively uncontentious; the second is stronger, and potentially more contentious [Wooldridge and Jennings, 95].

The weaker, and perhaps the most general way in which the term agent is used, refers to a kind of UNIX-like software process which exhibits the following properties:

- **Autonomy** - agents operate without the direct intervention of humans and have some kind of control over their actions and internal state.
- **Social ability** - agents interact with other agents (and possibly humans) via some kind of agent-communication language.
- **Reactivity** - agents perceive their environment and respond in a timely fashion to changes which occur in it.
- **Pro-activeness** - agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

This weak notion of agency is basically that used in the emerging discipline of agent-based software engineering [Genesereth and Ketchpel, 94] and describes agents as being able to “communicate with their peers by exchanging messages in an expressive agent communication language”.

For some researchers - particularly those working in AI - the term 'agent' has a stronger and more specific meaning than that sketched out above. These researchers generally mean an agent to be a computer system that, in addition to having the properties identified above, is either conceptualised or implemented using concepts that are more usually applied to humans. For example, it is quite common in AI to characterise an agent using mentalistic notions such as knowledge, belief, intention, and obligation [Shoham, 93]. Some AI researchers have gone further and considered emotional agents [Bates et al., 92]. Another way that agents have been given human-like attributes is by visual representation in the form of an animated face [Maes, 97].

Agent technologies also offer models of social cooperation. Using agent-based approaches, economists have constructed informative (if not completely predictive) models of economic markets. Agent technologies have exerted an increased influence on the design of distributed computing systems, the construction of internet search tools and implementation of cooperative work environments.

From an architectural viewpoint, [Luger and Stubblefield, 98] argue that a single architecture cannot account for all intelligent behavior. Instead, intelligence results from the cooperation of highly specialised agents. [Minsky, 85] outlines such a model in which the mind consists of a collection of specialised agents. Each agent contributes a particular ability to such tasks as understanding data or high-level problem solving. Other proponents of this multi-agent view of the mind include Hebb [Hebb, 49], Selfridge [Selfridge, 59], Fodor [Fodor, 83], Brooks [Brooks, 89] and Dennett [Dennett, 91].

When compared with the technologies discussed in the previous sections, agents are a young research area. However, a number of agent based systems have emerged; for example the ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems) project [Jennings et al., 96a], a European Commission funded ESPRIT II project aimed at developing an architecture, software framework and methodology for multi-agent systems for industrial applications in the area of power system control supervision. Another example is the ADEPT (Advanced Decision Environment for Process Tasks) project [Alty et al., 94], which provides an agent-based infrastructure for managing business processes. Both of these systems will be examined in further detail from an architectural perspective in Chapter 4.

3.2.7 Review of approaches

In this section, the six approaches to intelligent assistance previously discussed are contrasted to assess their comparative strengths and weaknesses in order to appraise their suitability as approaches to implementing intelligent assistance in software project planning tools.

Expert Critiquing Systems are different to Expert Systems in that ES only accept the problem statement as input and provide their machine generated solution as output, by applying rules in its knowledge base to the problem under consideration. ECS take as input a user proposed solution and a statement of the problem and attempt to offer an opinion on this solution. ECS do not necessarily solve problems for the user, they simply point out errors and sub-optimal conditions in the user-proposed solution that otherwise may remain undetected. ECS are particularly well suited to complex problem domains (such as project planning) as they do not always have an optimal solution and the problem cannot be precisely specified before attempting a solution [Fischer et al., 91]. In contrast to ES, ECS can function with only a partial understanding of the task, as they can provide support by applying generic domain knowledge, whereas ES are inadequate in situations where it is difficult to capture sufficient domain knowledge, because they leave the human out of the decision process and all 'intelligent' decisions are made by the computer.

Both ES and ECS have been applied in various problem domains. For example, the EXPIDER system [Shen et al., 97] successfully applied ES technology to the domain of VLSI design and the RISKMAN2 tool [Moynihan et al., 94] implemented an ECS in the domain of software project risk. Both of these technologies have proved to be of benefit to the user and can be seen to have a complementary approach, where ES assist the user in arriving at a solution and ECS enhance solutions by providing a critique of them. This dual approach to problem solving can potentially overcome the difficulties of operating in a situation where domain knowledge may be incomplete or inconsistent. However, it is worth noting that most implementations of ES's have been for well understood domains, which lend themselves more easily to capturing all necessary decision making data in the form of rules or other similar representation.

In contrast to ES and ECS, where domain knowledge in problem solving is embedded in a knowledge base, Decision Support Systems provide a framework for users in which models of the domain may be built, data gathered and decisions arrived at through informed analysis. Traditionally DSS do not have ES style prescriptive knowledge bases, rather they assist the user in analysing and evaluating assumptions underlying a problem by using models, but do not propose or critique a solution. In

the complex 'data rich' domain of large scale software development projects, DSS could prove useful in the gathering and analysis of project data, in an attempt to evaluate models of potential solutions. The results of the DESSERT project [Tierney and Davison, 95] in the telecommunications domain affirm this proposition. However, DSS may be more useful to the software project manager if coupled with the advisory and critiquing capabilities of both ES and ECS.

A view of an ITS would be that a persons problem solving skills would be represented by a set of production rules. Errors in problem solving efforts can be explained by the absence, incorrectness, or misuse of one of these rules. Intelligent computer-aided instruction seeks to identify the missing or incorrect rule and then teach the learner that skill or rule. ITS 'reconstruct' a problem solving process in a data driven way whereas an ES 'executes' a problem solving process in a goal driven manner. [Kamsteeg and Blerman, 89] illustrate this by the example of using an ES as the domain knowledge component of an ITS, which results in the writing of a new ES specifically designed to use the domain knowledge component of an ITS. This is due to having to add knowledge (e.g. incorrect knowledge, more levels of detail) and change knowledge representation to make it more explicit. In the context of providing training support for software project managers, the ITS approach would be a primary candidate. However the ITS approach would be less useful in the context of tool support for project managers, as this is removed from the notion of training support.

The Blackboard concept is a general model of problem solving and is particularly useful where there exists a number of (independent) knowledge sources. It provides a framework in which these knowledge sources may work cooperatively to assist the user in arriving at a solution. In common with DSS, the blackboard approach does not propose or critique a solution. It provides a framework for arriving at a solution state. The blackboard concept has been successfully applied within a number of approaches: DSS blackboards have been used in a number of systems, including DESSERT project mentioned earlier. In ES projects such as ESHELL blackboards were used to enhance working memory of the ES; and in ECS such as RISKMAN2, blackboards are used as a framework for each critic (or knowledge source) to add its advice and inspect the state of the developing solution. It is clear from previous research that blackboards are

both a viable and enhancing technology around which to base a future intelligent assistant systems.

Intelligent Agents are a relatively new and potentially exciting aspect to intelligent assistance systems. The agent properties of autonomy, reactivity and pro-activeness which fit naturally with the characteristics of the technologies discussed above. Blackboards are a natural candidate for use by agents, as they can represent ('autonomous') knowledge sources each of which cooperates via a blackboard to arrive at a solution state. Agents may also function as critics (as in the 'reactive' nature of ECS) or expert advisors (where an agent represents a 'pro-active' mini ES), where each agent is a specialist in some aspect (or sub-domain) of the greater problem domain being addressed. The former (critic mode) of agents was explored in conjunction with the blackboard model in the RISKMAN2 project and the latter (expert mode) was tested during the ADEPT project. From the research outlined previously and other current work in the area of agents it is apparent that agent based technology has a significant role to play in the development of future intelligent assistant systems.

3.3 Proposal for a new Intelligent Assistant

It is the proposition of this research that, in the complex domain of software project planning, a useful framework to support the project manager in the decision making process is a hybrid of a number of techniques including DSS, ES, ECS and the blackboard model. It has therefore been proposed [O'Connor et al., 97b] [O'Connor and Renault, 98] to incorporate the information gathering and analysis techniques of DSS, with the ability of ES to propose possible solutions using expert knowledge and best practices and the power of ECS to critique the possible solutions, thus providing the project manager with every facility to make an informed and quality decision.

It is considered that an agent based system will provide for an approach which enables the inter-working of a variety of well understood techniques within a single underlying framework - that of an agent-orientated system as illustrated in figure 3.6.

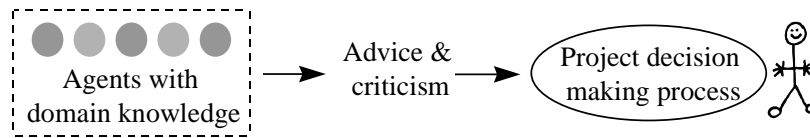


Figure 3.6 - Decision Making Process

The system proposed therefore is composed of a library of intelligent software agents - where each agent would play the role of a ‘mini-expert system’ or ‘mini-critiquing system’, each with an associated knowledge base. These agents would utilise the blackboard model of problem solving to communicate and thus converge on possible solution states and examine those states to assess their suitability given current conditions. This agent-orientated system would operate within the overall framework of a Decision Support System, which would provide for the gathering and analysis of data regarding a project and the development of models of the project with the aid and critique of the agents, as illustrated figure 3.7.

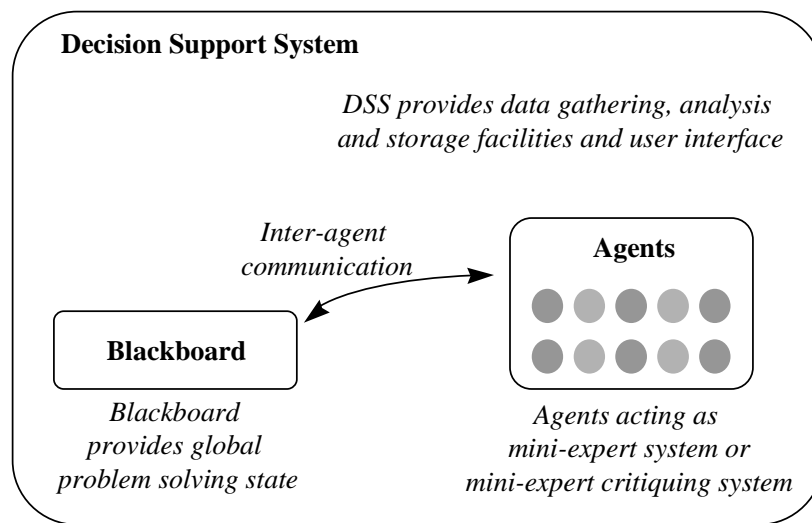


Figure 3.7 - Decision Making Framework

The major perceived benefits of this approach are the facilitation and improvement of the quality of decision making by a software project manager by reducing information overload and augmenting the cognitive limitations and bounds of the decision maker. This hybrid method of assistance, coupled with the architectural properties of intelligent agents (dynamic and distributed objects), present an ideal strategy to implement intelligent assistance system for use in software project planning.

In addition to the properties outlined above, an agent-orientated architecture is a natural choice to address the issue of heterogeneous client-server systems development. Recent research in Java-based agents [Watson, 97] [Caglayan and Harrison, 97] and mobile Java-based agents [Lange and Oshima, 97] has concluded that they are a viable technology on which to establish a platform independent agent-orientated architecture. To address the client-server issue, Orfali [Orfali and Harkey, 98] has successfully demonstrated the use of CORBA (Common Object Request Broker Architecture) as a basis for developing platform independent client-server systems, including agent-orientated systems.

To provide the necessary flexibility for the proposed system and to tackle the issues above, it is considered that both Java and CORBA provide an appropriate framework on which to base the system.

3.4 Summary

This chapter has described several commonly used approaches to intelligent assistance. They are contrasted to gain a better understanding of how to approach intelligent assistance in the domain of software project planning. Finally, a proposal for an agent-orientated hybrid approach to implementing an intelligent assistance tool for software project planners was presented.

Chapter 4 contains a review of the architectural aspects of intelligent systems and the trend towards distributed client-server platform independent systems. A number of intelligent assistant systems are investigated and the concepts of agent-based architectures for intelligent assistant presented.