An agent-based approach to managing distributed, multi-platform software development projects

Rory O'Connor, Eamon Gaffney and Tony Moynihan
School of Computer Applications
Dublin City University
Ireland
{roconnor, egaffney, tonym}@compapp.dcu.ie

Abstract

This paper describes work undertaken within the context of the P3 (Project and Process Prompter) Project which aims to develop the Prompter tool, a 'decision-support tool to assist in the planning and managing of a software development project'. Prompter will have the ability to help software project managers to assimilate best practice and 'know how' in the field of software project management and incorporate expert critiquing to assist with solving the complex problems associated with software project management. This paper focuses on Prompters agent-based approach to tackling the problems of distributed, platform independent support.

1. Introduction

To support software project managers, organisations have sought to develop a number of information technology systems to assist with various aspects of the management of their software processes. Many tools exist in the market today that assist a project manager in addressing some of these objectives. These tools fall into three broad categories:

Project planning - tools which are concerned with the scheduling aspects of planning a project, and pay less attention to organisation and methodological aspects of management.
Process management - which support the framework and rules of management of the projects process.
Risk analysis - specific tools which are often used as ancillary tools at specific stages during a project.

Although many project management systems are currently available, the enormous scope and complexity of current software systems means moving beyond the current state of practice (for example Microsoft Project or PERT charts, etc.), as such systems do little to support the 'average' project manager. In addition, many of these systems fall short of supporting the project manager in his/her decision making processes and do not offer assistance in representing knowledge about plans and designs, or provide mechanisms for reasoning about plans and designs in flexible ways. In the light of this and having completed a survey of commercial tools, we have identified several areas which would be of benefit in assisting the project manager:

- The ability to reason about projects plans, analyse alternatives and select the most suitable course of action.
- The ability to track the history of a project and capture knowledge gained, and reuse this knowledge as an aid to future project decision-making.
- Assist the project manager in adherence to standards, industry best practices and implementation of company policy.
- The ability to intelligently manage and analyse large amounts of project data.

In addition, as we move closer to the next millennium, software project managers face new and more complex challenges. Recent technological developments such as improved communications channels and the widespread use of Internet technology has lead to the development of the 'virtual project team', where team members are physically distributed throughout an organisations locations. This has brought about a requirement for project management support systems to address the needs of distributed software development projects. This emphasis on distributed software development coupled with the trend in client-server based computing has lead to an additional requirement for project management support systems to address the issues of multiple platforms, both in terms of hardware and operating systems.

The aim of the P3 project is to build the Prompter tool which will address the issues outlined above. Prompter will assist software project managers in their decision-making process, by helping them to assimilate best practice in the field of software project management and incorporate expert critiquing which will assist project managers in solving the complex problems associated with software project planning.

In this paper we describe the results of implementing a prototype of the Prompter tool. Our research approach consists of two key elements. Firstly the design of an agent-based approach to knowledge management and support within the context of a software project management tool. Secondly, the implementation of distributed, platform-independent agents using the OMG/CORBA model.

2. Agent-based architecture

The objectives of the section is to give an overview of the intelligent agent-based architecture chosen for the Prompter tool and outline the reasons for its choice as a solution technology for software project management tools.

2.1 Architecture overview

The architecture of the Prompter tool is illustrated in Figure 1. Here the user interacts with the tool via a GUI, with a number of project management expert agents below which advise and assist the user. In the following sections we will discuss the concept of agents and there use as expert assistants in a software project management tool.

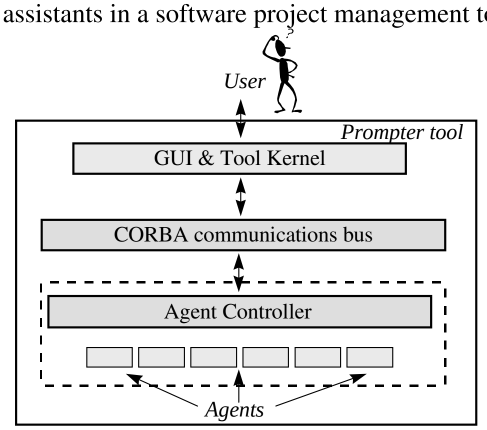

Figure 1 Prompter architecture

2.2 Agent overview

The area of Intelligent Agents [12] has emerged in recent times as a 'hot topic' in several branches of computing research from artificial intelligence to information systems. Intelligent agents are entities capable of autonomous goal orientated behaviour in some environment, often in the service of larger-scale goals external to themselves. An agent contains structures that enable representing knowledge, representing and achieving goals, interacting with the environments and coping with unexpected occurrences. To date most agent implementations have been in the area of Internet search and filtering agents [6], however agent-based technology has been applied to the supporting project management and decision making in the other domains. For example, in the domain of telecommunications management with projects such as ADEPT [1] and with electricity supply management with the ARCHON [5] project. Regardless of the application domain, agents may be characterised by the following attributes [3]:

Autonomy - agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
Reactivity - agents perceive their environment and respond in a timely fashion to changes that occur in it.
Collaborative behaviour - agents interact with other agents and humans in order to solve problem.
Proactiveness - agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.
Inferential capability - agents can act on an abstract task specification using prior knowledge of general goals.
Mobility - being able to migrate from one location to another, to solve problems there.

The various agent technologies existing today can be classified as being either Single-Agent or Multi-Agent systems. In Single-Agent systems, an agent performs a task on behalf of the user or some process. While performing its task, the agent may communicate with the user as well as with local or remote system resources, but it will never communicate with other agents. In contrast, the agents in a Multi-Agent system (MAS) not only communicate with the user and system resources, but they also extensively communicate and work with each other, solving problems that are beyond their individual capabilities. Such distributed MAS provide an ideal basis to develop agents for software project management.

2.3 Agents for project management

Given the characteristics of software project management, we consider the most natural way to model the decision making process is as a collection of autonomous distributed problem solving intelligent agents (or multi-agent system). The choice of agents as a candidate technology was further motivated by the following observations about the domain of software project management:

- The domain involves an inherent distribution of project data and problem solving capabilities.
- Interactions between problem solving entities are sophisticated and include information sharing and coordination.
- The domain cannot be completely prescribed from start to finish, therefore the problem-solving entities need to respond to changes in the environment.

When collectively viewed, these observations support the use of intelligent agents as a strong solution candidate to implement the inter-working decision support framework described above. In the P3 project these problem-solving capabilities are encapsulated in distributed intelligent agents that act as expert advisers helping users to define the process life cycle as well as providing continuous assessment on progress and quality of the project.

Each agent is an expert in a particular area of project management and continually analyses project parameters, making the user aware of any possible weakness and helping to mitigate the associated risks. The agents use their knowledge to advise the user on possible alternatives and advice on potential problems, as well as providing justification for their advice. Agents will encapsulate knowledge and best practice in the field of software engineering and in addition will have embedded the guidelines of quality standards such as ISO 9000 and the Capability Maturity Model.

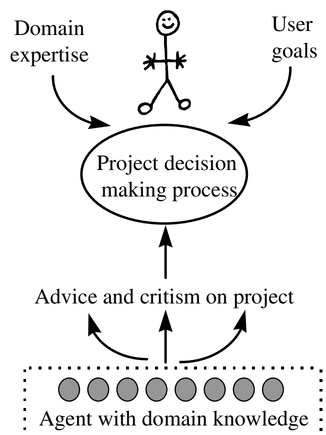

Figure 2 Prompter decision making paradigm

The decision making paradigm in Prompter can be seen in Figure 2 - two entities, agents and a user, working together to contribute what they know about the domain to solving some problem and hence make a quality informed decision. The users primary role is to generate and modify solutions; the agents is to analyse those solutions and produce a critique and advise on a possible solution for the human to apply in the next iteration of this process. In this scenario agents constantly 'watch' the actions of the user by way of monitoring project parameters as the user developing project plans and inputs actual project data. When an agent has all the information it needs, it will proceed with its analysis and produce its conclusions. These conclusions would be given to the user in terms of advice/criticism on the current/predicted future project situation, and also may be used as input data to other agents.

For example, there may be a number of agents who specialise in the selection of the most appropriate process model (lifecycle) for a particular project. The agents could have a set of criteria based on certain attributes of the project such as: the problem domain, product, available resources, personnel, and organisational attributes. These attributes would be examined and for each process model a comparative rating produced, indicating the most appropriate choice of process model. The agent would then communicate these findings to the user and other agents.

2.4 Distributed agents

With the increase in globalization, distributed information systems are rapidly becoming the norm. These systems cross geographical, and often organisational boundaries and are typically broadly heterogeneous in both hardware and software terms. In distributed systems, the idea of an agent is seen as a natural metaphor, and by some as a development of the concurrent object programming paradigm [2]. Much of the attention that currently surrounds agent-based technology is related to the phenomenal growth of the Internet. In particular there is great interest in 'mobile agents' that can move around a (local or wide area) network of machines on a users behalf. Such mobile agents have been successfully implemented in General Magic's Telescript language [11] and is closely related to the functionality provided by languages such as Java. To date much of the well publicised mobile agents fall into the category of Internet search and filtering agents such as BarginFinder, ShpoBot and CyberYenta.

The expert advisory agents for project management described in section 2.2 provide a mechanism to tackle the issues of developing a distributed platform-independent multi-agent system. Each agent contains the structures that enable representing knowledge, representing and achieving goals, interacting with the environments and coping with unexpected occurrences. As these the agent p paradigm encapsulates both problem solving and communications capabilities it provides an ideal candidate for deployment in a distributed fashion, provided that an appropriate transport bus is provided for agent communications. Section 3 describes the choice of OMG's CORBA Object Request Broker [8] as the communications bus on which to implement this distributed multi-agent system.

2.5 Agent representation

With the development of agent theories comes the question of agent representation and of agent languages [13]. Our concern is to effectively represent knowledge about the domain of software project management and provide a means for this knowledge to be used to advise and guide the software project manager in the decision making process. The encapsulated nature of multi-agent systems allows for a mix of agent representations be to implemented - provided a common interface is available. For the purposes of this paper we will introduce one agent representation language, Jess (Java Expert System Shell).

Jess [9] is a clone of the popular expert system shell CLIPS (C Language Integrated Production System) [4] written entirely in Java. CLIPS origins date back to a NASA project in the mid 1980s to establish an expert system language. Originally the primary representation methodology in CLIPS was a forward chaining rule language based on the Rete algorithm, however more recently both the procedural and object-orientated paradigms have been incorporated into CLIPS.

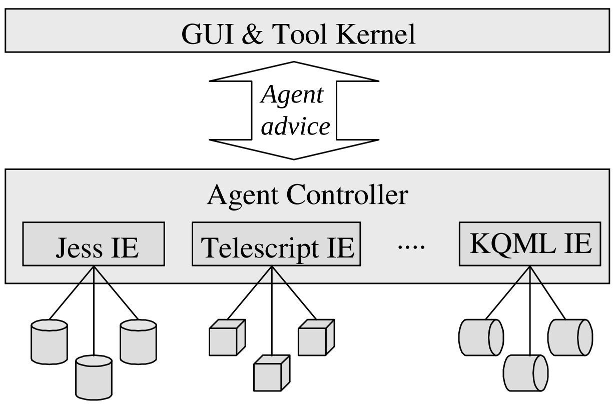

Figure 3 Agent architecture

The integration of a number of agents representations into the overall tool was also investigated. The architecture was constructed in a manner to allow the agents remain distinct from the tool, with an agent controller coordinating them. Within this controller there are a number of inference engines (IE), one for each type of agent representation. For the initial prototype there were two IE's constructed - the first was for standard forward chaining production rules and the second using Jess/CLIPS language. This facilitated having a narrow interface between the agents and the rest of the tool. This provides the advantage of allowing a number of agent representations to be added into the system without affecting the behaviour of existing system components. This architecture is illustrated in figure 3.

3. Distributed architecture

In this section we will discuss the use of CORBA and Java as mechanisms to implement the distributed architecture described earlier.

3.1 CORBA distributed bus

The CORBA bus allows transparent access to distributed objects over a heterogeneous network of machines and operating systems. CORBA distributes messages via its Object Request Broker (ORB) transparently between registered objects. The ORB receives requests from a 'client' to send a message to an object. The broker locates the object referred to by the client and delivers the message to that object. This style of architecture combined with the flexibility of CORBA provides a unique solution to the requirements of independence of implementation language, capacity for evolution and interfacing ability.

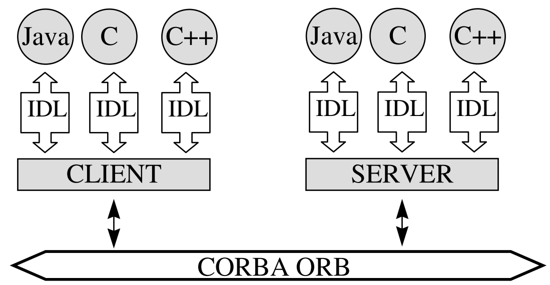

Figure 4 CORBA

OMG's CORBA is a standardised specification for the creation of distributed (client/server), object-oriented systems. Figure 4 shows a high level diagram of the CORBA structure. The client may be written in Java and the server written in C++ and neither would be aware of the difference. This is done using the ORB, which is a layer of software that allows developers to define objects and specify narrow interfaces (using CORBA's Interface Definition Language or IDL) through which these objects may be accessed. These interfaces allow the abstraction of all the underlying implementation details of the objects, their methods and any other information about the class. This allows applications to be distributed and heterogeneous - distributed meaning the various objects can be hosted across many computers and heterogeneous meaning many languages and many operating systems can be used, yet all operating together transparently. The ORB smoothes over the system differences between components of application. It can been described as an Object Bus, a software equivalent to the computer hardware bus [10] .

Objects can be implemented in various ways depending on the implementation language used. Thus in order to enable all objects to interact, CORBA must view them at a higher level of abstraction i.e. through interfaces. The mechanism used to implement these interfaces is IDL (Interface Definition Language) defined by OMG. This language allows classes, their respective attributes and methods and any other information to be defined. The IDL interface contains information such as where the methods implementation is situated and the parameters it requires etc. Neither the client or the server are aware of the mechanisms used to communicate method calls. All routine of communications handled at runtime by the ORB.

3.2 Java for mobile code

Java is an object-oriented architecture neutral portable multi-threaded dynamic language. Java was designed to support applications on networks, such as the internet, which made it an ideal choice for the Prompter tool. In addition it is fully object-orientated and therefore provides a natural choice for integration with CORBA. These properties make Java an ideal language to implement the Prompter tool.

Java compilers generate machine independent bytecode for what is called a virtual machine. This virtual machine is a layer of software on a machine, which takes the Java bytecode and executes them. This results in code that executes the same no matter what its underlying architecture is. Thus Java bytecodes can be shipped over the net and are guaranteed to function the same on all platforms. Java has achieved wide spread acceptance in the programming world [14]. Since Java is a mobile code system it can also be considered as an option for implementation of mobile agents.

Mobile code refers to the ability to transfer executable binaries to wherever they are needed. Though the basic Java support for bytecode migration implies Java code mobility, this capability is not really viable other that in applets. With the standard implementation of Java, all Java objects reside on a single host. Also, Java lacks mechanisms for transmitting arguments from one host to another. In contrast the fundamental premise of CORBA is that an object on one host can invoke a method of an object on another host. CORBA passes references rather than objects thus avoiding plunging into the issues of object migration. CORBA also provides a persistent object service that is not possible with Java alone.

For the purposes of Prompter, all the tasks of a distributed multi-platform system can be satisfied by using both Java and CORBA together. Java will allow CORBA objects to run on any system. Some suggest [7] that Java is the ideal language for writing client server objects. Its built in multi-threading and garbage collection makes it easy to implement robust objects. The two structures compliment each other. Java deals with implementation transparency and CORBA provides the services not covered by Java. It links the Java portable application environment and the rest of the world.

4. Prototype Development

An early prototype of the Prompter tool was developed for a number of reasons: Firstly as a proof of concept - the prototype assist in highlighting any flaws in the proposed architecture and investigated the feasibility of the distributed architecture based on CORBA and Java. It also assisted in identifying possible communication, data storage and knowledge representation problems that may occur in the development of the tool

The implementation strategy for this prototype was to develop the tool from the top level down i.e. to implement the prototype in Java initially to reduce complexity of code. It was then planned to introduce CORBA communications code at a later stage. This gave an easy transition from design into actual code.

On completion of the initial coding of the prototype in Java, the IDL interfaces were introduced. At this stage it became clear that many of the class structures would have to be altered as IDL did not support many of the rich structures specified during the design phase. An example of this can be illustrated using a simple 2D array structure. IDL does not allow undefined sized 2D arrays, so these arrays must be flattened into what are called "sequences" in IDL. This problem gets more difficult on more complex structures needing major restructure of the classes.

The version of CORBA used was Iona's OrbixWeb v2.0. The Implementation code had to be separated from the calling mechanisms and wrapped in a server class, with each server been given a name, so the client will know how to identify it. The next stage was to package up all method calls into a client class. This involved the creation of a client class to bind to the server. This class creates a reference for the server by specifying its name. The ORB is responsible for finding where the server resides and passes the reference back to the client. All calls to this server are then intercepted by the ORB and routed to the server's machine.

The prototype consisted of a number of servers in the agent architecture - Agent Controller, Blackboard, Agents themselves. The advantage here is that each of these servers are asynchronous from, each other and from the clients that call them, making them almost completely independent. Thus on the majority of occasions when the client call is made it can continue with its own process, the call is one-way and so does not have to wait for the calls completion.

Some issues that arose with the introduction of CORBA was the increased complexity of the code and the increase length of execution time. Since CORBA uses references to objects and servers, the code becomes more complex to debug. Another problem was the overhead in running the software - when the software was run on one machine the server calls were almost instantaneous. However, as more distribution was introduced the overhead became more dependent on the state of the network. This was expected however, but it did lead to a lot of waiting by some clients on results from the servers and generally slowing the tool down. This was most evident when the tool started up and led to the creation of all the servers. The ORB initially has to search for the location of the servers. This means requesting information form the network DNS (The network supervisor). This lead to unwanted delay. This problem was overcome by specifying exactly what server were where i.e. by giving it exact IP addresses.

A prototype system was successfully developed and deployed in a distributed manner over a number of networked machines. This prototype system was demonstrated to several project management tool users (potential future customers) and received a positive response. Work is currently underway on developing a full scale version of the Prompter tool.

5. Conclusions

Software development projects are becoming more complex in both their structure and the problems which they address. As a consequence, the tools used for the management of software development projects need to cope with the increased demands on project managers by supporting their role as decision makers. In addition, the emerging requirement for tools to tackle the issues of distributed multi-platform software development projects need to be addressed.

In this paper we have described the Prompter tool which addresses the issues outlined above by supporting the integration of knowledge support and expert critiquing in a distributed platform independent intelligent agent based system. The implementation of a prototype leads us to believe that the use of knowledge based guidance will be a valuable addition to the traditional support tools available to software project managers.

References

[1] J.Alty, D.Griffiths, N.Jennings and E.Mamdani, “ADEPT - Advanced Decision Support Environment for Process Tasks: Overview and Architecture”, In proceedings of the 1994 BCS Expert Systems conference, British Computer Society, 1994.
[2] G.Agha, P.Wegner and A.Yonezawa, Eds, “Research directions in Concurrent Object-Orientated Programming”, MIT Press 1993.
[3] J.Bradshaw, “An Introduction to Software Agents”, In Software Agents, MIT Press, 1997.
[4] J.Giarratano and G.Riley, “Expert Systems Principles and Programming”, PWS Publishing, 1994.
[5] N.Jennings, J.Corera, I.Laresgoiti, E.Mamdani, F.Perriollat, P.Skarek and L.Varga, “Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control”, IEEE Expert, Vol. 11, No. 6, 1996.
[6]P.Maes, “Software Agents”, In Proceedings of the Unicom conference on Agents and Intelligent User Interfaces, London, March 1997.
[7] R.Orfali, D.Harkey, J.Edwards, “CORBA, Java and the Object Web”, Byte, October 1997.
[8] R.Orfali, D.Harkey, J.Edwards, “Instant CORBA”, John Wiley & Sons, 1997
[9] M.Watson, “Intelligent Java Applications”, Morgan Kaufmann, 1997.
[10] M.Weiss, A.Johnson and J.Kinsey, “Distributed Computing: Java, CORBA, and DCE”, Technical Report Open Systems Foundation, 1996.
[11] J.White, “Telescript Technology: The foundation for the electronic marketplace”, White paper, General Magic Inc., USA, 1994.
[12] M.Wooldridge, N.Jennings, “Agents Theories, Architectures and Languages: A Survey”, Lecture Notes in Artificial Intelligence, Springer-Verlag, 1995.
[13] M.Wooldridge and N.Jennings, “Intelligent Agents: Theory and Practice”, Knowledge Engineering Review Vol. 10 No. 2, 1995.
[14] K.Srinivas, V.Jugannathan and R.Karinthi, “Java and Beyond Executable Content”, IEEE Computer, June 1997.

Acknowledgements

This research was supported by the Fourth Framework programme of the European Commission under ESPRIT project 22241.

The authors gratefully acknowledge the value gained from discussions regarding this work with their research partners: Marty Sanders, Robert Cochran, Brian McCarthy, Mark Johnston (Catalyst Software), Annie Combelles, Christophe Floch, Tristan Renault and Jean-Michel Douieb (Objectif Technologie), Martine Pauleito, Chantal Jourdain (Schneider Electric) and Vassilis Kopanas (Intracom).