A Software Agent Framework for the Support of Software Project Management

RITA NIENABER
UNIVERSITY OF SOUTH AFRICA
AND
ELSABE CLOETE
UNIVERSITY OF SOUTH AFRICA

Numerous software development projects do not live up to expectations or sadly fail. This can be seen in the fact that software projects often do not comply with the traditional standard measurements of success, namely time, cost and specifications. Traditionally, individual software projects were executed at a single location. However, due to globalisation and advances in computing technologies, this has changed, and software projects are developed and deployed in distributed and collaborative environments. This means that traditional project management methods cannot and do not address the added complexities found in a distributed environment, such as efficient task scheduling, tracking and monitoring, as well as effective sharing of information and knowledge among project contributors. High levels of collaboration, task interdependence and distribution have become essential across time, space and technology. In this paper the utilisation of stationary and mobile software agents is investigated as a potential tool to improve software project management processes. We also propose and discuss a software agent framework to support distributed software project management. Although still in its initial phases, this research shows promise of significant results in enabling software developers to meet market expectations, and produce projects timeously, within budget and to users' satisfaction

Categories and Subject Descriptors: D1.3 [**Programming Techniques**]: Concurrent Programming - *Distributed Programming; Concurrent Programming;* K6.1 **[Management of Computing and Information Systems**]: Project and People Management - *Systems development; Strategic information planning;* K6.3 **[Management of Computing and Information Systems**]: Software Management - *Software process*.
General Terms: Design, Experimentation, Management
Additional Key Words and Phrases: Software agent computing, Software project management, Collaborative distributed software projects.

---

1. INTRODUCTION

Software applications are integrated into almost every business application today. It is the quality, effectiveness and efficiency of these applications that determine the success or failure of many business solutions. As a result, businesses often find that they need to attain a competitive advantage through the development of software projects that support crucial business activities. The quality of a software project is determined by the quality of the software development process. Improvements in the development process can result in significant improvement in software quality [Schwalbe 2000].

Over the past few decades, software projects generally failed to come up to user expectations, were commonly delivered late, and mostly ran over the set budget. Indeed, much of this still holds true today, and it has alerted software developers and managers to the fact that these issues have to be addressed in concrete terms, and as a result the field of *Software Project Management (SPM)* has evolved. SPM involves the management of all aspects and issues that are involved in the development of a software project, namely scope and objective identification, planning, evaluation, project development approaches, software effort and cost estimation, activity planning, monitoring and control, risk management, resource allocation and control, as well as managing contracts, teams of people and quality. Initially, traditional *Project Management* (PM) techniques were applied to the development of software projects. Different standard project management approaches exist, which are applicable to different areas of SPM, such as PRINCE 2, BS 6079. However, over time PM methods seemed to lack in the ability to address the unique characteristics of the software development arena [Hughes and Cotterell 2002]. This led to the development of SPM as an independent application area and field of study.

Author Addresses:
R.C. Nienaber, Department of Computer Science and Information Systems, University of South Africa, P O Box 392, UNISA, 0003, South Africa; nienarc@unisa.ac.za.
E. Cloete, Department of Computer Science and Information Systems, University of South Africa, P O Box 392, UNISA, 0003, South Africa; cloete@unisa.ac.za.

In order to address the existing shortcomings in managing software projects, practitioners also attempted to apply several Software Engineering principles to different SPM processes [Lethbridge and Laganiere 2000]. They explored standard structured analysis and design methods, and also incorporated object-oriented approaches to overcome the aforementioned shortcomings [Gelbard et al. 2002; Lethbridge and Laganiere 2000]. Yet disappointment remained since many software projects still failed to comply with the triple constraints of scope, time and cost [Oghma:Open Source 2003]. The triple constraints refer to the fact that the failure of software projects can mostly be attributed to the fact that they are not delivered on *time* and do not meet the expectations of the client *(scope)*, and as a result have *cost* implications.

The SPM environment is rapidly changing due to globalisation and advances in computing technology. This implies that the traditional single project, which was commonly executed at a single location, has evolved into distributed, collaborative projects. The focus in the SPM processes has clearly shifted from the position that it held two decades ago. Consequently, tools for effective sharing of information and knowledge among project contributors, as well as efficient task scheduling, tracking and monitoring are sorely needed. High levels of collaboration, task interdependence and distribution have become essential across time, space and technology [Chen et al. 2003].

A promising solution to addressing software management problems in a distributed environment is offered by *software agent technology*. According to this technology, software agents are used to support the development of SPM systems in which data, control, expertise, or resources are distributed. Software agent technology provides a natural metaphor for support in a team environment, where software agents can monitor and coordinate events and meetings and distribute documentation [Balasubramanian 2001].

SPM skills, especially in the distributed computing environment, are greatly in demand. Moreover, there is a desperate need for technologies and systems to support the management of software projects in these environments. Our research is therefore aimed at software practitioners and software developers, but will also be beneficial to researchers working in the field of SPM.

In this paper the use of software agents is investigated as a potential tool to improve the SPM processes. We concern ourselves with the question of how software agents can be used to improve SPM processes in a distributed environment. As a result, we propose a software agent framework to support distributed SPM. Although our research is not yet complete, initial indications are that it will be significant in enabling software developers to meet market expectations, which will bring about savings in cost, time and effort.

Section 2 contains a background study and a discussion on research in this area. Section 3 provides a generic-type framework for one of the key SPM processes. This framework can be adapted to support all SPM processes and further extended using a (similar) multi-agent grid structure framework to coordinate the individual processes. Finally, Section 4 presents a conclusion.

2. BACKGROUND

2.1 Software Agents

A software agent is a computer program that is capable of autonomous (or at least semi-autonomous) actions in pursuit of a specific goal. The autonomy characteristic of a software agent distinguishes it from general software programs. *Autonomy* in agents implies that the software agent has the ability to perform its tasks without direct control, or at least with minimum supervision, in which case it will be a *semi-autonomous* software agent. Software agents can be grouped, according to specific characteristics, into different software agent classes. Literature does not agree on the different types or classes of software agents. For example, Krupansky [2003] distinguishes between *ten* different types of software agents, while the Oghma Open Source [2003] web site identifies sixteen different types of software agents. Because software agents are commonly classified according to a set of characteristics, different classes of software agents often overlap, implying that a software agent might belong to more than one class at a time. For the purpose of this research, we distinguish between two simple classes of software agents, namely *stationary agents* and *mobile agents*. Agents in both these classes might, or might not have, any or a combination of the following characteristics: a user interface, intelligence, adaptivity, flexibility and collaborative properties.

Whether or not an agent has a *user interface*, depends on whether it collaborates with humans, other agents or hosts. User interfaces are commonly only found where agents interact with humans. According to Woolridge [2001] intelligence implies the inclusion of at least three distinct properties, namely *reactivity*, *proactiveness* and *social ability*. *Reactivity* refers to the agent's ability to perceive its environment and respond timeously to changes that occur in order to achieve its design goals. *Proactiveness* is the agent's ability to take initiative in its environment in order to achieve its design goals. *Social ability* alludes to the collaborative nature of the agent. There are different definitions to define the collaborative nature of software agents. For the purpose of this paper we use Croft's [1997] definition in which the *collaborative nature* of a software agent refers to the agent's ability to share information or barter for specialized services to cause a deliberate synergism amongst agents. It is expected of most agents to have a strong collaborative without necessarily implying other intelligence properties. *Adaptivity* is a characteristic that can also be regarded as an intelligence property, although it is not counted as a prerequisite to identify an agent as intelligent. *Adaptivity* refers to

an agent's ability to customize itself on the basis of previous experiences. An agent is considered *flexible* when it can dynamically choose which actions to invoke, and what sequence, in response to the state of its external environment [Pai et al. 2000].

We consider a stationary agent to be a piece of autonomous (or semi-autonomous) software that permanently resides on a particular host. Such an agent performs tasks on its host machine such as accepting mobile agents, allocating resources, performing specific computing tasks, enforcing security policies and so forth.

We consider a *mobile agent* to be a software agent that has the ability to transport itself from one host to another in a network. The ability to travel allows a mobile agent to move itself to a host that contains an object with which the agent wants to interact, and then to take advantage of the computing resources of the object's host in order to interact with that object. Full autononomy, migratability and collaborativeness are the most important characteristics that should be imbedded in each mobile agent. When a mobile agent possesses these three intelligence requirements, it is often referred to as a *robot* [Krupansky 2003].

2.2 Software Project Management

The IEEE defines SPM as the process of planning, organizing, staffing, monitoring, controlling, and leading a software project [IEEE Standards Board 1987]. A more detailed exposition shows that SPM involves the planning, monitoring and controlling of people and processes that are involved in the creation of executable programs, related data and documentation [Elec 4704 2003]. Figure 1 illustrates these issues in a framework that contains the key elements in the field of SPM. We distinguish between three key elements: project stakeholders, project management knowledge areas, and project management tools and techniques.

The project *stakeholders* are the people involved in all the different project activities and include the project sponsor, project team, support staff, customers, users, suppliers and even opponents. Good relationships, as well as communication and coordination between all of these stakeholders, are essential to ensure that the needs and expectations of stakeholders are understood and met. *Knowledge areas* include the key competencies concerned during the software project management process. The *core functions*, namely scope, time, cost and quality management lead to specific project objectives and are supported by the *facilitating* functions. The facilitating functions represent the means through which different objectives are to be met and include human resource management, communication, risk, and procurement management. Stretched across all these knowledge areas are the project management *tool and techniques* (on the right-hand side of the framework diagram). These are used to assist team members and project managers in carrying out their respective tasks.

This framework refers to the SPM of a single project, in which the SPM manager allocates tasks and gives instructions to various role players. However, as mentioned earlier, factors such as business globalisation, rapid technology advancement, and distributed team membership have influenced the SPM environment to such an extent that traditional tools and techniques supporting the key management areas are no longer adequate [Chen et al.2003]. For example, the focus of the communication and cooperating mechanisms has changed. As a result, the communication between various role players is not sufficiently supported in this environment to coordinate and schedule activities, and support document distribution. The 1995 Standish Group study found that the three major factors related to information technology project success were user involvement, executive management support and a clear statement of requirements [Krupansky 2003]. All these depend on having good communication and coordination skills among the stakeholders. Poor, ineffective or untimely communication, contradictions, omissions, failure to notify all of meetings and decisions, and failure to store information are often cited as reasons for projects failing or running over time. Traditional reporting tools use a simple passive reporting mechanism, which does not provide sufficient reporting support to a collaborative distributed system [Chen et al.2003]. Communication is, however, enhanced and supported by the use of a common repository. A paper-based repository has several disadvantages, such as retrieval delays, lost documentation and error-proneness, but most of all, may result in insufficient project documentation in the distributed environment. Another common problem in communication is that many project processes, contexts, rationales, or artefacts may not be captured at all. An electronic repository might overcome some of these disadvantages.

2.3 Software Agents in SPM

Software agent technology is being explored as a promising way to support and implement complex distributed systems. Using software agents for SPM processes is a new field of research and as such, literature in this regard is not commonly available. However, some work on using agents has been done to address at least some of the aspects of SPM. In this section, the authors briefly consider how agent technology is currently being deployed in SPM by considering some application examples.

The first application that we mention is intended for the broader project management environment, and is not specific to the SPM environment. Nevertheless, we refer to this example as it applies agent technology to *scheduling tasks*, which are common to both environments.

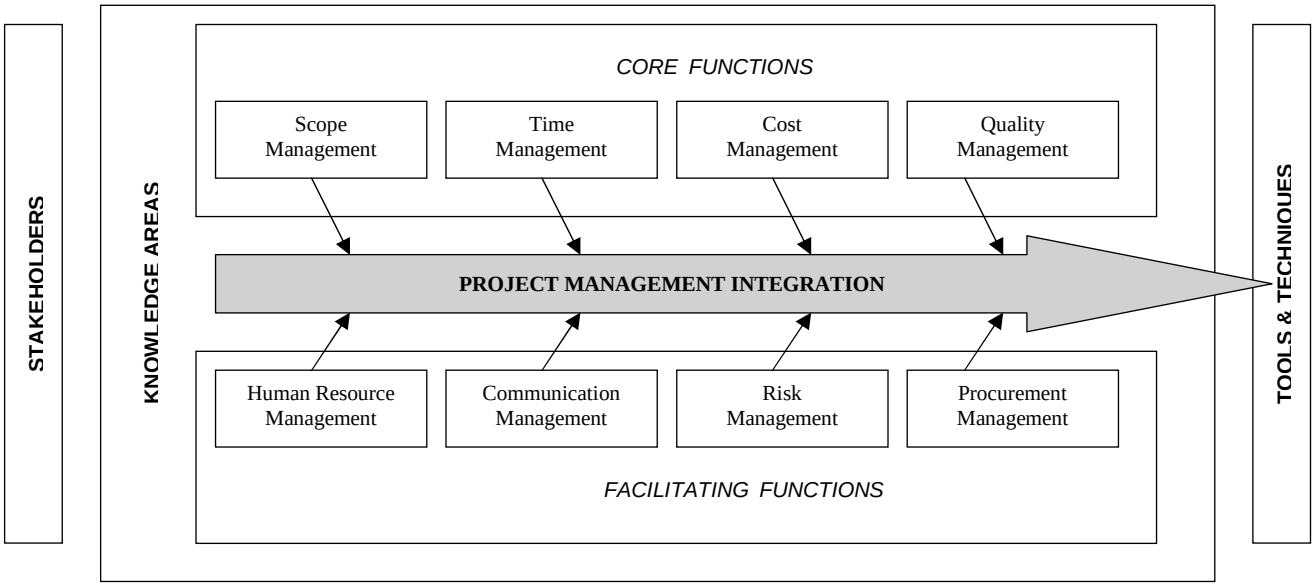

Figure 1: *Software Project Management Framework (adapted from Schwalbe).*

Furthermore, the authors find it worthy of mention it since it is focussed on in the distributed environment. In recent work, Sauer & Applerath [2003] presented an approach that involves using a generic agent framework to support the scheduling tasks within the supply chain in the PM environment. The framework allows for the consistent design of agents that reside on several levels of the organization. To prevent communication overhead (found in earlier multi-agent systems), agent *teams* are formed. All the agents in a team then collaborate to solve a specific scheduling task on a particular level. Furthermore, every agent (in its personal capacity) is also responsible for a specific schedule (the schedule of the resources that it represents). Therefore each agent is provided with the scheduling knowledge that is necessary to create or maintain the schedule without contacting the members of the team.

In another example, Maurer [1996] proposes a system (the *CoMo Kit*) in which methods and tools were developed to plan and manage complex workflows, especially in design domains. According to this system, tasks can be decomposed into subtasks and for every task, several alternative decompositions (methods) can be defined. Every task is associated with a set of agents, humans or computers, which are able to solve it. The problem-solving process, for example the application of methods to tasks, is distributed via a local area network. The proposed system uses agent technology as a tool for planning, coordinating and designing process execution. This approach follows a centralised black-box agent approach. The system architecture consist of a *modeller*, which does project planning; a *scheduler*, which supports project execution and manages information produced; and an *information assistant* that allows access to the current state of the project. During SPM, the *modeller* gathers information through interaction with the project manager or other stakeholders, and as a result presents a model of this information to the *scheduler* as input. The scheduler then manages agendas that contain the tasks to be carried out by an agent. To work on the task, the agent can access all relevant information (using the information assistant) for solving the problem.

3. SPM FRAMEWORK SUPPORTED BY SOFTWARE AGENT TECHNOLOGY

As described earlier, the software project management environment has changed in the past decade into a dynamic and complex environment where flexible and adaptive behaviour and management techniques are required. Agent-based solutions are exactly applicable to this environment since they are appropriate in highly dynamic, complex, centralised as well as distributed situations. In addition to the advantages of distributed and concurrent problem-solving, agent technology has the advantage of sophisticated patterns of interaction, namely cooperation, coordination and negotiation [Hall et al. 2003].

Before discussing our proposed SPM framework, we briefly reconsider the distinct knowledge areas and practices entailed in software project management (illustrated in Figure 1), to emphasise the focus of our work for this paper. The SPM management areas consist of four objective functions and four facilitator functions. The solution presented by Sauer and Applerath [2003] primarily focuses on the Time Management and certain aspects of the Communication Management functions. Maurer's solution [1996] is applicable to the Scope Management, Time Management and to a

certain extent the Communication Management functions. We believe that each of these key processes/functions could successfully be addressed by following a black box approach that is based on agent technology. Each black box consists of collaborative software agents ensuring cooperation, coordination and synergy between the different black boxes. Within such a black box a component-based development approach is followed. According to this approach, we use multiple (simple) agents, each with a particular objective, rather than fewer (complex) agents of which each has a long list of tasks to accomplish. For the purpose of this paper, we discuss our approach to only one of the SPM key processes, namely *Communications Management*, and describe the agent framework to accomplish the so called black box for this process.

Communications Management in a software project is an enabling and supporting action that ensures timely and appropriate generation, collection, dissemination, storage and disposition of project information [Schwalbe 2000]. Effective communication and sharing of information and knowledge among project contributors are needed. Schwalbe [2000] identifies five distinct functions associated with Communications Management, namely (1) communications planning; (2) information distribution; (3) performance reporting; (4) administrative closure; and (5) teamwork support. The *communications planning* function determines the *who*, *when* and *how* of the project, while the *information distribution* function entails disseminating information to keep all the stakeholders informed. *Performance reporting* alludes to the generation of reports such as status, progress and forecasting reports, while the *administrative closure* function involves project archiving and formal acceptance of reports. Finally the *teamwork support* function refers to the functions pertaining to collaborative project tasks, and hence includes the scheduling of meetings for these collaborative tasks. It therefore facilitates a collaborative working environment as well as document distribution.

To describe how software agents can be used to address the different functions of *Communications Management*, we use a set of *agent teams* to address the individual functions and then define specialised software agents operating within these teams, or on their own where applicable. In defining these specialised software agents, we find that it is less intricate to design the behaviour of each agent. Furthermore, the specialised agents also make it possible to describe the various interactions between different agents explicitly, which in turn reduces the general complexity of the agent system. The various programming patterns [Aridor and Lange 1998; Kendall et al. 2000; Tahara et al.1999] available, accomplish specific agent-associated tasks, such as creation, migration, suspension, and collaboration. The design of the overall system, based on components (specialised agents) simplify the design and programming of agents. The following specialised working agents are used:

- *Messaging agent:* an agent responsible for carrying messages between different agent teams. A messaging agent has strong collaborative characteristics and is by nature a mobile agent since the different agent teams may work in a distributed environment.
- *Personal assistant agent (PA agent)*: an agent that supports an individual stakeholder to accomplish his or her tasks by providing maximum assistance. This agent also has a collaborative nature, and relies on other agents to provide it with the information that it needs to sustain its owner. The PA agent is not computer-bound, but human-bound, as its stakeholders may be required to work on different computers when working in a distributed environment.
- *Task agent:* an agent that supports a specific project task. This agent collaborates with other objective and facilitator functions to support a specific task. This agent is commonly invoked by a PA agent to allow a stakeholder to work on a specific task, and is continuously monitored by a monitoring agent.
- *Monitoring agent:* an agent responsible for monitoring tasks, reporting back to the communications planning and *information distribution function*s where scheduling and rescheduling of tasks as well as the notification of stakeholders can take place. A monitoring agent is mobile, with intelligence, flexibility and strong collaborative properties.
- *Client agent:* a stationary agent responsible for a specialised task such as information retrieval or gathering. Client agents may or may not have intelligence, depending on their specific task, but must have a collaborative nature to interact with other agents in their agent team.
- *Team Manager agent:* an agent that is responsible for managing a team of agents, ensuring coordination between the sub-tasks of the different members of a team to accomplish the objective of the agent team.

Figure 2 illustrates the main operations in the Management Communications function and how agent teams cooperate to accomplish the objectives of these operations.

The software project manager, or other designated stakeholders, interacts with the communications planning function through a special user interface. This user interface, which sits on top of the *communications planning* function, uses *personal* agents, *task* agents and *messaging* agents. The interface assigns a *personal* agent to the user who has supervision rights over other *personal* agents. During interaction with the interface, the user defines team members or relevant stakeholders as well as the tasks that are assigned to them, and defines milestones, objectives, et cetera.

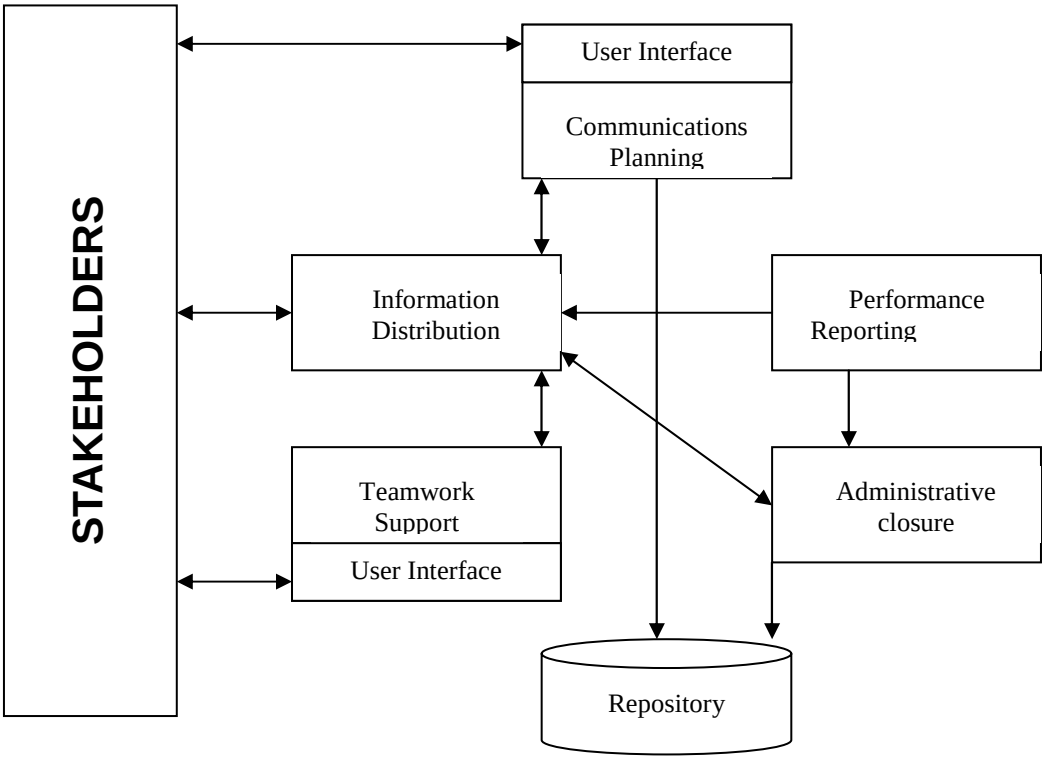

Figure 2: *SPM Communication Management function supported by software agent technology.*

The interface then assigns a *personal agent* to each person, to be invoked with a user name and password. (For simplicity's sake, the username and password could be the same as a person's network login Id and password, but the choice depends on the individual, or the manager, should he or she decide differently for the sake of security.) Required schedules and resources may be allocated at this stage or omitted if it is assumed that the *Time Management* agent system will do detailed scheduling. *Client* and *task* agents are used for automation where necessary, for example to do resource allocation, or calculations. Once the user has entered the required information into the system, *messaging* agents take the information to a central repository and the *information distribution function*

The *information distribution function* uses an agent team that consists of *messaging* agents, *task* agents, *client* agents and a *team manager* agent. The agent team of this function accepts incoming *messaging* agents from the user interface and uses its own *messaging* agents to interact with the stakeholders, the *teamwork support function* and the *administrative closure function*. It uses *client* agents to gather information from the incoming *messaging* agents and *task* agents to perform information integration and coordination.

In addition to the three primary intelligence properties, *client* (and *task*) agents at this level must also be adaptable in the sense that they remember specific properties of personal agents from previous work on the project, or even from previous projects and as a result, adjust their computing (based on a generic model) to integrate these characteristics. The above intelligence properties also imply flexibility. Developing these agents with the suggested intelligent properties is not a simple task, but since generic patterns exist for many of the other agents, more time can potentially be spent on this part of the development of the SPM tool.

As before, *task* agents are included for specialised computing tasks. For the *information distribution function*, task agents may or may not be included at this level, depending on the elaborativeness of the *client agents.* We advocate the use of task agents to simplify the design and improve the maintenance of the SPM tool software. As mentioned before, the *client* agent typically has a number of functions including interacting with (and thus receiving) incoming *messaging* agents, understanding (interpreting) incoming information, translating incoming information to a syntax that makes it processable, processing the incoming information, and deciding on distribution conduct (based on its generic approach to handling information as well as previous knowledge and experience). The *client* agent is also tasked with the responsibility to interact with the outgoing *messaging* agents, which must disseminate the processed information, and must also send the information to the *administrative closure* function that interacts with the central repository as well as with the *teamwork support function*. To simplify the design of a *client* agent, these individual tasks can be designed as *task* agents reporting to the *client* agent via *the team manager* agent.

The *performing reporting* agents, responsible for generating reports, commonly include client agents and messaging agents. Should this function use the same approach as advocated above, *task* agents are also to be included. A decision on whether or not to follow this approach depends on the complexity of the expected reports. The *messaging* agents interact with the *information distribution* function and *administrative closure* functions. In the first instance, mobile agents provide the reports to the *information distribution function* for dissemination, while they also present the report information to the *administrative closure* function for storing and archiving procedures, maintaining system persistence.

The *teamwork support* function is primarily responsible for collaborative scheduling tasks. Agents associated with scheduling are *monitoring* agents, *personal assistant* agents, *client* agents (and *task* agents where applicable), as well as *messaging* agents. *Messaging* agents are defined as before. *Monitoring* agents are responsible for monitoring the incoming messages from *messaging* agents in order to determine the necessity or urgency to suggest new or earlier meeting schedules than those already being scheduled during the previous communication rounds, or by the *teamwork support* function. The primary responsibilities of the *client* and *task* agents are to facilitate teamwork, perform scheduling task on teamwork, and distribute collaborative documents. When an individual team member works on a collaborative document, his or her *personal assistant* agent must be cognisant of any extraordinary circumstances when the user is falling behind schedule. This could for example be done by special-prompting-task-agents asking specific questions or *monitoring* agents comparing set dates to real dates. The *personal assistant* agent passes this information to the (manager) *monitoring* agent, which either sends the agent to the general *client* agent at this level, or makes special suggestions with regard to extraordinary meetings to be scheduled. A user interface is available at this level through which team members can interact with the collaborative task environment.

The *administrative closure* function interacts between the *performance reporting* function and the central repository. It also keeps a history through the use of *monitoring* agents to coordinate incoming reports before storing or archiving the information to the central repository. As expected, this function includes both *messaging* agents and *client* agents (potentially also *task* agents to assist the *client* agents) to coordinate the incoming reports and archiving processes.

4. CONCLUSIONS

The advances in computing technology have evolved over the past decade to a point where distributed computing has become the *de facto* working platform. This has changed the characteristics of SPM, and as a result, the traditional methods and techniques of SPM do not meet the new requirements posed by this new working platform. Software agent technology, although primarily applied to other fields, such as e-commerce, information retrieval and network management, is ideally suited to meeting the new challenges faced by the SPM characteristics such as appropriate tools for effective sharing of information and knowledge among project contributors, as well as efficient distributed task scheduling, tracking and monitoring mechanisms. In this paper we investigated an approach to using software agent technology to address these challenges. We also focussed on one of the key elements of SPM and designed a generic agent framework to address all the tasks of this key element. This framework forms a basis for other key elements, and could easily be adapted into individual frameworks and then coordinated by an overall multi-agent system to achieve the objectives of SPM. Our framework followed an approach of agent teams being composed of specialised software agents, each tasked with a manageable (atomic) task. This implied that the complexity of creating and maintaining tasks could be greatly reduced. Although we have not yet completed the programming of the proposed system, we believe that our solution is significant, based on our experience in other fields that advocate component-based development. We do, however, recognize the fact that programming of the model will have to be completed and the model thoroughly tested against other SPM tools before its true value will become apparent.

5. REFERENCES

ARIDOR, Y., AND LANGE, D.B. 1998. Agent Design Patterns: Elements of Agent Application Design. In *Proceedings of the 2nd International Conference on Autonomous Agents.* Minneapolis/St. Paul, USA. 108 - 115.

BALASUBRAMANIAN, S. BRENNAN, R. W. AND NORRIE, D. H. 2001. An Architecture for metamorphic control of holonic manufacturing systems. *Computers in Industry*. Vol 46, 1, 13-31.

CHANDRASHEKAR, S., MAYFIELD, B. AND SAMADZADEH, M. 1993. Towards automating software project management. *International Journal of Project Management.* Elsevier Science Ltd.

CHEN, F, NUNAMAKER, J. F., ROMANO, N. C. AND BRIGGS, R. O. 2003. A Collaborative Project Management Architecture. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.

CROFT, D.W. 1997 Intelligent Software Agents: Definitions and Applications. http://www.alumni.caltech.edu/~croft/research/agent/definition/

EARNSHAW, R. AND VINCE, J. (Eds) 2002. Intelligent agents for mobile and virtual media. Springer.

ELEC 4704 - Software Project Management. Department of Electrical and Information Engineering. University of Sydney. http://www.ee.usyd.edu.au/elec4704/lec-01.html Accessed May 2003.

GARDINER, A. 2002 Implementing PRINCE2 in a Business Change Environment. *Project Magazine*. Vol 4, 2

GELBARD, R. PLISKIN, N. AND SPIEGLER, I. 2002. Integrating systems analysis and project management tools. *International Journal of Project Management.* Elsevier Science Ltd.

HALL, G, GUO, Y. AND DAVIS R. A. 2003. Developing a Value-Based Decision-making Model for Inquiring Organizations. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.

HUGHES, B. AND COTTERELL, M. 2002. Software Project Management. Third Edition. McGraw-Hill.

IEEE STANDARDS BOARD. 1987. IEEE Standard for Software Project Management Plans. IEEE Std 1058.1-1987. 16pp. PDF: ISBN 0-7381-0409-4, SS12138.

KENDALL, E.A., KRISHNA, P.V., SURESH C.B. AND PATHAK, C.V. 2000. An Application Framework for Intelligent and Mobile Agents. *ACM Computing Surveys*. Vol 32. 1.

KRUPANSKY J.W. 2003. What is a Software Agent. http://agtivity.com/agdef.htm Accessed May 2003

LAZANSKY, J., STEPANKOVE, O. MARIK, V. AND PECHOUCHEK, M. 2000. Application of the multi-agent approach in production planning and modelling. *Engineering Applications of Artificial Intelligence.* Vol 13, 3, 369-376.

LETHBRIDGE, T. C. AND LAGANIERE, R. 2001. Object-oriented Software Engineering: Practical Software development using UML and Java. McGraw-Hill.

LINNHOFF-POPIEN, C. AND HEGERING, H. 2000. Trends in Distributed Systems: Towards a Universal Service market. Lecture Notes in Computer Science. *Proceedings of 3rd International IFIP/GI Working Conference*, USM 2000 Munich, Germany, Sept 2000.

MAURER, F. 1996. Project Coordination in Design Processes. *Proceedings of the 5th International Workshops for enabling Technologies*: Infrastructure for Collaborative Enterprises (WET ICE'96) IEEE

OGHMA: OPEN SOURCE. Types of Software Agents. http://www.oghma.org/. Accessed May 2003

PAI, W.C., WANG, C.C. AND JIANG, D.R. 2000. A software development model based on quality measurement. *Proceedings of the ICSA 13th International Conference*. Computer Applications in Industry and Engineering, 40-43.

PROJECT MANAGEMENT INSTITUTE (PMI) STANDARDS COMMITTEE. 2000. A Guide to the Project Management body of Knowledge (PMBOK).

SCHWALBE, K. 2000. Information Technology Project Management. Thompson learning.

SAUER, J AND APPELRATH, H. 2003. Scheduling the supply chain by teams of agents. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.

TAHARA, Y., OHSUGA, A. AND HONIDEN, S. 1999. Agent System Development Method Based on Agent Patterns. In *Proceedings of the 21st International Conference on Software Engineering*. Los Angeles, CA, USA.356 - 367.

THE STANDISH GROUP. http://www.standishgroup.com/

WOOLRIDGE M. 2001. An Introduction to MultiAgent Systems. John Wiley AND Sons. Chichester, UK.

WOOLDRIDGE, M. J., JENNINGS, N. R. 1999. Software engineering with agents: pitfalls and pratfalls. *IEEE Internet Computing*, 20 -27.

ZANONI, R. 2003 Project Management Model for physically distributed Software development Environment. *Proceedings of the 36th Hawaii International Conference on System Sciences.* Big Island, Hawaii.