
INTERNATIONAL JOURNAL OF COMPUTATIONAL MATHEMATICAL IDEAS
ISSN: 0974-8652 / http://satyanspublications.weebly.com/ijcmi-vol4-no3-2012.html
VOL 4-NO3-PP 1001-1006 (2012)

A Multi-Agent Architecture for Cooperative Software Engineering

[1]S.V.Achuta Rao, [2]Dr.Kiran Kumar Reddy, [3] K.Anand Kumar
[1] Research Scholar, Bhagawanth University,Ajmeir,RajahStan,India
achutaraosv@gmail.com
[2] Professor, CSE Department, at Krishna University,Machilipatnam
kirankreddi@gmail.com
[3] Assoc. Professor, vikas college of engineering & technology,nunna,vijaywada
anand_madhuk2007@yahoo.co.in

ABSTRACT

This paper looks at how Cooperative Software Engineering (CSE) can be supported. We first investigate the process aspects by presenting a traditional process architecture supporting CSE. Then we propose a multi-agent architecture for CSE, which is better in terms of simplicity and flexibility, and particularly useful in modeling and providing support to cooperative activities. We describe an industrial scenario of CSE, and show how to apply the proposed architecture to this scenario. The scenario is based on a software development and maintenance process for a Norwegian software company.

***Keywords*: Computer-Supported ooperative Work, Cooperative Software Engineering, Software Process Technology, Multi-Agent Systems**

1. INTRODUCTION

Most of the work in the software process community has been focusing on how to make people work together in an organized and planned way (partly pre-planned). For high-level processes with little details, it is likely that it is possible to make people work in this manner. However, the development of software involves people that cooperate to

Solve problems and to do actual work. These kinds of processes are very hard to support by traditional software process support tools, because the focus will be more at cooperative aspects than pure coordination of work. In this paper we introduce architecture to provide support for cooperative software engineering.

Computer-Supported Cooperative Work (CSCW)

It is a multidisciplinary research area focusing on effective methods of sharing information and coordinating activities. CSCW systems are often categorized according to the time/location matrix (synchronous/asynchronous and non-distributed/distributed). We may add an extra dimension to the CSCW typologies, considering different kinds of cooperative work in the order of increasing complexity of the process support they need.

Ad-hoc cooperative work such as brainstorming, cooperative learning, informal meetings, design work, etc.. Process modeling support here is implemented through awareness triggers.

Predefined/strict workflow, in traditional Office Automation style represented by simple document/ process flow

Coordinated workflow, such as traditional centralized software maintenance work onsisting of check-out, data-processing, check-in, and merge

steps. There exist several systems supporting coordinated workflow (mostly prototypes).

Cooperative workflow, such as decentralized software development and maintenance work conducted in distributed organization or across organizations. Here the shared workspace and the cooperation planning are the main extra factors from the process point of view. Example of a system supporting distributed organizations and processes is Oz [3]. By Cooperative Software Engineering (CSE) we mean large-scale software development and maintenance work which falls into the last two categories in the above list. Because of the rapid spread of World Wide Web as the standard underlying platform for CSCW systems, more software companies are moving from the traditional centralized working style to the decentralized one. In the decentralized

CSE, communication, negotiation, coordination and collaboration among the various participants are more complicated, because people are not only geographically distributed, but may also work on different platforms, at different times, with different process models. A better understanding about CSE processes is needed as well as a full range tool support of such processes. The research in this area will help the software industry to change the work style to take full advantage of WWW, and will enrich the research area of Software Process Technology (SPT) in which so far a centralized work style has been often assumed implicitly. Compared with the traditional architecture, an agent-based architecture is advantageous in terms of simplicity and flexibility, and particularly useful in modeling and providing support to cooperative activities [5]. In this paper, we try to integrate the areas of CSCW and SPT in a multi-agent architecture. First we investigate the process aspects of CSE by presenting a traditional process architecture supporting CSE. Then we propose a multi-agent architecture for CSE, which is an extension and specialization of the more general architecture for all CSCW. This architecture will then be applied to an industrial CSE scenario.

2. A TRADITIONAL PROCESS ARCHITECTURE SUPPORTING CSE

The key issues of CSE are group awareness, concurrency control, communication and coordination within the group, shared information space and the support of a heterogeneous, open environment which integrates existing, single user applications. All these are related to the software process. Within the SPT community there has been research on each of the issues, but from a slightly different point of view. To see how CSE is supported by a traditional architecture, we present the process support in such a architecture. This architecture is usually realized by a **Process-sensitive Software Engineering Environment, a PSEE**, with a spectrum of functionalities and associated **process tools**. The support is needed in three main areas; at the Process Modeling template level (defining process in PML), at the Instance level (adding detail to process model), and for Enactment and monitoring. To full fill the goal as a process support architecture for CSE, some underlying components are needed. These components make it possible to apply the architecture in a distributed, heterogeneous environment. First, a portable platform infrastructure for PSEE-based client tools are needed (candidates are HTML/CGI and Java). Second, the architecture must offer an integrated environment for tool operation and communication. Third, we need facilities for distribution of tools and workspaces (candidates are CORBA or DCOM). Fourth, we need a community memory or experience base to store template process models (a candidate is Experience

Factory [1]). Figure 1 presents a general PSEE architecture for CSE. Its cooperative support is provided through a shared workspace where files and

parts of the process model are

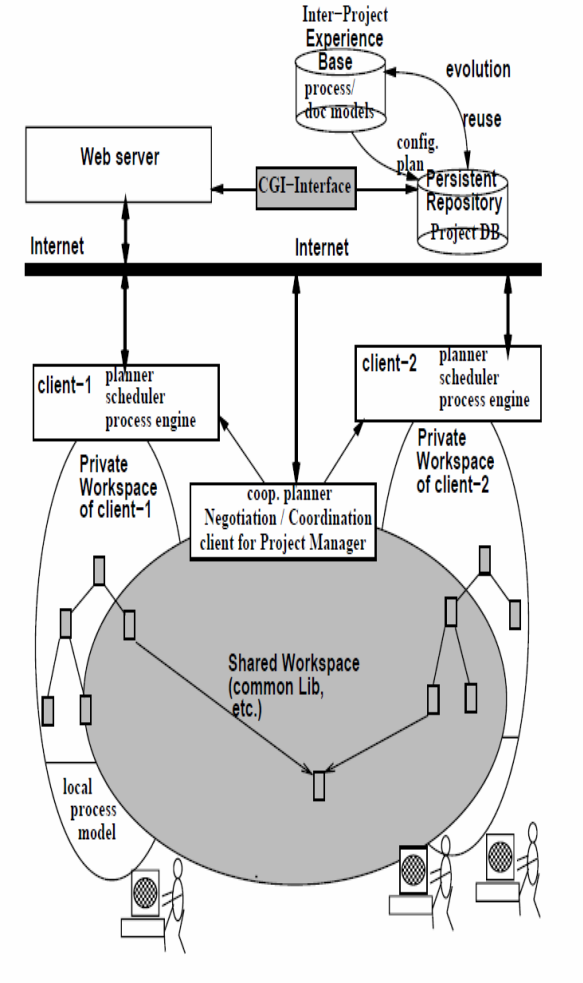

Fig1. A general process architecture supporting CSE

stored and shared. The private workspaces are provided with tools for planning, scheduling and inaction of the process model. The shared workspace provides support for cooperative planning and negotiation, and for coordination through cooperative protocols. The shared workspace is managed by a project manager. The architecture is web based, and repository and experience base support is provided through a web-server and a CGI-interface to the repositories. There are, however, several problems with such PSEE/CSE architecture. First, it is too centralized and has too much flavor of a centralized database surrounded by a fixed number of applications. Second, too homogeneous models are used assuming one common PML. This means that one PML must be used by all involved partners, although this is no necessarily the best solution. Third, it is hard to change process tools and models. Due to the distributed and open setting, we should allow dynamic reconfiguration of process models, as well as for process tools. Forth, a open-ended spectrum of process tools may be needed to offer better support for cooperative work, than what classical PSEE architecture can offer. As will be seen in the next section, a multi-agent architecture seems more appropriate for a general CSE.

3 MULTI-AGENT ARCHITECTURE FOR CSE

The previous section shows how complex a CSE environment could be. Similar situations exist in other areas such as Distributed Artificial Intelligence, Business Process Management, and Electronic Commerce. Nowadays, it is believed that the Multi-Agent Systems (MAS) are a better way to model and support these distributed, open-ended systems and environments. A MAS is a loosely-coupled network of problem solvers (agents) that work together to solve a given problem. The main advantages of MAS are:

Decentralization: being able to break down a complex system into a set of decentralized, cooperative subsystems. In addition, many groups of organizations are inherently distributed.

Reuse of previous components/subsystems: That is, building a new and possibly larger system by interconnection and interoperation of existing (sub) systems, even though they are highly heterogeneous. Thus, we do not request a common PML, so different PMLs can be used in different subsystems.

Cooperative Work Support: being able to better model and support the spectrum of interactions in cooperative work, since software agents can act as interactive, autonomous representatives of humans.

Flexibility: being able to cope with the characteristic features of a distributed environment such as CSE, namely incomplete specification, constant evolution, and open-endless. In the remainder of the paper we try to model the problem area CSE as a MAS consisting of four components

3.1 Agents

In this paper, an **agent** is a piece of autonomous software created by and acting on behalf of a user (or some other agent). It is set up to achieve a modest goal, with the characteristics of autonomy, interaction, reactivity to environment, as well as pro-activeness. The whole process (and meta-processes) of CSE is carried out by groups of people, using

tools, such as production tools, process tools, and communication tools. Each participant can create a set of software agents to assist him/her in some particular aspects. There are also some system agents created by default for the administrative purpose in the architecture. We can perceive the following types of agents:

System agents: These cover default agents for the administration of the multi-agent architecture, such as creation and deletion of agoras. In some cases more specific system agents are needed. Monitor agent's track events in workspaces and agoras in order to collect relevant measurements according to predefined metrics. Repository agents can provide intelligent help for searching for information.

Local agents: To assist in work within local workspaces. These agents act as personal secretaries dealing with local process matters such as production activities as well as to define, plan, and enact process models.

Interaction agents: To help participants in their cooperative work. Such agents can be viewed as shared process agents, and they include four subclasses.

Communication agents are used to support a spectrum of more high-level communication facilities. All information flow, also simple communication, uses agents as the underlying communication mechanism to make the architecture clean and simple. Negotiation agents verbalize their demands (possibly contradictory) to move towards an agreement through the process of joint decision making. Coordination agent's support, e.g., a project manager issuing a work-order that involves a group of developers; or a higher-level manager being called in to mediate between negotiating agents to reach an agreement. Mediation agents are used to help negotiating agents reach an agreement. In doing so, mediation agents may consult the Experience Base, act according to company policies, or ask a project manager (human) for help to make decisions.

3.2 Workspaces (WS)

A **workspace** is a place where human and software agents access shared data (usually files) and tools which can be private or shared by a group of people. In addition, interaction between users and software agents takes place in workspaces. The simplest form of a workspace can be a file-system provided with services to read and write files. A more advanced workspace can provide file versioning, access to of some repository, awareness services, web support etc. BSCW [4] is one example of an advanced web based workspace implementation. Agents can access data in the workspace either directly or indirectly through tools.

3.3 Agoras

An **agora** is a place where software agents meet and interact, but can also be a market place where agents "trade" information and services. Agoras should provide agents with more intelligent means to facilitate their interaction. The main purpose of the agora is to facilitate cooperative support for applications and agents. Below we propose the following preliminary functionalities that any agora should support.

1. Inter-Agent Communication:

This is not simple information-passing; it rather conveys intentions, goals, beliefs, and other mental states to form the foundation of negotiation and other complex interactions. An agora should facilitate agents to announce their capabilities and to get in touch with agents capable of doing specific tasks. The following services are needed to facilitate inter-agent communication:

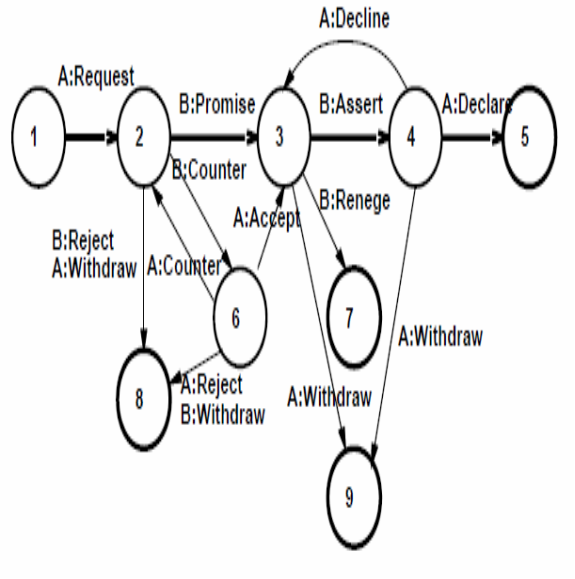

Fig2: an example of speech art

An agora should provide a predefined set of speech-acts (conversation types), such as proposal, counter-proposal, acceptance, rejection, confirm, deny, inform etc. The various speech-acts types will define how agents can interact with each other. In many cases a speech act is represented as a state

transition diagram as the one shown in figure 2. The speech-acts act as shared process model for how agents should interact. Figure 2 shows states of a conversation between two agents A and B. States are shown as circles while transition between one states to another is shown as arcs. The terminal state 5 indicate a successful conversation and the bold lines shows the path of a successful conversation.

An agora should specify a common syntax for messages transmitted through the agora, so that the recipient can analyze the contents of a message. An agora should specify a common semantic of an agent language. One part of this semantic is defined through speech-acts, i.e. what state transitions of a conversation that a message will cause. The semantic also ensures that software agents interpret the same words similarly.

An agora should specify pragmatics for agents. This means that agents shall not lie to other

2. Inter-Agent Negotiation:

The progress of a negotiation depends mainly on the negotiation strategies employed by the agents involved, but agoras should provide mechanisms to minimize communication overheads.

An industrial scenario

This scenario is based on the software development and software maintenance process from a real Norwegian software company, in this paper called Acme Soft. The company's products exist on various operating system platforms, including Microsoft Windows NT and various UNIX platforms. In this scenario we by software development mean the development of future releases and updates of products, whereas by software maintenance we refer to the correction of defects in released software. Common to these processes are a production and testing process which builds the products for requested platforms and the delivery process which creates the distribution media and ships products. An overview of the main activities in the scenario process is shown is figure 3. Corresponding responsible groups are listed below the activity name. The **Development process** focuses on work that is directly related to changes of software products and the planning and scheduling of these changes. The three main process steps are: 1) Release and update planning, 2) Scheduling, and 3) Implementation.

The **Maintenance process** is triggered by a one of the following maintenance reports; Software Query Reports (SQRs): Error report or desired, Release Problem Reports

(RPRs): Internal problem reports, or Production orders:

Requests from customers for a given product or product revision. The maintenance agreements define priorities system for SQRs and RPRs, with five levels from Critical down to May not be implemented named P0 to P5. Based on this classification, the correction phase of SQRs and RPRs is divided into the five following process steps: 1) Registration (by the development department),

2) Estimation of resources (which developer, effort),

3) Send out (send SQR/RPR to developer),

4) Correction (actual problem fix done by developer), and

5) Module testing (by developer). The **Production and testing process** starts after a freeze in development code or after defect corrections, or when customers request a delivery revision built for a specific platform. The process consists of the three following steps:

1) Production, 2) Testing, and 3) Verification the **Delivery process** consists of activities to store products on distribution media and to ship products. This process is initiated when a product release or update is made available, and on customer demand. The delivery process can be divided into two main activities: 1) Delivery and 2) Shipping.

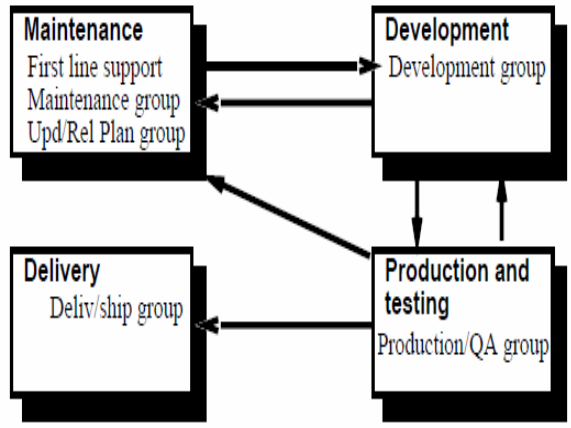

Fig3: scenario process

4. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced an agent-based architecture to solve CSE problems. This architecture

consists of four main components: Agents, Workspaces, Agoras and Repositories Agents provide flexible and dynamic support to cooperating users, as well as help for doing every-day work. Agents can easily respond to a changing environment (learn; adopt based on experiences in the Experience-Base etc.). It is widely accepted that real software processes evolve over time, so our process support must adopt and cope with such changes. To enable agents to cope with process changes, they will need to learn from prior experiences. In our architecture this is introduced through repositories (Experience Bases) as well as agents can learn on their own. Agoras and workspaces are introduced to support agent interaction and grouping of agents, respectively. Our architecture has been applied on one specific scenario.

We believe, however, that our multi-agent CSE architecture is applicable on various situations, processes and organizations. One concrete example is to support meta-process activities, such as discovering/planning process models, negotiation about the process model and the real-world model and assignment of resources to a instantiated process model. Our architecture's main contribution is to give process support where traditional SPTs often fail in respect changing environment and unexpected events. Further work with describe formalities, implementation of prototypes, and experiment with more industrial scenarios will show if this is the case. Another outcome of our research will be to identify disadvantages of MAS.

5. REFERENCES

[1] Victor R. Basili, G. Caldiera, Frank McGarry, R. Pajerski, G. Page, and S. Waligora. The Software Engineering Laboratory – an Operational Software Experience Factory. In Proc. 14th Int'l Conference on Software Engineering, Melbourne, Australia, pages 370– 381, May 1992.

[2] Israel Ben-Shaul and Gail E. Kaiser. A paradigm for decentralized process modeling. Kluwer Academic Publishers, Boston/Dordrecht/London, 1995.

[3] Israel Ben-Shaul and Gail E. Kaiser. A Paradigm for Decentralized Process Modeling. Kluwer Academic Publishers, Boston/Dordrecht/London, 1st edition, 1995.

[4] R. Bentley, T. Horstman, and J. Trevor. The World Wide Web as enabling technology for SCW: The case of BSCW. Computer Supported Cooperative Work: The Journal of Collaborative Computing, 7:21, 1997.

[5] Anthony Chavez and Pattie Maes. Kasbah: An Agent Marketplace for Buying and Selling Goods. In Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi- Agent Technology, London, UK, April 1996.