# CSCW in Software Development: Collaboration among Humans and Artificial Agents through Dialogs

Milton Pires Ramos [1], Cesar Augusto Tacla [2], Gilson Yukio Sato [2], Emerson Cabrera Paraiso [3], Jean-Paul A. Barthès [4]

[1] *TECPAR - Paraná Institute of Technology, Artificial Intelligence Division,Curitiba, Brazil*

[2] *UTFPR - Technological Federal University of Paraná, CPGEI,Curitiba, Brazil*

[3] *PUCPR - Pontifical Catholic University of Paraná, PPGIa, Curitiba, Brazil*

[4] *UMR CNRS Heudiasyc, Université de Technologie de Compiègne, Compiègne, France*

*milton.ramos@tecpar.br, {tacla, sato}@utfpr.edu.br, paraiso@ppgia.pucpr.br, barthes@utc.fr*

***Abstract***

*In this paper, we discuss the construction of dialogs for Personal Assistant Agents that are in charge of the interface between users and a Multi-Agent System. Such a system aims at providing support for small teams developing software collaboratively. Small teams have specific needs such as the integration of free or open-source tools or the support to elaborate project documentation. Considering such specific needs, we have elaborated an architecture implemented using a Multi-Agent platform. We present the structure offered by the platform for handling dialogs with users and discuss some implementation details. We also give examples of the dialogs that represent interactions between members of small software development teams and their Personal Assistant Agents. We consider that the use of Personal Assistant Agents can help small teams handle documentation issues in an integrated and undemanding way.*

***Keywords:*** *Multi-Agent Systems, Cognitive Agents, Collaborative Software Development, Small Teams, Dialog Management*

## 1. Introduction

Research efforts in Computer Supported Cooperative Work (CSCW) applied to the Software Development (SD) domain mostly address the needs of large distributed teams. Hence, their focus is mainly on infra-structure and environment for enhancing collaboration by improving communication and awareness. Small collocated teams are seldom taken into account. Because small teams may include fewer than ten members, time-consuming routine activities could hinder the software development initiative.

Moreover, the research in CSCW usually gives priority to supporting the software development team (e.g. [6] , [7] [9] [18] and [22] ), but issues concerning coordination, time scheduling control, and documentation are often neglected.

The specific needs of small teams are also disregarded. Constrained budgets make small teams work with tools that are less expensive (e.g. freeware or open-source) and less integrated or stand-alone. Thus solutions integrating (or encapsulating) such tools are desirable.

In order to investigate issues related to the use of CSCW systems to support small teams developing software, we have started the CSCW-SD (Computer Supported Collaborative Work in Software Development) project. The ultimate goal of the project is to improve the productivity of small teams developing software by means of a collaborative work platform.

One of the first activities in the CSCW-SD project was to propose the architecture of a system to support small collocated teams developing software. The architecture aims at integrating different tools that are currently used in this team [4] . One of the most significant aspects of the integration is a unified interface that allows its users to interact with different tools using the very same interface. In the proposed architecture, an agent (the Personal Assistant) implements this interface. The Personal Assistant can interact with its user using several mechanisms like point-and-click or menu-driven interfaces, but more efficient interfaces can be envisaged.

In this paper, we aim at presenting a dialog mechanism allowing users to interact with the tools using commands and demands typed in natural language. The dialog mechanism between users and their agents augments cooperation in the project, because the agents work in a proactive way looking for information about the projects under developed and ongoing tasks, fulfilling some documentation, instigating collaboration between the team members and facilitating the reuse of experiences or code. We also discuss the use of CSCW systems to support software development and how such systems neglect the specific requirements of small teams. Then, we present the architecture of the multi-agent system we have been developing to support small teams. The architecture is implemented using a multi-agent platform that provides a dialog mechanism that we describe here.

## 2. CSCWSD for Small Teams

In this section, we discuss how CSCW systems are used to support software development (SD) and how specific requirements for small teams are often neglected. Most groupware approaches for supporting SD focalize onto communication and awareness issues. However, small teams usually do not experience this kinds of problem, facing instead, problems concerning documentation and division of work.

### 2.1 CSCW in Software Design

Groupware systems have been used to support Software Design (SD) by providing features to facilitate communication and awareness between members of distributed teams.

Cook and Churcher in [6] present a collaborative software engineering architecture, called CAISE, that provides extensible real-time support for collaboration between developers and tools. Such a support is constituted by features to provide software developers with awareness of other developers' locations, to alert developers regarding potential conflicts and the need for close collaboration.

TagSEA [22] is a collaborative tool to support asynchronous software development. The support is provided by a lightweight source code annotation tool that enhances navigation,

coordination, and capture of knowledge relevant to a software development team. TagSEA employs annotations written by developers as comments embedded in the code to provide landmarks for readers in order to facilitate navigation and coordination. The system also allows developers to share source code as text instead of requiring developers to copy-paste snippets or make external references to the code being discussed.

Palantir [18] is a workspace awareness tool that complements existing configuration management systems by providing developers with insight into other workspaces. The tool informs a developer about changes other developers have done on artifacts, calculates a measure of severity of those changes, and graphically displays the information.

Fitzpatrick et al. [7] describe a system that integrates an event notification system, a chat and a tickertape tool. The idea is to integrate formal (CVS notification) and informal (chat) communication. The event notification system receives messages from the CVS and presents them in the tickertape tool. Using the chat, developers can discuss the message from de CVS.

The systems we just described, illustrate the concern of the research in CSCW in supporting the activities of large distributed teams. Although comprehensive, such a concern neglects or lets relatively unexplored a set of other concerns that we consider relevant to the research domain. One concern is the lack of tools for supporting project managers. Most researches focus on supporting the development staff and neglect managerial issues like coordination, time scheduling control, and documentation. Another concern refers to the fact that most researches rely on suggesting their own tools, forcing new users to abandon their current tools. For small teams, sometimes constrained by their small budgets, less expensive solutions are more appropriate. Solutions integrating (or encapsulating) their current tools (in general open source ones) should be a priority. The last concern is that the particularities of small collocated teams are not taken into account. Such teams have specific needs that have less importance or even do not exist in larger teams.

### 2.2 Requirements for Small Teams in SD

Large distributed teams and small collocated teams in SD usually share some problems, but they also present differences regarding, for example, the availability of resources and the division of work. Analyzing such differences, it is possible to define some requirements that are particular for designing tools to support small teams in SD.

- Members of collocated teams rely mostly in face-to-face communication; hence communication tools are less necessary in such teams.
- Although playing the same roles that members of large teams play (e.g. programmers, testers, analysts), members of small teams usually have to play several roles simultaneously during the software development life-cycle.
- As members of small teams play multiple roles, they tend to be overworked. In such circumstances, some tasks that are perceived as second class tasks, such as documentation and management of source code are often neglected.
- Small teams usually have a limited budget, thus they would prefer to use less expensive (e.g. freeware or open-source) and less integrated or stand-alone tools.

Among all such requirements, we focused our work in the integration of tools used by a small team to support the development of its activities. As we consider that one of the most significant aspects of such integration is a unified interface to the user, we decide to explore

this point. Our approach involves the use of a Personal Assistant equipped with a mechanism to dialog with its user.

## 3. The CSCW-SD Architecture

In order to contextualize the discussion, we present the architecture we proposed to support small collocated teams developing software [4] . We called this architecture CSCW-SD.

The CSCW-SD architecture has been applied to a small SD team composed of five engineers and three to five other external collaborators (depending on the project). The engineers work together in the same office, but external collaborators can work in different offices, organizations or cities.

To facilitate project management, the team uses two web-based applications. The first application is used as an environment to support collaborative work - the Project Management Tool (Figure 1). One of its functions is to support document elaboration and management by keeping meeting minutes or allowing the collaborative elaboration of the project glossary of terms. Another function of this environment is to support project management. It allows task allocation and management, meeting arrangement, and automatic notifications about these events.

The second application is a Code Version Control (CVC) system. It is a system for collaborative version control of software source-code that allows tracking modifications in software code.

The CSCW-SD architecture aims at integrating both tools. One of the major aspects of such integration is the unified interface for both applications mediated by an agent. Another aspect is the encapsulation of the applications used by the team. This architecture is based on a Multi-Agent System (MAS).

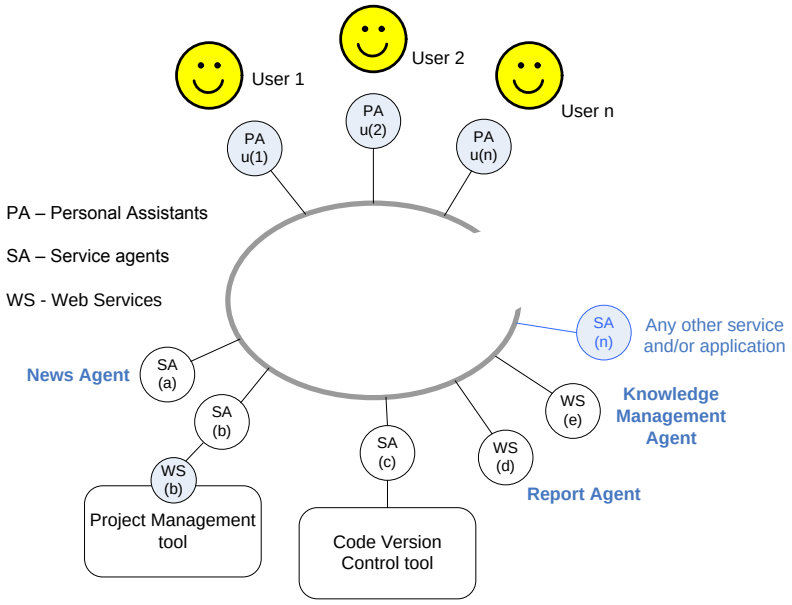


**Figure 1. The CSCW-SD Proposed Architecture (updated/adapted from [4] )**

MAS have been used to improve groupware tools ([21] [25] ) by (i) improving the exchange of information among participants and the control over workflows and procedures; (ii) providing support and convenient user interfaces to CSCW systems [17] .

In this research, we use a Multi-Agent platform called OMAS – Open Multi-Agent Systems [2] that lets us implement two types of agents: Service Agents (SA) and Personal Assistant Agents (PA). The former provide a particular type of service corresponding to a set of specific skills, and the latter is in charge of interfacing humans to the system.

In our approach, all users have their own PA, whatever their role in the community (managers, technicians, secretaries and so on). A PA has a crucial function, being dedicated to:

- understanding the needs of its master (i.e. the user owning the PA) needs;
- acting pro-actively to anticipate the master's needs;
- mobilizing SAs to execute a command or demand from its master;
- integrating information from different sources to solve a problem;
- presenting the information intelligently and in a timely manner;
- mediating all information exchanges among team members (who are considered information sources);
- organizing the documentation of its master with the help of a SA;
- capturing and representing the team members' operations, helping them in the process of preserving and creating knowledge [17] .

More specifically, a PA can help its master proactively by looking for information about the projects under developed, ongoing tasks, fulfilling some documentation, instigating collaboration between the team members and facilitating the reuse of experiences or code.

In our approach, SAs encapsulate the applications that support the development team and provide specific services like checking for modifications in the contents of the applications (checking for a new code version) or updating documents (extracting information from minutes into a report).

The implementation of the CSCW-SD platform can help the team keep the required documentation updated, ensuring software quality; improve the cooperation inside the team, and induce knowledge management by the possibility of reusing experiences or code.

Using a unified interface to interact with the system calls for the use of open dialogs, i.e. one cannot use traditional point-and-click features. The next section describes two approaches for the PA's interface.

## 4. The PA's Interface

In past projects, we developed two different approaches for the PA's interface. The first one uses an ontology-based speech interface, called SpeechPA, for controlling the interface and negotiating information with the user [17] .

In a MAS environment, ontologies have been used to help interpreting the context of messages sent by other agents, and to keep a computational representation of knowledge useful at inference time. The ontologies however, may also be used for facilitating the interaction between user and PA. The information they contain is a source of knowledge for conversational interfaces, enhancing the quality of the assistance the PA can offer. In SpeechPA, we propose an ontology-based speech interface for PAs. Our main goal is to reduce the user's cognitive load while improving the quality of assistance.

Traditionally, developers propose graphical-oriented interfaces involving the use of menus, sub-menus, dialogue-boxes, and so on. Often this approach is inappropriate or at least not very appealing, leading to an assistance of poor quality. An attractive solution is the conversational interface. Conversational interfaces as defined by Kölzer [11] , let users state what they want in their own terms, just as they would do speaking to another person. Conversational interfaces aim at supporting large-vocabulary spontaneous spoken language exchanges in a fluid dialogue between user and agent, according to Oviatt and Adams [16] and Komatsu and Morikawa [12] . Of course, the interaction is more complex, but the complexity is handled by the system. Conversational interfaces let users concentrate on their main activity and, once in a while, let them exchange spoken words with the PA.

When one thinks about a speech-based interface, speech recognition immediately comes to mind. The speech recognition technology has advanced quickly in the last decade and is now used routinely in commercial projects (see [13] for further details). Speech recognition is an extremely complex process, quite error prone, and cannot be implemented today without a great deal of knowledge about what the utterances are likely to be.

The results from an automatic speech recognition engine may be quite far from what the user actually said; the differences may be lexically and syntactically significant. Also, as stated by Jurafski and Martin [10] , people address computers differently than they address other humans, trying to adapt to what they perceive as limitations of the machine. Regarding the last point, we believe that if we could make the computer interact with people in a more realistic way, we could reduce their cognitive load.

In order to reduce the interaction and, consequently waste less time, we limited the space of dialogue utterances to directives speech act classes [19] —inform, request, or answer— since they define the type of expected utterances in a master-slave relationship. This strategy reduces the number of turn-takings since some speech acts will not be used by the PA, example of which are acknowledgement acts ("Thank you" or "Have a nice trip"). As shown by Flammia [8] , the latter may represent up to 30% of turn-takings.

We also defined a policy of presentation, filtering and delaying messages to be presented to the user. Less important messages (such as "The email was sent" or "The printer is out of service") are kept in a repository and are not systematically presented. The SpeechPA interface is shown Figure 2.

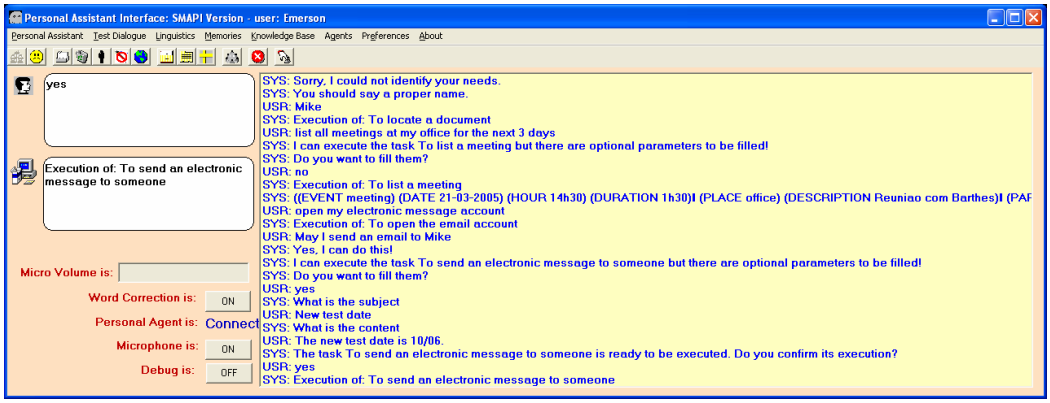


**Figure 2. Snapshot of SpeechPA**

Figure 3 shows the major components of our architecture. It has three parts: (i) graphical and speech user interface (GSUI) modules; (ii) linguistic modules; and (iii) agency modules.

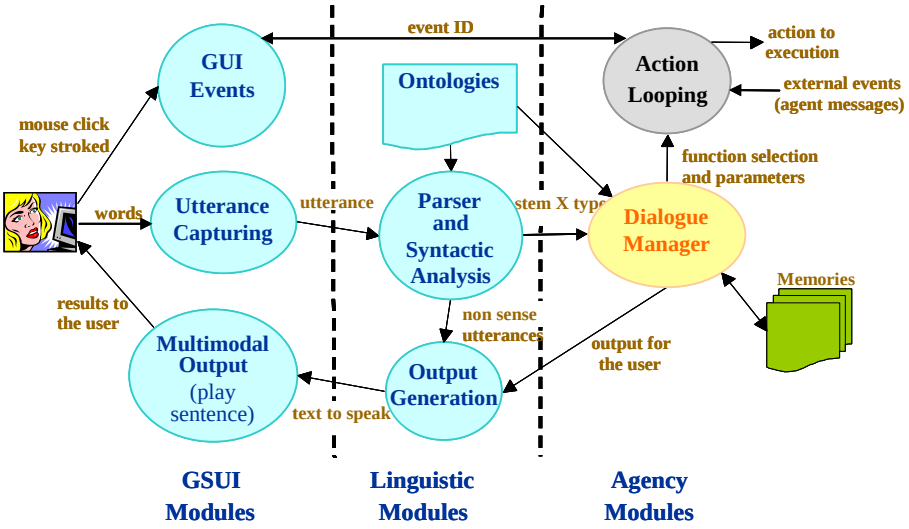


**Figure 3. Interface Architecture Overview**

The process starts by capturing the utterances using a commercial automatic speech recognition engine that returns the recognized result for each word. The Utterance Capturing module concatenates all the words forming an utterance. A process running independently analyzes each utterance. Due to local noise interference or bad pronunciation, the utterance may be lexically and/or syntactically different from the words actually said. Initially, we are using the utterance as it is, extracting a list of known disfluencies.

Interpreting an utterance is done in two steps: (i) parsing and syntactic analysis; and (ii) ontology application. The results are sent to the dialogue manager continuously, or back to the user when they do not make sense.

The parsing algorithm works top-down. It replaces each utterance stem with its syntactic category (verb, noun, adverb, etc) with the help of a lexicon file and a set of grammar rules. The grammar rules were divided in order to classify an utterance into one of three categories: order, question or answer. If a sentence is not well formed, or if it is out of the domain, then it is classified as a nonsensical utterance. In such a case, the user is invited to reformulate his sentence. Nonsensical utterances occur rarely since our system tries to act with minimal information, but nevertheless may occur.

The mixed-initiative and task-oriented dialogue mechanism is coordinated by the Dialogue Manager module. Each dialogue session is conducted as a task with sub-tasks and can handle several tasks simultaneously.

When the user requests an action, the Dialogue Manager tries to execute it, creating a task that is dispatched by the Action Looping module. However, if the initial utterance lacks crucial information—e.g., an action parameter—the Dialogue Manager starts sub-tasks to complete the action list, asking additional information from the user. To do that, it uses an action library. Once the action list is complete, the PA executes it, with eventual support from other agents.

Although talking is a privileged mode, the agent interface is multimodal. Thus, the Action Looping is responsible for merging all modalities (e.g., button click and speech).

The second approach for the PA's interface uses conversations graphs and is the one used in the CSCW-SD project. So, the next section describes how PA dialogs are handled in the OMAS platform.

## 5. OMAS and Dialog Structure

The Personal Assistant Agent (PA) paradigm is a special and distinguished characteristic of the OMAS platform [2] . This type of agent is built to be a real assistant or surrogate of his master, or, as proposed by Negroponte [15] , to act as a "digital butler." It means that we expect a specific relationship between the PA and his master (the user). Part of the relationship is established by the dialog capabilities available from the platform, coupled with the construction of a particular ontology. The user and her PA use a practical dialogue—which means that they are pursuing specific goals or solving tasks cooperatively—as defined by Allen et al. 0. The rest is developed in the internal PA structures.

All OMAS dialogs are implemented using conversation graphs (Figure 4) that model the possible states of the dialog (S0 to S3) and use rules (boxes named rules) at each state to analyze the situation (boxes named facts) and trigger transitions (arrows) until an action state (S4) is reached (a known task/action is found) or the dialog is abandoned.

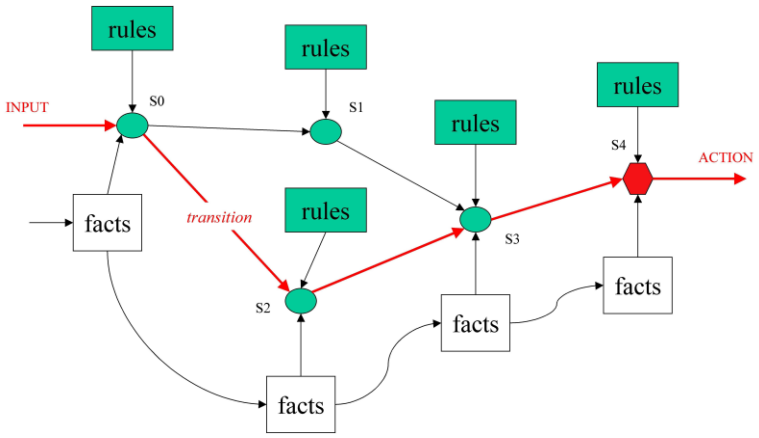


**Figure 4: Dialog Mechanism using Conversation Graphs (information states) [5]**

The top-level dialog has two parts, a very general loop, called Main Conversation (MC), and a sub-graph ranking the possible tasks to be executed (Figure 5).

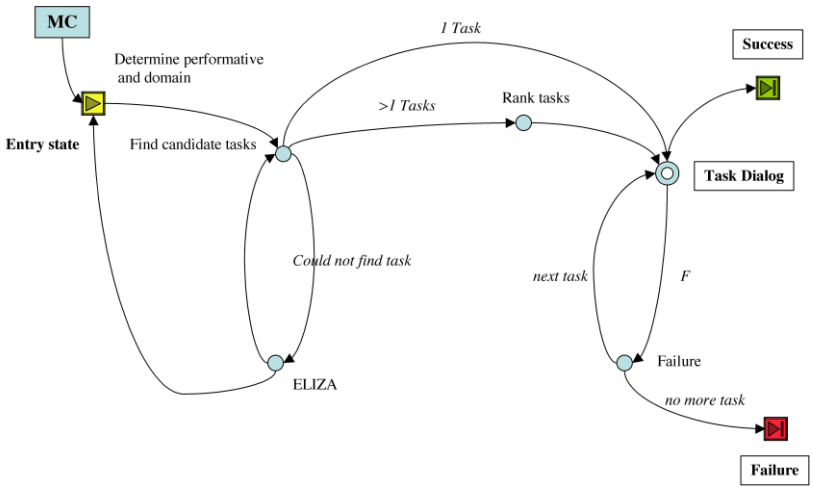


**Figure 5: OMAS Main Conversation (MC) Process Sub-dialog**

When the PA cannot make sense of the master's request it enters a state of small talk, Eliza style [26] , to avoid drab negative answers.

In order to use this powerful mechanism already available in the OMAS platform, one needs to implement the master/PA dialogs for a specific situation. To implement an application, at first, it is necessary to design the dialog models.

## 6. CSCW-SD Dialog Structure

As presented by Campagnolo et al. [4] , the initial objective of the CSCW-SD project is to support the documentation activities of small teams developing software, and in particular, the construction and management of the Small Project Management Plan (SPMP). SPMP is an alternative model for software project management proposed by Land and Waltz [14] , compliant with the main software quality standards. It organizes the documentation of a software project and provides a complete check list of all documents and registers that must be stored and maintained in order to ensure the quality of a software project. The focus here is on small software development teams in which documentation tasks are also important, but resource consuming.

Some use cases involving the execution of tasks in CSCW-SD are shown thereafter, describing the master's intention, the ways to express it, the available services, pre-established tasks, and finally the correspondence between the ways to express the master's intentions and the established tasks.

Use Case 1 – Collaborative construction of the SPMP (for any member of the project team)

**master intention:**
to collaborate with other team members in the construction and maintenance of the SPMP, writing parts of the document.
**ways to express:**
- add this text to the project X SPMP, in Section Y;
- this text could be important in Section Y of project X SPMP;
- put this in the project X SPMP.

**pre-established tasks:**
A Service Agent drives the Project Management Tool, via WebServices (REST), with the tasks of writing, searching, verifying recent contributions, statistics of participation in a SPMP project.
**correspondence:**
The PA has to interpret the master´s intention in the dialog (several expression formats), associate this with a known task of an agent in the community, or ask the group if someone is able to know what it means.
**results:**
The SPMP document grows gradually, and the project manager should only revise it whenever a formal version is required.

Use Case 2 – Team activities follow-up (mainly for project managers)

**master intention:**
to check the activities of the project team, verifying the degree of members' participation, in order to keep members engaged and to push people to collaborate.
**ways to express:**
- How is it going on project X?
- Is project X running as we would expect?
- What could you tell me about project X?

**pre-established tasks:**
A Service Agent drives the Project Management Tool, via WebServices (REST), with the tasks of checking and storing the statistics of people participation in a project (access frequency, number of contributions in texts, and so on). Another Service Agent drives the Code Version Control Tool, via WebServices (REST), also with the same tasks on the members' participation in the code generation.
**correspondence:**
The PA interprets the master's intention in the dialog according to several expression formats and associates it with known tasks of one or more agents in the community.
**results:**
The project manager can receive a report of people participation on the project periodically or on demanding.

Use Case 3 – Periodical team report of activities (for any member of the project team)

**master intention:**
to be prepared to produce a periodical report of activities of a given project.
**ways to express:**
- my activities today for the project X are …
- put this on the project X report:
- give me the activities report draft for the project X this month.

**pre-established tasks:**
The Report Agent receives pieces of information sent voluntarily by the project team members that reflect their activities in the project for a period of time. The agent can organize the contributions composing a draft report which can be used to show the project advancement and also as a contribution for the project SPMP.
**correspondence:**
The PA can signal the need for reporting activities and mediate the contributions to the Report Agent. The PA can also, proactively, ask the Report Agent for the draft report close to the time his master will need the document.
**results:**
It is an easy way to construct a report of activities, even if someone will need to revise the draft in a more elegant format. The team manager can receive the draft report before asking it.

The examples of dialog situations we described lead to several complex conversation graphs, with some sub-dialogs used to clarify or to complement the information needs before a task can be fired.

The OMAS platform also allows building multilingual systems where one task can be fired from several conversation graphs each one built in a different language. However, this is not currently the focus of the CSCW-SD project.

## 7. Implementation

The CSCW-SD architecture has been used experimentally to support a small team developing R&D projects using artificial intelligence and knowledge engineering techniques. This test-bed uses OMAS as the MAS platform and Web Services to access the features of the two web-based application tools (the project management and the code version control).

Within the OMAS platform, one first needs to build a library of tasks that can be performed by the system. Each task contains linguistic markers taken in part from the domain ontology. Figure 6 shows the definition of a task for obtaining the latest news from an agent handling news (News Agent).

```
  (defindividual "task"
    ("task name" "obtain last news")
    (:doc :en "Task to show recent news.")
    ("performative" :request)
    ("dialog" _get-last-news-conversation)
    ("index pattern"
     (:new "task index" ("index" "news") ("weight" .51))
     (:new "task index" ("index" "last") ("weight" .41))
     (:new "task index" ("index" "facts") ("weight" .21))
     (:new "task index" ("index" "events") ("weight" .21))
     (:new "task index" ("index" "new") ("weight" .31))
       ))
```

**Figure 6: Definition of a Task for Obtaining Latest News**

During a dialog, such markers are used to rank the different tasks in a decreasing order of plausibility. Then, a threshold is used to eliminate tasks with low plausibility, and the first and most promising task is executed. This process is executed by the OMAS platform. Each task refers to a specific dialog that is then executed, most of the time to acquire enough data to trigger the task. Such task dialogs depend on the specific application and must be written manually. OMAS provides a dialog writing mechanism helping build the different states of the dialog and expressing the rules associated with the state. The example of Figure 7 declares a sub-dialog to obtain the latest news. In the entry-state of the conversation, a message is sent to the NEWS agent, and if there is a failure, then a transfer is made to the next state, _gln-sorry, or in case of success the news are printed and the sub-dialog is exited with a :success marker.

```
  (defsubdialog
    _get-last-news-conversation
    (:label "Get the more recent news")
    (:explanation
     "This dialog is meant to show the last news.")
    )

  (defstate
    _gln-entry-state
    (:label "Get last news dialog")
    (:entry-state _get-last-news-conversation)
    (:explanation
     "Sends message to the news agent, no arguments are necessary.")
    (:send-message :to :NEWS :self PA_TYLER :action :get-last-news
:args `(((:language . :en))))
    (:transitions
      (:on-failure :target _gln-sorry)
      (:otherwise :exec (print-news (read-fact moss::conversation
:answer))
                  :success))
    )
```

**Figure 7: Sample of the OMAS Language for Building Dialog States**

Of course, some states require complex handling. An escape mechanism is provided if the dialog language is not capable of expressing all the rules attached to a state.

In the same way, Figures 8 and 9 show task definitions for the situations described in Use Cases 1 and 2. Both tasks are performed by the Project Management Agent, which drives the Project Management Tool.

```
  (defindividual "task"
    ("task name" "write on SPMP")
    (:doc :en "Task to include some text in a specific section of a
SPMP document.")
    ("performative" :request)
    ("dialog" _put-text-on-SPMP)
    ("index pattern"
     (:new "task index" ("index" "write") ("weight" .8))
     (:new "task index" ("index" "put") ("weight" .6))
     (:new "task index" ("index" "project") ("weight" .4))
     (:new "task index" ("index" "text") ("weight" .2))
     (:new "task index" ("index" "important") ("weight" .2))
       ))
```

**Figure 8: Definition of a task for writing some text on a project SPMP (Use Case 1)**

```
  (defindividual "task"
    ("task name" "follow-up a project")
    (:doc :en "Task to verify/follow the activities of a project
team.")
    ("performative" :request)
    ("dialog" _get-project-team performance)
    ("index pattern"
     (:new "task index" ("index" "follow-up") ("weight" .9))
     (:new "task index" ("index" "going on") ("weight" .6))
     (:new "task index" ("index" "running") ("weight" .4))
     (:new "task index" ("index" "tell") ("weight" .3))
     (:new "task index" ("index" "project") ("weight" .3))
     (:new "task index" ("index" "expected") ("weight" .2))
       ))
```

**Figure 9: Definition of a task to follow-up the activities of a project team (Use Case 2)**

The Personal Assistant Agent (PA) will interpret phrases like "Please, put the text below in the section I of project Galaxis SPMP" as a call to the known task _put-text-on-SPMP. Then the PA will ask for the Project Management Agent to execute the task.

However, if the PA is not sure about what the master is asking for, it will trigger sub-dialogs to gather more information. For example, phrases like "How Galaxis is going on?" or "This information is important for the project Galaxis." will trigger sub-dialogs, where the PA can ask for clarification or information to fulfill gaps in its interpretation. At the end of the

sub-dialog, would the PA identify the task its master is asking for, it will ask a Service Agent to execute the task. Otherwise, the PA will try to improve the interpretation of its master request with the help of Service Agents, or it will decide that the request cannot be understood (failure).

The dialog described in the Use Case 3, in fact, requests two tasks for the NEWS Agent. In the first task, a user reports the activities she has recently done (Figure 10), and in the second, a user asks to the NEWS Agent a report of the activities performed in a given period of time (Figure 11).

```
  (defindividual "task"
    ("task name" "report my recent activities")
    (:doc :en "Task to send the activities done to the News
Agent.")
    ("performative" :request)
    ("dialog" _write-my-activities-conversation)
    ("index pattern"
     (:new "task index" ("index" "activities") ("weight" .9))
     (:new "task index" ("index" "project") ("weight" .7))
     (:new "task index" ("index" "write") ("weight" .5))
     (:new "task index" ("index" "put") ("weight" .5))
     (:new "task index" ("index" "today") ("weight" .4))
       ))
```

E.g., triggering phrase: "My activities in the project Galaxis today were:" or "Please, put this in my report activities, project Galaxis."

**Figure 10 – Definition of a task to inform my recent activities (Use Case 3)**

```
  (defindividual "task"
    ("task name" "build a report of activities")
    (:doc :en "Task to ask for a report of activities of someone or a
project, in a period of time.")
    ("performative" :request)
    ("dialog" _get-report-of-activities-conversation)
    ("index pattern"
     (:new "task index" ("index" "activities report") ("weight" .8))
     (:new "task index" ("index" "project report") ("weight" .8))
     (:new "task index" ("index" "give me") ("weight" .6))
     (:new "task index" ("index" "month") ("weight" .4))
       ))
```

E.g., triggering phrases: "Please, give me the activities report of project Galaxis last month."

**Figure 11 – Definition of a task to ask for a report of activities (Use Case 3)**

Developing and testing the dialogs is not overly difficult, but does require a little practice and some attention to choose appropriated words for the construction of the dialogs. The

examples shown here are useful in the context of the project management. The needs of the test team have oriented the research to develop more specific tasks and dialogs.

## 8. Conclusions and Future Work

This paper insisted upon developing a system that provides a unified interface for using collaborative tools. To do so, we advocated using dialogs in the framework of CSCW-SD, and gave an example of how this can be done. We consider that the use of PAs can help small teams handle documentation issues in an integrated and undemanding way. In this context, an adequate mechanism to handle the dialog between masters and PAs plays a key role.

We believe that this approach will improve collaboration among small team members developing software, specifically collaboration in the elaboration of project documentation. There is still a long way to go in this direction, but the CSCW-SD project has provided an adequate framework to guide the on-going development of the proposed architecture.

The PA could improve such collaboration by providing a unified interface for the several tools used by the team. In doing so, the PA is able to decrease the team members' workload by reducing the number of different interfaces they must deal with.

The PA could also improve collaboration by keeping a representation of its master's activities. With this information, the PA could be able to answer demands made by another PA. For example, if a user needs someone that has done a specific modification in the software code, it could ask her PA to ask to the other PAs who did the modification. It could create an opportunity to collaborate without increasing significantly their masters' workload.

A representation of the master's activities could also help the team manager. Such a representation could be used to help the elaboration of activity reports. It could also help follow the development effort.

Other Service Agents are under construction to increase the range of services available for the community. One of these Service Agents under development is capable of helping developers document their code (written in Java). The agent will be capable of writing comments into the code automatically. It will be also capable of extracting comments written by developers and putting them into the documentation of the whole collaborative project under development.

The integration of Web and Agent technologies can be useful for developing environments to support collaborative design [20] . We consider that to improve the interaction between humans and agents inside such an environment, better interfaces are necessary. Our investigation aims at improving such interfaces by applying a dialog mechanism.

In the long term, it is realistic to envision mixed teams of humans and agents. Researches have been exploring such a possibility ([23] [24] ). Although preliminary, we think that our investigation can be also relevant in the efforts to effectively team humans and agents.

## References

[1] Allen J, Ferguson G, Stent, A. "An Architecture for More Realistic Conversational Systems". In: Proceedings of Intelligent User Interfaces 2001 (IUI-01). Santa Fe , New Mexico: IEEE; 2001. p.1-8.

[2] Barthès JP. "OMAS - A Flexible Multi-Agent Environment for CSCWD". In: Proceedings of the 13th IEEE Computer Supported Cooperative Work in Design. Santiago, Chile: IEEE; 2009. p. 258-263.

[3] Barthès JP, Ramos MP. "Agents assistants personnels dans les systèmes multi-agents mixtes: Réalisation sur la plate-forme OMAS". RSTI série TSI 2002; 21:4. Ed. Hermes/Lavoisier. p. 473-498.

[4] Campagnolo B, Tacla CA, Paraiso EC, Sato G, Ramos MP. "An architecture for supporting small collocated teams in cooperative software development". In: Proceedings of the 13th IEEE Computer Supported Cooperative Work in Design. Santiago, Chile: IEEE; 2009. p. 264-269.

[5] Chan-Lam V. Conception et réalisation d'une base de données ensembliste. Thèse de Doctorat, Université de Technologie de Compiègne, Décembre 1979.

[6] Cook C, Churcher N. Modelling and Measuring Collaborative Software Engineering. In: Proceedings of the Twenty-Eighth Australasian Computer Science Conference, volume 38 of Conferences in Research and Practice in Information Technology. Newcastle, Australia: Australian Computer Society, Inc.; 2005. p. 267-276

[7] Fitzpatrick G, Marshall P, Phillips A. CVS Integration with Notification and Chat: Lightweight Software Team Collaboration. In: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work. Banff, Alberta, Canada: ACM; 2006. p. 49-58.

[8] Flammia G. Discourse segmentation of spoken dialogue: an empirical approach. PhD thesis at MIT, 1998, 152p.

[9] Jiang T, Ying J, Wu M, Fang M. An Architecture of Process-centered Context-aware Software Development Environment. In: Proceedings of the 10th Computer Supported Cooperative Work in Design. Nanjing, China; 2006. p. 1-5.

[10] Jurafsky D, Martin J. Speech and language processing. An introduction to natural language processing, computational linguistics and speech recognition. New Jersey: Prentice Hall; 2008.

[11] Kölzer A. Universal dialogue specification for conversational systems. In: Proceedings of IJCAI 99 – Workshop on Knowledge and Reasoning in Practical Dialogue Systems. Stockholm, Sweden: 1999.

[12] Komatsu T, Morikawa K. Entrainment in the Rate of Utterances in Speech Dialogs between Users and an Auto Response System. Journal of Universal Computer Science 2007; 13(2), p. 186-198.

[13] Kotelly B. The art and business of speech recognition. Boston: Addison-Wesley; 2003.

[14] Land SK, Walz JW. Practical Support for ISO 9001 Software Project Documentation. New Jersey: IEEE Computer Society; 2006.

[15] Negroponte N. Being Digital. New York: Alfred A. Knopf; 1995. 243 p.

[16] Oviatt S, Adams B. Designing and Evaluating Conversational Interfaces with Animated Characters. In: Cassell J, Sullivan J, Prevost S, editors. Embodied Conversational Agents. Cambridge: MIT Press; 2000. p. 319-345.

[17] Paraiso EC, Barthès JPA. An Intelligent Speech Interface for Personal Systems in R&D Projects. Expert Systems with Applications 2006; 31, p. 673-683.

[18] Sarma A, Noroozi Z, Van der Hoek A. Palantir: Raising Awareness Among Configuration Management Workspaces. In: Proceedings of the 25th International Conference on Software Engineering (ICSE). Portland, USA: IEEE; 2003. p. 444-454.

[19] Searle JR. A taxonomy of illocutionary acts. In: Proceedings of Language, Mind and Knowledge, Vol. 7. Minneapolis: University of Minnesota Press; 1975. p. 344-369.

[20] Shen W, Hao Q, Li W. Computer Supported Collaborative Design: Retrospective and Perspective. Computers in Industry 2008; 59(9), p. 855-862.

[21] Shen W, Wang L. Web-Based and Agent-Based Approaches for Collaborative Product Design: an Overview. International Journal of Computer Applications in Technology 2003; 16(2/3), p. 103-112.

[22] Storey MA, Cheng LT, Bull I, Rigby P. Shared Waypoints and Social Tagging to Support Collaboration in Software Development. In: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. Banff, Alberta, Canada: ACM; 2006. p. 195-198.

[23] Tweedale J, Ichalkaranje N, Sioutisb C, Jarvisb B, Consolib A, Phillips-Wrenc G. Innovations in multi-agent systems. Journal of Network and Computer Applications 2007; 30(3), p. 1089-1115.

[24] Urlings P, Sioutisb C, Tweedale J, Ichalkaranje N, Jainet L. A future framework for interfacing BDI agents in a real-time teaming environment. Journal of Network and Computer Applications 2006; 29(2-3), p. 105-123.

[25] Wu S, Ghenniwa H, Shen W. User Model of a Personal Assistant in Collaborative Design Environments. In: Agents in Design 2002. Sidney: MIT; 2002. p. 39-54.

[26] Weizenbaum J. ELIZA - A computer program for the study of natural language communication between man and machine. Communications of the ACM 1966; 9:1. p. 36-45.