Application Design Based on Work Ontology and an Agent Based Awareness Server

Rosa Alarcón and David A. Fuller

Pontificia Universidad Católica de Chile, Computer Science Department
Vicuña Mackenna 4860, 6904411, Santiago, Chile
{ralarcon,dfuller}@ing.puc.cl

Abstract. Collaborative Systems support user groups enabling a *shared environment* and informing of its state and changes through a mechanism called *awareness*. We believe that these changes and their consequences can be understood from an interpretation context, this approach allows us to infer the *relevance* of the awareness information and control its delivery trough different channels. The mechanism is enabled through a Multi-Agent System (MAS). We built a collaborative scheduler to test these ideas and we present a case study based on its use. We found that those ideas significantly ease the construction of different interfaces for the same problem while application usability is not impacted.

1 Introduction

Collaborative systems allow groups of users to share software artifacts, objects and self-representations enabling for them a virtual shared environment whose state and changes are presented through a mechanism called Awareness [1,2,3].

Users must explore the environment in order to discover any change made, or they can be notified automatically with the risk to be flooded (if every change is notified). To avoid this risk the current approach consists in choosing a set of interesting events to be informed about, but when the environment is dynamically constructed, those objects are created on the fly, so the notification can even be nonexistent.

We believe that based on their semantics, every action that occurs in the shared environment can be automatically notified to users. That is we need to understand the context where those actions occur (situational context [4]).

Context is described from the user perspective: user availability preferences and the interaction capabilities of user's devices (user context) and from the shared perspective: the organization of people, activities and resources (work context).

Both contexts allow us to infer the user willingness to be exposed to awareness information given the actual device interaction capabilities, the user preferences and the relevance of actions.

In a very abstract level, actions can be seen as simple, generic operations that transform the state of context: add, update, delete (add person, delete person, etc)

In this way, actions that occur in the shared environment are described trough those operations and any attempt to change the state of an element of the shared environ ment (e.g. delete an activity) causes a relevance and consequences estimation based on the work context and are informed to the users based on each user context.

Due to the distributed and personalized nature of our awareness proposal we choose to implement the awareness mechanism as a Multi-Agent System (MAS).

An agent is a software process that controls its own actions (autonomy), perceives its environment (react) and can take the initiative to perform an action (proactively) in order to accomplish tasks on behalf of its user [5]. A MAS system can be seen as a collection of possibly heterogeneous agents.

The proposed awareness server (AwServer) is a MAS that conforms to the standards defined by FIPA (Foundation for Intelligent Physical Agents) [6].

Every user has associated a pair of agents, one of them estimates the relevance of events perceived in the shared environment (based on the work context) and the other schedules the awareness delivery (based on the user context). Both tasks are implemented as rule-based inference processes.

We have built AwServer as an independent functionality layer, where a java component allows the shared environment to propagate actions to the AwServer (e.g. e.g. update user location, update designer team, add activity A, etc.), to listen for notifications from the server (e.g. john logs in the application, john has been removed from the designers team, john's activity A finished and reach its goals, etc.) and to perform specific queries to the agents (e.g. where is the user?) (fig1).

We have built a prototype system that uses AwServer to create a collaborative scheduler (CScheduler). The scheduler has three interfaces: a web application, an email-based application, and an SMS application.

In this paper we present a case study that allow us to have an insight of the impact of the approach in the construction and design of collaborative software as well as the users interaction trough the system, users satisfaction, application usability, support for the task at hand (to negotiate meetings) and interface features (AwServer awareness information delivery).

We found that the use of AwServer can help designers to obtain an abstract definition of the activities that describe the task at hand (a task context). This abstraction makes possible to easily build different interfaces for the same application: a servlet based web pages, a java application, an email notification system or a SMS notification system. All of them only implement interfaces, but the "plumbing" that implements the task context is made once.

The result suggests that application usability is not significantly impacted with the use of different channels of communication; the user seems to perceive and evaluated them all as separated channels.

Additionally we realized that it is possible to take advantage of agent technology to foster collaboration trough a previous agent collaboration that reduces users work burden. We plan to implement a full version that tests these features.

Section 2 presents the principal characteristics of the architecture that must be taken into consideration when building applications based on AwServer, complementary information can be obtained in [7]. Section 3 describes the process of creating an application based on AwServer. Section 4 presents our findings when building such application. Section 5 describes a case study we conducted using the application and finally, section 6 presents our conclusions.

2 MAS Server Architecture

AwServer manages the delivery of awareness information among a shared application and a group of users, trough a set of devices previously defined for each one.

The AwServer is a multi-agent system that conform FIPA specification and comprehends three kinds of agents: a Proxy Agent (PA) a Work Context Agent (WCA) and a User Context Agent (UCA).

A pair of WCA-UCA agents is associated to each user with the goal of manage his or her awareness information (fig 1), from the user perspective; they behave as a unity, an agency that provides a service.

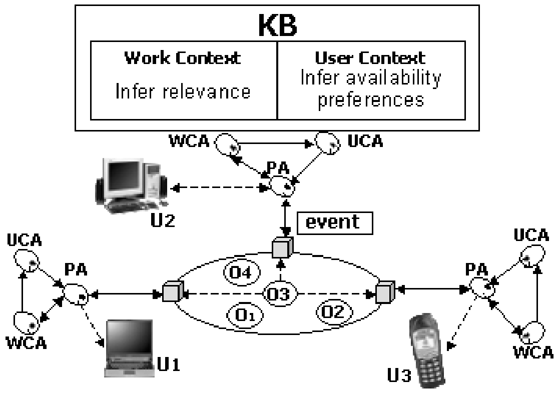


Fig. 1. AwServer architecture. *WCA* agent infers the relevance of events occurred within a *shared application* (the oval in the middle of the figure) based on the *work context*. *UCA* agent infers the *user availability* to perceive this information. *PA* agent propagates the events occurred in the shared applications to the *WCA-UCA* pairs trough an interface called *AgentListener* denoted by the shadow boxes in the border of the oval

Each action performed within a shared application is propagated to the proxy agent (PA), which in turn broadcasts a message to every pair WCA-UCA informing the event. A unique PA can manage all the messages or there can be a PA for each user because this kind of agent does not perform reasoning tasks and behaves more like a dumb robot. It is possible also to have a PA agent in a small portable device like a palm.

Both agents, WCA and UCA, behave cooperatively to interpret the message received from PA and to determine when and where (through which device) the user they represent will receive a notification informing about the event.

Agents' knowledge implementation (knowledge base and reasoning engine) has been designed separately from agents behavior, this approach allow us to refine or change the inference strategy without making any change to the agents' code.

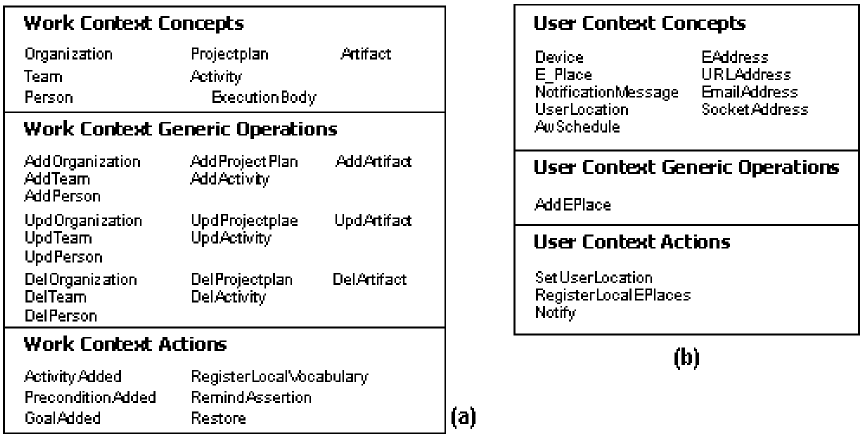
The next sections present the agents' knowledge and behavior respectively.

2.1 Agents’ Knowledge

In AwServer agents were designed as autonomous entities that communicate among them through messages in order to accomplish a specific task. Messages are written in ACL, which is a FIPA standard for agent communication.

Based on the semantic of the messages agents execute different behaviors (fig 3) (e.g. one agent can reply a query that asks for user location). Nevertheless, based on its knowledge an agent can infer that a behavior must be performed.

So, WCA infers user actions’ relevance based on its knowledge of work context which is represented as *work ontology*. An ontology is an explicit formal specification of the terms in a particular domain and the relationships among them [8]. User actions can be generic (add, update, delete an ontology element) or specific (register local vocabulary, inform that an activity has been added, etc.) (fig 2.a)



Work Context Concepts

Organization	Projectplan	Artifact
Team	Activity	
Person	ExecutionBody	

Work Context Generic Operations

AddOrganization	AddProjectPlan	AddArtifact
AddTeam	AddActivity	
AddPerson		
UpdOrganization	UpdProjectplae	UpdArtifact
UpdTeam	UpdActivity	
UpdPerson		
DelOrganization	DelProjectplan	DelArtifact
DelTeam	DelActivity	
DelPerson		

Work Context Actions

ActivityAdded	RegisterLocalVocabulary
PreconditionAdded	RemindAssertion
GoalAdded	Restore

(a)

User Context Concepts

Device	EAddress
E_Place	URLAddress
NotificationMessage	EmailAddress
UserLocation	SocketAddress
AwSchedule	

User Context Generic Operations

AddEPlace

User Context Actions

SetUserLocation
RegisterLocalEPlaces
Notify

(b)

Fig. 2. Agent’s knowledge represented through ontologies. Both ontologies comprise a set of *concepts*, *operations* to be performed over those concepts and *agents actions*. *WCA* knowledge is represented through a *work context ontology* (a), *UCA* knowledge is represented trough a *user context ontology* (b).

When WCA determines the relevance of an event, it asks UCA to inform the user about the event. Similarly, based on knowledge collected from the user, UCA infers the time and logic device (email, cellular phone, the shared application interface, etc.) where to deliver a notification informing the event occurrence.

Work context ontology considers concepts related to people, activities and group resources as well as the relationships among them. Notice that work activities have associated an “ExecutionBody” representing the ongoing execution of an activity.

Similarly user context ontology comprehends concepts that describe the situation of a user when he or she interacts with a shared application. This situation is described in terms of the interaction capabilities that the devices through which users contact shared environments offer.

Devices are considered as electronic or virtual places (e_Place) where users reside and can be contacted (a cellular phone, an email account, a shared application, etc.)

2.2 Agents' Behavior

Agents interact following social conventions; their communication is based on messages. Messages content can be predicates, concepts or actions to be performed over elements of the *work context ontology* for WCA agents and *user context ontology* for UCA agents (fig 2).

When PA requests WCA or UCA to execute a generic operation (add, update, delete) they must acknowledge the request, informing if the operation could be accomplished. If an agent does not understand a message, it informs that event to the requester. An operation execution consists of updating agents knowledge (e.g. add a person, update a team, etc) and asserting the result (success/failure) in its own knowledge base (fig. 3 A, F)

Other actions cause agents to execute different behaviors, for example the action "Set User Location" causes UCA agent to change its knowledge about where the user resides currently (that is which e_Place) or which e_Place has abandoned (fig. 3 G).

UCA receives a request to perform the Notify action from WCA agents whenever WCA determines that an event occurred in the shared environment must be notified and infers its relevance (fig. 3 E, J).

WCA agents can be asked to restore its knowledge from a log file in case of server breakdowns (fig. 3 A) and to remind the assertions or facts of its knowledge bases, which makes possible for end users to inspect agents reasoning (fig. 3 D, I).

It is possible also to wait for user acknowledgment, PA implements a *Waiting* behavior, which is an execution suspension until a responding message arrives.

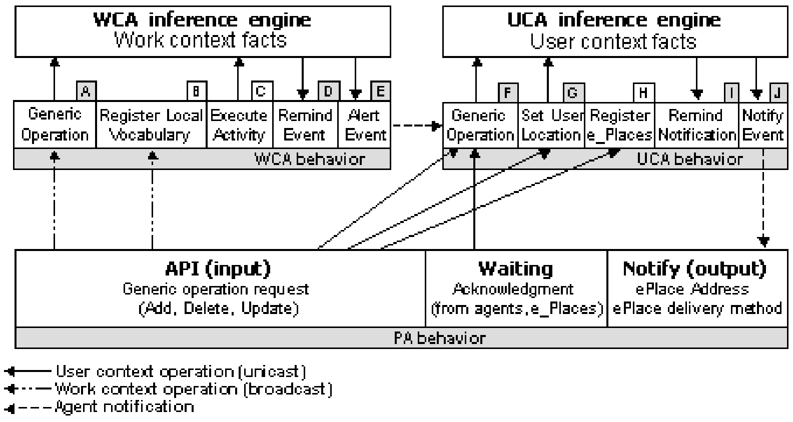


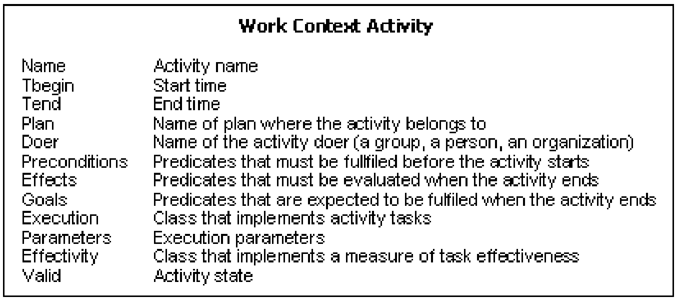
Fig. 3. Each agent can react to a message performing a specific behavior (*A, B, C, F, G, H*). *WCA* and *UCA* interact with an *inference engine* also and react to those inference processes by executing a behavior (*D, E, I, J*). *PA* implements the sensor (*input*) and effector (*output*) functions of the agency

When agents receive the request of register the "Local Vocabulary" and the set of user "e_Places", they simply registers them as dictionaries extending in this way the concepts and predicates they can understand to include the shared application itself as well as the effective user places where they can be contacted.

The consequences of these actions are extremely important as we describe in the next section. But previously we will describe how AwServer treats work activities.

2.3 Work Ontology: Activities

We conceptualized work activities based on the enterprise ontology of [8]. An activity is a concept or a class that has a *name*, must be executed during a *time interval* (start time and end time), is part of a *project plan* (which implies that it must be executed while the project plan is running), must be performed by a *doer*, has *preconditions* to fulfill in order to start its execution, causes some *effects* when executed, has *goals* to achieve and can be *executed* (fig. 4). Its execution can require some *parameters* and has an *effectivity* criterion that makes possible to measure its efficiency.



Work Context Activity

Name	Activity name
Tbegin	Start time
Tend	End time
Plan	Name of plan where the activity belongs to
Doer	Name of the activity doer (a group, a person, an organization)
Preconditions	Predicates that must be fullfiled before the activity starts
Effects	Predicates that must be evaluated when the activity ends
Goals	Predicates that are expected to be fulfiled when the activity ends
Execution	Class that implements activity tasks
Parameters	Execution parameters
Effectivity	Class that implements a measure of task effectiveness
Valid	Activity state

Fig. 4. Work context activity definition. An activity is described by a name, a length time (*Tbegin*, *Tend*), forms part of a *project plan*, is executed by a *doer*, has *preconditions* to fulfill, *effects* generated, *goals* to achieve, an *effectivity* measure a validity (*valid*) criteria and an execution body (*execute*) which can require *parameters*

Finally an activity has different states, when it is added through an "AddActivity" event (fig. 3 A), its *validity* is verified, that is, WCA checks if its starting and ending time falls between the running time interval of the project plan where the activity belongs to and if it is a viable time interval (the interval has not already passed or is an empty interval).

If it is not valid then the "AddActivity" operation is considered as a failure and the fact is notified to the users, otherwise the activity's *preconditions* are tested, again if the preconditions are not met, the AddActivity operation fails.

If preconditions are satisfied WCA executes the activity (fig. 3 C) and waits until it reach the activity's ending time. When this occurs, the agent tests the *effects*, *goals* definitions and *effectivity* criterion; again if goals are not met it is considered a failure.

Even though we did not implement a complex activity execution it is possible to execute activities in a sequential, parallel or cyclic fashion.

The benefit of having WCA agents executing activities is that activities can be complex enough to request agents to execute behaviors specific to the shared application without the burden of implementing a separate agents' environment that with agents that understands and communicate among them based on these ontologies.

WCA supports the execution of work activities that are defined in the Local Vocabulary by describing those activities in terms of logical preconditions and consequences (*ExecutionActivity* behavior). The approach allows us to implement contingent actions in order to facilitate the collaborative process (the activity execution) as the agents can engage in a FIPA Contract Net protocol to facilitate the preconditions and effects achievement.

Of course, an activity can have an empty execution body; in this case activities preconditions and goals are also verified at the starting and ending time.

3 Work Ontology Based Application Design

As we can see, work context ontology is a very abstract representation of work. It includes persons, teams, organizations, activities, project plans and resources or artifacts. Those concepts and the relationships among them describe work practice in a very high level, so it must be grounded according to the needs of every shared environment.

Applications based on work context must define a grounded instance of work context ontology or "Local Vocabulary". This vocabulary consists of concepts, predicates and work activities definitions and represents an ontology that defines the semantics of the application itself.

In order to test our ideas we designed a collaborative scheduler application (CScheduler) (fig. 5) [9].

CScheduler is a groupware application that aims to support the negotiation of a fitting time-interval for appointments of engineering project groups.

Users of CScheduler work on projects that have defined some *milestones* to achieve so they must negotiate work *appointments* that must be held before the milestone date has reached.

The attendants of those meetings can be the whole group, a part of them or can include people that belong to others groups (for example consultants).

CScheduler users can also define private meetings (preset meetings) where the attendant is himself or herself and possibly an outsider (for example a dentist).

It is also desirable to define *alerts* that remind teammates (inviters and invitees) the nearness of an appointment.

When a *teammate* receives a meeting invitation, he or she must answer the inviter *accepting*, *refusing* or *requesting a new date* for the appointment. Inviters can *cancel* an appointment, otherwise appointments remains in a *pending* (no any answer was received) or an *expired* state (the time has run out).

So the local vocabulary comprises nine principal activities as well as their execution body: fix appointments, answer appointments and modify those answers, set alerts, set milestones, add groups and modify them, add teammates and modify them (two section hexagons of fig. 5, where the upper part correspond to the activity name and the lower part the execution body name).

FixAppointment activity precondition is that inviter and invitees must be free (that is without an appointment accepted for any of them), so that it is necessary to declare and implement Free(Person, TimeInterval) predicate in order to allow the agents to determine if the activity can be executed (that is to run the Execution class) (fig. 6a).

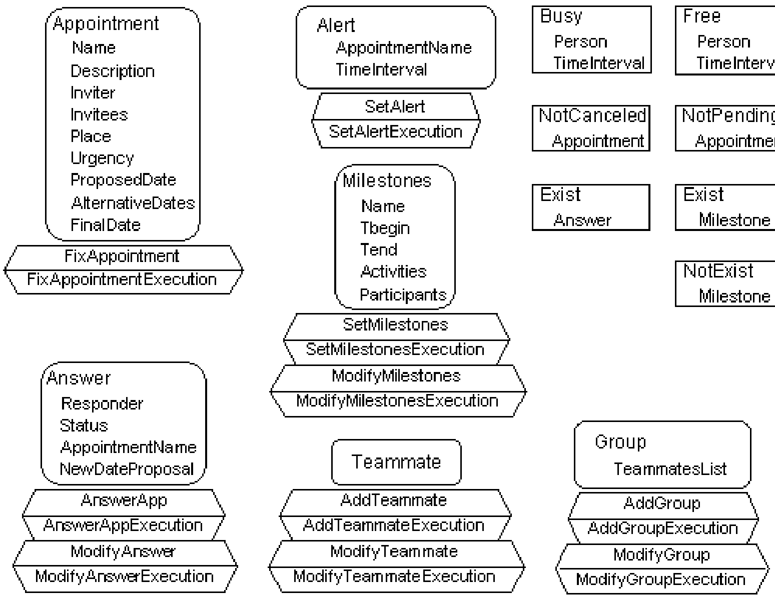


Fig. 5. Work ontology based CScheduler design. *Activities* are represented by a divided hexagon where the upper part is the activity name and the lower part the execution body name. *Concepts* are represented by rounded rectangles and *predicates* by rectangles. *Concepts name* appears in the upper part of the rounded rectangle and their *slots* or *attributes* are listed below. *Predicates names* are presented in the upper part of the rectangles and their *parameters* appears below

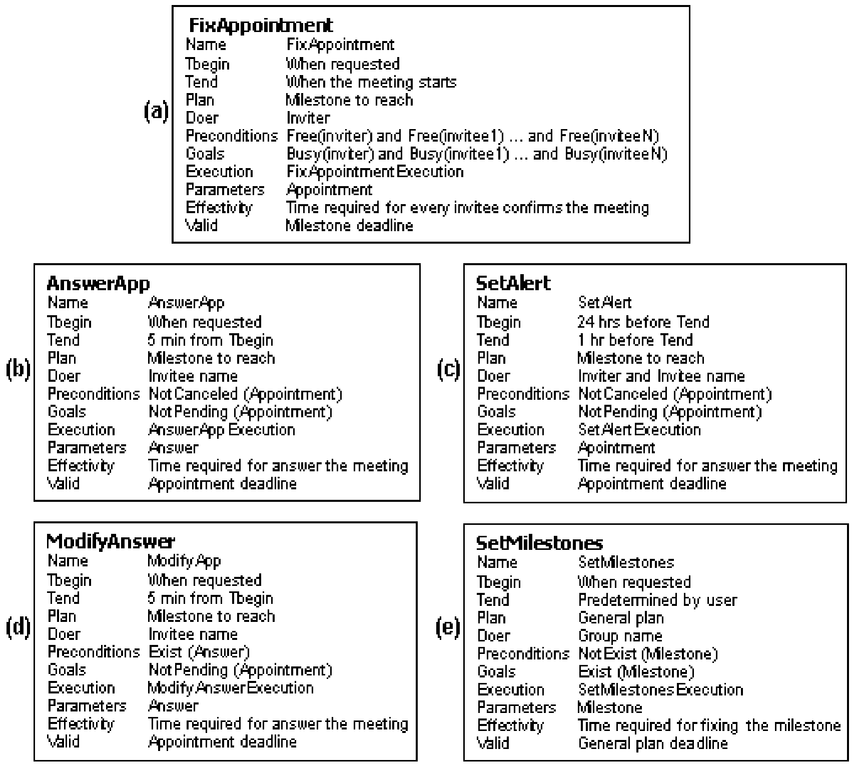
If the precondition is met, WCA agent executes the activity and once it has ended (Tend) goals are evaluated, that is, the predicate Busy (Person, TimeInterval) is executed. Finally, the execution of the activity takes as a parameter the appointment, it can be denoted like: FixAppointmentExecution (Appointment).

The precondition of AnswerApp activity is quite simpler; it is only necessary to have the appointment not canceled (by the inviter). Similarly, the goal here is to change the status of the appointment from pending (not answered yet) to another one (refused, accepted, etc) (fig. 6b).

The ModifyAnswer activity requires that the answer already exist and the consequences of executing this activity are similar to the AnswerApp activity (fig.6d).

SetAlert activity's precondition is to have the appointment not canceled (fig.6c).

The objectives of an alert are two: to remind the inviter and invitees of a meeting nearness and to remind invitees that they must answer the invitation, so they accept, refuse or request a new date for the appointment. The first objective has the goal of having the user aware of the nearness of a meeting, which we cannot determine (an alert can be sent to a cellular phone forgot at home) and the second objective is achieved when invitees changes the status of their invitations so this remains as not pending.



(a)

FixAppointment	
Name	FixAppointment
Tbegin	When requested
Tend	When the meeting starts
Plan	Milestone to reach
Doer	Inviter
Preconditions	Free(inviter) and Free(invitee1) ... and Free(inviteeN)
Goals	Busy(inviter) and Busy(invitee1) ... and Busy(inviteeN)
Execution	FixAppointmentExecution
Parameters	Appointment
Effectivity	Time required for every invitee confirms the meeting
Valid	Milestone deadline

(b)

AnswerApp	
Name	AnswerApp
Tbegin	When requested
Tend	5 min from Tbegin
Plan	Milestone to reach
Doer	Invitee name
Preconditions	NotCanceled (Appointment)
Goals	NotPending (Appointment)
Execution	AnswerApp Execution
Parameters	Answer
Effectivity	Time required for answer the meeting
Valid	Appointment deadline

(c)

SetAlert	
Name	SetAlert
Tbegin	24 hrs before Tend
Tend	1 hr before Tend
Plan	Milestone to reach
Doer	Inviter and Invitee name
Preconditions	NotCanceled (Appointment)
Goals	NotPending (Appointment)
Execution	SetAlertExecution
Parameters	Apointment
Effectivity	Time required for answer the meeting
Valid	Appointment deadline

(d)

ModifyAnswer	
Name	ModifyApp
Tbegin	When requested
Tend	5 min from Tbegin
Plan	Milestone to reach
Doer	Invitee name
Preconditions	Exist (Answer)
Goals	NotPending (Appointment)
Execution	ModifyAnswerExecution
Parameters	Answer
Effectivity	Time required for answer the meeting
Valid	Appointment deadline

(e)

SetMilestones	
Name	SetMilestones
Tbegin	When requested
Tend	Predetermined by user
Plan	General plan
Doer	Group name
Preconditions	NotExist (Milestone)
Goals	Exist (Milestone)
Execution	SetMilestonesExecution
Parameters	Milestone
Effectivity	Time required for fixing the milestone
Valid	General plan deadline

Fig. 6. CScheduler activities definition. CScheduler main activities are to *fix an appointment*, to *answer invitations*, to set alerts and to *define milestones*

An alert becomes its activity 24 hrs before the meeting has been fixed, and is sent every day from starting, the last day is sent 18 and 8 hrs before the meeting takes place. Once the second goal is reached, the alert is sent 2 hrs before the meeting fixed time.

SetMilestones activity has the precondition of having the milestone not previously fixed and the goal is simply to fix a milestone (fig.6e). As we can see, Milestones are project plans that form part of a General project plan. The latter plan has a length of weeks or months within all Milestones must be fitted. Milestones have a starting and ending time and activities must be accomplished within this time interval.

Additionally, Milestones can be modified and this action impacts in every activity already running.

AddGroup and AddTeammate activities has the precondition of not having exist previously the group or teammate respectively, it goal is very simple to have them exist. The time length for accomplish these activities is similar to the AnswerApp activity time interval.

ModifyGroup and ModifyTeammate activities are also similar to ModifyAnswer activity, they only require that the group or teammate already exist and the consequences of executing these activities are similar to AddGroup and AddTeammate activities.

Activities have defined also doers, which are the responsible of performing the activities, the owner of the activities in some way.

In the case of FixAppointment activity, this action must be performed by the inviter, which means that he or she will be notified if the activity creation fails (precondition, execution or goals fails). AnswerApp and ModifyAnswer activities are of responsibility of each invitee, SetAlert activity is responsibility of inviter or invitees respectively, and the responsibility of SetMilestones is assigned to the whole group.

As we observe those activities requires the definition of seven concepts: appointment, answer, alert, milestone, group, teammate and time interval (rounded rectangles of fig. 5) and seven predicates: Busy (Person, TimeInterval), Free (Person, TimeInterval), NotCanceled (Appointment), NotPending (Appointment), Exist (Answer), Exist (Milestone) and NotExist (Milestone) (rectangles of fig. 5).

When invitee WCA agent receives an invitation to make an appointment, it determines if the user is free or has an appointment already made for that date, in the latter case, the agent creates a list of suggestions based on the nearest slot of time available and sends its proposals to the inviter WCA agent.

Once the inviter WCA agent has collected all proposals, it determines the closest date that fits all users agendas (where everybody is Free) and sends these suggestions to the user asking he or she to pickup a final date. If the inviter chooses one of them, his or her WCA agent sends an acknowledge message to every invitee agents, which in turn fix the appointment for their users and mark its status as pending.

In this way we take advantage of having an activity executed by agents that conforms FIPA standard: they had predefined complex protocols of interaction like FIPA Contract Net Protocol that we can use in order to foster collaboration among users.

Activities are currently implemented as classes that contain attributes and get/set methods as well as the code that implements the Execution behavior and is loaded lately for the AwServer, this feature is optional. Of course is possible to automate the activities definition even to implement a graphical interface that will help collaborative application designers to devise the task context easily.

User context must also be grounded for use; it is necessary to create instances of specific e_Places for each user as well as to define their availability preferences. We present those e_Places in the next section.

4 Building an Application

In order to test our ideas we built a reduced version of CScheduler that implements the core functionality; we also conducted a case study that is described in section 5.

AwServer was built over JADE, a java-based agent development environment, so that the local vocabulary was also designed in java according to the jade content language and ontology support features defined in JADE.

Agents in JADE run in a container called JVM (java virtual machine) and the shared environment or CScheduler can run in another JVM (a java application, an applet or a servlet) or can be a completely different application (such as a php web site).

PA agent listens events in a java socket; shared applications send those events through a java-based component called Agency Listener (shadow boxes in fig. 1).

The component implement a java socket that writes in PA's socket and can receive messages from PA as well. Messages have ACL format and can be easily converted into java classes for facilitating their manipulation.

For simplicity we built the application using servlet technology. Servlets are java application that run in the side of a web server and are able to generate dynamic web pages. It is possible to modify the component to include a second socket were a non-java application can write and listen to or write an ad-hoc component as its functionality is simple.

We used Apache 1.3.22 for the web server and Tomcat 4.1 which implements a JVM where servlets run and can be reached trough http protocol, we also used Mysql 3.23.38 for the database.

As seen in fig. 3, WCA agents perform an Execute behavior that requires testing preconditions, executing the activity and evaluating activity goals.

Agents' behaviors are java threads and we exploited this characteristic to implement work activity management trough threads. That is, WCA agents evaluate preconditions if they are fulfilled a thread that executes the code defined by a descendant of the class ExecutionBody is created (the class is loaded and executed); additionally the agent waits until activity ending time to evaluate goals (the Execute thread is suspended and waits for the remaining time).

Preconditions and Goals are implemented as monomials formulas ($a_{\wedge}b_{\wedge}c\ldots_{\wedge}n$) from predicates that can be evaluated individually for values of true or false, for example:

$$\text{Free (inviter) and Free (invitee1) and} \ldots\ldots \text{Free (inviteeN)} \qquad (1)$$

The definition of a predicate requires also the construction of a method that allows agents to determine its value.

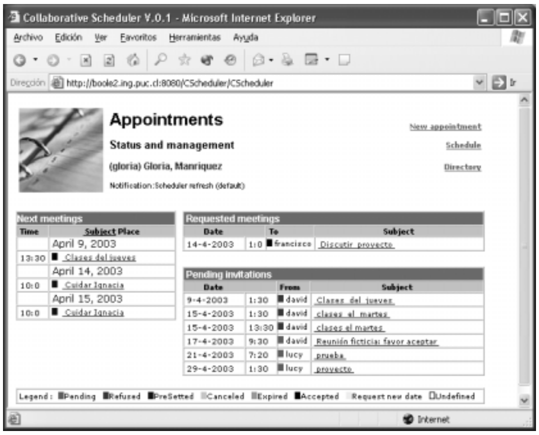
Fig 7 shows the screenshot of CScheduler Status and Management functionality; we can see three sections or inboxes for messages of group members. The left table shows appointments already made (they are confirmed or are private meetings).

The upper right table shows appointments requested for the user (in this case Gloria) to other members (in this case Francisco) the date and time of the appointment and the status of the meeting (red which means refused).

The lower right table shows the invitations to meetings that user has received, 6 in this case, the inviter (David and Lucy) the subject of the meeting, its date and its status.

Fig 8 shows a screenshot of Schedule function, we can see a calendar and a detail of the week currently selected, both of them are synchronized. The detail allows them to add a personal meeting and shows all the appointments the user has made and the invitations he or she has received as well as their status.

Users e_Places were also built as java classes that defines e_Places and e Addresses for each user. Users that own cellular phones had an SMS e_Place where a method (delivery_method) that sends a message from web was implemented.


Collaborative Scheduler V.0.1 - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección http://boole2.ing.puc.cl:8080/CScheduler/CScheduler
Ir
Internet

Appointments

Status and management

(gloria) Gloria, Manriquez

Notification:Scheduler refresh (default)

New appointment
Schedule
Directory

Next meetings

Time	Subject Place
	April 9, 2003
13:30	■ Clases del jueves
	April 14, 2003
10:0	■ Cuidar Ignacia
	April 15, 2003
10:0	■ Cuidar Ignacia

Requested meetings

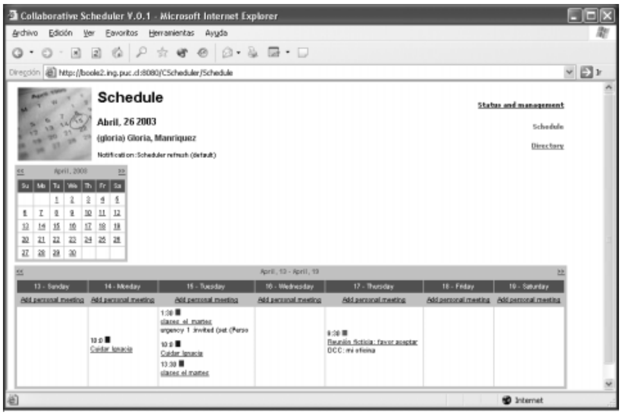
Date		To	Subject
14-4-2003	1:0	■ francisco	Discutir proyecto

Pending invitations

Date		From	Subject
9-4-2003	1:30	■ david	Clases del jueves
15-4-2003	1:30	■ david	clases el martes
15-4-2003	13:30	■ david	clases el martes
17-4-2003	9:30	■ david	Reunión ficticia: favor aceptar
21-4-2003	7:20	■ lucy	prueba
29-4-2003	1:30	■ lucy	proyecto

Legend: ■Pending ■Refused ■PreSetted ■Canceled ■Expired ■Accepted ■Request new date □Undefined

Fig. 7. CScheduler screenshot: *Status and management* function.


Collaborative Scheduler V.0.1 - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección http://boole2.ing.puc.cl:8080/CScheduler/Schedule
Ir
Internet

Schedule

Abril, 26 2003

(gloria) Gloria, Manriquez

Notification:Scheduler refresh (default)

Status and management
Schedule
Directory

<< April, 2003 >>

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

<< April, 13 - April, 19 >>

13 - Sunday	14 - Monday	15 - Tuesday	16 - Wednesday	17 - Thursday	18 - Friday	19 - Saturday
Add personal meeting	Add personal meeting	Add personal meeting	Add personal meeting	Add personal meeting	Add personal meeting	Add personal meeting
	10:0 ■ Cuidar Ignacia	1:30 ■ clases el martes urgency 1 invited (set (Perso 10:0 ■ Cuidar Ignacia 13:30 ■ clases el martes		9:30 ■ Reunión ficticia: favor aceptar OCC: mi oficina		

Fig. 8. CScheduler screenshot: *Schedule* function.

We take a similar approach for email accounts and java based applications. As users used mainly a web page, which is based on pull technology, we did not implement a delivery _method for this place and additionally agents did not choose this place for communications in any case for the same reason.

5 Case Study

In order to test our ideas we conducted an experimental study where the subjects were organized in groups to test the CScheduler version described in section 4. We designed the evaluation experience based on the classifications defined in [10] and [11].

The objective of the experience was to have an insight of users interaction trough the system, users satisfaction, support for the task at hand (to negotiate meetings) and interface features (AwServer awareness information delivery).

This is an evaluation prototype; the evaluation setting was naturalistic with minimal manipulation, we participate only to solve configuration or usage problems (some subjects asked how to perform functionality at the beginning of the experience).

Participants should negotiate an amount of 8 meetings. Their communication was mainly asynchronous although they also engaged in a sort of synchronized communication protocol when each participant was able to make a meeting (they were aware that this protocol was currently taking place).

This was a formative experience with the purpose of gaining insight in the construction of shared applications based on work ontology and AwServer and to learn from the impact of this technology in end users.

We collected both quantitative and qualitative data and compare them in order to determine if what users declared (perceptions) were close with what effectively had happened. For quantitative data we took the Scheduler database and the logs written by agents. These logs include the messages agents send each other, the inferences they had, the actions they actually executed and a status trace log.

For qualitative data we take usability tests as well as a perception test were we obtain information regarding users sense of group belonging, events occurred during the experience, sense of task progression and sense of users involvement.

Subjects. Subjects were 14 undergraduate and graduate students of the same course at the Pontificia Universidad Católica de Chile.

Students know each other but were not necessarily friends; most of them were Chilean but there were also foreigners. They were very familiar with email, SMS and web technologies and have different personal agendas to attend (they take different courses).

Communicational Resources. Regarding participants communicational resources, 11 own a cellular phone; all of them have web and email access from campus laboratories and the majority from their home computers (13). 3 of them have a desktop computer assigned at the campus with a fixed IP number.

From those who own a cellular phone, 8 have not enabled the SMS services, email or messages from the web. Those with a home computer have IP numbers randomly assigned; some of them have DNS service (3) while others did not.

Procedure. Students were organized in three groups of 4 and 5 peer members, each member was chosen randomly. Two groups were assigned to use AwServer based CScheduler version (Group1 and Group2) and the third group use a web based version without AwServer (Group3).

Groups were asked to negotiate 8 meetings during three weeks, at the end of which they were asked to take a questionnaire.

There were no deadlines defined to reach consensual meetings (we did not define milestones).

Results. Subjects using CScheduler qualified the web-based application as easy to use and understandable in general. We made direct and indirect questions about scheduler characteristics and all of them but one responded accurately (What kind of answers your invitees gave to you? What kind of states can have a message?).

Users declared to be satisfied with the functionality of the scheduler although they required more control. We asked users to qualify the easiness of use of all functionality of the system and both kind of groups those who use the AwServer version and the simple web version without AwServer qualified the system functionality in a similar fashion (fig. 9a).

Users tend to fixed meetings were a part of the group was invited not the whole because they has some task to perform that does not involve the group, they missed to notice that this was a general experiment and use the scheduler to negotiate appointments that they actually need (or make sense for them). Although they were informed that email system was used to deliver notifications some of them in fact answer these notifications.

Nevertheless, some members of groups behave in a very passive fashion expecting appointment invitation in order to answer but without requesting them a meeting. We asked users to express how much active did they believe was their groups and themselves in the experience in a direct manner (asking him or her to rate themselves) and indirectly (asking him or her to report the number of meetings they requested, the number of meetings they responded as well as their status). In general users perceived themselves as having a poor activity (fig 9.b) while they were really more active and close to the goal (arrange 8 meetings) than they reported.

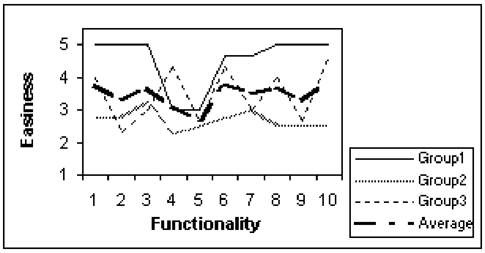
In the case of groups that used the version that include AwServer, they received a bulk of almost 22 emails that informed about the actions that occurred in the shared environment: a precondition has met or failed, an activity has met or failed, a goal has accomplished or failed. Most users found this overwhelming and confusing because of the format of messages, they were too formal and without interpretation: We reported a "goal failure" instead of advising that "appointment x fixed for the date y having z, q, p attendants, was not confirmed by any".

On the other hand the control group reported a lack of acknowledge, they visited almost daily the site in order to find out what has happened. In this group only one member requested appointments from the others, the others remained always passive.

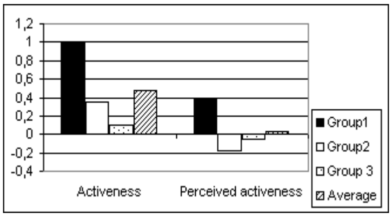
Additionally they reported to be satisfied with the application and suggested minor changes that make the site more usable (arranging the menu disposition, the possibility of a more detailed description of the meeting purpose or the causes for rejection) while the other focused more in functional details (modify appointments, adding cyclic personal appointments, manipulate their directories in order to include other members, reduce the number of messages, presented in a better format, etc.)

In the case of users that received acknowledge by cellular phone they reacted visiting the web site or trying to figure out what is the meaning of the message because they felt that something was wrong.

In all three groups the sense of group was established in relation to those users who were more proactive and to users that already engaged in the task (by answering appointments requests). In relation to the number of meetings requested and invitations answered for each one, Groups 1 and 2 reported activity closer to the reality than Group3.



(a)



(b)

Fig. 9. CScheduler screenshot: Schedule function

6 Conclusions and Further Research

The information obtained from end users was important for us because it shows us that the use of different channels of communication is not necessarily a source of distraction or cognitive overload. The amount of information obscured, not interpreted can certainly overload users and can be a source of frustration.

On the other hand, acknowledge from teammates can help users to notice the group activity only if this information arrives trough a channel close to the user (a cellular phone, an email alert, the application interface). When this information is sent to passive mediums like email, users expect it to be meaningful; otherwise they neglected them more as time passes.

Nevertheless there are some notifications not important enough even to be notified about, and must be presented only when users explicitly request to examine the status of the application, then they are important as activity indicators. Finally, usability characteristics are related to each user interface users seams to perceive and evaluated them all separately.

The experience was very instructive for us, because we realized that having a unique definition of activities that describe the task at hand, that is the local vocabulary, we were able to design easily different interfaces for the application: a servlet based generated web pages, a java application that fully implements the scheduler, a java application that only alerts of the arrival of a message, an email notification system and a SMS notification system. All of them implements only interfaces but the “plumbing” is made once.

In the case of complex applications that uses push technology (a java based application for example) it is also possible to design the application in a reactive fashion, that is because we can determine unambiguously the semantics of a particular message, we can embed components in the application that react only to the messages they are interested for example, reloading the screen to present a new invitation if the user is currently watching the appointments status screen, updating the names of users that are already logged in the system, expiring appointments whose goals has failed, etc. Again the plumbing required is far less demanding.

We plan to implement Cscheduler´s full functionality and to conduct again this experience in order to contrast our findings. A third evaluation will be realized with a version that includes the learning of user preferences.

References

1. Dourish, P., Belloti, V.: Awareness and coordination in shared workspaces. In Proceedings of CSCW '92 (1992) 107–114
2. Sohlenkamp, M.: Supporting Group Awareness in Multi-User Environments through Perceptualization. GMD Research Series; No. 6. (1999)
3. Fussell, S., Kraut, R., Lerch, F., Scherlis, W., McNally, N., Cadiz, J.: Coordination, overload and team performance: Effects of team communication strategies. In Proceedings of CSCW'98, (1998) 275–284
4. Schmidt, A., Implicit Human Computer Interaction Through Context Personal Technologies Volume 4 (2&3), June 2000. pp. 191–199
5. Maes, P.: Agents that reduce work and information overload. Communications of the ACM, 37 (7) (1994) 31–40
6. Burg, B.: Foundation for Intelligent Physical Agents. Official FIPA presentation, Lausanne, February 2002. See `http://www.fipa.org for further details`
7. Alarcón, R., Fuller, D.: Intelligent Awareness in Support of Collaborative Virtual Work Groups. Lecture Notes in Computer Science (Joerg M. Haake, Jose A. Pino, eds.), Vol. 2440, pp. 168–188, Springer-Verlag 2002
8. Gruber, T.R.: A translation approach to portable ontology specification. Knowledge Acquisition. Vol. 5 (1993) 199–220
9. Malone, T., Grant, K., Turbak, F., Brobst, S., Cohen, M.: Intelligent information-sharing systems. Communications of the ACM, 30 (5), 1987, 390–402
10. Pinelle, D., and Gutwin, C.: A Review of Groupware Evaluations. Proceedings of Ninth IEEE WETICE 2000 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Gaithersburg, Maryland, June 14–16
11. McGrath, J.E. Methodology Matters: Doing Research in the Behavioral and Social Sciences. In Readings in Human-Computer Interaction: Toward the Year 2000, 2nd Ed. Baecker, R.M., Grudin, J., Buxton, W.A.S., Greenberg, S.(eds.), Morgan Kaufman Publishers, San Francisco. 1995, 152–169