Ecole Nationale Supérieure des Mines
SAINT-ETIENNE

Industrial Engineering and Computer Sciences Division (G2I)

STEPS TOWARDS MAKING CONTEXTUALIZED DECISIONS : HOW TO DO WHAT YOU CAN, WITH WHAT YOU HAVE, WHERE YOU ARE

O. BUCUR, P. BEAUNE, O. BOISSIER

Septembre 2005

RESEARCH REPORT
2005-700-013

MINISTÈRE DE L'ÉCONOMIE
DES FINANCES ET DE L'INDUSTRIE

Steps Towards Making Contextualized Decisions: How to Do What You Can, with What You Have, Where You Are[*]

Oana Bucur, Philippe Beaune, Olivier Boissier

Centre G2I/SMA, Ecole NS des Mines de Saint-Etienne,
158 Cours Fauriel, Saint-Etienne Cedex 2, F-42023, France
{bucur,boissier,beaune}@emse.fr

Abstract. Context-aware applications need facilities for recognizing and adapting to context to provide useful and user-centered results. There are several problems to be addressed when building context-aware applications, two of which being how to *define and manage* all available contextual information and how to *distinguish relevant* from non-relevant context for a given task. In this paper, we focus on the second problem and propose a context definition and model for context-aware multi-agent system architecture. We exploit this model to build agents that learn to select relevant context and to use it when making decisions.

Introduction

The rise of pervasive computing has stressed the importance of *context*. As defined in [7], this concept consists in "any information that can be used to characterize the situation of an entity", where an entity can be "a person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves". This definition underlines two of the major problems in current context-aware applications: the *amount of contextual information* that must be managed ("*any* information […] used to characterize the situation" gives an idea about how difficult must be the task of defining and managing it) and the *relevance of that information* ("…information that can be *used to*…", "…that is considered *relevant*…"), even though the definition does not specify how to choose among all available context information the one that is relevant, it assumes that this task must also be completed.

Existing works usually handle these problems at the *design phase*: the system's designer is the one that defines what information will be part of the application's *context* and makes the choice of what will be considered as relevant in that precise application. As the amount of information available is usually very large, the designer has to build a huge ontology of concepts that will be used in that system, and then make an a-priori choice of what is relevant.

[*] Paper submitted to International Journal of Pervasive Computing and Communications, August 2005

The major problem with the a-priori choice of what is relevant is that the applications have changed. Classical applications were clearly defined, had a specific purpose and a specific context of use that didn't change very much and was easily predictable. The new generation of pervasive and ubiquitous computing changed all this, in that applications are now situated in a dynamic environment, where it is more difficult (if not impossible) to predict the changes and configurations that might happen. Considering this, a static approach to the environment (and to what might be called context, in general) of an application is not appropriate anymore. The designer can not predict all possibilities a-priori, so the need for the application (in relation to its user) to deal with dynamic and unpredictable environments emerges.

In this paper, our goal is to draw a common base for context-aware applications, by alleviating the system's designers task in what concerns the choice of relevant information by transferring it to the user or the application itself. The intended results are to allow contextualized decisions by taking into account what is relevant in the current situation. As a first step, we will deepen our motivations and present a global picture of our picture. Then we will present the proposed context representation and the context aware system architecture fulfilling our goal. We will then describe the benefits of this architecture by the use of intelligent agents to learn relevant context users.

We detail the architectural construction of such a system and illustrate it with a case study of an open and interoperable context-aware agenda management using Multi-Agent technologies. The resulting MAS is made of several meeting schedulers agents called mySAM (my Smart Agenda Manager). mySAM assists its user in fixing meetings by negotiating them with other mySAMs and by using context knowledge to decide whether to accept or reject a meeting proposal made by another agent. Knowledge about how to select relevant context and how to use it to deal with a meeting proposal is acquired through individual and multi-agent learning.

After the description of this case study, we will compare our work to existing approaches and conclude.

Motivation and approach

As we mentioned before, the focus of our work is the question of relevance, in all its interesting points: how to define what is relevant, how to choose between relevant and non-relevant information and how to use relevant information when making decisions.

Mostly, when context is used to make behaviours context-adaptable, it is used in an ad-hoc manner, without trying to propose an approach suitable for other kind of applications (ref to **[17]**, **[15]**, **[24]**, **[23]**, etc.). Some research propose a general architecture on context-aware applications, like CoBrA(Context Broker Architecture), proposed by Chen et al.**[2]**, Socam, by Gu et al **[26]** or the work of Coutaz et al. ([5]). The context broker and interpreter that are proposed are motivated by the acquisition of certain information from heterogeneous sources. However they don't address the problem of the representation of context in a semantic and generic manner. Moreover they aren't interested in the reasoning on context knowledge based on this representation. There are several works that also argue the need to separate the managing of context from reasoning with context to determine the behavior to adopt in each situation ([2], [26], etc.). In this way, the management of context can be reused between application, as there is just the reasoning engine that can vary from one application to another.

To address those problems, we argue that there are some necessary steps to be taken, as to: (i) make a *distinction between managing context and reasoning using context* to make decisions (the necessity of this distinction was argued beforehand), and (ii) dynamically *adapt the selection of relevant information*, according to the task (finality) at hand. These two directions lead to the definition a layered architecture (cf. Fig.1) in which a clear distinction between managing context (layer 1) and reasoning using context (layer 2) is done. The system architecture is structured along four layers. Layer 0 addresses the management of the context sources whereas layer 1 is dedicated to the general management of everything that can be considered as context which is provided by these different context sources. Layer 2 is dedicated to the context-based reasoning. In order to allow a personalized and decentralized approach, we use multi-agent technologies on this layer so that reasoning capabilities in order to dynamically adapt the selection of relevant context and to decide appropriately are embedded in every agent. Agents interact with the context-manager layer that is able to answer context-related queries. We thus disconnect all that is related to context acquisition and management (done by layer 1) from reasoning with context knowledge, done by the agents. Over the last layer are dedicated applications which use agents' abilities to reason with context, or in which context can be directly used by the application itself.

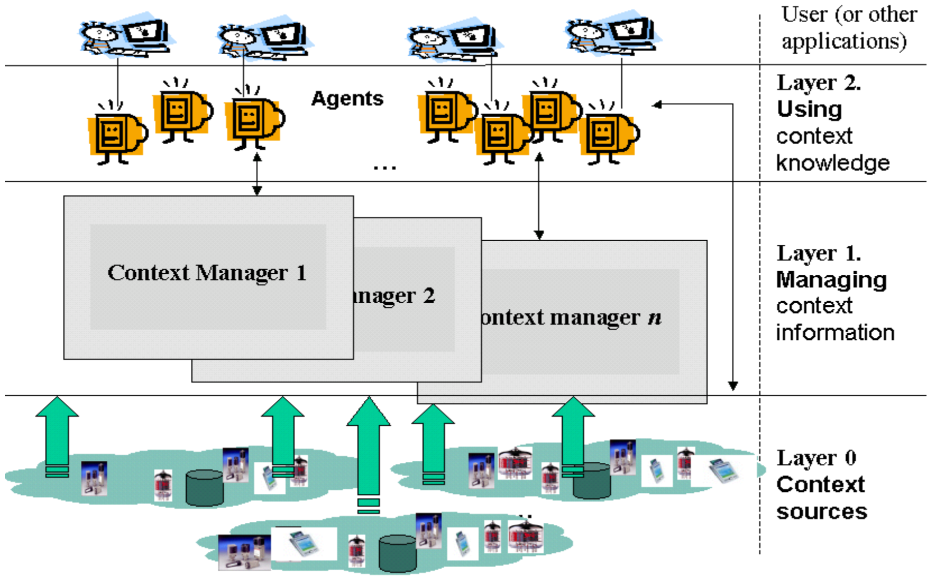

User (or other applications)
Agents
...
Layer 2. Using context knowledge
Context Manager 1
nager 2
ontext manager n
Layer 1. Managing context information
Layer 0 Context sources

Fig. 1. Global architecture for Context-aware applications

Given this global architecture that structure our approach according to the step (i) pointed above, let's go back the second step (ii). The main problem in context-aware applications is how to select (choose) *relevant* from non-relevant information for a specific task. As explained in introduction, our aim is to have this selection of relevant information done by the application itself, at runtime, and not in advance, by the designer. We consider that changes in environment are so unpredictable in context-aware applications, that such an a-priori choice of what is and it is not relevant is not adequate. The solution, as we see it, is to make this selection at runtime, and, if at all possible, improve the selection mechanism at each occasion. Our approach leads to the definition of context-aware multi-agent applications that may be composed of several "societies" of heterogeneous and situated agents. Those agents will be used to select relevant information for the task they need to accomplish, as they can be context-sensitive thanks to their interaction with the layer 1. In our system, agents will learn to discriminate relevant from non-relevant context and to make appropriate decisions based on relevant information. They will also able to share amongst themselves the way they use context knowledge in solving similar problems. In this way, we not only embed the selection phase into the application itself (in this case, represented by an agent), but we dynamically improve this phase (through learning) to obtain better performance. Choosing in advance what might be relevant for an application can give quite average results, as there are applications (especially highly dynamic ones) in which such a prediction of future states of the environment can not be done.

To sum up our approach, what we focus on is: a general management of everything that can be considered as context and adding reasoning capabilities to agents in order to dynamically adapt the selection of relevant context and to adapt the decision making process. Given this global picture, and before describing in more details this architecture, we have first to define an *explicit representation of context* so that agents can interrogate the context management layer on one hand, and also to exchange knowledge and reason about context.

Basic definitions and our model of "context"

We will first describe what we understand by "context", and then explain how context is represented.

Basic definitions

Agents are situated entities, meaning that they sense their environment and act accordingly. In this section we define the basic components of what we call "context": the finality (goal) of its use, the elements that are part of it and the relevance of those elements given a specific finality.

Finality

Going back to Dey's definition, we can define context as a cluster of information relevant for a specific purpose. Before going further, let us explain what we consider as purpose (or finality, how we will call it further on). The *finality*, **f**, is the goal for which the context is used at a given moment, the focus of the activity at hand. A *finality* is the information that is the most interesting for the application at a given moment, for example: deciding what to do with an offer, explaining an action, understanding a conversation, etc. All of these are finalities that determine the way the application will consider context and act upon it, therefore compel the activity of that application.

It is the designer of the system (or the user himself) who defines the finalities that will need to be fulfilled with the use of context. For example, in an agenda-management application, finalities can be: answer to a meeting proposal, cancel a meeting, negotiate a meeting, etc. In a tourist guide application, finalities could be: propose sightseeing tour, find a museum/church/restaurant/…, etc. The set of finalities (goals to be fulfilled) is defined in association with the application at hand, as this set is the one that will guide the selection of what is relevant and what is not for that specific task.

As mentioned before, context is defined by a set of attributes that are relevant in a given situation given a finality. In what follows we will define the general structure of a context attribute.

Context attribute

Usually, in literature, an element of context is either seen as a simple "label" that will have an associated value (like in [27], [1], [22]or [8]), or as a more complex structure, that also includes the specification of the behavior or action to be followed when a certain value is reached (like in [9]). The problem with this kind of modeling context is that the context-management task is very much connected to the reasoning-with-context one. We see this as a problem because, due to this type of management, the context needs to be re-defined and managed in a different manner for each application. Therefore, the distinction between the *context-management* task and the *reasoning-with-context* task imposes a clear separation between the *definition* of a context attribute and the specification of the behavior to be adopted in certain situations.

Thus, what we intend is to develop a specific structure within which we are able to define a context attribute. This structure's goal is to specify the pattern of a context attribute, and not to point out which conduct should be adopted for a specific value of the context attribute. Moreover, we consider a context attribute much more complex than just an attribute-value structure. In order to describe any context information, no matter how complex it might be, we extend the simple 'attribute-value' approach.

We consider that a *context attribute* ***(a)*** designates the information defining one element of context, e.g. "ActivityLocation", "NamePerson", "ActivityDuration". Each context attribute has at least one value at a given moment, the value depending on several entities to which the attribute relates. An entity is an instance of a "person, object or place" (as mentioned in [7]), but can also be an activity, an organizational concept (role, group, another agent, etc). For instance, the context attribute "DevicesAvailableInRoom" defines the devices that are located in a room. When trying to instantiate this attribute, the needed parameter will be the specific room that interests us. A "PersonIsMemberOf" context attribute will take a person as input, and will return (possibly) several groups in which that person takes part.

The *definition domain* of context attribute $\boldsymbol{a}$, $\boldsymbol{V_a}$ is the set of possible values that $\boldsymbol{a}$ may take (example: $\boldsymbol{V_{Hour}}$ =[0,24[, $V_{\text{DevicesAvailableInRoom}}$={PC, PDA, smart phone, printer, Xerox, phone, white board,..}).

We can therefore associate to each context attribute an instantiation function called ***valueOf***. We define *valueOf* for a context attribute $\boldsymbol{a}$ as being the function from cartesian product $\boldsymbol{A}$ and $\boldsymbol{P_a}$ to $\boldsymbol{P(V_a)}$, where **A** is the set of all attributes, $\boldsymbol{P(V_a)}$ is the power set of $\boldsymbol{V_a}$, and $\boldsymbol{P_a}$ is the set of parameters needed to compute the value of $\boldsymbol{a}$. It is obvious that some context attributes (as "PersonIsMemberOf", "Supervises", "DevicesAvailableInRoom", etc.) can take not just one, but a set of values. This is why we allow the function *valueOf* to take values in $\boldsymbol{P(V_a)}$, and not only $\boldsymbol{V_a}$.

Relevance

Since not all attributes are relevant for a finality, we define predicate ***isRelevant(a,f)***, that states that the attribute $\boldsymbol{a}$ is relevant for the finality $\boldsymbol{f}$. Let's call ***RAS***(*f*) the subset of $\boldsymbol{A}$ which defines the *Relevant Attribute Set* for the finality $\boldsymbol{f}$:

$$\boldsymbol{RAS}(f) = \{\ a \in \boldsymbol{A} \mid \boldsymbol{isRelevant}(a,f)=\text{true}\ \}.$$

We call an *instantiation of context attribute* $a \in \boldsymbol{A}$, the pair (a,v) where v is the set of values $v \in \boldsymbol{P}(\boldsymbol{V_a})$ of a at a given moment. For instance, (Day, {14}), (roleOfPersonInGroup, {Team Manager}), (PersonIsMemberOf, {MAS Group, Center_X, University_Y}) are all instantiations of the respective context attributes: *Day*, *roleOfPersonInGroup*, *PersonIsMemberOf*. We call $\boldsymbol{I}$ the set of instantiated context attributes as

$$\boldsymbol{I} = \{(a,v) \mid a \in \boldsymbol{A} \wedge \boldsymbol{valueOf}(a)=v\}.$$

The *Instantiated Relevant Attribute Set* of a finality $\boldsymbol{f}$, ***IRAS*(f)**, is the set of instantiated context attributes relevant to finality $\boldsymbol{f}$:

$$\boldsymbol{IRAS}(\text{f}) = \{(a,v) \mid a \in \boldsymbol{RAS}(\text{f}) \wedge (a,v) \in \boldsymbol{I}\}.$$

Let's notice that in related work (**[15]**, **[23]**, **[24]**), the notion of "context" is often understood as being what we defined as the IRAS. To explain the difference between RAS and IRAS and the reason to make this distinction, let's consider the following example. Given the finality f = "deciding whether to accept or not a meeting", RAS(f)={"RoleOfPersonInGroup", "ActivityScheduledInSlot"} is considered, i.e. role played by the person who made the proposal and if the receiver has something already planned for that time slot. The resulting IRAS for a student may be $\text{IRAS}_{\text{student}}$(f)={(RoleOfPersonInGroup, {teacher}), (ActivityScheduledInSlot, {Activity001})} and for a teacher $\text{IRAS}_{\text{teacher}}$(f)={ (RoleOfPersonInGroup, {student}), (ActivityScheduledInSlot, {Activity255)}. As we can see, the difference between IRAS of student and teacher may lead to different rational decisions. Usually, RAS is similar for different users when needed to make decisions (for the same finality), but the decision itself is IRAS-dependent.

Taking into account the definitions that we proposed so far we are now able to describe the representation of a context attribute.

Representing context

Given previous definitions, our aim is to represent context attributes in a general and suitable manner for any kind of applications that need to represent and reason about them. What we focus on is to have a generic and complex representation for a context attribute, representation that will assure in this way the interoperability, but also the possibility to represent no matter what context attribute, not just the simple ones. Several representations of context exist: contextual graphs (**[1]**), XML (used to define ConteXtML **[22]**), contextors ([4]) or object oriented models (**[9]**). All these representations have strengths and weaknesses. As stated in **[10]**, lack of generality is the most frequent weakness: some representation are suited for a type of application and express a particular vision on context. There is also a lack of formal bases necessary to capture context in a consistent manner and to support reasoning on its different properties. A tentative answer in [10] was the entity-association-attribute model, which is an extension of the "attribute-value" representation, where contextual information are structured around an entity, every entity representing a physical or conceptual object. We based our proposal on this idea and on ontology as the explicit way to represent it. To take into account the need for generality, and also considering the fact that we aim at having several MAS, each dealing with different contexts (that we will need to correlate in some way), an ontology-based representation seems reasonable. This is not a novel idea, the work done by Chen *et al.* (**[2]**) being just one example of defining context ontologies using OWL([18]).

Usually, when context is represented using ontologies, what is said to be "context" are the properties associated to each concept (or entity, how we will call it further on). For example, when defining a meeting ontology, we can define the status of a meeting as a property ("*statusMeeting*") of the entity Meeting. This property can also be considered a context attribute. This representation is simple and does not allow more complex context attributes to be represented in the same manner. For example, if we want to define an attribute that will have the value "true" if two meetings are taking place in the same time, we need to define this attribute not as a property (as it will be connected to two meeting entities – in order to check when they take place), but as a class in itself. This approach will confer a rather heterogeneous representation of context attributes, as properties, classes, instances. Our need for a generic representation of *all* context attributes made us propose a slightly different approach.

We define two ontologies: a ***domain ontology*** and a ***context ontology***. The domain ontology is used to define all concepts that will be used in the current system. The context ontology is the one that contains all context attributes that will be managed by a context manager. Let us detail first the domain ontology and then, how we build the context ontology using the concepts defined beforehand.

Domain ontology

In our domain ontology, we define a class "#Entity" as the super class of all concepts, e.g. in our agenda-management case study, #Person, #Group, #Room, #Activity, etc. are subclasses of #Entity.

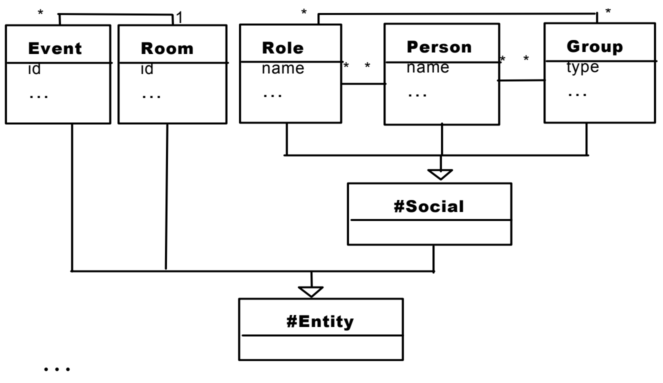


Fig. 2. A partial view on the concepts defined in a domain ontology

For example, in Fig. 2, we show how concepts as: #Event, #Room, #Role, #Person, #Group relate to one another and how all of them are sub-classes of the root of the domain ontology: the class #Entity. Classes as #Role, #Person and #Group are subclasses of the concept #Social. There are several other properties for each concept than just *name* and *id*, as, for example, for a #Person, we defined properties as: *interests*, *currentActivity*, *isMemberOf*, etc. Below you can see the OWL definition of the class #Person and #Social.

```
<owl:Class rdf:ID="#Person">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="#Social"/>
    </rdfs:subClassOf>
  </owl:Class>

<owl:Class rdf:about="#Social">
    <rdfs:subClassOf rdf:resource="#Entity"/>
  </owl:Class>
```

Using the concepts defined in this "domain ontology", we can further explain how we represent context in the "context ontology". We will first explain the structure of a context attribute, as we constructed it in the definitions and then clarify how we used that structure in building the context ontology.

Context ontology

According to our definition of a context attribute in section "Basic definitions", we propose to define a concept called "context attribute", which will encapsulate all needed information for defining and instantiating a context attribute. Instead of treating each attribute differently (based on its complexity), we define a class (called #ContextAttribute) that will typify a common description for all context attributes: the name of the attribute, the type of needed parameters(entities) for the instantiation, the V_a (values domain), if the attribute is allowed to take a number of values (or just one) (see Table 1 for the detailed description of the class). Each context attribute will be characterized by these properties, with different restrictions: the attribute "*RoleOfPersonInGroup*" will need a #Person and a #Group as entities (parameters) and will return a #Role when instantiated, while "*NamePerson*" will need #Person as a parameter and will take as value a String, and so on.

Table 1. Description of #ContextAttribute class

Property Name	Property Type	Domain	Range	Multiple values
name	Datatype	#ContextAttribute	String	No
noEntities	Datatype	#ContextAttribute	Integer	No
entitiesList	Object	#ContextAttribute	#Entity	Yes
valueType	Object	#ContextAttribute	#Entity	No
multipleValue	DataType	#ContextAttribute	Boolean	No

As it can be seen in Table 1, the class #ContextAttribute is a general description of the characteristics of all context attributes. Each attribute is afterwards represented as an instance of this class, with restrictions on each property, in order to have a clear description of the specified attribute. For instance, the context attribute *RoleOfPersonInGroup* is defined with the following values on the properties of the class #ContextAttribute: Name = "*RoleOfPersonInGroup*"; NoEntities = 2 (we need to connect this attribute to a #Person and a #Group); valueType = #Role (value for this attribute is an instance of the class #Role); multipleValues = "false" (a person can only play one role in a group); entitiesList = { #Person; #Group} (connected entities - of type #Entity - are instances of class #Person and of class #Group). Below is the OWL description of the context attribute characterizing the role played by a person in a group.

```
  <ContextAttribute rdf:ID="roleOfPersonInGroup">
        <nameAttribute>rolePlayed</nameAttribute>
        <noEntities> 2 </noEntities>
        <entitiesList rdf:resource="#Group"/>
        <entitiesList rdf:resource="#Person"/>
        <valueType rdf:resource="#Role"/>
        <multipleValues>false</multipleValues>
</ContextAttribute>
```

Given this structure, we can see the *valueOf* of a context attribute as defined in section "Basic definitions" like a function that takes as parameters *noEntities* entities (where each entity has the corresponding type defined in *entitiesList*) and returns one (or several) values of type *valueType.* For convenience and facility we can express a context attribute in the following manner:

nameAttribute (entitiesList) -> valueType(*)

meaning that the valueOf function of the context attribute "*nameAttribute*" needs as parameters a list of entities corresponding to the types mentioned in *entitiesList* and returns one (or several – this is what the star *(*)* stands for) values of type *valueType.*

In Table 2. we present a small part of the list of context attributes defined in the OWL ontology used for our case study application.

The attributes are grouped based on their specificity: attributes that are connected to a person, attributes related to time management, etc..

Table 2. Some of the context attributes defined in MySAM ontology

Person – related	**Time-related**
InterestsPerson :(Person)-> String	TimeZone : (Time) -> Integer
IsSupervisorOf :(Person, Person)-> Boolean	DayOfWeek : (Date) -> String
StatusPerson :(Person) -> String	TimeOfDay : (Time) -> String
Supervises : (Person) -> Person*	
RoleOfPersonInGroup :(Person,Group)-> Role	
Location - related	**Activity – related**
PersonIsInRoom : (Person, Room) -> Boolean	ActivityStartsAt:(Activity)->Time
PersonIsAtFloor : (Person, Floor) -> Boolean	ActivityEndsAt :(Activity)-> Time
PersonIsInBuilding:(Person,Building)-> Boolean	AcivityGoal : (Activity) ->String
PersonIsInCity : (Person, City) -> Boolean	ActivityDuration :(Activity) -> Integer
	ActivityParticipants: (Activity) -> Person*
Agenda - related	**Environment – related**
BusyMorning : (Agenda) -> Boolean	DevicesAvailableInBuilding : (Building) -> Device*
BusyAfternoon : (Agenda) -> Boolean	DevicesAvailableInRoom:(Room)->Device*
BusyEvening : (Agenda) -> Boolean	DevicesAvailableAtFloor : (Floor) -> Device*

As we can see, this ontology is made of two components: one that defines the domain (similar to all other "context" ontologies which are, in fact, *domain* ontologies) and another one, which is the description of all context attribute that will be managed in the current system (by the available Context Manager) – which is the actual *context* ontology in our case.

The representation extension that we propose allows a homogeneous representation of all context attributes in a system, therefore facilitating a generic management of those attributes, and not an ontology-dependent one. This representation has an increased expressiveness as it allows for any type of context attribute (no matter how complex) to be represented. So far, we do not perform any ontological reasoning on the ontologies that we defined, but we used ontologies (and not just XML) to define context attributes in the idea that this type of reasoning will be included in our approach. The part where this kind of reasoning will be useful is when we need to find associations between concepts defined in different context ontologies. A translation between context ontologies would be very helpful taking into account that is highly plausible that two systems will not use exactly the same ontological description for the same context or domain.

Context-aware MAS Architecture

Given the representation of context presented above, let's turn now to the detailed description of the layers 1 and 2 of proposed layered architecture (cf. Fig.1). It is composed of agents (cf. Fig. 4 for their architecture), each of them assisting one user and able to interact with local CMs (cf. Fig. 3). Each of the CMs manages one domain ontology and a set of context attributes. Being connected to the current state of the environment, each CM provides agents with context related information available in the system where the agent is located at that time. The CM (and not agents) is the one that computes the values of all context attributes in the environment. Agents learn how to recognize relevant context and how to act accordingly. We describe now the CM and the context-based agents that learn how to choose and how to use context.

Context manager (CM)

The main functionalities of a CM (fig. 3) are twofold: (i) to let the agents that enter the system, know which are the context attributes defined in the ontology and that are managed by the CM, and (ii) to compute the corresponding IRAS from the RAS given by the agents at some point of processing.

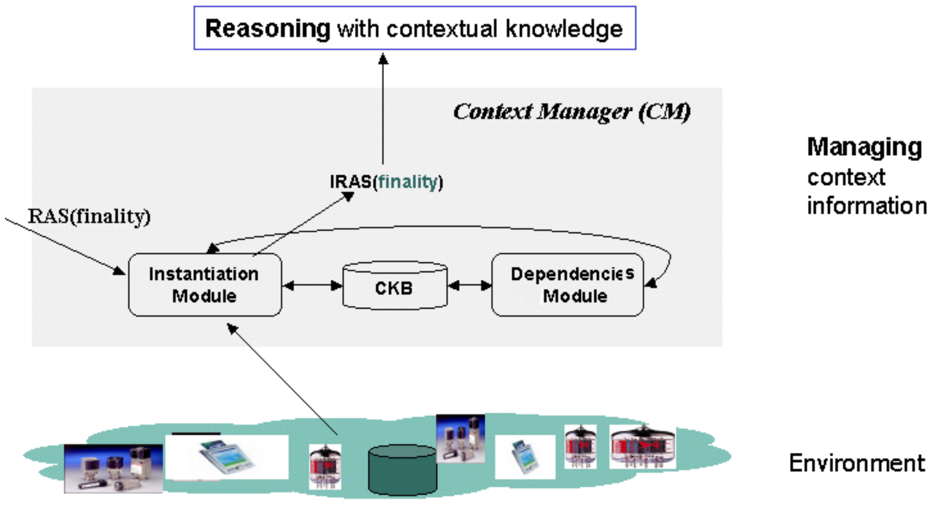


Fig. 3. Context manager architecture

The CM is composed of a *Context Knowledge Base* (CKB) that contains the ontology of the domain, defined as a hierarchy with #Entity as root, on one hand, and all context attributes defined as subclasses of the class #ContextAttribute as we defined in the previous section on the other hand. The management of the context corresponding to this ontology will be insured by the CM. The *instantiation* module computes the *IRAS(f)* for a given *RAS(f)*. The *dependencies* module computes the values for higher level context considering possible relations between context attributes.

Context-aware learning agent

The architecture of the context-based agents operating on the layer 2 is general and it is not restrained to the kind of application considered to illustrate our approach. Even though the agent has several other modules (negotiation, interaction, user interface, etc.), in what follows we will focus on the *selection* and *decision* modules (cf. fig. 4), as these are the ones dealing with contextual information.

The *selection* module aims at choosing relevant attributes for a certain finality f (RAS(f)). For example, for a finality relative to deciding whether accepting or not a "2 participants" meeting, the *selection* module builds the RAS={"ActivityScheduled

InSlot", "roleOfPersonInGroup"}; or, for a finality relative to a "seminar", the RAS can be {"ActivityParticipants", "Activity Description", "PersonInterests", etc}. By making this module explicit, we intend to show that agents are thus able to adapt their behavior dynamically and improve the system's performances.

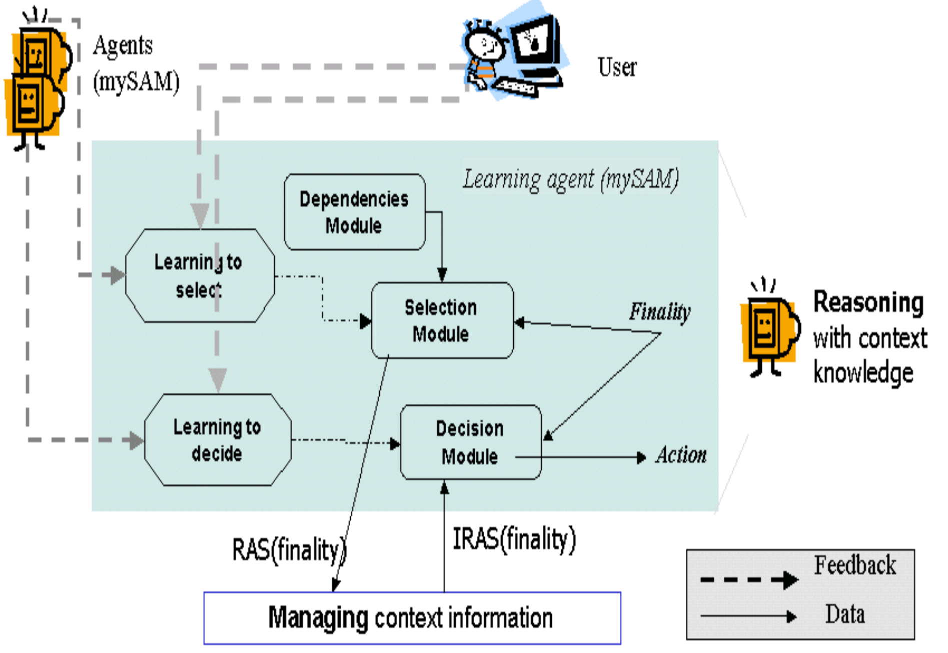


Fig. 4. Context-aware agent architecture

The *decision* module decides whether to accept or reject a meeting based on the IRAS resulting from the respective RAS instantiation. Going back to the previous example, if we have nothing planned for that time slot and if the person that proposes this meeting is our hierarchical superior, the decision module will propose to accept the meeting.

Both the selection and the decision modules have associated learning methods (individual as well as cooperative learning) that improve their behavior. We use the term "*individual*" learning to designate the learning done by the agent only in relation with its user. "*Cooperative*" or "*multi-agent*" learning is realized through exchanging knowledge (concerning either the *selection* task or the *decision* one) between agents. We will show in the following how by using individual learning and knowledge sharing techniques, agents are able to make a better selection of RAS sets for a given finality and to take better decisions based on obtained IRAS.

Learning to reason with context to make decisions

Several approaches have been proposed recently ([29], [31]) concerning multi-agent learning. The use of individual learning methods in MAS is already known as indispensable if we desire to have adaptable and efficient agents. Multi-agent learning is being considered for quite a while already, and different methods were already proposed. What we propose here are methods to collectively learn to choose the "right" context and what to do with it. In what follows, we present the learning modules for the selection of relevant attributes and for using context knowledge for decision-making.

We want to underline that the presentation of these modules is made from a general point of view, without going into details in what concerns the specific learning method that is used for mono-agent learning (this choice is usually made taking into account the application). Our objective is to argue the necessity to use mono and/or multi-agent learning (as described in [24], [31]) when dealing with context (for both selection of relevant context and decision-making task) and to imagine multi-agent learning mechanisms suitable for context-aware applications. We are not proposing a new learning method, but show how using existing methods can improve the decision making process in a context-aware system.

Learning to select what is relevant for a decision (RAS)

Learning how to choose among context attributes which are the relevant ones could be very important in an application where the amount of available context information is too large and the effort needed to compute the values for all those attributes is not acceptable in term of efficiency. We improve the agent's capacity to choose the relevant information to assess this phase by an individual learning method (in connection to user's feedback) and also by a collective one.

For *individual learning* (mono-agent learning) on how to choose among context attributes those who are relevant for a given situation, the agent uses the *feedback from the user*. In our case study, the user chooses attributes which he considers as being relevant for the situation, before making a decision. When the user chooses more attributes as being relevant for a type of meeting for instance, the agent memorizes them. Next time the agent will have to deal with the same type of meeting, it will be able to propose to its user all previously known relevant attributes, so that the user adds or deletes attributes or uses them such as they are. For example, the agent received a proposal for a "family meeting". Considering this as a finality for which it has to choose a RAS, the agent needs to find the RAS("family-meeting"). Itthen asks for the CM to instantiate it, in order to obtain the IRAS("family-meeting"), and to make the decision based on this IRAS. Let's suppose that, for now, the RAS("family-meeting")={"ActivityStartsAt"}. MySAM will propose the user this RAS and will ask if it is enough to make the decision based only on this set of relevant context attributes. The user can further add to the RAS the attribute "ActivityImportance", so that not only the start time of the meeting will be relevant. The new RAS("family-meeting") is now {"ActivityStartsAt", "ActivityImportance"}. Next time a family meeting proposal will be received, the agent will list this new RAS, so that the user can choose to remove one of the attributes or to add new ones.

In our approach, we have improved the method used in individual learning for selection of relevant context attributes by a knowledge sharing between agents : agents may use the RAS that other agents in the system have already learnt in the situation they face. . If an agent does not know which attributes could be relevant for the situation in which it is, probably for the first time, it can ask other agents which are the attributes which they already know as being relevant. In the same way, if an agent had not had a lot of feedback from its user on attributes in a specific situation, it can again try to improve its set of relevant attributes, by asking for others' opinion. In the example given above, the agent can choose to ask other agents for their RAS("family-meeting"), then make the union of all RAS received as a response to its request, and propose this enlarged set to its user for further modifications. This method adds to the ability of the agent to propose new possible relevant attributes. The user's task becomes easier, in the way that he just has to check for the relevance of the proposed attributes, without going through the whole list of available context attributes to select possible interesting ones.

Learning to use instantiated context (IRAS) to make decisions

Using the IRAS retrieved from the CM is similar to a simple decision making process. To improve it, we used here also mono and multi agent learning to enhance the agent with the ability to evolve and to become more and more user-personalized.

Mono agent learning of how to use relevant context can be, therefore, any machine learning method developed in Artificial Intelligence which is suitable to the type of application that we want to develop. For our case study we chose the classification based on association (CBA) tool developed in the Data Mining II suite ([6]). In the "Implementation and Results" section we explain our choice and we give further details in what concerns this method.

The difficulty of multi-agent learning is due to the fact that agents must have a common apprehension of the manner in which to use context attributes and knowledge. To be able to understand what other agents say, every agent must use the same 'language', the same manner of encoding information. They will therefore need to share at least some parts of the domain ontology, to make sure that agents understand the contents of exchanged messages. For multi-agent learning on how to use IRAS(f), we chose a very similar knowledge sharing method to that presented before. The protocol proposed for this distribution can be the same, the difference consists in the knowledge that agents choose to share with others: they can choose to share only the solution to the problem and not reveal the way they reasoned to find that solution, or they can share directly their problem-solving method, so others can use it for themselves. The choice depends on the system. If agents should be cooperative and it is considered that no privacy matters should be taken into account, then the second solution could be faster, in the sense that next time the agent can simply use the rule, and not ask again for the solution. But if, as considered in our case study, the agents should not share all their criteria for accepting or rejecting a meeting (for sure we don't want the boss' agent to know that we refuse the meeting because we don't feel like getting up that early…), then sharing just the solution (, an "accept/reject" decision) could be preferable.

Using both individual and multi-agent learning for choosing and using relevant context in making decision improves our system's openness. Agents can come in and out of a society of agents with the context adjusting accordingly and without causing problems to the general use of the system. The decision making process is as well an individual process as a collective one. Agents try to solve the agenda managing problems on their own first, and then ask for opinions and propose a solution accepted by a majority of agents in the system, based on a voting procedure.

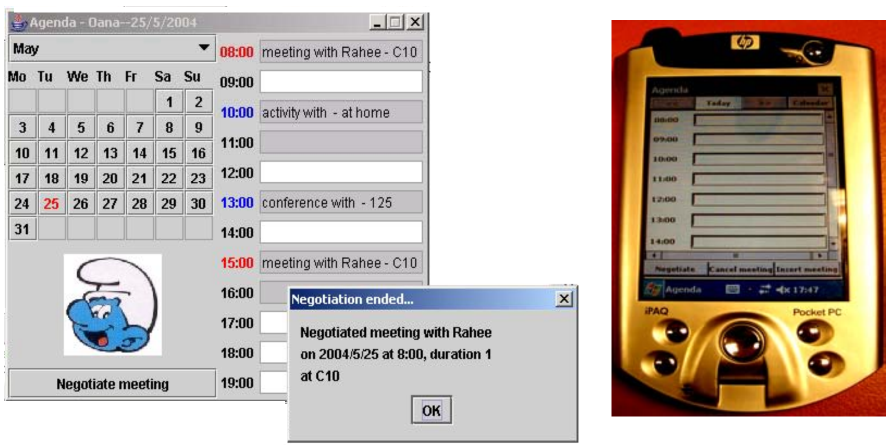
We underline the fact that, for choosing RAS(f), as well for making decisions based on the IRAS(f), the user has the possibility to approve or reject the agent's proposal. What the agents do is to propose solutions (actions), but it is always the user who makes the final decision. Of course, the goal would be to have an auto-customizing agent that will finally get the user's trust and will act on its own. In decision support system the idea of autonomy is not to be considered; the system is supposed to support the decision making process, not to bypass the user's approval. However, for multi-agent systems, one of the challenges is to develop autonomous agents that are able to accomplish tasks as a replacement for human involvement. For now, we propose a decision *support* agent (not an *autonomous* one), considering that giving the user the possibility to reject the agent's proposals helps increasing the chances for the agent to be accepted without difficulty. In what follows, we detail implementation issues and give some results we obtained when testing our case study application.

Implementation and results

In order to validate our proposal, we developed a case study system (as detailed in section 2), using a multi-agent system containing several mySAM agents and one CM. To deploy our agents we used the JADE platform ([11]).

JADE (Java Agent DEvelopment Framework) is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines (which not even need to share the same operating system) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. JADE is completely implemented in Java language and the minimal system requirement is the version 1.4 of Java (the run time environment or the JDK). Besides being the most used platform in MAS community at this time, JADE also proposes an add-on for deploying agents on hand-held devices, called LEAP.

We used JADE to develop mySAM agents, capable of handling the negotiation task for scheduling meetings. Each mySAM agent is a JADE agent with a graphical interface (see Fig. 5) that allows a user to manage her agenda. This graphical interface has been simplified for mySAM agents on a HP iPAQ 5550 Pocket PC. We underline the fact that thanks to the layered system's architecture, we are able to develop context-aware agents that are light-weighted, therefore adapted to hand held devices. This is due to the division of functionalities: context *management* done by an external entity (not the agent itself) and all task-related *reasoning* and *learning* done at the agent level. MySAM agents deployed on PDAs execute only the negotiation task, without any learning methods. Learning is done by a remote agent situated on a PC. This makes multi-agent sharing process even more crucial, as an agent might find itself disconnected from its remote agent that deals with all learning and reasoning modules. In this case, the agent is forced to ask for help from other agents, in order to be able to aid its user.


Agenda - Oana--25/5/2004

08:00	meeting with Rahee - C10
09:00	
10:00	activity with - at home
11:00	
12:00	
13:00	conference with - 125
14:00	
15:00	meeting with Rahee - C10
16:00	
17:00	
18:00	
19:00	

Negotiation ended...

Negotiated meeting with Rahee on 2004/5/25 at 8:00, duration 1 at C10

Fig. 5. MySAM interface – on PC (left image) and on HP Pocket PC (right image)

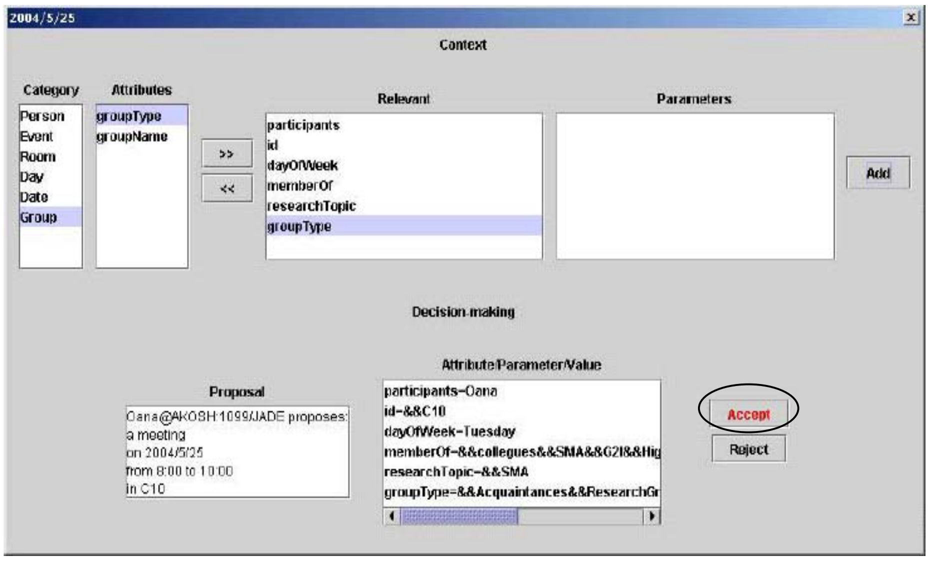
As soon as an agent receives a meeting proposal, it lists to the user the context of that proposal (see Fig. 6), which contains: the proposal, the known relevant attributes to that situation (RAS(f)), their instantiation (IRAS(f)) and the suggested decision (accept/refuse). The suggestion is written in red in the interface – circled on the print screen image in Fig. 6 (in that case, for example, mySAM proposes to accept the meeting). The user can add or remove context attributes to/from current RAS(f) and restart the instantiation process (to obtain the new IRAS) and the search for the appropriated decision in that case.

Each meeting proposal that is accepted or rejected by user is added to an example data base. This example base is used further on by the agent to improve its performance (in predicting user's decision) by learning. MySAM agents use CBA (Classification Based on Association) algorithm to *learn how to use* relevant context (for acceptance or refusal of meeting proposals).

CBA (v2.1) has many unique features, the one that interests us being the classification and prediction using association rules. It builds accurate classifiers from relational data, where each record is described with a fixed number of attributes. This type of data is what traditional classification techniques use, e.g., decision tree, neural networks, and many others. It is proved to provide better classification accuracy (compared to CBA v1.0, C4.5, RIPPER, Naive Bayes) [6]. We used this method also because it generates behaviour rules comprehensive for both agents and humans. We have developed a module for the conversion of the rule from the CBA format:

```
Rule 6:   busyAfternoon = false
          durationEvent = 60
          groupType = work
```

```
                  -> class = yes
```



Person
Event
Room
Day
Date
Group

groupType
groupName

participants
id
dayOfWeek
memberOf
researchTopic
groupType

Oana@AKOSH:1099/JADE proposes:
a meeting
on 2004/5/25
from 8:00 to 10:00
in C10

participants-Oana
id-&&C10
dayOfWeek-Tuesday
memberOf-&&collegues&&SMA&&G2I&&Hig
researchTopic-&&SMA
groupType=&&Acquaintances&&ResearchGr

Fig. 6. Interface for selecting relevant context attributes for a precise situation

into a CLIPS format. CLIPS [3] is a productive development and delivery expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. Although CLIPS also provides two other programming paradigms: object-oriented and procedural, we used the simple rule based one. Rule-based programming allows knowledge to be represented as heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. A rule in CLIPS is specified in the following style:

```
(defrule Rule6
     (busyAfternoon  false)
     (durationEvent  60)
     (groupType  work)
            => (store CLASS yes))
```

"Class" specifies whether the agent should accept or refuse the proposed meeting. After transforming rules obtained with CBA in CLIPS rules, we could use Jess ([13]) inference engine. Jess was originally inspired by the CLIPS expert system shell, but has grown into a complete, distinct, dynamic environment of its own. Using Jess, one can build Java software that has the capacity to "reason" using knowledge supplied in the form of declarative rules. Jess is small, light, and one of the fastest rule engines available. The core Jess language is still compatible with CLIPS, in that many Jess scripts are valid CLIPS scripts and vice-versa.

Each agent has a module that deals with the rules generated by CBA in order to find out which rule can be applied in a specific context. The process is the following: giving a finality *f*, mySAM chooses the RAS(*f*), then asks for the IRAS associated with the RAS (the request is made to the CM). The obtained IRAS constitutes the current relevant context to be taken into account when making decisions. MySAM asserts as facts the current context in the form:

```
(defrule startup  =>
     (assert ( "attribute_1"  "value_attribute_1"))
…    (assert ( "attribute_n"  "value_attribute_n")) )
```

Then, mySAM starts Jess inference engine to find the value of “CLASS”. The value can be ‘Yes’ (meaning that the meeting proposal should be accepted), ‘No’ (meaning that the meeting proposal should be refused) or ‘Unknown’ (no rule matched the specific context).

When no rule matches the specific context (no rule is activated when using Jess), mySAM uses *a multi-agent knowledge-sharing* process to find out how to use this specific context (IRAS) to decide. The agent starts a voting system: it asks all known agents in the system for their opinion on the situation (the situation being defined as {***f***, *IRAS*(***f***)} = (***f***, {(attribute_1, value_attribute_1),…, (attribute_n, value_attribute_n)})) and counts each opinion as a vote for “accept”, “reject” or “unknown”. The agent then proposes to the user the decision that has the most votes. Agents consider an “unknown” result as a “reject” (by default, an agent will propose the user to reject all meeting proposals that neither it, nor other agents know how to handle). We chose to use this “voting” procedure because it was easier than trying to share the rules for themselves and because, in this kind of application, the privacy is important. Not all agents will want to share their decision-making rules, but an “accept/reject/unknown” answer is reasonable to be shared. In this way, agents do not explain how they inferred that decision, but just what the decision is. However, if rule sharing is to be taken into account, there are several important details that should be addressed: using common vocabulary, understanding the encoding of the rule, managing privacy, etc.

The context manager (CM) is also implemented as a JADE agent. CM is a special agent in the system that has access to the domain ontology that defines the concepts that characterize the domain and the context attributes that the CM will manage. CM provides context knowledge to all agents that are active in the system. The ontology was created using Protégé 2000 (Protégé is a free, open source ontology editor and knowledge-base framework [21]) and the agent accesses the ontology using Jena ([12]), a Java framework for building Semantic Web applications. Jena provides a programmatic environment for RDF, RDFS and OWL, including a rule-based inference engine.

Agents interactions in the system are quite simple: mySAM agents can query the CMA using a REQUEST/INFORM protocol, the meeting negotiations between mySAM agents are done in a PROPOSE/ACCEPT/REJECT manner.

When testing mySAM we were able to draw several conclusions, such as:

- using a selection step to choose the RAS for a situation helps in having smaller and more significant rules. Using all attributes to describe a situation is not only difficult to deal with, but also unnecessary. We tested our hypothesis on a set of 100 examples. For 15 context attributes used, we obtained an overall classification error of 29.11% and more than 40 rules. When we split the example set on several finalities (“family-meeting”, “friends-meeting”, “work-meeting”), and for each situation we take into account a limited number of context attributes (7 for a meeting with family, 11 for the two others), the error becomes 7.59% and the number of obtained rules drops to an average of 15;
- sharing with other agents just the solution (accept/reject) for a situation is enough, because the agent that received the answer will then add this situation to its examples base, from where it will then learn the

appropriate rule. Even if it will take longer to learn that rule than just having it immediately provided by others, the privacy problem is this way solved, because we share just the answer to a specific situation, and not the reasoning that produces such an answer. Also, the user has the possibility to decide otherwise (if the decision is not suitable for him/her). Whereas, if agents shared rules and they added those rules directly into their reasoning modules (without user's consent), this could introduce a significant amount of errors in agent's decision making.

Related work – comparison

Our study is centered on multi-agent systems, but our definition and vision on context is based more on approaches in other domains, like Artificial Intelligence, decision support systems, human-machine interaction, ubiquitous computing, etc.

We can mention here some definitions given by: Persson [20] ("context is […] the surrounding of a device and the history of its parameters"), Brezillon [1] ("the objective characteristics describing a situation,[…] the mental image generated, […] the risk attitude" used to make decisions) or Turner **[28]** ("Any identifiable configuration of environmental, mission-related and agent-related features" used to predict a behaviour). We proposed in basic definitions section a definition that is quite similar to all these definitions in the sense that it is based on: (i) the elements that compose the context and (ii) its use, i.e. the finality (the goal we want to achieve) when using this context. The definition we proposed takes into account those two dimensions of context (its use and its elements); it also explains what each dimension is and how to properly define it when designing a context-based MAS.

In MAS, the notion of context is used to describe the factors that influence a certain decision. In similar applications (meeting schedulers), context means: type of event, number of attendees, etc. (Calendar Apprentice [17]), activity, participants, location, required resources (Personal Calendar Agent [15]), system load, organization size (Distributed Meeting Scheduler [24]), time, user's location, etc. (Electric elves [23]). None of these works mention the idea of context but they all use "circumstances" or "environmental factors" that affect the decision making.

In making Calendar Agent ([14]), Lashkari et al. use the notion of context, but they assume that relevant context is known in advance, so that all contextual element that they have access to is considered relevant for the decision to be made. This is the major problem with the way context is used: these approaches are not fitted for an application independent way of handling context, because they do not provide a general representation of context knowledge and methods to choose between relevant and non-relevant context elements for a specific decision.

The main difference and contribution of our work is in the sense that we propose a MAS architecture based on an ontological representation of context and that can permit individual and multi-agent learning on how to choose and how to use context. It is not the choice of an application that generated this architecture, but MySAM is just a case study to validate our approach.

Conclusions and future work

In this paper, we have presented a definition of the notion of context, notion that is used in most of the systems, usually without precisely and explicitly taking it into account. We have proposed an ontology-based representation for context elements and a context-based architecture for a learning multi-agent system that uses this representation. We then validated our approach by implementing a meeting scheduling MAS that uses this architecture and manages and learns context based on the definitions and representation we proposed. We proved that embedding the selection of relevant information into the agent itself improves the agent's performance. Furthermore, adding just a very simple learning method to the selection module makes the agent behave more appropriately, by revising its behavior dynamically.

As future work, we aim to extend this framework for context-based MAS to be used for any kind of application that considers context when adapting its behavior. The context manager has to be able to deal *automatically* with all context-related tasks (including the computation of context attributes values) and to share all his context-related knowledge. In order to make this possible, our future work will focus on representing and managing: (i) how to compute the values for derived context attributes (how to define the *valueOf* function), (ii) dependencies that can exist between context attributes (how to compute higher level context information from lower level one), (iii) the importance of different attributes in different situations (making a more refined difference between relevant and non relevant attributes, using a degree of relevance, not just relevant/non-relevant distinction).

In what concerns learning agents, the framework will provide agents with at least one individual learning algorithm and all that is needed to communicate and share contextual knowledge (how to choose, compute and use context to make decisions).

In order to prove the generality of our approach, we will implement it to solve different other scenarios of context-aware applications that can use multi-agent technology.

References

[1] Brezillon, P. – "Context Dynamic and Explanation in Contextual Graphs", In: Modeling and Using Context (CONTEXT-03), P. Blackburn, C. Ghidini, R.M. Turner and F. Giunchiglia (Eds.). LNAI 2680, Springer Verlag (http://link.springer.de/link/service/series/0558/tocs/t2680.htm). pp. 94-106.

[2] Chen, H., Finin T., Anupam J. – "An Ontology for Context-Aware Pervasive Computing Environments", The Knowledge Engineering Review Volume 18 , Issue 3, p. 197 – 207, 2003.

[3] CLIPS (C Language Integrated Production System) - http://www.ghg.net/clips/CLIPS.html

[4] Coutaz J., Ray G. –"Foundations for a theory of contextors", In Proc. CADUI02, ACM Publication, pp. 283-302, 2002.

[5] Coutaz J. et al – "Context is key", Volume 48 , Issue 3 (March 2005) The disappearing computer, SPECIAL ISSUE: The disappearing computer, Pp: 49 – 53, 2005.

[6] Data Mining II – CBA - http://www.comp.nus.edu.sg/ ~dm2/

[7] Dey, A., Abowd, G.– "Towards a better understanding of Context and Context-Awareness", GVU Technical Report GIT-GVU-00-18, GIT, 1999.

[8] Edmonds B. – “Learning and exploiting context in agents”, in proceedings of The First International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, pag. 1231-1238.
[9] Gonzalez A., Ahlers R. – “Context based representation of intelligent behavior in training simulations”, Transactions of the Society for Computer Simulation International, Vol. 15, No. 4, p. 153-166, 1999
[10] Henricksen K., Indulska J., Rakotonirainy A.– “Modeling Context Information in Pervasive Computing Systems”, Proc. First International Conference on Pervasive Computing 2002, p. 167-180, Zurich.
[11] JADE (Java Agent Development framework) : http://jade.cselt.it/
[12] Jena Semantic Web Framework - http://jena. sourceforge.net/
[13] Jess: http://herzberg.ca.sandia.gov/jess/index.shtml
[14] Lashkari Y., Metral M., Maes P.– “Collaborative Interface Agents”, Proc. of the Third International Conference on Information and Knowledge Management CIKM'94, ACM Press, 1994.
[15] Lin S., J.Y.Hsu – “Learning User’s Scheduling Criteria in a Personal Calendar Agent”, Proc. of TAAI2000, Taipei.
[16] Matsatsinis N.F., Moratis P., Psomatakis V., Spanoudakis N. – “An Intelligent Software Agent Framework for Decision Support Systems Development”, ESIT ´99 (European Symposium on Intelligent Techniques).
[17] Mitchell T., Caruana R., Freitag D., McDermott J., Zabowski D.– “Experience with a learning personal assistant”, Communications of the ACM, 1994.
[18] OWL - http://www.w3.org/2004/OWL/
[19] Ossowski S., Fernandez A., Serrano J.M., Hernandez J.Z., Garcia-Serrano A.M., Perez-de-la-Cruz J.L., Belmonte M.V., Maseda J.M. – “Designing Multiagent Decision Support System. The Case of Transportation Management”, ACM/AAMAS 2004, New York, pp 1470-1471.
[20] Persson P.– “Social Ubiquitous computing”, Position paper to the workshop on ‘Building the Ubiquitous Computing User Experience’ at ACM/SIGCHI’01, Seattle.
[21] Protégé 2000 - http://protege.stanford.edu/
[22] Ryan N.– “ConteXtML: Exchanging contextual information between a Mobile Client and the FieldNote Server”, http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html
[23] Scerri, P., Pynadath D., Tambe M.– “Why the elf acted autonomously: Towards a theory of adjustable autonomy “ , First Autonomous Agents and Multi-agent Systems Conference (AAMAS02), p. 857-964, 2002.
[24] Sen S., E.H. Durfee – “On the design of an adaptive meeting scheduler”, in Proc. of the Tenth IEEE Conference on AI Applications, p. 40-46, 1994.
[25] Sian S. S. – “Adaptation Based on Cooperative Learning in Multi-Agent Systems”, Descentralized AI, Yves Demazeau & J.P. Muller, p. 257-272, 1991.
[26] Tao Gu, Xiao Hang W., Hung K.P., Da Quing Z.– “An Ontology-based Context Model in Intelligent Environments”, Proc. of Communication Networks and Distributed Systems Modeling and Simulation Conf., 2004.
[27] Turney,P. – “The identification of Context-Sensitive Features: A Formal Definition of context for Concept Learning”, 13th International Conference on Machine Learning (ICML96), Workshop on Learning in Context-Sensitive Domains, p. 53-59.
[28] Turner, R. – “Context-Mediated Behaviour for Intelligent Agents”, International Journal of Human-Computer Studies, vol. 48 no.3, March 1998, p. 307-330.
[29] Sian S. S. – “Adaptation Based on Cooperative Learning in Multi-Agent Systems”, Descentralized AI, Yves Demazeau & J.P. Muller, p. 257-272, 1991
[30] Willmott S., Calisti M., Rollon E. –“Challenges in Large-Scale Open Agent Mediated Economie”, Proc. of Workshop on Agent Mediated Electronic Commerce AAMAS 02, july 2002.

[31] Weiss G., Dillenbourg P.– “What is “multi” in multi-agent learning?”, P. Dillenbourg (Ed) Collaborative-learning: Cognitive, and computational approaches, p. 64-80, 1999