A community based metaphor supporting EUD within communities

Marco P. Locatelli
Università degli Studi di Milano-Bicocca
Viale Sarca 336/U14
Milan, Italy
locatelli@disco.unimib.it

Carla Simone
Università degli Studi di Milano-Bicocca
Viale Sarca 336/U14
Milan, Italy
simone@disco.unimib.it

ABSTRACT
The paper proposes an approach to EUD focusing on communities: on the one hand, users are considered as a community of cooperating actors; on the other hand, applications and devices are considered as a (artificial) community that, by interacting with those actors, cooperate to support their collaboration. This idea facilitates an integration based on modularization and abstraction, by applying to the technology principles that are derived from human collaboration: in so doing, it offers a uniform approach that allows users with different technical capabilities to operate at different levels of abstraction by applying the same conceptual tools. The community metaphor inspires a model that helps the users community integrating devices and applications according to its needs. In turn, the model defines a layered structure of primitives: from the basic ones up to the domain and application dependent ones, passing through the domain and application independent primitives. The approach is illustrated by an example in the education domain: the interaction flows in an augmented classroom can be defined and adapted by communities where teachers, tutors and students organize their collaboration.

Categories and Subject Descriptors
H.5.3 [Group and Organization Interfaces]: Collaborative computing

General Terms
Design, Human Factors

1. INTRODUCTION
We are witnessing an increasing number of situations where users need to negotiate their behavior, and a suitable technology to support it, with other users, as members of more or less consolidated communities. This might happen in a corporate context where part of the activities can be autonomously organized by groups of people under the hypothesis that some corporate outcomes or time constraints are satisfied; the same can happen in the education domain where all the involved roles (teachers, students, tutors, supervisors, etc.) interact to prepare and manage a course and the lessons it encompasses; finally, the same can happen within communities whose members interact through the WEB to reach their goals (sharing knowledge, guaranteeing mutual support, or simply sharing a common interest). These communities can take various forms and have a varying life span: however all of them have to establish some basic behavioral rules and conventions to survive and be effective toward their members, and to organize how existing technologies are to be integrated in order to improve the community capability to reach its goals. In fact, even in the case in which the community is part of a broader organization, very rarely the latter is likely to invest to support it and very often the community itself asks for a high degree of self-organization. In this paper we take for granted that the community exists in one of its possible phases [6] and that some technologies have been identified as useful for its functioning. Now, the problem for the community is to define (or even redefine) how the cooperative behavior of its members is harmonized and supported by the integrated behavior of these technologies.

Empirical studies, especially in the CSCW framework, have uncovered that the mechanisms supporting coordination among cooperative actors can be described as the composition of artifacts and protocols in combination with the awareness of what is going on in the context [9]. Artifacts contain the information useful to characterize the state of affairs in cooperation; the protocols describe how this information has to be maintained and what has to be done in front of predefined situations (the combination of artifacts information and of coordinative events generated by the actors and the context); finally, awareness is promoted by an additional and specific kind of information that can be derived from the context and helps orient the protocol execution in a context-aware way. From a complementary perspective [3], those artifacts can be part of a Common Information Space (CIS) that cooperating actors access to get the information needed to execute coordinated activities and to define their meaning in the cooperation context.

The point of this paper is that since actors are familiar with the above mechanisms (actually, they have been distilled from the observation of the way in which actors cooperate) it should be natural for them to apply the same principles not only to express their mutual behavior but also how the
technologies interact to support it. In other words, we propose to use the same concepts identified within human cooperation to build a metaphor supporting the “cooperation” of different pieces of technology (applications, devices, and so on). On the one hand, this approach is a way to orchestrate the services (or better, the interaction capabilities) these pieces of technology make available; on the other hand, this orchestration is smoothly integrated with the coordinative mechanisms characterizing actors cooperation since both are defined at the same time, by means of the same conceptual tools (see Figure 1). Actually, actors and pieces of technology are “members” of the same community, obviously with different roles and intentionality. In this way, we see pieces of technology as actors with limited behaviors, instead of constraining human behaviors to make them compatible with the capability of the technology. Accordingly, the linguistic tools we are proposing are different from typical scripting languages that are basically machine oriented: instead, we aim to bring to the technology the “controlled flexibility” typical of human cooperation.

2. THE METAPHOR
According to the above approach, we have defined a model, called Community-Aware-MAS (CASMAS), that is based on the metaphor of human cooperation. The concept of community is a first class object (see Figure 2): the members of the community, called entities, are associated with a CIS, called fulcrum, that contains the coordinative information, possibly articulated in coordinative artifacts [7]. Moreover, the fulcrum contains the protocols, that are expressed in terms of behaviors to be assigned to each entity to make it an active member of the community: the use of behaviors is similar as in the Actor Model [2]. Like this model, communication is asynchronous but it is not message based. Instead, entities post a request in the fulcrum, other entities will react to this request according to the behaviors assigned to them. An entity can be member of more than one community: in this case it can play the boundary leadership [10] and allow the exchange of information among the communities the entity belongs to. An entity can be located in one (or more) awareness space (see Figure 2). An awareness space is where awareness information is propagated from a source along the space structure (topology), possibly changing its intensity [in the line of [8, 4]]. Typically, information about presence is propagated in the physical space, while domain dependent information can be propagated in a logical space:

Figure 1: CASMAS provides a uniform view of cooperation.

Figure 2: CASMAS model.

3. THE LANGUAGE
The above model has associated a language to specify entities, their behaviors and the awareness management. This language takes the declarative form of facts and rules (if-then constructs). This choice is motivated by the possibility to express behaviors in a highly modular way, without the need to define complex and exhaustive control structures [5]. Moreover, the basic constructs of the language can be uniformly used to express entities’ behaviors at any level of detail (see Figure 3) where each level uses the primitives defined at the level(s) below. From the basic constructs provided by CASMAS one can construct the domain/application independent primitives, like post a request, assert-a-fact, copy and update-a-space. At the level immediately above, one can express the interfaces (services) provided by domain independent applications/devices: for example, the basic I/O primitives of a interactive table or the basic functionalities of a Document Management System. This corresponds to build a sort of wrapper for the applications/devices to be orchestrated. Then the domain depen-
dent behaviors can be defined at the highest level, possibly with different degree of visibility: the latter are identified by the needs of the community. In fact, they depend on the kinds of technical skill characterizing different classes of users, on the complexity of the orchestration to be realized, and so on. The basic point is that any user applies the same declarative pattern at each level either to define/modify/compose the primitives available at that level, or to enrich them by using the primitives of the levels below. A best practice is to organize the domain application levels so that it is clear what they make available and the knowledge required to operate at their level: the knowledge only, since the language is always the same.

4. THE METAPHOR AT WORK
Consider a scenario where a class splits into groups in order to develop projects related to a specific course. According to the proposed approach, the interactions involving teacher and students can be supported by the integration of entities behaviors, which belong to a number of communities: one contains the teacher and all the students (called Course); the others are identified by each group of students (one generic community like these ones is called Student). Within each Student community, a student plays the role of project-leader while the other students play the role of participant. To support role based awareness, each Student community owns a Roles space with two locations, one for each of the two roles. In the following entities are denoted by the role they play (in italic). In Figure 4 the various linguistic items required to implement the the scenario are located at the appropriate abstraction level.

project-leader and teacher are the only entities that are always connected to the Course community while the other students are connected only when they participate to a lesson. Among other things, project-leader is in charge of notifying the Student community under its responsibility that a deadline has been defined in the Course community (usually by the teacher). This leads to the following pre-defined rules for project-leader:
// bridge rule to transfer
// information between communities
when there is a deadline in Course
then copy deadline to Student

// rule to notify the community members
when there is a deadline in Student
then make aware of the deadline in Roles
By the second rule, project-leader generates the awareness information about the deadline from the leader site in Roles; the information is then propagated with a lower intensity to the participant sites where participants are situated: they perceive the awareness information only if a participant explicitly declared its interest by setting an appropriate threshold of sensitivity to this kind of information.

participant behavior is predefined so as to have the deadline shown on some device (not specified here) if it has been made aware of this event, as expressed by the following rule:
// rule for participant
when aware of a deadline in Roles
then inform ACTOR about deadline
where ACTOR is set by default to the instance of the student playing that role.

Instead, the project-leader is informed only if it did not delegate its role to a participant (there is no delegation):
// rule named RESP-AWARE for project-leader
when aware of a deadline in Roles
and there is no delegation in Student
then inform ACTOR about deadline
The last rule shows the use of high-level primitives (e.g., inform) that implement the desired effects (e.g., to inform someone) and hide the complexity of the implementation (and coordination) they imply. In fact, for example, inform implies that participant posts a request to show the deadline to the person it represents and that, according to the current configuration, the appropriate devices accomplishes this task.

Two rules (not shown here) manage the effects of a delegation: one moves the delegated person to the project-leader site in the Role space; the other moves it back to the participant site when the delegation is revoked.

A predefined behavior can be personalized by a rule defined by a user (through Drools [1]). For example, one of the project-leaders is abroad: he needs to delegate his role (as shown before) but he still wants to be informed of the deadline via a SMS. The following rule overwrites the default behavior defined by the above RESP-AWARE rule.
// rule personalizing a project-leader behavior
when aware of a deadline in Roles
and there is a delegation in Student
then send SMS to ACTOR about deadline

4.1 New functionalities development
One of the students owns advanced skills about rule development. Because his community would like to have a (virtual) post-it displayed on the smart desk for each approaching deadline, the skilled student makes this feature available at the domain dependent level (see Figure 3); in so doing, this solution can be available to other interested communities too. Some steps have to be done; first, the student defines what an “approaching deadline” is: the community agrees that a deadline is approaching when it will expire in the next 10 days.
// condition definition
approaching deadline= deadline:Deadline(daysFromToday<10)

Then he defines a function to create a post-it for a deadline that is approaching:
// function definition
create deadline post-it=show text box,
set text to deadline, bgcolor to "yellow"
where show text box is a device dependent service provided by the smart desk: when it is invoked the smart desk displays the text box in an empty position.

According to an agreed convention the student chooses to render a post-it as a text box with a yellow background: this means that each text box with a yellow background will be considered a post-it by this students community. A perhaps more professional solution would be to define an explicit post-it type instead of using an affordance prop-
<table>
<thead>
<tr>
<th>Domain DEP.</th>
<th>Conditions</th>
<th>Functions</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart Desk</td>
<td>deadline</td>
<td>inform, create deadline</td>
<td>All rules (- “last rule”)</td>
</tr>
<tr>
<td>Device/App. DEP.</td>
<td>show, move “last rule”</td>
<td>GSM Gateway send SMS</td>
<td>DMS upload, download</td>
</tr>
<tr>
<td>Domain/Device/App. INDEP. CASMAS</td>
<td>assert-a-fact, update-a-space, post a request, copy</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>assert, retract, modify, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Mapping of conditions, functions and rules to the abstraction levels.

Finally, he defines the domain dependent rule that creates a deadline post-it for each new approaching deadline:

```plaintext
// rule definition for the smart desk
when approaching deadline
then create deadline post-it.
```

After this community used the post-it feature for a while, the students agreed that it would be better if the post-it is displayed in a particular area of the screen. Consequently, the skilled student defines an application dependent rule for the smart desk that manages the post-it positioning.

```plaintext
// rule definition for the smart desk
when text box with bgcolor equal "yellow"
then move text box at the top-left area
```

An alternative solution could be that the skilled student redefines the application dependent function for the smart desk to include the post-it positioning.

```plaintext
// function definition
create deadline post-it=show text box,
set text to deadline, bgcolor to "yellow",
position to "top-left"
```

5. CONCLUSIONS
The metaphor/language proposed to define the integration of applications/devices useful for a (partially) self-organized community is based on two ideas: to propose the concepts actors normally use in coordinating their activities and in associating an action to a given situation. The language allows for different levels of adaptation: from modifying/constructing the left part of a rule up to define richer primitives to be used in the right part. The next steps concern the definition of a visual framework that supports the presentation of the community’s support structure (which behaviors are associated to which entity at which level), to go beyond the currently used rule editor and to extend the experimentation beyond groups of students with different skills.

6. REFERENCES