End-users' integration of applications and devices: a cooperation based approach

Marco P. Locatelli and Carla Simone

Abstract T

he current organizational and technological evolution suggests to conceive tailorability and EUD also in terms of integration of off the shelf applications and devices that support collaboration. To this aim this chapter proposes an approach that leverages the ability of actors to coordinate their activities and then grounds integration on the notion of cooperation. The resulting technological environment is presented and illustrated through a case derived from an ongoing project. Some considerations derived from a short experimentation conclude the chapter.

Key words: EUD, Tailorability, End-user, Cooperation, Coordination, Awareness, Integration

1 Introduction

Environments supporting tailorability and End Used Development (EUD) are called nowadays to offer their functionalities in a different scenario with respect to earlier approaches and solutions [12]. From a technological point of view, we are witnessing an explosion of open source software applications: they are not always reaching the desired level of robustness and quality of documentation but surely they constitute a reference asset that is going to increase its quality and re-usability. In any case, today almost all available applications guarantee their openness through API (Application Programming Interface).

Marco P. Locatelli
University of Milano-Bicocca, viale Sarca 336, Milan, Italy, e-mail: locatelli@disco.unimib.it

Carla Simone
University of Milano-Bicocca, viale Sarca 336, Milan, Italy, e-mail: simone@disco.unimib.it

More interestingly, from the organizational point of view, companies need to be flexible enough to timely react to external (and sometimes internal) changes. This means that an organization survives if it balances in an effective way two complementary activities: the management of the corporate processes and information that support its mission and guarantee its overall robustness; and the management of the day-by-day activities that guarantee the fulfillment of those general goals by a distributed problem solving and coordination that involves groups of people acting at the frontiers of the organization itself and then needing a more flexible way to organize their joint work. Thus, we are witnessing an increasing number of situations where users need to negotiate their behavior with other users, as members of more or less consolidated communities [19]. In all these different situations, communities can take various forms and have a varying life span [2]: however, all of them have to establish some basic behavioral rules and conventions to survive and be effective toward their members and their hosting organization. To this aim, the technologies the organization makes them available (typically, corporate information systems) have to be flanked by (compositions of) technologies that, altogether, better respond to their local needs: technologies that are conceived to support local activities without a strong focus on persistency, efficiency, uniform adoption, and are instead more focused on requirements as heterogeneity, malleability, immediate appropriation, user control and the like. This view motivates a shift of interest from tailorability of single (collaborative) applications, e.g. [11], (possibly adopted across the organization) towards the tailorability of *local compositions of existing applications*, as a complement to the former kind of tailorability [20].

The point we want to make is that, irrespective of any details of how it is conceived and implemented, integration is usually proposed in terms of *mutual control* among applications, as for example in Web Services Architectures (SOA) [9] and by the so called mashups [21]. This approach takes the point of view of the professional programmers who reason in terms of system functionality and are skilled enough to deal with all the technical details implied by this kind of composition. On the contrary, end-users are driven by their needs to coordinate their individual and collaborative activities according to the aims, rules and conventions of the community they belong to. So, why not to take the *cooperation* point of view in defining a framework that supports end-users in composing the applications they deem as useful for their local needs and why not to leverage on the long tradition of collaborative work studies? Empirical studies, especially in the CSCW framework, have uncovered that the mechanisms supporting cooperation can be described as the composition of artifacts and protocols in combination with the awareness of what is going on in the context (e.g., [18]). Since actors are familiar with the above mechanisms it should be natural for them to apply the same principles not only to coordinate their mutual behavior but also to define how useful applications might interact to support it since both kinds of interactions can be defined at the same time, by means of the same conceptual tools [16] (see Figure 1 (left)). This approach is in the line of component based tailorability [20] but in addition it offers an explicit way to compose/integrate pieces of software by means of a clear and hopefully friendly metaphor. In this approach, we aim to bring to the technology the “controlled flexibility” typical of human cooperation instead of constraining human beings to conceive their own behaviors in terms of the capability of the technology.

This chapter presents how we exploited the CASMAS framework[1], which supports this view of integration and illustrates its application in a case derived from an ongoing project in our University. Some considerations derived from this short experimentation conclude the chapter.

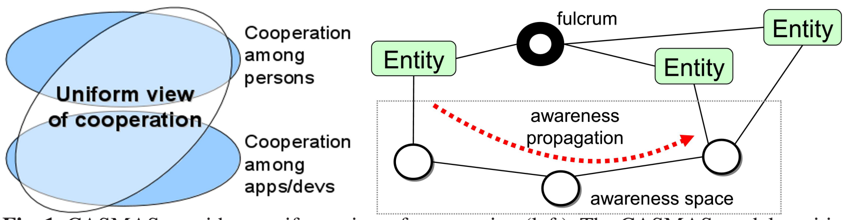
Uniform view of cooperation
Cooperation among persons
Cooperation among apps/devs
fulcrum
Entity
Entity
Entity
awareness propagation
awareness space

Fig. 1 CASMAS provides a uniform view of cooperation (left). The CASMAS model: entities can represent either persons or applications/devices (right).

2 The metaphor and the language underlying the framework

Suppose that a set of applications and devices have to be integrated to support a set of actors cooperating through them. According to the Community-Aware-MAS (CASMAS) framework they all are represented as *entities* and become members of a *community*: as such, they coordinate their behaviors through a Common Information Space (CIS) [3], called *fulcrum*[2], that contains the coordinative information, possibly articulated in coordinative artifacts [13]. Moreover, the fulcrum contains the *behaviors* that are dynamically assigned to each entity to make it an active member of the community: the use of behaviors is similar as in the Actor Model [1]. Like this model, communication is asynchronous but it is not message based. Instead, when an entity *posts a request* in the fulcrum, other entities will *react to this request* according to the behaviors currently assigned to them. In addition, entities promote mutual awareness by being located in an *awareness space* where awareness information is propagated from a source entity along the space structure (topology) [17, 4]. An awareness space can represent the physical space, but also be a logical space: for example, it can represent a set of roles and their relationships as edges connecting them. Figure 1 (right) sketches a community, its fulcrum and awareness space. An entity can be member of more than one community and can be located on more than one awareness space: in this case it allows the exchange of information among the communities the entity belongs to.

[1] A previous formulation was presented in [7].

[2] This term is used to emphasize the pivotal role of this space in cooperation.

The CASMAS framework has associated a language to specify entities, their behaviors and the awareness management. This language takes the declarative form of facts and rules (when-then constructs), which offers the possibility to express behaviors in a highly modular way, without the need to define complex and exhaustive control structures [14, 15]. The rules constituting an entity's behavior express *what* the entity is expected to do *when* some conditions are satisfied [15]. These conditions are matched against the facts contained in the community's fulcrum and in the entity's local memory, and against the awareness information from the community's spaces where the entity is located; the action(s) the entity should do updates either the community fulcrum/awareness spaces or the memory of the entity itself.

The integration of a software application/device is realized by inserting a fact in the memory of the entity representing it and by defining the behavior of this entity. The fact contains pairs attribute-value that specify the information the application/device makes available for sake of coordination with, and awareness promotion for, the other entities of the same community; the entity's behavior expresses conditions (among others) on the concrete application/device attributes (when) and invokes some of the functions the application/device exposes to the community (what): actually, the entity is a sort of wrapper that mediates between the concrete application/device and the integration environment (community). For example, if a DMS (Document Management System) is defined so as to provide the number of downloads in one of its attributes and exposes a function to build a report of some occurred events, users may define rules such as:

```
# a semi-formal rule to express the behavior of a DMS
when
 today's downloads are more than 1000
then
 share the report of the today downloads
   with the community
```

Actually, the CASMAS framework uses the Drools[3] rule editor and its capability to express rules in semi-formal terms. This possibility realizes a sort of self documentation and facilitates rules formulation and understanding. This is particularly important since end-users (in the true spirit of EUD) share with other end-users single rules up to whole behaviors: thus documentation plays an important role in reuse or adaptation of exiting rules/behaviors.

When a user wants to define or modify a coordination/awareness pattern, she is facilitated by the high modularity of the overall approach. In fact, behaviors are clearly assigned to entities and then any definition/modification of a rule constituting a behavior affects very delimited portion of the system: the behavior itself and those of the entities involved by the rule at hand. This makes it easier to cross-check the consistency of the definition/modification since it is univocally specified which entity is in charge of making true each condition and of providing each function constituting a rule [16].

[3] www.jboss.org/drools

3 The framework put to work

The basic constructs of the language can be uniformly used to express entities' behaviors at any level of abstraction (see Figure 2) where each level can use the primitives defined at the level(s) below. The language (constituting the first level) provides the following basic constructs (only the ones managing coordination and awareness are presented): *assert, retract, modify* a fact in a fulcrum, *move* an entity in a space, *makeAware* about an information, *AwareOf* to test if an entity perceives some kinds of awareness promoting information. The next level makes available *domain/application/device independent* primitives, like *postRequest* and *copyFact*, and predicates like *Request* and *Response*: they express coarser and recurrent pieces of behaviors that are likely to facilitate integration in terms of coordination of generic components. The level immediately above contains 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 DMS (as described in the previous section). Then the *domain dependent behaviors* are defined at the highest level, possibly with different degree of visibility of the levels below, according to the needs of the community. In fact, this visibility depends on the kinds of technical skill characterizing different classes of users, on the complexity of the integration to be realized, on the organizational norms, 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/predicates available at that level, or to enrich them by using the primitives/predicates of the levels below [16]. A best practice is to organize each level so that it is clear what it makes available and the knowledge required to operate at this level: the knowledge only, since the language is always the same.

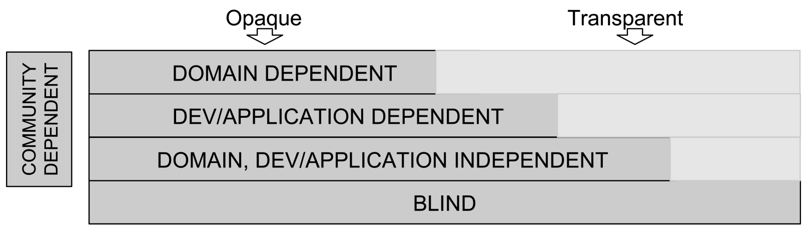
Opaque
Transparent
COMMUNITY DEPENDENT
DOMAIN DEPENDENT
DEV/APPLICATION DEPENDENT
DOMAIN, DEV/APPLICATION INDEPENDENT
BLIND

Fig. 2 Abstraction levels.

A framework supporting tailorability and EUD has to clarify how the different kinds of users are expected to use it. According to the EUD literature, there are at least three kinds of users: *developers*, *power-users*, and *end-users* [12]. In CASMAS developers are expected to set up and maintain the overall framework: in particular, they are in charge of the two bottom levels of Figure 2. Hence, they define the basic primitives/predicates that are domain independent in that they express the elemental interactions with the fulcra and the awareness spaces, their applications and devices.

In the previous section we mentioned the basic ones but nothing prevents a specific instantiation of the framework to add new primitives/predicates if needed. Developers are also in charge of constructing the wrappers for the functionalities offered by the applications/devices to be coordinated: they can be used in the right part of the rules at the next higher level of abstraction. These two activities require technical skills that the other kinds of users are not expected to possess. The highest level is the realm of end-users and power users. Power-users are end-users with technical skills suitable to manage the dynamic configuration of the coordinated applications, mainly by adding new behaviors or high level primitives/predicates. In principle any end-user is enabled to manage dynamic configurations: however our experience and the literature (e.g., [8]) show that often end-users delegate the construction of difficult interactions to more skilled users. In other words, end-users may need a "human proxy" between them and the tailored application: power users are the best candidates to play this role because they belong to the community, understand the needs of their "costumers" and know about the rules and conventions holding within the community itself. They are then able to express them in terms of behaviors, i.e., sets of rules, that can be made available to the other community members. End-users are expected to tailor existing behaviors by adapting the set of rules they contain. More experienced end-users can build completely new behaviors by writing new set of rules, possibly with the help of some power-users. In this paradigmatic allocation of responsibilities end-users are not expected to act as "systems integrators" or "software architects": instead, they are called to think in terms of which kind of coordinated behavior they are able to offer and want to get from the community in response to their common needs. This behavior has to be negotiated among the actors (as it happens irrespectively of the available technology) and then properly implemented by using the features made available by the framework.

4 Case study

In the context of the GAS (Grandi Attrezzature Scientifiche) - Intelligent Building project of our University we set up a room with some facilities for group work: an interactive desk, some interactive whiteboards, a localization technology, small and medium size interactive screens, and artifacts "augmented" by means of localization tags to make them traceable in the room (for example, books with a localization tag attached). The above interactive devices allow actors to use software applications to support group work and (new) content production by accessing their own contents or the contents provided by other people.

The case study we present in this chapter involved a class of fourteen students divided in three groups (two groups of 5 persons and one group of 4 persons). The students were familiar with computer usage and owned a background in psychology (one person per group own advanced skills in computer usage; two of them have been involved in doing power-user tailorability, see Section 4.2.4). The supervisor asked to groups to prepare a report on a common topic in psychology. The groups met twice a week for about two months to collaboratively prepare the report with the support of the supervisor. From time to time the supervisor made some contents available to the class to orient the work of the three groups. To support this supervised collaboration the room was configured as follows: the desk is reserved to the supervisor and runs an application to manage personal contents and possibly make them publicly available to the class; each group owns a whiteboard providing a work space where to manage group, personal, or publicly available contents. During three meetings students and supervisor were told about the CASMAS framework and language as well as about the features made available by the basic configuration. Then they started to autonomously use this framework: when tailoring was needed students tried to solve the problem by their own and came to us only to validate their solution or to ask for our help. Students were familiar with the DMS that managed the content/documents they needed: it was configured so that single actors and groups can have their own repositories. The rest of this section describes how the basic configuration (referred to as "system") was constructed and then tailored by the students to respond the needs arisen during its usage.

4.1 Configuring the system

According to the approach presented in Section 2, we (as developers) implemented the initial system configuration that includes the following fulcra and awareness spaces :

- `group X fulcrum`: a community fulcrum for each group (see Figure 3 (left));
- `class fulcrum`: a community fulcrum for all the actors involved, i.e. students and the supervisor;
- `contents space`: a space to promote awareness of available contents (see Figure 3 (left)) ;
- `room space`: a space representing the physical localization inside the room of all the involved actors to manage awareness promotion about their presence.

The `contents space` contains a `group-X content` node for each group: these nodes are linked to the `public content` node to support the propagation of awareness information about the availability of group and public content, respectively. Actually this is a simplification of the actual `contents space`: it is sufficient to discuss the usage of the framework presented in this chapter.

In addition, the configuration contains the entities that stand proxy for the three groups, the supervisor and for the following "technological actors": one entity for the interactive desk (`IDesk`); one entity for each interactive whiteboard (`IWB-X`); one entity for the public wall (`PWall`); one entity for each group to manage its interactions with the DMS (`DMS-X`).

Figure 3 (left) considers only the portion of the whole configuration that will be used in the rest of this chapter and shows how the above entities are connected to

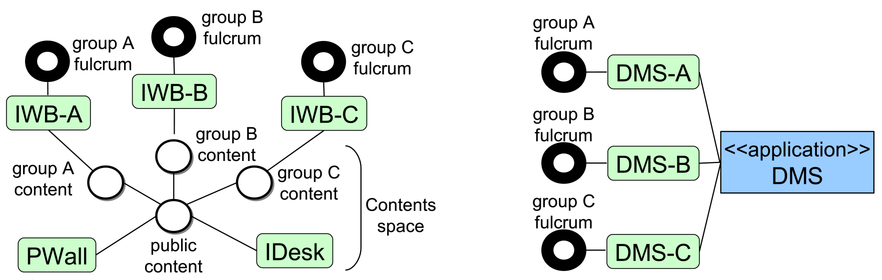

Fig. 3 Basic configuration (left). Multiple entities associated to one shared application (right).

the fulcra and to the `contents space`. Moreover, groups members are linked to the fulcrum related to their group, and the supervisor is linked to the fulcrum related to the class (these links and the latter fulcrum are omitted in the figure).

In a cooperative setting, some applications (typically, a DMS as in our case) are shared among several users and groups as a resource provided by the organization they belong to. However, users and groups might want to define different local interactions with this resource, according to their needs. In CASMAS this is realized by associating more that one entity to a single application instance and to connect each of them to the appropriate fulcrum (see Figure 3 (right)).

Each entity of the basic configuration has associated a “standard” behavior. In the following we illustrate a small portion of system behavior concerning a case of awareness promotion that involves `IDesk` and `PWall`. Patterns related to other forms of cooperation will be shown later on. When a new content is moved from the personal to the public area of the interactive desk (typically, by the supervisor), the `IDesk` uses the `contents space` to make the other entities aware of this event (*makeAware* primitive, see Section 2). The related awareness information is propagated in the space from the `public content` location because the `IDesk` is situated there, and reaches all the `group X content` locations. `PWall` becomes aware of the new available content because it is situated at the `public content` node and owns (by design) the following rule (rule RD1_PWALL) that realizes the following reaction: `PWall` shows the new content to make it publicly available.

Rules are hereafter presented using the semi-formal notation of Drools.

```
rule "RD1_PWALL-show publicly available content"
when
 aware of content available on space CONTENTSSPACE
then
 show content
end
```

where `aware of` is a construct defined at the CASMAS language level. The `show` action is defined at the application dependent level. Notice that the `IWB-Xs` do not react to the presence of any new available content because the developers did not define a similar rule in their behavior, even though this kind of awareness promoting information reaches all the locations where they are located.

4.2 Tailoring the system

During the case study we observed different situations in which power users or end users felt the need to modify the basic system configuration giving rise to different kinds of tailorability.

4.2.1 End-user tailorability

Two of the three groups improved the support provided by the system. Members of group 'A' recognized it was useful for them to have the public contents available for future use. Members of group 'B' identified a different requirement: they would have the public contents visualized also on their local IWB in addition to the public wall. In this way the group could elaborate their report having these contents ready at hand during their collaboration. Then Group 'A' defined two rules that illustrate a typical cooperation pattern between two applications through the fulcrum. By the first one (rule R2_IWB) the `IWB-A` posts a store request to `group A fulcrum` when it is aware of the new available content.

```
rule "R2_IWB-privately store public content"
when
 aware of new content available in space CONTENTSSPACE
then
 post a "store" request of this content to the group
end
```

`post` is a construct at the domain/device/application independent level, as already mentioned in Section 2.

By the second rule (rule R2_DMS) the `DMS` responds to the "store request" by storing the content through the functions of the DMS application. The content is stored in the current activity area of the repository.

```
rule "R2_DMS-privately store public content"
when
 there is a "store" request from the group
 group is carrying on an activity
then
 store content contextually to the current activity
end
```

`request` is a fact at the domain/device/application independent level, as well as `post`; `current activity` is a fact defined at the domain dependent level that models the current activity and its attributes (e.g., its name) that the group is carrying on. In the basic configuration used in the case study the `current activity` fact was asserted as a static information useful to allocate to a specific area the contents generated during the case. In a more general situation, this naturally dynamic information could be obtained from a WFMS application for which `current activity` is defined as one of its attributes (see Section 2).

Group 'B' needed to define only one rule to let the `IWB-B` show the content when it becomes aware of a new available content (rule R3_IWB).

```
rule "R3_IWB-show public content"
when
 aware of content available on space CONTENTSSPACE
then
 show content
end
```

After group ‘B’ used the system for a while, its members realized that every time a public content was available on their IWB they naturally focused on that content; in order to make its fruition easier they also minimized the current content to have more space for the new content. Hence, the group decided to change the above rule (rule R3_IWB) to minimize the current content area before the public content was shown, by adding the “as main content” clause to the action part of the rule

```
 ...
 show content as main content
 ...
```

Group ‘B’ found group ‘A’ “storing behavior” very useful. Accordingly, group ‘B’ updated the behavior of their `IWB-B` with the functionality implemented by group ‘A’ . In this way they did not only viewed the public content but also had the content stored in the DMS.

4.2.2 Domain language as support of tailorability

As mentioned in Section 2, the semi-formal language supported by Drools to define a Domain Specific Language (as Drools calls it) can be used both to offer a more user friendly version of the CASMAS language and to construct a semi-automatic documentation of system behavior [16]. Users appreciated this possibility because this made easier and more natural for them to conceive the rules, and also because the semi-formal language allows one to hide some technical details (as we will discuss in Section 5). We present here as an example the formal version of rule R3_IWB defined for the `IWB-B` by group ‘B’:

```
rule "R-F-1"
when
 AwareOf(space=="$CONTENTSSPACE$",
  type=="content available", contentURL:awarenessData)
then
 application.minimizeContent();
 application.showContent(contentURL);
end
```

This formal rule is automatically derived from rule R3_IWB by means of the following mappings where the semi-formal version is associated (by ‘=’) with its formal counterpart and the texts in curly brackets define the rule parameters to be instantiated: in this case, the type of awareness information and the awareness space where the information is perceived.

```
[when]aware of {awareness type} on space {space name}=
 AwareOf(space=="{space name}",
  type=="{awareness type}", contentURL:awarenessData)
[then]show content as main content=
  application.minimizeContent();
  application.showContent(contentURL);
```

Notice that the semi-formal formulation does not contain any reference to the information content: in fact, the rule does not define any specific condition on it. However, the content appears as an attribute in the formal specification. Although this kind of parametrization might be considered simple and not so powerful, it allows one to adapt it to compose other rules managing other types of awareness information, e.g. about an updated content. The Drools factory supports users in building the mappings through a graphical user interface (that generates the above textual mapping structure) although in a not completely satisfactory way: we will come back to this points in the conclusions.

Finally, the action `show content as main content` ([then] mapping) expresses the users' needs *in one sentence*: in fact, "as main content" implies that all the other contents have to be minimized. The abstraction of a sequence of actions into a single sentence is very powerful also to avoid accidental omissions, in particular when some actions have to be done only to maintain system consistency. Rule R2_DMS is an example as the response to a request implies the request deletion. This rule is translated into the following formal rule

```
rule "R-F-2"
when
 request:Request(assertedIn=="$GROUP$",
   service=="store", contentURL:params)
 CurrentActivity(assertedIn=="$GROUP$",
   activityName:name)
then
 application.store(contentURL, activityName);
 retract(request);
end
```

by means of the following mapping:

```
[then]store content contextually to the
 current activity=application.store(contentURL,
  activityName); retract(request);
```

In our experience, to start using the semi-formal language end-users need the assistance of a power user: then, they start using the mappings to build new semi-formal rules. Only after a longer experience they define the needed mappings by their own, typically by adapting the ones they had used before.

4.2.3 Tailoring information provided by applications

During the case study the students came to understand how to extend the set of attributes that the developers defined to convey information from the DMS to the coordination space (i.e, groups fulcra) since they didn't find a solution by themselves. During the discussion to give an answer to this request they appreciated the fact that this was possible in a way that they could understand. In fact, it would only require the invocation of some pertinent application method(s) as shown by the following example (RAPP-1) that we used to illustrate the solution.

```
rule "RAPP-1"
when
 there is a "number of docs" request from the group
then
 post number of docs to the group
end
```

where the post of “number of docs” information should be defined in the mapping as:

```
int numberOfDocs = application.getDocsCount(dir);
assert(new DirDocsCount(assertedIn="$GROUP$",
  directory=dir, count=numberOfDocs));
```

The students told us that this kind of tailoring would require the help of the developers: however, they felt they could formulate the related request with an idea of what it would require to be implemented. Power users instead were confident that, if the application methods were sufficiently well documented, they would be able to handle the simplest cases by their own.

4.2.4 Power-user tailorability

The awareness promotion about new content produced by other groups was a feature required by the participants since they like to stay informed. However, end-users were not able to define this behavior by themselves, mainly because it is not simple for them to implement a message starting from the URL of the content to be shown. Hence, a couple of recognized power-users committed themselves to cooperatively build the new behavior for their colleagues. The two power users negotiated the following assignments, according to their skills. One of them defined a new rule to be added to the `IWB-X` behavior, which shows the desired message when this device is aware of a new content (thus, using a previously unexploited information). The second power user defined the function that creates the new message by extracting information from the URL of the awareness information content. Although this power user was not a professional developer, she was able to look for some tutorials on the Web[4] to implement the needed function, called `createMessage`, by adapting standard patterns they make available.

```
rule "R1_IWB-show awareness about groups' content"
when
 AwareOf(space=="$CONTENTSSPACE$",
  type=="group content created",
  contentURL:awarenessData)
then
  # show content awareness
```

[4] e.g. www.exampledepot.com/egs/java.net/ParseURL.html

```
 String awMessage = createMessage(contentURL);
 application.showAwarenessMessage(awMessage);
end

function String createMessage(String contentURL) {
 URL url = new URL(contentURL);
 String documentName = url.getFile();
 String message = documentName + " is available";
 return message;
}
```

`createMessage` is a function that she defined at the domain dependent level, since it defines for the specific community how the actors are informed about a new content produced by other groups. To adopt this new behavior each group uploaded it in its own fulcrum so that the IWB of each group acquires this behavior and behaves accordingly to the community's agreement. The above kind of division of labor was observed in several circumstances: it is favored by the if-then structure of the language, which defines a natural "interface" between the tasks of the two power users.

5 Lesson learnt and future developments

The experience we got from the case study has been encouraging and fruitful: the community metaphor and the cooperation based approach was naturally appropriated by the participants since they were able to recognize coordinative practices in their every day learning activities and take them as starting point to tailor the given system configuration. Sometimes users had some difficulties in separating the functional aspects of an application (what it can do) from its coordination and awareness behavior (when and how it should do something). This kind of difficulty emerged when users collaborated with developers to include a new application functionality (as reported in Section 4.2.3) or a totally new application (not reported in this chapter) into the system: often users suggested functionalities that include coordination or awareness aspects. However, this problem almost disappeared when users wanted to tailor how the application contributes to the overall system: in this case they had to recognize which basic functionality was offered by the application, and which new coordinative behavior had to be defined. The case study confirmed that any adoption of tailorability and EUD environments requires an assisted learning process: however, we perceived an increasing appropriation of the approach because the kind of support users requested was toward the solution of more complex problems of integration and tailoring [10]. At the same time the case study raised a set of basic requirements oriented to improve the usability of the approach. The first issue is about the development of a graphical interface to represent the current configuration of the system in terms of the integrated applications/devices and the proxies of their collaborating users. This graphical interface would facilitate system development and tailorability, and support rules organization. Actually, the levels of abstraction described in Section 3 and the modularity based on the notion of community of entities are a sound basis towards rules organization at each level. A second issue concerns rules definition. Here it is not a matter of syntax comprehension: users correctly understood existing rules; nor of compliance with the (semi-)formal language: Drools offers a rich syntax driven editor. When users came to us for getting support, we observed that they did not own a strategy to transform their intuition of what each entity should do into well-formed rules [5] and in so doing they encountered recurrent problems: typically, users do not list all the conditions that characterize the situation in which an action has to be executed; or they forget to retract facts that no longer represent valid situation; or they are confused about where to memorize information (i.e., facts) either in their private memory or in the shared fulcra, and by default choose the second alternative; or finally, they apply the traditional pattern of interaction based on request/response also when the post/react pattern would be more appropriate since the capabilities of the current collaborative context are not fully known. We are currently collecting the problems that the past and ongoing experiences of usage are putting in evidence [14]: the aim is to enrich the graphical interface with suitable guidelines that would warn users of erroneous or at least dangerous rules formulations. A last and even more challenging issue concerns the test of rules consistency in presence of the tailoring of a given configuration [6]. Again, the rules organization mentioned above offers an initial strategy based on modularity: however, more has to be done to support the check of consistency. A possible solution could be based on the visualization of rules dependencies, like cells dependencies in a spreadsheet: here however the solution has to take into account the different levels of abstraction in which rules are organized. In improving the CASMAS framework along the above lines we will continue to adopt the iterative method based on users feedbacks, thanks to the technological setting made available by the GAS-Intelligent Building project.

6 Acknowledgments

This work has been supported by the GAS-Intelligent Building project and with the financial support of F.A.R. 2010.

References

1. Agha, G.: Actors: a model of concurrent computation in distributed systems. MIT Press, Cambridge, MA, USA (1986)
2. Andriessen, J.: Archetypes of knowledge communities. In: P.v. van de Besselaar, G. De Michelis, J. Preece, C. Simone (eds.) Second Communities & Technologies Conference (C&T2005), pp. 191–214. Springer (2005)
3. Bannon, L., Bodker, S.: Constructing common information spaces. In: ECSCW'97: Proceedings of the fifth conference on European Conference on Computer-Supported Cooperative Work, pp. 81–96. Kluwer Academic Publishers, Norwell, MA, USA (1997)

4. Benford, S., Fahlén, L.: A spatial model of interaction in large virtual environments. In: ECSCW'93: Proceedings of the third conference on European Conference on Computer-Supported Cooperative Work, pp. 109–124. Kluwer Academic Publishers, Norwell, MA, USA (1993)
5. Berti, S., Paternò, F., Santoro, C.: Natural development of ubiquitous interfaces. Commun. ACM **47**(9), 63–64 (2004). DOI 10.1145/1015864.1015891
6. Burnett, M., Cook, C., Rothermel, G.: End-user software engineering. Commun. ACM **47**(9), 53–58 (2004). DOI 10.1145/1015864.1015889
7. Cabitza, F., Locatelli, M.P., Simone, C.: Cooperation and ubiquitous computing: an architecture towards their integration. In: Proceeding of the 2006 conference on Cooperative Systems Design, pp. 86–101. IOS Press (2006)
8. Dixon, D.R.: The behavioral side of information technology. International Journal of Medical Informatics **56**, 117 – 123 (1999). DOI DOI: 10.1016/S1386-5056(99)00037-4
9. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA (2005)
10. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A.G., Mehandjiev, N.: Meta-design: a manifesto for end-user development. Commun. ACM **47**(9), 33–37 (2004). DOI 10.1145/1015864.1015884
11. Fogli, D.: chap. End-User Development for E-Government Website Content Creation, pp. 126–145 (2009). DOI 10.1007/978-3-642-00427-8_8
12. Lieberman, H., Paternò, F., Wulf, V. (eds.): End User Development. No. 9 in Human-Computer Interaction Series. Springer Netherlands (2006)
13. Locatelli, M.P., Loregian, M.: Active coordination artifacts in collaborative ubiquitous-computing environments. In: B. Schiele, A.K. Dey, H. Gellersen, B.E.R. de Ruyter, M. Tscheligi, R. Wichert, E.H.L. Aarts, A.P. Buchmann (eds.) Ambient Intelligence, European Conference, AmI 2007, Darmstadt, Germany, November 7-10, 2007, Proceedings, *Lecture Notes in Computer Science*, vol. 4794, pp. 177–194. Springer (2007). DOI 10.1007/978-3-540-76652-0_11
14. Locatelli, M.P., Simone, C.: Integration of services based on the community metaphor: some guidelines from an experience of use. In: The 2nd International Workshop on End User Development for Services (EUD4Services). Torre Canne (Brindisi), Italy (2011)
15. Myers, B.A., Pane, J.F., Ko, A.: Natural programming languages and environments. Commun. ACM **47**(9), 47–52 (2004). DOI 10.1145/1015864.1015888
16. Repenning, A., Ioannidou, A.: What makes end-user development tick? 13 design guidelines. In: H. Lieberman, F. Paternò, V. Wulf (eds.) End User Development, *Human-Computer Interaction Series*, vol. 9, pp. 51–85. Springer Netherlands (2006). DOI 10.1007/1-4020-5386-X_4
17. Rodden, T.: Populating the application: a model of awareness for cooperative applications. In: CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work, pp. 87–96. ACM, New York, NY, USA (1996). DOI 10.1145/240080.240200
18. Schmidt, K., Simone, C.: Coordination mechanisms: towards a conceptual foundation of cscw systems design. Comput. Supported Coop. Work **5**, 155–200 (1996). DOI 10.1007/BF00133655
19. Wenger, E.: Communities of practice; learning as a social system. Systems Thinker **9**(5), 2–3 (1998)
20. Wulf, V., Pipek, V., Won, M.: Component-based tailorability: Enabling highly flexible software applications. Int. J. Hum.-Comput. Stud. **66**(1), 1–22 (2008). DOI 10.1016/j.ijhcs.2007.08.007
21. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development. IEEE Internet Computing **12**, 44–52 (2008). DOI 10.1109/MIC.2008.114