A Task-Driven Design Model for Collaborative AmI Systems

R.F. Arroyo[1], M. Gea[1], J.L. Garrido[1], and P.A. Haya[2]

[1] Universidad de Granada
{robfram,mgea,jgarrido}@ugr.es
[2] Universidad Autónoma de Madrid
Pablo.Haya@uam.es

Abstract. Ambient intelligence (AmI) is a promising paradigm for human-centred interaction based on mobile and context-aware computing, natural interfaces and collaborative work. AMENITIES (a conceptual and methodological framework based on task-based models) has been specially devised for collaborative systems and is the starting point for a new design proposal for application to AmI systems. This paper proposes a task-based model for designing collaborative AmI systems, which attempts to gather the computational representation of the concepts involved (tasks, laws, etc.) and the relationships between them in order to develop a complete functional environment in relation with the features of AmI systems (collaborative, context-aware, dynamic, proactive, etc.). The research has been applied to an e-learning environment and is implemented using a blackboard model.

1 Introduction

Ambient intelligence (AmI) has become the next step in the user-centred approach of computer applications. These applications incorporate technology into an omnipresent and transparent infrastructure for the implementation of smart environments. AmI places the emphasis on user-friendliness, more efficient services and support for human and group interaction [1]. This paradigm is based on emerging technologies, such as ubiquitous computing, collaborative systems and intelligent user interfaces for natural interaction[2]. Although interest in this technology and its benefits are high, it is difficult for there to be the required infrastructure to develop these applications which fulfill the requirements.

In order to characterize this interaction paradigm, a brief review of the features for this kind of system is presented below.

Context-Awareness. Context is defined as "any information that can be used to characterize an entity's situation. An entity can be a person, place, or physical or computational object relevant to the interaction" [4].

Natural User Interface. Mobility and ubiquity demand better user-friendly interfaces which use natural mechanisms and allow the user to focus on the task. Spoken dialogue has also been considered as a natural method to interact with the environment, and context could also be used to resolve communication ambiguities [5].

Collaborative spaces. Most of the AmI scenarios are oriented towards group-work whereby users interact with each other to achieve common goals. These collaborative tasks are present due to many users working together to fulfill an activity, or alternately, the system takes part in the users' tasks to give additional information, performing actions or guiding the user workflow while the users interact in the system.

Dynamic Space. These active spaces mean that people work in a constantly changing context [6] (such as for example changes in group members, scenario devices, etc.). Two different methods can be used: context pull requests the required context information when it is needed (by the user, system, etc.), and context push is a mode where the context information is forwarded to subscribed users.

Proactiveness. One of the requisites of AmI scenarios is for there to be reactive behaviour. In this case, it should not be necessary to focus the user's attention on dialogue interaction all the time. New behavioural user models have therefore been proposed [7] which are useful for inferring the user's action.

Shared Knowledge. Knowledge about the context of the user engaged in the scenario is highly distributed. This information is bound to the users, their locations, the community in which they are involved [8].

Usefulness. This new paradigm offers a different computational environment which is a long way from classical desktop applications. The development of applications to solve everyday tasks is important. Several aspects must be considered such as trust, collaboration, effectiveness, etc. One example of a useful application is the Stick-e Notes [9].

These features demand a robust and comprehensible model to describe and implement AmI scenarios. Section 2 of this paper introduces the main methods used by different authors to describe AmI systems. Section 3 then presents a comprehensive and straightforward method for describing these AmI features. Section 4 illustrates the integration of this proposal on a centralized implementation based on a blackboard. Finally, Section 5 describes an example based on an e-learning collaborative scenario.

2 Modelling Approaches for AmI Systems

2.1 Methods for AmI Specification

The complexity of AmI design is closely related with the mechanism for describing its inherent features. Suitable methods for describing these properties in a straightforward way should therefore be applied and this would allow us to accomplish a more correct analysis and development. Some of these approaches are described below.

Theoretical models. Most of the proposed theoretical models in human-computer interaction (HCI) are based on the human information processor model

[10]. For AmI environments, the user(s) attention(s) is/are shared on several simultaneous activities using artifacts to achieve certain goals. Activity theory (AT) [21] deals with human social interaction using tools in the context of a community in which the fundamental unit of analysis is human activity. The activity is the smallest meaningful unit for human action. Activities are embodied so as to accomplish a goal (objective) using tools within a community. This theory has been successfully applied in order to understand collaborative works (CSCW). A complementary theory comes from situated social interaction [11]. Theories and ontology-based methods are well suited for a better understanding of AmI environments but it is also necessary for development processes to deal with these concepts.

Scenarios. AmI is viewed as a natural human-centred interaction, and in this way, its benefits are suggested by defining envisioned scenarios [12]. Scenario-based design [13] is a well-known technique for problem understanding and requirement elicitation, and it has sometimes been proposed as an alternative method for task-based design. However, the natural narrative language is difficult for non-expert users to handle, and it is necessary to translate these conclusions into other intermediate (graphical) notation such as a UML use case diagram, which is used to identify functional requirements for the AmI context.

Task and workflow models. Task and workflow models. Task modelling transforms user activities and related data into structured pieces of task-based knowledge. Any model usually covers the representation of the task execution (giving temporal constraints) with objects and the involvement of actors playing roles. However, context data is not present at this stage. A mixed approach is presented in [14,3], where the notation used is a mixture of scenario description (using a graphical notation) and task modelling (using actors, messages, synchronization mechanism, etc.) in a two-step process. Alternately, workflow defines precise work processes to predict their execution and management [22]. Workflow is oriented towards processes and time execution whereas tasks focus on the user and expected behaviour. The inclusion of more context-dependent aspects of these activities is currently being researched and this is crucial for AmI design.

Middleware. A consistent middleware layer is required to support the infrastructure requirements for AmI development. [15] proposes a context layer which is a middleware based on a blackboard metaphor that stores a global data structure representing a model of the world, where any relevant information is maintained. This layer is also used for an asynchronous information querying mechanism. This approach has the following benefits: it is a loose-coupling mechanism between producer and consumer, interpretation is consumer-dependent, one participant is not aware of the others, and the model is easily extendable.

2.2 AMENITIES: Methodological and Conceptual Framework

So far, we have reviewed the most important AmI features and different mechanisms for representing them. One of the most relevant issues is context-awareness, and most authors agree with the importance of understanding these concepts and reflecting them in the design. In order to address the development of collaborative AmI systems, we have used a methodology (called AMENITIES [17]) based on tasks and behaviour models and used for studying and developing cooperative systems. It focuses on group-related concepts and has been successfully applied to collaborative systems, describing complex and dynamic organizations and shared resources [16,18].

AMENITIES is based on tasks and behaviour models, with tasks being the main concept of any system modelled using this methodology. Figure 1 shows the main concepts of AMENITIES in addition to their relationships using an UML class diagram. According to this conceptual framework, an action is an atomic unit of work. Its event-driven execution may require/modify/generate explicit information. A subactivity is a set of related subactivities and/or actions. A task is a set of subactivities intended to achieve certain goals. A role is a designator for a set of related tasks to be carried out. An actor is a user, program, or entity with certain acquired capabilities (skills, category, etc.) that can play a role in the execution (using artifacts) of (or responsibility for) actions. A group performs certain subactivities depending on interaction protocols. A cooperative task is one that must be carried out by more than one actor, playing either the same or different roles. A group is a set of actors playing roles and organized around one or more cooperative tasks. A group may comprise (i.e. be formed of) related subgroups. A law is a limitation or constraint imposed by the system that allows it to adjust the set of possible behaviours dynamically. An event is based on its common software definition, which is *an occurrence or happening of significance to a task or program.* An organization consists of a set of related roles. Finally, a cooperative system is composed of organizations, groups, laws, events and artifacts.

For each particular system, these concepts are described in a suitable way using an UML extension (called COMO-UML) [20]. This has the following benefits: concepts (such as roles, capabilities, laws) are blended on a modelling notation to express collaborative issues, reflecting dynamic aspects (changes in responsibilities, interruptible tasks), social structure (roles, capabilities) and impositions (the rules governing the community). We shall now use this approach to collect AmI features and reflect them in a specific design.

3 AmI Design Model

This design model aims to state the computational representation of the concepts involved (tasks, laws, etc.), and the relationships between them in order to develop a complete functional environment in terms of the features of AmI systems (collaborative, context-aware, dynamic, proactive, etc.). AMENITIES embraces the main concepts to be considered when designing collaborative systems and is the starting point for our new proposal for designing AmI systems. In the following section, we will propose a stepwise method for translating these abstract concepts into a computational model.

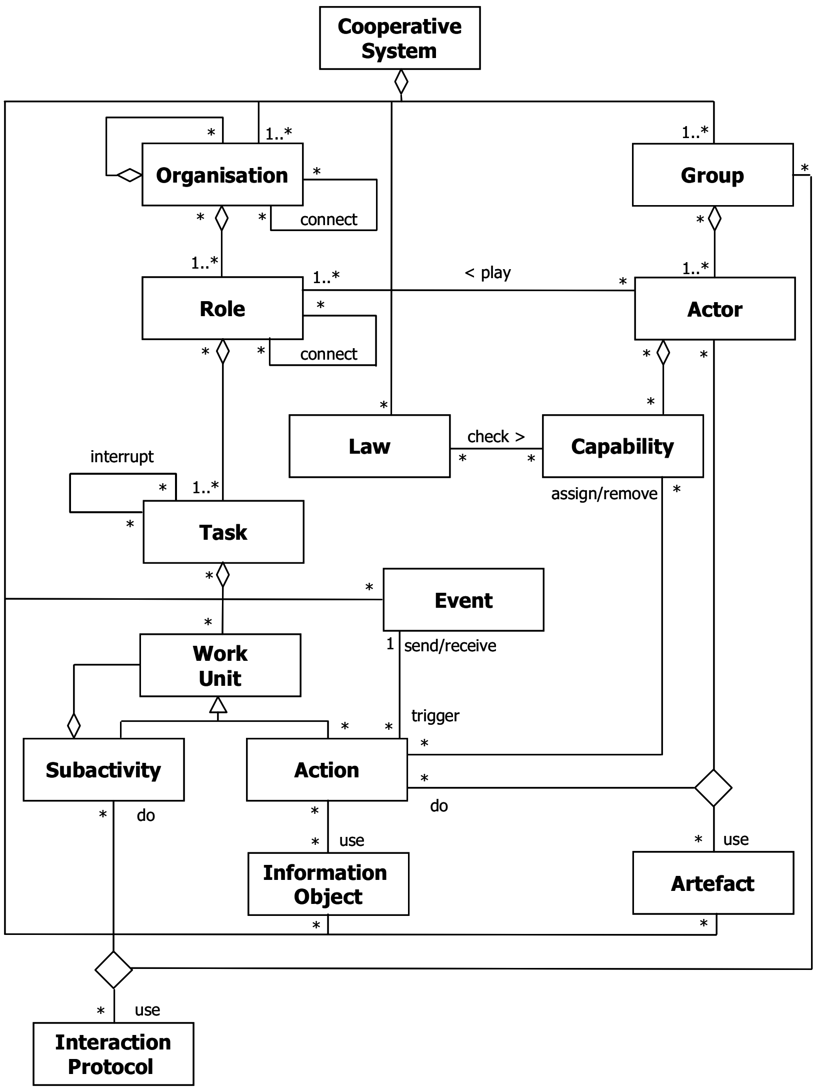

Fig. 1. AMENITIES main concepts.

3.1 Model Basis

We define an **object** as *the basic logic abstraction of everything*, either physical or not. Our objective is to treat all things as objects (lamp, student, person, bulb or terminal), regardless of their true kind or nature, establishing a homogeneous representation as the basis for the design. An object can store additional information (normally attribute-value pairs). In this way, a lamp interface should specify its ability to be switched on and off, and storing and asking about its current state (among others). As our intention is to design dynamic environments, we must establish a method for relating behaviours and properties to objects in order to concisely specify how objects work. An **interface** is defined as a specification of a set of properties and/or methods which are exported by objects that implement this interface. Physical objects are abstracted into logical ones, and these will be the ones introduced into the system. We then proceed to separate the object properties from its behaviour. This distinction enables us to split the system in two: object *implementation* collected in its entities, and its *specification*, collected in the interfaces. Figure 2 shows the proposed design and an example of a device (`computer001`) encapsulated in an object (`terminal`) working as an e-mail client and instant messaging client (interfaces `emailClient` and `IMClient`).

Until this point, we have objects and interfaces to work with. In addition to the creation of this type of entity, we can link them following these rules:

1. *Interfaces can be related to any object except any interface.* We must remember that the interfaces provide the behaviour of the objects, not of other interfaces. An interface without an object is meaningless.
2. *Objects can be related to any number of objects and interfaces.* This implies two things:
   – An object can be related to many objects: we can define new objects as addition, aggregation, or any kind of hierarchy. For example, a lamp comprises a bulb, and this is represented by the model by saying that the bulb is connected with the lamp.
   – An object can be related to many interfaces: there is no specification about interface complexity. For example, we could define an interface for a messaging system, and another for a mail client, and have an object computer that implements both interfaces.

Associations between entities (links) could be tagged to add additional information about the nature of the relation.

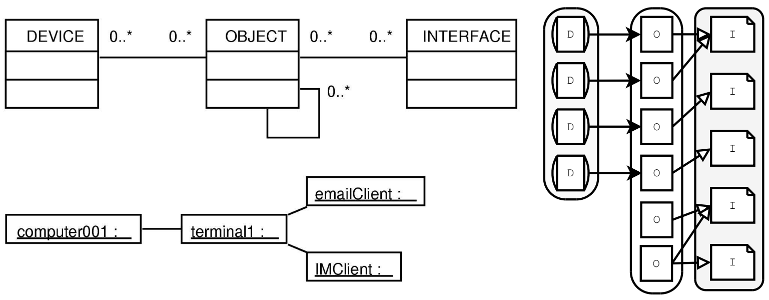

Fig. 2. Separation of devices, objects and interfaces. Composition of objects and interfaces.

In order to illustrate this, we shall consider an example concerning object composition. We shall start with a simple classroom model, consisting of illumination, a door and the cooling system. The illumination models onto an abstract object called `lights`, which is composed of three instances of the lamp class. The room comprises the cooling system, lights and door, and each object has its corresponding interface. It should be noted that since each lamp is considered to be the same as the others, they all have the same interface. The resulting diagram is shown in Fig. 3.

With this model we capture context by means of both:

- object attributes (e.g. the light status).
- the proper relationships between objects, for example, a person is inside a room if there is an `in` relation connecting these two entities (as shown in the figure).

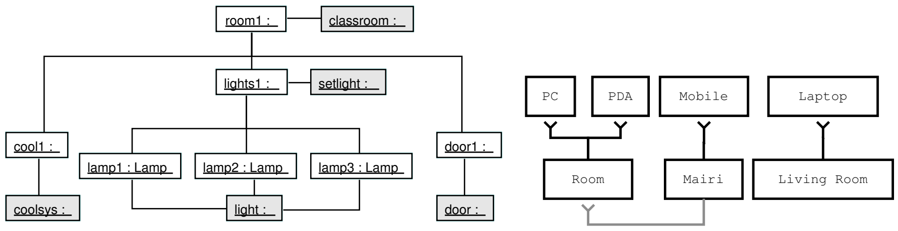

Fig. 3. Example of classroom composite (objects in clear background, interfaces in shaded background). Context relative to placement stored into relation between objects.

3.2 Defining Modelling Entities

Due to space limitations, this section describes only some modelling entities related to the concepts introduced in Sec. 2.2. These more complex objects are also represented in our design model. They may also contain other types of restrictions which can be described in the following sections.

Law An example of a law is that a class cannot start if the teacher is absent. This object has the following properties: *self-information* for identifying the law itself; *preconditions*, comprising a set of conditions that must be satisfied in order for the law to be fulfilled; *actions* that are performed once the law has been fulfilled; and finally, a *logical expression* connecting previously defined preconditions by means of logical operators and possible events (with or without parameters) producing changes in system activity. If this logical expression has not been specified, then the AND operator between preconditions is assumed by default.

A law is graphically described according to the scheme in Fig. 4. In order to create the logical expressions, we need a set of operations. Although a logical set is completely defined with only the operators *or*, *and* and *negation*, we will define more for simplicity when expressions are created.

Precondition This comprises *self-information*, a set of *restrictions* specifying a list of particular attributes that candidate objects to be chosen must have in order to carry out system activities. These restrictions consist of elements with the attribute to be evaluated, a condition to be satisfied for this attribute, and a field indicating the obligatory nature of the restriction; a *logical expression* on the defined restrictions or an implicit *logic AND* (if the logical expression has been omitted). The obligatory nature can be used as a preference criterion for the candidate object choice. For example, when we stablish the preference of one classroom over another due to capabilities or equipment, we are speaking about preconditions.

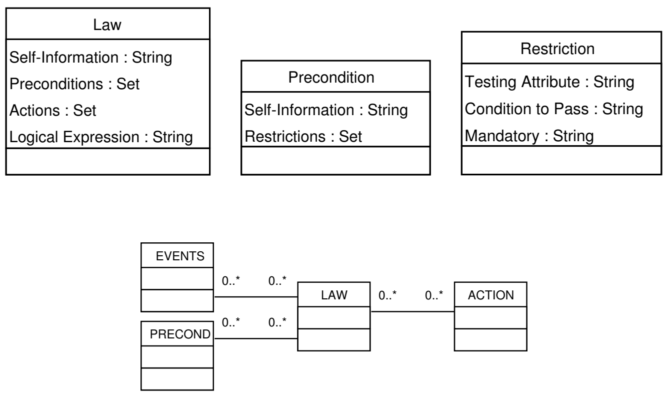

Fig. 4. Components of laws, preconditions and restrictions. Law definition as object composition.

Subactivity We use the term subactivity to refer to elaborated groups of actions or simpler ones. Analyzing the needs of a subactivity, we can define its components as a *finite state machine (FSM)*, which stores the behaviour of the subactivity; links to *roles*, determining which ones take part in this subactivity; links to *outcoming events*, determining what events are generated; links to *incoming events*, determining the ones needed; links to *actions*, specifying what actions are done by the subactivity; and links to *subactivities*, determining what subactivities are called from this one. Figure 5 shows a schematice representation of the subactivities. When we speak about a student doing his homework, we are referring to a subactivity.

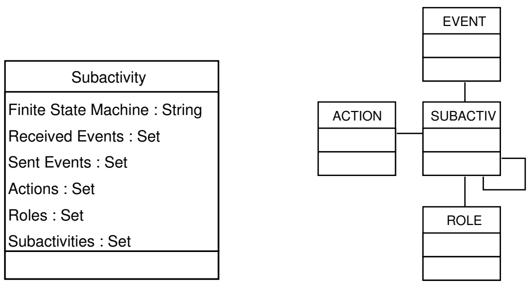

Fig. 5. Components of subactivities, and composition as objects where cardinality of all relations is `n` and tags are omitted for simplicity.

Event The information about the specific use of the event is specified in the link connecting this event with the object that uses it. This specification consists of the *type* of relation with the event, i.e. whether the event is being sent or received; a *roles* expression including at least one role or a composition of some of these using an exclusive OR operator, or the reserved word `any` followed by a list of roles to be excluded; and a list of *parameters* that might be necessary for certain events. For example, when a teacher leaves the classroom, an event is generated.

For example, if a student generates the event `EnterOnClass`, the entity of student will be linked to that event specifying on the link a type (`send`), a set of valid roles (for example, `student`), and a list of parameters if needed (for example, the classroom, `class03`).

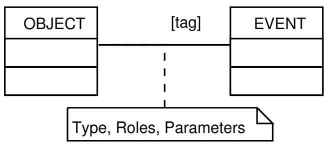

Fig. 6. Specification of events relationships.

Role Comprising interruptible subactivities with both event-triggered and law-controlled execution, a role is formed by the *interruptible specification* defining under what circumstances a specific subactivity is interruptible; a *task list* composed by a *starting law*, a *subactivity* and an *ending law*. Figure 7 shows a graphical representation of this. When we speak about teachers or students, we are using roles.

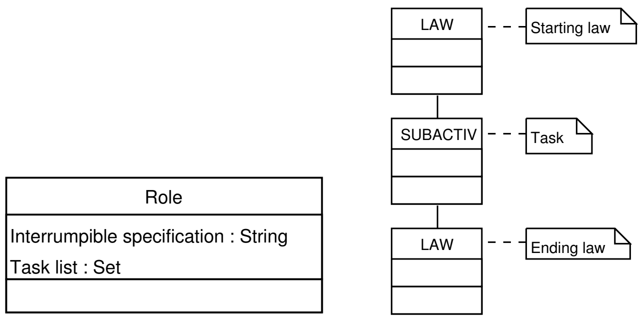

Fig. 7. Specification of roles, and one item of the task list.

4 Implementation Phase

Our design proposal has been specially devised for AmI Systems. In particular, this proposal is applied to U-CAT (ubiquitous collaborative adaptive training) - a Spanish research project for creating intelligent e-learning active spaces. The main goal is to develop an integrated environment that facilitates the realization of educational activities in arbitrary places by using different physical devices in different contexts and situations. These mechanisms should consider not only each user and group's features but also the characteristics of each specific situation or context, as well as information about the available devices for performing the teaching activities. We are currently using several labs to simulate active spaces.

4.1 Blackboard

The previously proposed design has been implemented using a blackboard-based architectural design. This architecture is based on a paradigm called blackboard, which stores the prominent information that is available about the environment at any time.

Since the heterogeneous mix of software and hardware entities of AmI scenarios imposes certain requirements, a global world model combined with an asynchronous communication mechanism is therefore the best approach for achieving complex interactions between components. A blackboard architecture has been adopted in order to implement AmI spaces (laboratories). This blackboard receives and returns information in XML specifications using HTTP protocol. The blackboard is mainly able to store a generic entity and relationships with a basic push/pull information mechanism. Figure 8 shows the blackboard architecture.

There are two different kinds of clients that interact with the blackboard: producers and consumers. These are distinguished according to whether they contribute to or obtain information from the blackboard. When the producers need to communicate new changes, they modify the information stored on the blackboard. There are two ways that consumers can find out what new information is available: they can either consult the blackboard to see if there are any new changes or they can subscribe to blackboard modifications whereby they are notified of any modification. One of the advantages of the proposed paradigm stems from the fact that it is not necessary for each client to be aware of the existence of the remaining components; each client only knows the location of the blackboard and the part of the model that they are interested in. This approach loosely connects the different components on two levels: a temporal level and a spatial level. On one hand, *clients do not need to be synchronized*, which means that a producer can make changes to the model and finish its execution. A consumer can then make a request to the blackboard and retrieve the change since it has been stored. On the other, when a client makes a modification on the blackboard, he/she will *not be aware of the users affected by that change.* Each client interacts with the blackboard as if they were the only one and so development is easier.

This blackboard model [15,19] is provided with a solver system capable of solving imposed restrictions, and can be functionally expanded as new blackboard clients are added. The blackboard solver has been widely used with great success. The addition of a complete representation of an AmI system to the blackboard allows a functional implementation of a modelled system.

Consequently, we can translate our conceptual model to this blackboard. The model will be mapped as a set of objects representing actors, rules, etc. reflecting the current state of the underlying AmI scenario. This architecture allows different clients to request information from the blackboard in order to perform actions, so their synchronization and correctness will be guaranteed if the blackboard reflects the current state in a suitable way so as to manage shared resources.

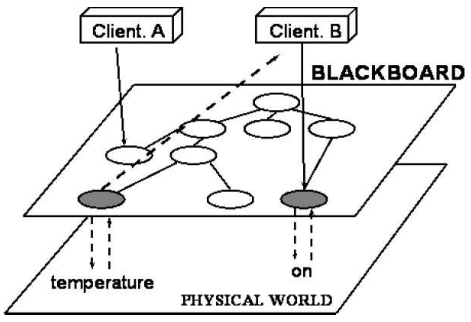

Fig. 8. Representation of U-CAT blackboard architecture.

5 Case Study

In our e-learning AmI environment, an extract of a complex scenario might be described as:

> "Mairi came back home after her class, wanting to do her homework in her room. As long as her classmates are also doing theirs, they can start solving the exercises in a collaborative way, assisted by the system, which allows commentaries and explanations to be exchanged. Any classmate connected to the system can join the brainstorming session, regardless of where they are or what device they are using to interact with the system (a PC, a PDA, a mobile phone, etc.). When Mairi selects an exercise, the system starts searching class notes and related external additional information and informs her of which classmates solved it."

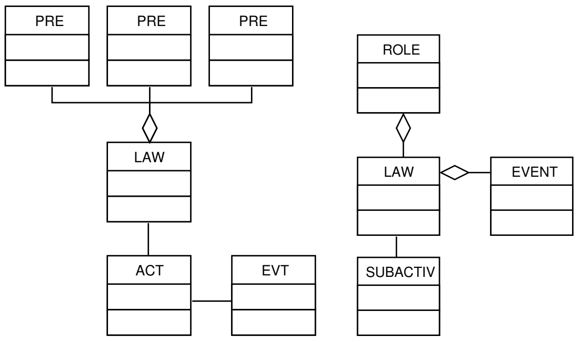

Fig. 9. Representation of object relationship inside case study example.

We can see how the previously described entities are present in this scenario: student role, task/subactivity (`DoHomework`), collaborative for all the classmates. This subactivity execution is triggered by the event `StartHomework` and controlled by the law `HasHomework`. The law is formed by certain preconditions which check the real existence of exercises to be done, and the presence of a device which is useful for doing homework (`NumberHomework`, `HasACapableTerminal`). When a student starts or finishes an exercise, some events are generated into the system (`ExerciseStarted`, `ExerciseEnded`). The system executes its associated subactivity, which looks for information (`SearchFor`) to assist the student. It will also run the subactivity of establishing communication between students when required. If dynamic changes occur in the system, e.g. when a teacher moves from his/her office to the classroom, the proposed design and implementation provide the support for reflecting this fact and launching the appropriate actions (e.g. switch on the lights updating the context of this object). In the same way, when Mairi enters her room, an event is triggered, updating the information stored to reflect the new current context, as a system actor has changed their location on the system.

Storing Information into Blackboard As mentioned above, all the information must be specified in XML when interacting with the blackboard implemented by our system. For educational and clarification purposes, we shall therefore show how part of the law and one precondition are represented in blackboard XML notation. Figure 10 shows a template for introducing entities onto the blackboard using their XML notation, and a fragment of a law using that syntax.

```
<entity name="NAME" id="ID" type="TYPE">
  <property name"NAME">
    <paramSet name="NAME" id="ID">
      <param name="NAME"> TEXT </param>
      ...
    </paramSet>
  </property>
  ...
  <relation name="NAME"
            destination="DESTINATION" id="ID">
  </relation>
  ...
</entity>
```

```
<entity name="Law" id="1001" type="10">
  <property name="[Info]">
    <paramSet name="Self-Information" id="1">
      <param name="Name">HasHomework</param>
    </paramSet>
  </property>
  <relation name="precondition"
    destination="NumberHomework" id="2">
  </relation>
  <relation name="precondition
    destination="HasACapableTerminal" id="3">
  </relation>
  [... actions ...]
  [... logical expression ...]
</entity>
```

Fig. 10. Skeleton of blackboard XML notation for objects, and a fragment of law.

Joining Model Specification and Solver Once all the XML information has been stored on the blackboard, dynamic processing clients attached to the blackboard can operate with the data. This allows the solver to find object goals and to establish preferences between possible candidates as established in the preconditions included in the laws. For example, given a specification of all the existing components in the system, the solver is capable of determining which documents are the most suitable for Mairi to resolve the exercise she is currently solving. The system is then responsible for informing her of the existence of such documents. Notification is another task that the system must accomplish. Depending on the underlying technology available, sending a message, an SMS, etc. will be selected as the specified modelling stage. When all the exercises are finished, the system is responsible for deciding the actions to be performed. If no other classmate is chatting to Mairi (since we have modelled that the student must check the exercises after they have all been solved), the system decides to close the shared brainstorming area and start an exercise checker, adopting the role of academic examiner.

6 Conclusions and Future Works

This paper focuses on ambient intelligence as a new step towards human and group-centred interactions. In this direction, current methods should cover new relevant features such as context-awareness, collaborative work, dynamic or shared knowledge.

Starting from AMENITIES as a conceptual and methodological framework, this paper has proposed a new task-driven design model that allows AmI system features to be taken into account. This proposal focuses on object abstraction as a way of developing a simpler but powerful design model for these systems. This model is implemented on a blackboard model based on a client-server architecture which is able to represent the AmI world consistently. Blackboard clients are added (acting as solvers) in order to determine partial solutions for these constraints, and to provide these systems with required functionality. This two-step model states the computational representation of the concepts and the relationships involved in AmI systems, covering relevant features such as cooperativeness, roles, context-awareness, or shared resources. We conclude that this model, which has been developed from a theoretical conceptual framework, is able to represent the main concepts for AmI systems, supporting their unique features and obtaining an implementable design. Complex scenarios cannot yet be fully managed as the model has not been completed. The model, however, has proved to be a solid model design layer and is both extensible and adaptable. Future work shall be directed towards fulfilling a general scenario, describing context-aware information in detail, and connecting it with new solve features to enhance proactiveness, dynamic space and collaborative spaces. We also plan to extend the methodology to integrate these aspects into the general framework.

7 Acknowledgements

This research is partially supported by a Spanish R&D Project TIN2004-03140, Ubiquitous Collaborative Adaptive Training (U-CAT).

References

1. C.K. Hess, R.H. Campbell: An application of a context-aware file system. Pers Ubiquit Comput (2003) 7: 339-352
2. G. Riva, F. Vatalaro, F. Davide, M. Alcañiz.: Ambient Intelligence. IOS Press, 2005
3. F. Paternò: Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, Nov, 1999
4. A.K. Dey, G.D. Abowd, P.J. Brown, N. Davies, M. Smith, P. Steegels: Towards a better understanding of context and context-awareness. Workshop of Context-Awareness (CHI-2000).
5. G. Montoro, P.A. Haya, X. Alamán. Context adaptive interaction with an automatically created spoken interface for intelligent environments. IFIP Conference on Intelligence in Communication Systems (INTELLCOMM 04). Lecture Notes in Computer Science (LNCS-3283). 2004

6. R. Aldunate, M. Nussbaum, R. González, An Agent-Based Middleware for Supporting Spontaneous Collaboration among Co-Located, Mobile, and not necessarily Known People. Workshop on "Ad-hoc Communications and Collaboration in Ubiquitous Computing Environments" ACM CSCW 2002.
7. N. Oliver. Towards Perceptual Intelligence: Statistical Modeling of Human Individual and Interactive Behaviors. PHD Thesis. MIT Media Lab, 2000
8. M.R. Tazari, M. Grimm, M. Finke. Modeling user context. 10th International Conference on Human-Computer Interaction (HCII), June 2003.
9. J. Pascoe, Nick Ryan, and David Morse: Issues in Developing Context-Aware Computing. HUC'99, LNCS 1707, pp. 208-221, 1999.
10. S. Card, T. Moran, A. Newell. The Psychology of Human-Computer Interaction. Hillsdale, NJ: Erlbaum, 1983
11. A. Takeuchi, T. Naito: Situated Facial Displays: Towards Social Interaction. SIGCHI Conference on Human factors in computing systems, 1995
12. K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, & J-C. Burgelman: Scenarios for Ambient Intelligence in 2010 Final Report Compiled by February 2001 (ISTAG) IPTS-Seville. ftp://ftp.cordis.lu/pub/ist/docs/istagscenarios2010.pdf (Nov 2001).
13. J.M. Carroll. Five reasons for scenario-based design. Interacting with Computers 13 (2000) pp 43-60.
14. W. G. Philips, N. Graham: Workspaces: A Multi-Level Architectural Style for Synchronous Groupware. DSV-IS, 2003. Lectures Notes in Computer Science, 2844. Springer Verlag, 2003
15. P. A. Haya, G. Montoro, X. Alamán. A prototype of a context-based architecture for intelligent home environments. International Conference on Cooperative Information Systems (CoopIS 2004), Lecture Notes in Computer Science (LNCS 3290), 2004.
16. M. Gea, J.L. Garrido, F.L. Gutiérrez, R. Cobos, X. Alamán: Representación del comportamiento dinámico en modelos colaborativos: aplicación a la gestión del conocimiento compartido. Revista Iberoamericana de Inteligencia Artificial, Vol 24, 2004
17. Garrido J.L., Gea M. & Rodríguez M.L.: Requirements Engineering in Cooperative Systems. Requirements Engineering for Socio-Technical Systems. Chapter XIV. IDEA GROUP INC. (USA), 226-244, (2005).
18. Garrido, J.L., Paderewski, P., Rodríguez, M.L., Hornos, M.J. & Noguera, M.: A Software Architecture Intended to Design High Quality Groupware Applications. 4th International Workshop on System/Software Architectures (IWSSA'05) - Proceedings of the 2005 International Conference on Software Engineering Research and Practice (SERP'05) , LAS VEGAS (USA), 59-65, (2005).
19. P.A. Haya, X. Alamán, G. Montoro: A Comparative Study of Communication Infrastructures for the Implementation of Ubiquitous Computing. UPGRADE, The European Journal for the Informatics Professional, Vol 2, 5, 2001
20. Garrido, J.L. (2003). Especificación de la Notación COMO-UML. Technical Report LSI-2003-2. Departamento de Lenguajes y Sistemas Informáticos. University of Granada. Spain.
21. Nardi, B., Ed. Context and Consciousness: Activity Theory and Human-Computer Interaction. Cambridge, MA, MIT Press, 1996.
22. Ellis, C., "Workflow Systems", /Encyclopedia of Distributed Systems/, Dasgupta, P., and Urban, J., (eds). Kluwer Academic Pub., 1998.