Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 66 (2007) 25–43
Science of Computer Programming
www.elsevier.com/locate/scico

Definition and use of Computation Independent Models in an MDA-based groupware development process

José Luis Garrido[a,*], Manuel Noguera[a], Miguel González[b], María V. Hurtado[a], María L. Rodríguez[a]

[a] *University of Granada, Department of Software Engineering, E.T.S.I.I., c/Saucedo Aranda s/n, 18071 Granada, Spain*
[b] *Autonomous University of Madrid, Information Technologies, E.P.S., c/Tomás y Valiente 11, 28049 Madrid, Spain*

Received 15 March 2006; received in revised form 31 August 2006; accepted 13 October 2006
Available online 22 December 2006

Abstract

Groupware systems allow users to be part of a shared environment in order to carry out groupwork. Members of a group belong to organizations in which each one fulfils general and specific enterprise objectives. This paper presents a proposal, from the perspective of the CSCW (Computer-Supported Cooperative Work) systems, for modelling enterprise organization and developing groupware applications. This research work focuses on two specific models for the proposal: a conceptual domain model formalized through a domain ontology, and a system model built using a UML-based notation. The second stems from the first and each provides a Computation Independent View (CIV) with different objectives. Respectively, they allow a common vocabulary for knowledge sharing to be established, and organization functional requirements to be specified, particularly those concerning communication, coordination and collaboration. Furthermore, these models are part of a concrete MDA-based development process of groupware applications that is also introduced.

Keywords: CSCW; Groupware development process; MDA; Ontology; OWL; UML; Enterprise modeling; Software models

1. Introduction

Computer-Supported Cooperative Work (CSCW) [23] studies and analyzes coordination mechanisms for effective human communication and collaboration and also the technological systems supporting them. CSCW systems play an important role in enabling business application integration across organizations, since most enterprises are involved in cooperation processes and are also technology dependent. Nowadays, in order to stay competitive, enterprises must be capable of quickly adapting their business processes to the new dynamic environments [3]. In addition, as a result of technological evolution, enterprises must continuously rethink their business designs, and change their models taking into account new challenges (technology, interoperation, cooperation, etc.) [11]. Social, organizational and technological aspects influence functional requirements of a software system to be developed.

* Corresponding author.
E-mail addresses: jgarrido@ugr.es (J.L. Garrido), mnoguera@ugr.es (M. Noguera), miguel.gonzalez@uam.es (M. González), mhurtado@ugr.es (M.V. Hurtado), mlra@ugr.es (M.L. Rodríguez).

0167-6423/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2006.10.008

Groupware has been defined [15] as "*a computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment*". Groupwork is performed by users using distributed and powered groupware systems. These systems may be implemented by means of many current technologies (Internet, wireless networks, mobile code, ubiquitous computing, etc.) that promote their use in different contexts. The main functional requirements in the development of groupware applications are related to the following key areas [15]:

- *Communication*. This activity emphasizes the exchange of information.
- *Collaboration*. This is an inherent activity in the group context.
- *Coordination*. This is related to the integration and harmonious adjustment of the individual work effort towards the accomplishment of a greater goal.

On the other hand, the inherent complexity of CSCW systems requires a great deal of effort in specifications and development [20]. The development of groupware systems is more difficult than that of a single-user application. Consequently, methodologies and implementation techniques aimed at enhancing group interaction activities should therefore be applied. Model-Driven Development (MDD) [1] has been advocated by academia and industry for many years. Most of the popular and widely-used software engineering methodologies use models as the primary tool for developing software; hence this can claim to follow an MDD approach. MDD can be defined as "*an approach to software development where extensive models are created before source code is written*" [7]. By considering models as first-class entities, MDD aims at reducing the complexity of software production. A primary example of MDD is the Model-Driven Architecture (MDA) initiative of the Object Management Group (OMG) [36].

This paper presents a proposal for modelling enterprises and developing groupware applications from the perspective of the CSCW systems under a concrete MDA-based development process. We argue that special emphasis should be placed on the first stages of this process especially in order to (1) provide conceptual and system models, i.e. Computation Independent Models (CIMs) [36], and (2) define connections between these CIMs and software models. This entails sharing information about how an enterprise is organized (static description of its structure) and relevant aspects of its behaviour (dynamics, member responsibility changes, etc.). Two concrete kinds of CIMs are provided: ontology-based CIMs consisting of a conceptual domain model formalized through a domain ontology and concretized for each particular system using an application ontology [25]; and a system model built by using a UML-based CIM. The first allows a common vocabulary to be established (thereby enabling knowledge to be shared) and the system description to be validated automatically. The second CIM allows specifying functional requirements (especially those focusing on communication, coordination and collaboration) in a more flexible way through an accepted standard (UML), in order to state clear connections between system models and software architectural models [16]. The objective is to facilitate the subsequent software development.

The remainder of the paper is organized as follows. Section 2 presents a brief introduction to the MDA approach. Section 3 describes the starting point for our proposal on the basis of a conceptual domain model and its formalization. Section 4 describes a method for translating the previous conceptual model into a system model closer to the development of groupware applications. It is illustrated by means of an enterprise case study. Section 5 introduces the concrete MDA-based development process. Section 6 presents work in progress intended to carry out connections with another MDA/UML-based development standard in order to obtain additional benefits of our proposal. The final section summarizes the main contributions.

2. Introduction to MDA

MDA [36] is an approach to the development, integration and interoperability of IT (Information Technology) systems. It distinguishes between models designed independently of any technical considerations of the underlying platform, i.e. PIM (Platform Independent Model), and models that include such considerations, i.e. PSM (Platform Specific Model). In addition, at a higher level of abstraction, OMG defines Computation Independent Models (CIMs) [36] which focus on the domain rather than on showing details of the system structure. They provide a vocabulary familiar to the practitioners of the domain in question.

Although MDA supports the elaboration of sophisticated models describing (at various levels of abstraction) the applications to be developed, it does not include, however, precise rules or guidelines explaining how software

engineers can use them [36]. Consequently, in order to address software development for these systems, concrete contributions should be proposed.

To ensure a model-driven approach for software development, standards for representing a variety of specific models are being used or have emerged. Examples of such standards are UML (Unified Modeling Language) [40], MOF (Meta-Object Facility) [37], SPEM (Software Process Engineering Metamodel) [39], EDOC (Enterprise-Distributed Object Computing) [35], etc. In addition, a large number of MDA-compliant tools have been developed providing developers with the capacity to operate on models [7]. In this context, the proposal that we will present in the following sections centres on the definition and use of concrete CIMs as part of an MDA-based development process.

3. Conceptual framework

3.1. Foundations

One key issue in the business process modelling that may entail important time-savings is enterprise information modelling. Although enterprise modelling embraces various aspects (marketing, costs, strategy, etc.) [24], the focus is usually on describing how an organization is structured and operates. An enterprise model can be defined as "*a computational representation of the structure, activities, processes, information, people, behaviour, goals and constraints of a business, government or other enterprises*" [17]. Due to the nature of the cooperation processes, these should be integrated into an enterprise activity model which, in turn, might be directly connected to an organization model [20]. Nevertheless, a variety of terms are often used interchangeably to describe cooperative environment and organization functions. People involved in developing complex systems (as CSCW systems are) do not often agree on the terms used to talk about the entities that may appear in the organization. Furthermore, even when the same terms are used, the meanings associated with them may differ, i.e. the semantics.

A conceptual framework is therefore needed to exchange business process specifications between business people and software engineers using a common vocabulary. The resulting specification must be sufficiently clear, unambiguous and readily translatable into other representations. A conceptual framework is also useful for developers and stakeholders to discuss what entities may appear in this kind of systems. Methodologies aimed at enhancing this representation should be applied. The AMENITIES methodology [20] (intended to analyze, design and develop cooperative systems) is based on behaviour and task models. Most of the concepts present in an enterprise model are included in the conceptual framework provided by this methodology.

The use of ontologies to formalize a conceptual framework (such as the one provided by AMENITIES) improves both the system description and the enterprise modelling. It enables a common vocabulary, knowledge to be shared [16] and the reasoning about the entities described [5]. We have distinguished between the domain and the application level [25] in the ontology design.

In the following subsections, we present the AMENITIES conceptual framework. This is considered as a conceptual domain model. Then, we formalize this model (using a domain ontology) and instantiate it for each particular CSCW system (using application ontologies).

3.2. Definition of the conceptual framework

The conceptual framework proposed in AMENITIES can be represented using a UML class diagram (see Fig. 1) [20]. It establishes the basis for a common understanding between the different participants involved in the task of specifying how an enterprise is organized and operates. It is important to note that in addition to identifying the entities of a cooperative system and their relations, the methodology provides the way of modeling the groupwork itself. This conceptual framework is considered to be a pattern that includes the main common concepts present in CSCW systems, as well as the relationships between these concepts.

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 and so forth) that can play a

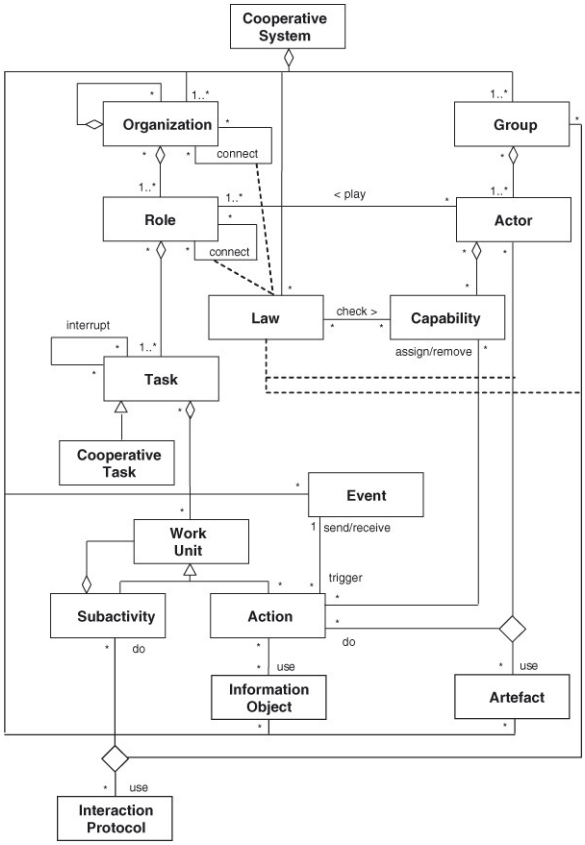

Fig. 1. AMENITIES conceptual framework for cooperative systems.

role in executing (using devices) or being responsible 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 be comprised of related subgroups. A *law* is a limitation or constraint imposed by the system that allows the set of possible behaviour to be adjusted dynamically. An *organization* consists of a set of related roles. Finally, a *cooperative system* consists of organizations, groups, laws, events and devices.

3.3. Formalization of the conceptual domain model in a domain ontology

The representations provided by the UML class diagrams are semi-formal and some explanatory texts should be attached. Furthermore, a representation such as a class diagram is not suitable for machine processing and validation, not to mention textual descriptions. Neither do graphical representations nor natural language specifications (possibly stored in different formats) facilitate interoperability.

In this respect, ontologies and languages developed for the semantic web (e.g. XML-schema [48], RDF [44] and OWL [42]) are proving to be a good tool for formalizing the information description in a machine-processable manner [31,42] and separating domain knowledge from operational concerns. As far as business process modelling is concerned, the enterprise model can be formalized using some of the languages mentioned above.

Fig. 2 shows an excerpt from the domain ontology definition for the conceptual domain model in OWL [42,30]: a cooperative system is formed by a set of *artifacts, events, groups, laws,* etc.; a *role* is a set with at least one task; also, a *role* is part of an *organization*. Statements concerning this appear in bold-type.

The notion of class and relation is shared with that of UML. However, the treatment and modelling differ somewhat. For example, in UML, associations cannot be defined as standalone entities, while in OWL they are defined as instances of a particular class called “Property”. In recent years there has been a lot of work concerning the mappings between elements in both paradigms [38]. Further details and foundations of how this can be accomplished can be

```
<!-- Domain ontology for AMENITIES conceptual framework -->
<owl:Class rdf:about="#CooperativeSystem">
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty>
    <owl:ObjectProperty rdf:ID="hasPart"/>
   </owl:onProperty>
   <owl:allValuesFrom><owl:Class>
    <owl:unionOf rdf:parseType="Collection">
     <owl:Class rdf:ID="Artefact"/>
     <owl:Class rdf:about="#Event"/>
     <owl:Class rdf:ID="Group"/>
     <owl:Class rdf:ID="Law"/>
     <owl:Class rdf:ID="InformationObject"/>
     <owl:Class rdf:ID="Organization"/>
    </owl:unionOf>
   </owl:Class></owl:allValuesFrom>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>...
```

```
<owl:Class rdf:ID="Role">
 <rdfs:subClassOf><owl:Restriction>
  <owl:allValuesFrom><owl:Class rdf:about="#Organization"/>
  </owl:allValuesFrom>
  <owl:onProperty><owl:TransitiveProperty rdf:about="#partOf"/>
  </owl:onProperty>
 </owl:Restriction></rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction><owl:allValuesFrom>
   <owl:Class rdf:about="#Task"/>
  </owl:allValuesFrom><owl:onProperty>
   <owl:ObjectProperty rdf:about="#hasPart"/>
  </owl:onProperty></owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
 <rdfs:subClassOf>
  <owl:Restriction><owl:onProperty>
   <owl:ObjectProperty rdf:about="#hasPart"/></owl:onProperty>
   <owl:minCardinality rdf:datatype="http://www.w3.org/2001/
XMLSchema#int">1</owl:minCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>...
</owl:Class>...
```

Fig. 2. Excerpt of domain ontology for AMENITIES conceptual framework in OWL.

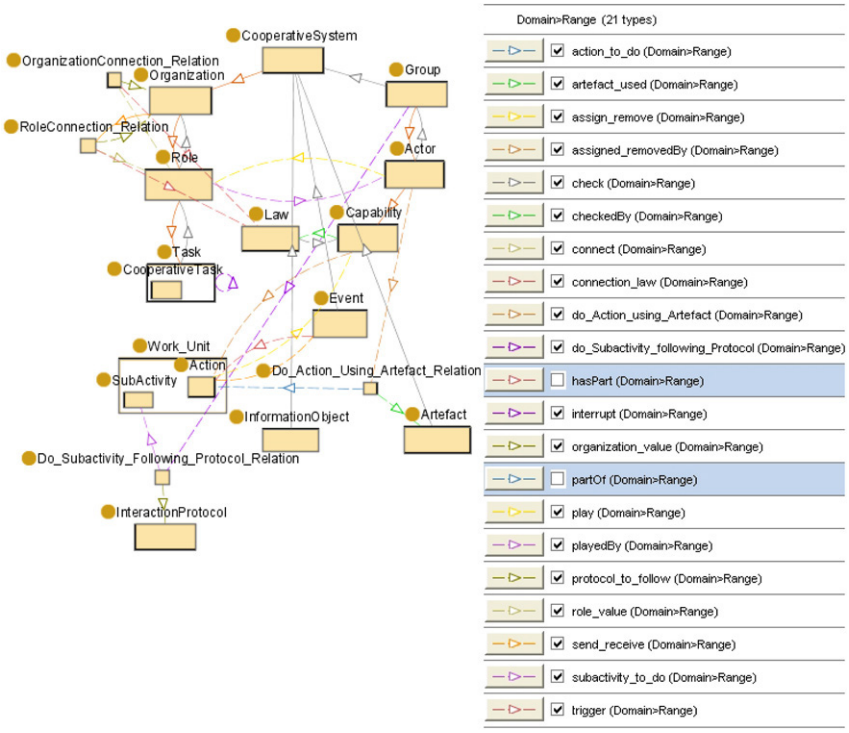

Fig. 3. Graphical representation of domain ontology.

found in [13]. Additionally, this description of the system can be shown to stakeholders, graphically. Fig. 3 depicts how OWL specification of Fig. 2 can be automatically translated into a graphical representation [28] similar to a class diagram. The “hasPart” and “partOf” relations have been omitted for readability reasons.

3.4. Application ontology

The classes described in the domain ontology are instantiated through an application ontology. The application ontology provides a CIM for each particular system. This ontology includes basic relationships (aggregation, association and specialization) between instances of classes.

Fig. 4 shows an example (also generated from an OWL specification) of an application ontology corresponding to three different companies which cooperate in order to agree whether to grant a mortgage which a client has applied for. This case study will be described in depth in the following section showing how the application ontology provides

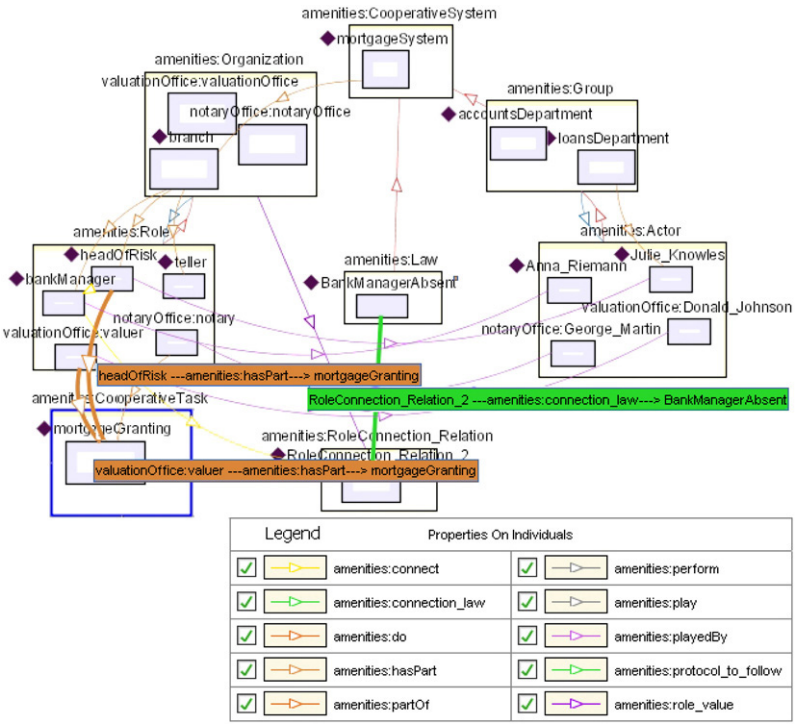

Fig. 4. Example of application ontology obtained from a domain ontology.

the basis for a different system representation more suitable for use in a software development process. Fig. 4 is an ontology-based model which makes explicit the instances that form part of the proposed cooperative system according to the conceptual domain model of Figs. 1 and 3.

4. The UML-based cooperative model

4.1. Motivations and foundations

The ontologies introduced in the previous section serve as a formal substrate against which to validate the elements to be added to the system description. The aim behind the development of groupware applications is to support processes based on interactions between cooperating users.

From the point of view of the software development, as part of the requirement modelling, these interaction processes might be described using, for example, workflows and/or role models, and allow stakeholders and developers to discuss and negotiate requirements. In order to address these issues, a system model, called Cooperative Model (COMO), is proposed. This system model describes the system regardless of its implementation and therefore it also turns out to be another CIM called COMO-CIM.

Most of the interaction processes expressed in this system model are covered by the ontology-based CIMs of Section 3, and the mutual correspondences are shown in the following subsections. However, more development-oriented aspects such as subactivity/action ordering (to be defined in tasks) and event sequence are not addressed in the ontology-based CIMs. That is why the COMO-CIM is useful for specifying a system from the point of view of its structure and behavior. The proposed COMO-UML notation [18] is based on UML state and activity diagrams (notations specially used for specifying software artifacts), and it is basically a graphical notation (operational) that integrates small declarative sections. Classes and associations in COMO-UML preserve the same semantics as in UML. The operational semantics for the COMO-UML notation using the CPN (Coloured Petri Nets) formalism is described in [19]. This semantics enables the verification of certain system properties (deadlocks, liveness, etc.).

Table 1
Correspondences between the ontology-based domain model elements and the COMO-UML elements

Ontology element	Abbreviated description	COMO-UML element
`<owl:Class rdf:ID="Role">`	Designator for a set of related tasks	Class `Role`
`<owl:Class rdf:ID="Law"/>`	Limitation or constraint imposed by the environment	Class `Law`

`<owl:ObjectProperty rdf:about="#connect">`
`  <rdfs:range rdf:resource ="#RoleConnection_Relation"/>`
`  <rdfs:domain rdf:resource="#Role"/>`
`</owl:ObjectProperty>`
	Defines a relationship between two roles	Association `connect`

`<owl:Class rdf:ID="RoleConnection_Relation"/>`		

`<owl:ObjectProperty rdf:ID="role_value">`
`  <rdfs:range rdf:resource="#Role"/>`
`  <rdfs:domain rdf:resource="#RoleConnection_Relation"/>`
`</owl:ObjectProperty>`		

`<owl:ObjectProperty rdf:ID="connection_law">`
`  <rdfs:range rdf:resource="#Law"/>`
`  <rdfs:domain rdf:resource="#RoleConnection_Relation"/>`
`</owl:ObjectProperty>`		

4.2. Method

In order to translate an application ontology into its corresponding COMO-CIM, which will be hierarchically structured, we propose a simple method based on the concepts and relationships of the domain ontology. This method consists of the four following steps: specification of the organization, role definition, task definition and specification of interaction protocols. Only for the first step, the main correspondences between both types of CIMs are described in detail.

4.2.1. Specification of the organization

This reflects the organization structure and constraints on the actors' behaviour. A company's internal structure is based on organizational roles that must be identified. In addition, in relation to coordination requirements, relevant relationships between roles are identified on the basis of constraints imposed by company rules and/or goals. Laws represent these constraints that the organization imposes to control role changes. Laws may also interrogate capabilities, which state skills/categories that actors have acquired. Both concepts are very helpful when defining and analyzing possible organization strategies (e.g. behaviour patterns) and group dynamics (representing group behaviour).

The previous domain ontology is reused in order to automatically generate this new representation. To achieve this, translation schemes between these different models must be clearly established. Table 1 shows some correspondences related to the specification of the organization between the elements in both types of model.

All the classes in the OWL ontology are mapped onto classes of the COMO-UML notation. The OWL *Role* class is mapped onto the COMO-UML *Role* class. The same is for the class *Law*. Additional classes and relations (*properties* in the OWL jargon) must be defined with respect to the 3-ary association *connect* (two roles, source and destination, and the law which governs the transition between them). This is because OWL language does not provide the necessary vocabulary so that n-ary relations may be defined [34]. In this case we have defined one fictitious class (*RoleConnection_Relation*) and two fictitious properties (*role_value* and *connection_law*) which link the subject of the relation (the source role) with its objects (the target role and the law that rules the change).

4.2.2. Role definition

Each organization divides the workload among its members, whereas each role establishes a connection between these and specific tasks (individual and/or cooperative). This step is intended to define each previous role by the set of

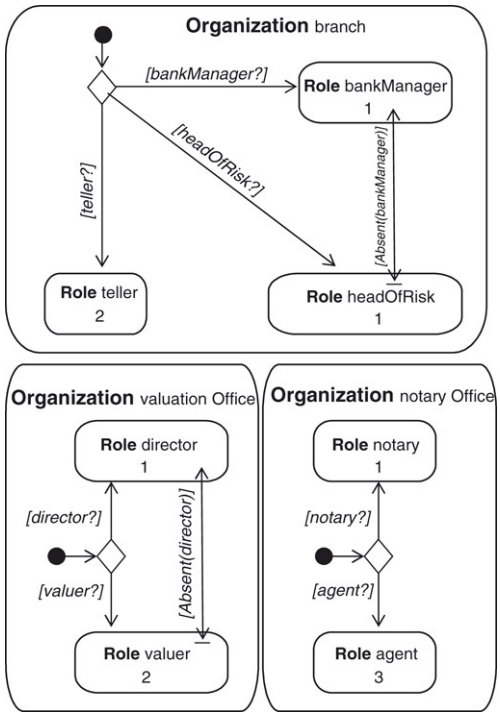

Fig. 5. Organization diagrams.

tasks that can/must be performed. The tasks involved are specified by taking into account relevant requirements that might affect the participants' behaviour. For instance, relevant information would be the event that triggers a task or interrupts it when it is being performed (denoting task priorities).

4.2.3. Task definition

Tasks define work that can be directly identified in relation to individual or group goals. In this step, each previously specified task is broken down into related subactivities. They describe cognitive capabilities that participants need to accomplish work. Each subactivity/action includes the specification of those responsible and optional roles needed to accomplish it.

4.2.4. Specification of interaction protocols

In this last step, the interaction protocols in the above task definition are described. In each non-structured collaborative activity, the type of protocol used to accomplish this objective between participants is explicitly specified. The identification of such protocols is extremely helpful since they identify system requirements such as type of communication required (synchronous and asynchronous) and type of communication channel (link, mailbox, etc.) for supporting collaboration.

4.3. Case study

We illustrate the previous method for obtaining the COMO-CIM using the above mentioned case study (i.e. three companies that cooperate in order to agree whether to grant a mortgage). This model stems from the application ontology shown in Fig. 4 using the COMO-UML notation. Again, only for the specification of the organization, the main correspondences between the main elements are described in detail.

4.3.1. Organizations

The case study comprises three organizations: branch, valuation office and notary's office. The organization diagrams obtained are shown in Fig. 5.

Table 2
Specification of the organization (application ontology level)

Ontology element	COMO-UML element
`<amenities:Role rdf:ID="headOfRisk">`	Role *headOfRisk*
`<amenities:Role rdf:ID="bankManager">`	Role *bankManager*
`<amenities:Law rdf:ID="bankManagerAbsent"/>`	Law *bankManagerAbsent*
`<amenities:connect>`	Association Connect
`<amenities:RoleConnection_Relation rdf:ID="roleConnection_Relation_1">`	

`<amenities:connection_law>`
`  <amenities:Law rdf:ID="bankManagerAbsent"/>`
`</amenities:connection_law>`	

`<amenities:Role rdf:ID="headOfRisk">`
`  <amenities:connect>`
`  <amenities:RoleConnection_Relation rdf:ID="roleConnection_Relation_1">`
`  <amenities:role_value rdf:resource="#bankManager"/>`
`  <amenities:connection_law>`
`  <amenities:Law rdf:ID="bankManagerAbsent"/>`
`  </amenities:connection_law>`
`  </amenities:roleConnection_Relation>`
`  </amenities:connect>`
`</amenities:Role>`	

For instance, the branch organization has three specialities: (*bankManager*, *headOfRisk* and *teller*) and one group with four members specified for each role as multiplicity value (one playing role *bankManager,* one playing the *headOfRisk* role and two more playing the *teller* role). Initially, laws (identified as *[<law>]*) checking capabilities (identified as *<capability>?*), e.g. *[bankManager?]* determine the role that is played by each member according to his/her professional category, specialization, etc. Members may dynamically change roles that they play as a result of various circumstances. One example of this requirement is that the organization imposes laws such as *[Absent(bankManager)]*, i.e. the actor playing the role *headOfRisk* can become the *bankManager* if the manager is absent.

In relation to the correspondences between the conceptual domain elements and the COMO-UML elements (see Table 1), the Table 2 shows how it is specified that an actor playing the role *headOfRisk* may start playing the role *bankManager* when the actor playing this last role is absent. This consists of a 3-ary relation comprising the source role, the target role and the law which rules the transition. It can be identified as the element *roleConnection_Relation_1*, an instance of the fictitious class *RoleConnection_Relation* (see Section 4.2.1). It implements the connection between the roles *headOfRisk* and *bankManager* through the fictitious relations *role_value,* with the value *bankManager*, and *connection_law*, with the value *bankManagerAbsent*. This constraint has also been highlighted in the graphical representation of the ontology-based application model (see Fig. 4).

4.3.2. Roles

Fig. 6 shows two obtained role definitions from the application ontology, i.e. the roles *headOfRisk* and *valuer* which belong to different organizations, respectively, *branch* and *valuationOffice*. The common task they are all involved in is the cooperative task *mortgageGranting*. This form of specification allows us to associate different context elements (events, actions, number of members) for each role involved in the same task. In this example, all roles (including this task) specify that no explicit event is required to be triggered, but at the same time, the organization requires that first, the mortgage has been applied for (law *[mortgageApplied]*), so that the task can be started. In addition,

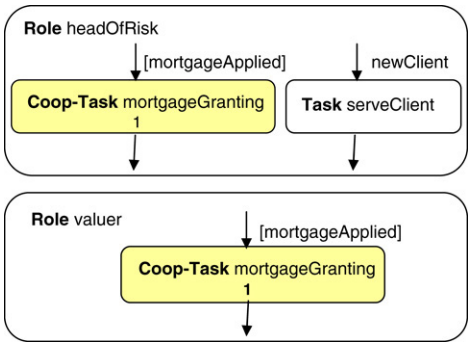

Fig. 6. Role diagrams.

for the role *headOfRisk*, when task *mortgageGranting* is being performed, it could be interrupted if a new client arrives at the office (individual task *serveClient*). The number of members required for the same cooperative task is specified with the multiplicity value associated with each occurrence of that task for each role (one *headOfRisk* and one *valuer*).

4.3.3. Tasks

As an example of mapping between models, Fig. 7 shows the task *mortgageGranting*. Since this is a cooperative task requiring more than one participant to accomplish it, it specifies that a collaboration requirement be satisfied. The COMO-UML notation enables us to specify temporal-ordered constraints of subactivities/actions by means of sequential (arrows) and concurrent (thick bars) constructions. Bifurcations (labelled with a diamond) denote a decision point. The current ontology-based models do not address the specification of this kind of information.

A subactivity is to be described in more detail. This notation assigns roles to subactivities/actions. Each subactivity/action includes the specification of those responsible roles needed to accomplish it. Task definition may also include relevant requirements about the task:

- information to be needed and used, even in a required status (e.g. *valuationReport*)
- coordination mechanisms
- organization politics (decision making, protocols, workload, etc.).

4.3.4. Interaction protocols

In order to carry out non-structured work, the interaction protocols between participants must be identified and associated to the corresponding subactivities. The role specification for each subactivity/action may comprise one of the following operators: optionally (*[]*), addition (+), inclusive-OR (|) and exclusive-OR (*X*). For instance, in Fig. 7 a decision-making (subactivity *decideConcession*) is performed by both the *bankManager* and *headOfRisk*. For this, they use a conversational protocol, face-to-face (synchronous communication) and a shared workspace where required information is available.

5. A systematic development process

In order to reach the objective of stating clear connection between previous CIMs and software models, this section proposes the MDA-based development process for groupware applications. It is made concrete defining specific MDA artifacts to be applied.

5.1. General scheme

The development process is depicted in Fig. 8, which is described below according to its correspondence with phases in a general software development process [10,47]. This process includes in its early phases the previously introduced CIM models, and in subsequent phases, connections with architectural software models already applied to real developments of groupware applications [21].

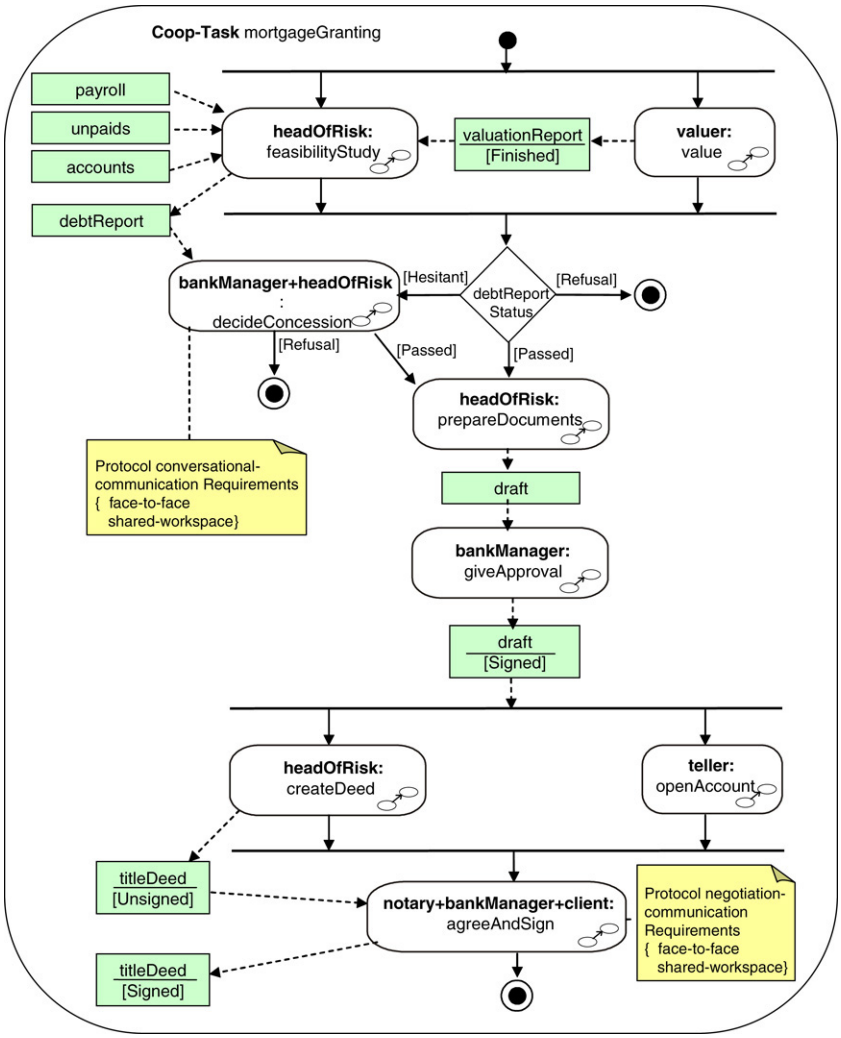

Fig. 7. Task diagram.

5.1.1. Requirements engineering

Requirements engineering is defined as "*a systematic process of developing requirements through an iterative, cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained*" [43]. Certain stages of the process such as analysis, viability and modelling, should provide a full description of requirements, including functional and non-functional requirements to be satisfied. Requirements engineering also embraces the development of abstract system models (like part of a detailed system specification) [33].

The starting point for this phase is the domain ontology described in Section 3. This model is assumed as the first CIM in the process whose instances are defined as application ontologies. The development of the models to be built in subsequent phases stems from the translation of each particular application ontology into an MOF-compliant model (i.e. the UML-based cooperative model introduced in Section 4). This is accomplished by the definition of a UML profile for standardizing the COMO-UML notation used. Although the resulting model continues to be another CIM model, this translation enables us to connect easily system models with software development models [8], since these last models might use the UML standard modelling language [16]. The output of this requirement phase will be a CIM, above called COMO-CIM. Since this CIM is a system model centred on specifying functional requirements, it includes parts to be identified directly with cooperative processes which must be supported by groupware applications/tools.

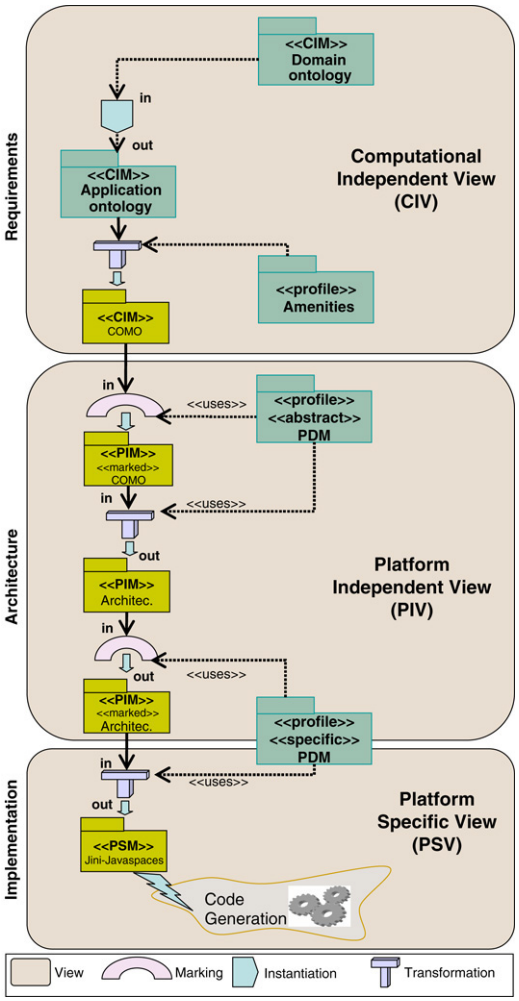

Fig. 8. MDA scheme for developing groupware applications.

5.1.2. Architectural design

An architectural design is defined [45] as the design of "*the overall structure of the software and the ways in which that structure provides conceptual integrity for the system*". The design of this architecture "*occupies a pivotal position in software engineering*" [12]. An architectural design [26] might specify the following basic elements: subsystems, components, interactions, ports, etc. Subsequently, architectural design is the next phase to address in the development process. In this way, the COMO-CIM must then be marked by taking into account an abstract PDM (Platform Description Model) [6] in order to provide a logical view for the architectural design [21].

This marking phase allows us to link domain elements (roles, cooperative tasks, etc.) represented in the COMO-CIM with architectural software elements. For example, let us consider information about the role (*bankManager*) that an actor must play in order to carry out the specific subactivity of approving the granting of a mortgage (*giveApproval*). This would result in a requirement to be satisfied by a groupware application if its software implementation is required. The resulting PIM (Platform Independent Model) [2] is stereotyped as <<marked>> and named COMO-PIM (COoperative MOdel-Platform Independent Model). This PIM represents architectural design aspects such as what subsystem will be in charge of implementing previous functional requirement.

5.1.3. Detailed design and implementation

The following step corresponds with a detailed design phase since architectural PIMs even exclude platform-specific details. Therefore, another marking step (according to specific PDM) is required in order to obtain a PSM. For example, an implementation of a groupware application using a Jini platform [21] must add the marks corresponding

Table 3
AMENITIES Profile: Stereotypes

Amenities diagram type	Stereotype	UML metaclass (base)	Parent
Organizations	OrganizationSpec	StateMachine	N/A
	Role	State	N/A
	ComputerRole	State	Role
	Law	Constraint	N/A
	BA_Transition	Transition	N/A
	Task	State	N/A
	Coop-Task	State	Task
	AME_Action	Activity	N/A
Task, sub-activities and interactions	ResponsableSpec	Class	N/A
	UnitOfWork	ExecutableNode	N/A
	AME_SubActivity	ExecutableNode	UnitOfWork
	AME_Action	ExecutableNode	UnitOfWork
	LocalCondition	Class	N/A
	Team	Class	N/A
	RoleBinding	Class	N/A
	Communication Req	Class	N/A
	Interaction ProtocolSpec	Class	N/A

to the Jini [46] architectural paradigm, such as "services", "leasing", "principals", or "events". Finally, this PSM shall be used to generate groupware applications. This result is called by some authors [4,8] as Platform Specific Implementation (PSI).

The system could be implemented in any other specific platform because MDA process would allow generating different PSMs depending on each final implementation platform. In addition, as MDA promotes, it is possible to restart this process at any phase using the corresponding generated model.

5.2. MDA artefacts

For standardization of the COMO-CIM we have chosen *profiling* as an alternative in order to extend UML metamodels. This is because it is a simple, direct and effective mechanism for capturing domain specific characteristics in the form of models [2,8]. The profile is based on UML 2.0 Superstructure [40] and it has been divided into three groups of use on the basis of the main diagrams provided by the COMO-UML notation:

– Organizations (specification of organizations, bidirectional and additive transitions, etc.)
– Roles (cooperative tasks, events, etc.)
– Tasks, subactivities and interactions (subactivity, units of work, communication requirements, etc.).

Tables 3 and 4 are respectively, a summary of the main stereotypes and tagged values defining this profile.
Fig. 9 shows the relationships between the profile and the UML metamodel packages [40]:

- The Kernel, "*represents the core modelling concepts of the UML, including classes, associations and packages*". *InformationObject, Capacity* and *Law* stereotypes are based on this package.
- The PrimitiveTypes "*have been defined for use in the specification of the UML metamodel. These include primitive types such as Integer, Boolean and String*". We have used these for tagged values (e.g. the *IsInterruptByAny* and *IsAdditive* booleans).
- The BasicActivities, "*supports modelling of traditional sequential flow charts. It includes control sequencing*". Additionally, IntermediateActivities provides concurrent control and data flow (e.g. *JoinNode*, *ForkNode*) and StructuredActivities, i.e. the traditional structured programming constructs (e.g. *ConditionalNode*, *LoopNode*).
- Communications, BasicBehaviours and BehaviourStateMachines introduce the different mechanisms to specify behaviours such as automata and Petri-net. For example, the BehaviourStateMachines package provides key elements like StateMachine, State and Transition which are the basis for OrganizationSpec, Role and BA_Transition stereotypes.

Table 4
AMENITIES Profile: Tagged values

Amenities diagram type	Tag	Stereotype	Type	Multiplicity
Organizations	Multiplicity	Role	Multiplicity	1
	IsBidirectional	BA_Transition	Boolean	1
	IsAdditive	BA_Transition	Boolean	1
Roles	Multiplicity	Coop-Task	Multiplicity element	1
	IsInterrupt ByAny	Interruptible-Task	Bolean	1
	Interruptby set	Interruptible-Task	Task	0..n
	Actions	InformationObject	AME_Action	0..n
	Information	AME_Action	InformationObject	1..n
Task, sub-activities and interactions	Makes	ResponsableSpec	UnitOfWork	1..n
	Comment	UnitOfWork	Comment	1
	Responsables	UnitOfWork	ResponsableSpec	0..n
	CommRequires	Team	CommunicationReq	1..n

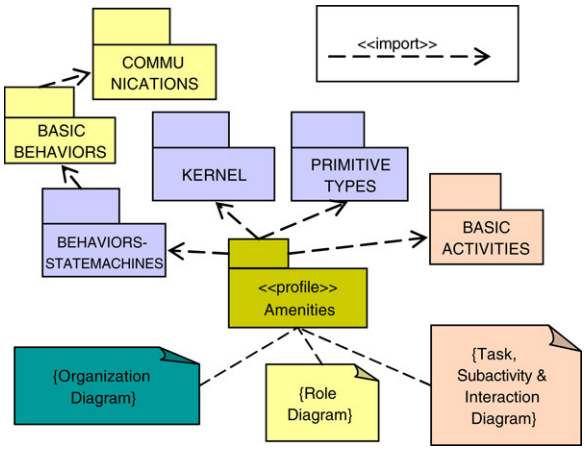

Fig. 9. Relation between Amenities Profile and UML 2.0 packages.

As defined by MDA, the transformation between models (i.e. from CIM to PIM, and from PIM to PSM) needs the source model to be marked according to the mapping defined for the target model [36]. Then, once this profile has been defined, we establish their correspondence with a software architectural model that is technology independent and specially devised for developing groupware applications. This architectural design proposal promotes the division/partitioning of the whole system into components (called subsystems) to facilitate its development, evolution and maintenance [21] (see Fig. 10). The *Identification* (users' access control), *Metainformation* (metadata management) and *Awareness* (contextual information) subsystems are always present for every groupware application. The application subsystem is obviously specific and is to be developed for each particular groupware application.

Table 5 shows basic mappings from some elements of Amenities Profile (for CIMs) and this target architecture (for PIMs). Concretely, *Role* and *AME_subactivity* stereotypes, respectively representing organizational and activity concerns, are considered to be translated into crucial architectural elements like components and subsystems.

With this mapping, a software architect can take the appropriate design decisions in order to accommodate the domain requirements and architectural restrictions [14]. We make it by means of properties such as *Arch_name*, to name the target element to generate, and *Subsystem_type* to indicate where to locate the element within a final system subdivision. In addition, several transformations could be automatic. This is the case for the *Role to Class* mapping which forces the obtained class to be encapsulated within the *MetaInformation* subsystem. In this way, this subsystem would be in charge of storing certain information specified for the system (possible roles to be played, laws constraining role changes, etc.) and supports all the functionality for managing metadata (new roles acquired, laws applied by the organization, etc.).

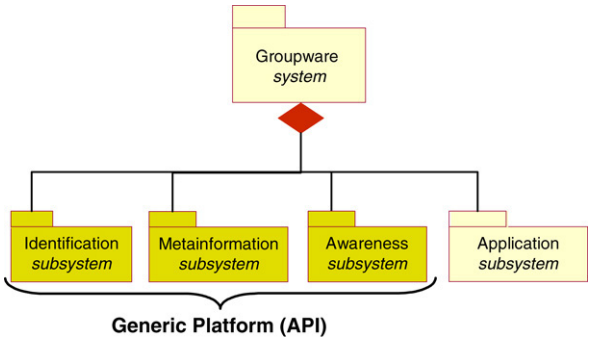

Fig. 10. Architecture for groupware application: component view.

Table 5
Example of mapping from CIM to PIM

Stereotype/Metamodel element	Added at	Target architectural element	Map comment	See associated properties
Role	CIM	<<Class>>	The <<subsystem>> component containing this element will be "MetaInformation"	Arch_name
AMEsubactivity	CIM	N/A	The <<subsystem>> component containing this element will be in the set: (Identification, Awareness, Application)	Subsystem_Type, Arch_name
Arch_name (property)	CIM to PIM mapping	If subsystem_type not in ("Identification", "Awareness", "MetaInformation") gives the value of attribute "name", else nothing.	It defines the name of the target architectural element.	
Subsystem_type (property)	CIM to PIM mapping	If subsystem_type in ("Identification", "Awareness", "MetaInformation") gives the value of attribute "name".	It defines the name of the target architectural element.	

5.3. Applying MDA to the case study

An example of use of this profile for the roles and transitions in the organization *valuationOffice* is shown in Fig. 11. It should be noted that the characteristics which are not present in UML, such as additive transitions, are obtained by means of tagged values such as *IsBidirectional* and *IsAdditive*. It is worth mentioning how the lack of current industry profiles covering all of these topics due to their more general aim (e.g. the EDOC Profile) made it necessary to adopt an ad hoc proposal for CSCW systems.

Then, considering this *valuationOffice* organization from the COMO-CIM and using the mapping (see Table 5) to the software architecture, the *Director* and *Valuer* roles would be transformed into two classes (entities belonging to metaInformation subsystem) with the same names by setting its *Arch_name* property.

We also show how we have addressed the specification of *mortgageGranting* cooperative task (see Fig. 7). Elements like *decideConcession, giveApproval and prepareDocuments* subactivities, which can't be translated automatically to an architecture, must be marked (see Fig. 12) with *Subsystem_type* property in order to decide

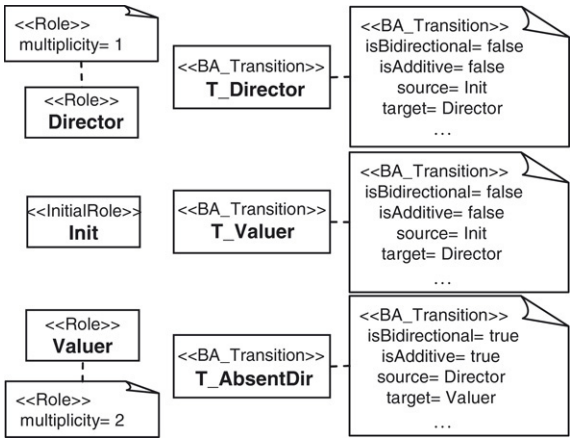

Fig. 11. *valuationOffice* organization instance: stereotypes values within COMO-CIM.

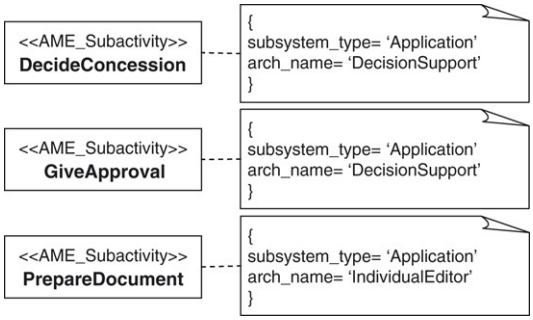

Fig. 12. Stereotyped subactivity instances after an architectural marking step.

whether they belong to a general purpose subsystem like *Identification* and *Awareness* or to a concrete *Application* subsystem. In this case, the two first subactivities could have been marked with *Subsystem_type="Application"* and *Arch_name="DecisionSupport"* meaning its functionality would be included in the same *Application* subsystem. This facilitates the coordination in order to reach the needed decisions by means of a shared workplace, for instance. The architect can decide that the third subactivity *prepareDocument* would be supported by means of a single-user publication tool. Thus, the marking would differ in *Arch_name property*, for example with "*IndividualEditor*" as value.

Once every relevant element has been marked (if not automatic mapping associated) to pass into the next phase, those marks would be processed in order to compose the corresponding target model (Architectural-PIM).

The next step, "detailed design and implementation", continues in the same way to complete the development process. We have achieved successful experiences by using the data-driven programming model provided by the JavaSpaces technology (based on Jini [46] and which implements the Linda coordination model [9]): a collaborative appointment book application for groupwork has been implemented [21]. There exist several similar experiences, which resolve these steps with other technological platforms (e.g. by using CORBA [32]).

6. Future work

Future work is targeted in four directions to complete the current proposal on the basis of its main weak points. The first target is improving CIMs in order to achieve an automatic translation between ontology-based and UML-based CIMs. For example, to enable the specification of temporal-ordered constraints of subactivities/actions in ontology-based CIMs. Currently this is only supported in UML-based CIMs. Some properties and expressive power of the UML-based cooperative model, such as those concerning task and event sequence, are not present in our domain and application ontologies and must be described in a more specific task ontology. Moreover, another approach would be to address the development phase connecting the ontologies with the software architecture directly.

Second, we consider EDOC in architectural and detailed design phases. This OMG proposal "*simplifies the development of enterprise distributed component-based systems by means of a modelling framework conforming*

Table 6
Architecture for groupware and EDOC

Our architectural proposal for groupware	EDOC	Aspects
Yes	Yes	PIM oriented
No	Yes (e.g. FCM, J2EE, CORBA)	(PSM) technology mappings
Yes	Yes	Component-based
Yes	Yes	RM-ODP computational viewpoint
No	Yes (ECA entities profile)	RM-ODP information viewpoint
Yes (identification, awareness and metainformation subsystems)	No	CSCW oriented
No	Yes (OMG's)	Standard

to MDA" [35]. The EDOC Profile provides a set of profile elements whose core is a technology independent profile, the Enterprise Collaboration Architecture (ECA) [41]. Seven significant aspects lead us to consider EDOC as one interesting alternative. Table 6 shows a comparative between the architectural proposal for groupware (used in Section 5.2) and EDOC. Note that the standardization is a strong point in EDOC since it uses the conceptual framework provided by the Reference Model of Open Distributed Processing (RM-ODP) [27]. However, EDOC aims for universality so that it lacks specific groupware-oriented artefacts, unlike our proposal.

Third, the aim is clearly to identify the requirements of the system so that it can follow the life of each requirement, in both a forwards and backwards direction in order to maintain an adequate requirements traceability.

Finally, we plan to incorporate quality evaluation in our proposal. To achieve this goal, it is necessary to identify the key issues in both development and evaluation processes in order to provide criteria that allow selection techniques to be applied.

7. Conclusions

In this paper, we have addressed the issue of connecting enterprise models to software models. A conceptual domain model has been formalized using ontology-based models. This conceptual domain model provides a common vocabulary of concepts and relations to be used in the specification of some aspects present in enterprise models. In addition, using ontologies enables changes in the description of a class to be documented (i.e. if it is incompatible with a previous version) and to state the mappings between the classes described in different models (i.e. if they are equivalent, disjoint, etc.). While the first is useful for questions of maintenance and evolution, the second is useful for questions of reuse and interoperability. At the same time, the underlying application ontology helps validate the system.

In order to translate the ontology-based models into system models which enable functional requirements of the software to be developed, a correspondence between the elements in the different types of models has been defined. The system model represents the system on the basis of its structure and behaviour. The expressive power of the used notation (COMO-UML), promoting participatory design, makes it easy for stakeholders to be involved in the requirement negotiation process. Several approaches and standards, such as BPMN (Business Process Modeling Notation), UML and so forth, have been proposed for business process modelling from the business analyst's point of view [22]. In particular, BPMN defines a BPD (Business Process Diagram) slanted to workflow descriptions for modelling private and collaborative (B2B) business processes [29]. In comparison with the COMO-UML and these other notations, the former additionally includes other modelling aspects [18]:

- Dynamic changes related to some classes in the problem domain, for example, the role played by an actor may change throughout a real situation and can be specified using the COMO-UML organization diagrams (see Section 4.3.1).
- Advanced specification (using operators such as optionally, exclusive-or and so forth) of different role responsibilities or functional capabilities in performing subactivities/actions. This is an extension to BPD and UML swimlanes (see Section 4.3.4).
- The possibility of specifying that tasks may be interrupted and which the interruptible tasks are. It influences how an actor is involved in concurrent work.

- Use of interaction protocol for specifying main interaction characteristics between actors that collaborate by performing non-structured work. These characteristics help to determine requirements for particular tools to be used in supporting collaboration processes (see Section 4.3.4).

Finally, likewise BPML (Business Process Modeling Language) by BPMN and other competing standards such as BPEL4WS (Business Process Execution Language for Web Services), in our case a concrete MDA-based development process has been introduced intended to support the generation of required groupware applications in a software development process. As far as this process is concerned, a conceptual framework is the starting point for creating two types of CIMs with different objectives. The formalization of the conceptual model itself is the first CIM. The final objective of this process is to state clear connections between the different parts of these CIMs models and software models which would fulfil specified requirements. For that purpose, several MDA artifacts have been created (a UML profile) and used (a groupware architecture, specific development platforms such as JavaSpaces, etc.). The profile has been defined for a UML-based notation called COMO-UML and used to create an MDA-compliant CIM model. In other words, the UML language is extended to a CSCW domain specific modelling language; in [16] a method is described in order to restrict the syntax of UML for domain modeling. We argue that this proposal also helps to reduce the gap between requirement modelling and the subsequent software implementation because the same language is used to model the CSCW system as well as the software in it. Benefits have also become apparent from the experience acquired in the development of real groupware applications. However, the process has not been applied strictly due to the lack of CASE tools (still not developed).

Acknowledgement

This research is partially supported by the R+D project TIN2004-08000-C03-02 of the Spanish MCYT.

References

[1] J. Aagedal, I. Solheim, New roles in model-driven development, in: 2nd European Workshop on MDA, Technical Report No. 17-04, September 2004.
[2] M.S. Abdullah, A. Evans, I. Benest, C. Kimble, Developing a UML profile for modelling knowledge-based systems, in: Proceedings of Model-Driven Architecture: Foundations and Applications, MDAFA 2004, 2004, pp. 202–216.
[3] L.F. Andrade, J.L. Fiadeiro, Agility through coordination, Information Systems 27 (2002) 411–424.
[4] J.P.A. Almeida, L. Ferreira Pires, M. van Sinderen, D. Quartel, A Systematic approach to platform-independent design based on the service concept, In: Proceedings Seventh IEEE International Conference on Enterprise Distributed Object Computing, EDOC 2003, Brisbane, Australia, September 2003, pp. 112–123.
[5] F. Baader, D. Calvanese, D. McGuineness, D. Nardi, P. Patel-Schneider, The Description Logic Handbook, Cambridge University Press, 2003.
[6] A. Belangour, J. Bézivin, M. Fredj, Towards Platform Independence: a MDA Organization, INRIA ATLAS de l'Université de Nantes, France, LGI-ENSIAS Rabat, Maroc, 2002.
[7] R. Bendraou, P. Desfray, M.P. Gervais, MDA components: A flexible way for implementing the MDA approach, in: ECMDA-FA 2005, in: LNCS, vol. 3748, 2005, pp. 59–73.
[8] T. Bloomfield, MDA, Meta-modelling and model transformation: Introducing new technology into the defence industry, in: ECMDA-FA 2005, pp. 9–18.
[9] N. Carriero, D. Gelernter, Linda in context, Communications of the ACM 32 (4) (1989) 444–458.
[10] J. Castro, M. Kolp, J. Mylopoulos, Towards requirements-driven information systems engineering: The TROPOS project, Information Systems 27 (2002) 365–389.
[11] H.-M. Chen, R. Kazman, A. Garg, BITAM: An engineering-principled method for managing misalignments between business and IT architectures, Science of Computer Programming 57 (2005) 5–26.
[12] L. Chung, D. Gross, E. Yu, Architectural design to meet stakeholder requirements, in: P. Donohue (Ed.), Software Architecture, Kluwer Academic, San Antonio, TX, 1999, pp. 545–564.
[13] D. Djuric, MDA-based ontology infrastructure, Computer Science and Information Systems (ComSIS) 1 (1) (2004) 91–116.
[14] M. Eichberg, MDA and programming languages, in: J. Bettin, G. van Emde Boas, C. Cleaveland, K. Czarnecki (Eds.), Workshop Generative Techniques in the Context of Model-driven Architecture, OOPSLA 2002, November, 2002.
[15] C.A. Ellis, S.J. Gibbs, G.L. Rein, Groupware: Some issues and experiences, Communications of the ACM 34 (1) (1991) 38–58.
[16] J. Evermann, Y. Wand, Toward formalizing domain modeling semantics in language syntax, IEEE Transactions on Software Engineering 31 (1) (2005).
[17] M.S. Fox, M. Gruninger, On ontologies and enterprise modelling, in: Proceedings of the International Conference on Enterprise Integration Modelling Technology 97, Springer Verlag, 1997.
[18] J.L. Garrido, Especificación de la notación COMO-UML, Tech. Rep. No. LSI-2003-2, Granada, Spain, University of Granada, Departamento de Lenguajes y Sistemas Informáticos, 2003.

[19] J.L. Garrido, M. Gea, A Coloured Petri Net formalisation for a UML-based notation applied to cooperative system modeling, in: P. Forbrig, et al. (Eds.), Interactive Systems — Design, Specification and Verification, in: LNCS, vol. 2545, Springer, 2002, pp. 16–28.
[20] J.L. Garrido, M. Gea, M.L. Rodríguez, Requirements engineering in cooperative systems, in: Requirements Engineering for Sociotechnical Systems, Idea Group, Inc., USA, 2005 (Chapter XIV).
[21] J.L. Garrido, P. Padereswki, M.L. Rodríguez, M.J. Hornos, M. Noguera, A software architecture intended to design high quality groupware applications, in: Proc. of the 4th International Workshop on System/Software Architectures, IWSSA'05, Las Vegas, USA, June 2005.
[22] P. Green, M. Rosemann, M. Indulska, C. Manning, Candidate interoperability standards: An ontological overlap analysis, Data Knowledge Engineering (2006), doi:10.1016/j.datak.2006.08.004.
[23] S. Greenberg, Computer-Supported Cooperative Work and Groupware, Academic Press Ltd., London, UK, 1991.
[24] M. Gruninger, K. Atefi, M.S. Fox, Ontologies to support process integration in enterprise engineering, Computational and Mathematical Organization Theory 6 (4) (2000) 381–394.
[25] N. Guarino, Formal ontology and information systems, in: N. Guarino (Ed.), Proceedings of FOIS'98, Trento, Italy, June, 1998, IOS Press, Amsterdam, 1998, pp. 3–15.
[26] C. Hofmeister, R.L. Nord, D. Soni, Describing software architecture with UML, in: Proceedings of the First IFIP Working Conference on Software Architecture, WICSA1, San Antonio, TX, February 1999.
[27] ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation X.901, X.902, X.903, X.904, "Open Distributed Processing - Reference Model", OMG, 1995–1998.
[28] Jambalaya 2.2.0 build 15 2005/07/07 15:04, The Jambalaya Project, More info at http://www.thechiselgroup.org/jambalaya.
[29] J.Y. Jung, H. Kim, S.H. Kang, Standards-based approaches to B2B workflow integration, Computers & Industrial Engineering 51 (2) (2006) 321–334.
[30] H. Knublauch, R.W. Fergerson, N.F. Noy, M. Musen, The Protégé OWL Plugin: An open development environment for semantic web applications, in: LNCS, vol. 3298, 2004, pp. 229–243.
[31] I. Lera, C. Juiz, R. Puigjaner, Performance-related ontologies and semantic web applications for on-line performance assessment of intelligent systems, Science of Computer Programming 61 (2006) 27–37.
[32] R. Maciel, et al., An MDA domain specific architecture to provide interoperability among collaborative environments, in: 19° Brazilian Symposium on Software Engineering, 2005.
[33] L.A. Macaulay, Requirements Engineering, Springer, 1996.
[34] N. Noy, A. Rector, Defining n-ary relations on the semantic web, Editor's draft 7 September 2005. http://smi-web.stanford.edu/people/noy/nAryRelations/n-aryRelations-2nd-WD.html.
[35] Object Management Group (OMG): UML Profile for Enterprise Distributed Object Computing (EDOC), OMG Document ptc/02-02-05, 2002.
[36] Object Mangagement Group: Model Driven Architecture (MDA) Guide, v1.0.1, OMG, omg/03-06-01.
[37] Object Management Group: Meta Object Facility (MOF) specification. OMG Document formal/2002-04-03.
[38] Object Management Group: Ontology Definition Metamodel (ODM). Sixth revised submission to OMG/ RFP ad/2003-03-40, document/05-08-01.
[39] Object Management Group: Software Process engineering metamodel v1.1 (SPEM), formal/05-01-06, January 2005.
[40] Object Management Group: Unified Modelling Language (UML) 2.0 Superstructure Specification, August 2003. Ptc/03-08-02, pp. 455–510.
[41] Object Management Group (OMG): UML Profile for ECA v1.0. formal/04-02-05, February 2004.
[42] M.K. Smith, C. Welty, and D.L. McGuinness, (Eds.), OWL Web Ontology Language Guide, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/. Latest version available at http://www.w3.org/TR/owl-guide/.
[43] K. Pohl, The three dimensions of requirements engineering, in: Proc. 5th Int. Conf. of Advanced Information Systems Engineering 1993, Paris, Springer, Berlin, 1993, pp. 275–292.
[44] G. Klyne, J. Carroll (Eds.), Resource Description Framework (RDF): Concepts and abstract syntax, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-concepts/.
[45] M. Shaw, D. Garlan, Formulations and formalisms in software architecture, in: J. van Leeuwen (Ed.), Computer Science Today: Recent Trends and Developments, in: Lecture Notes in Computer Science, vol. 1000, Springer-Verlag, Berlin, 1995.
[46] Sun Microsystems: Jini Network Technology. http://www.sun.com/software/jini/index.html.
[47] C.R. Turner, A. Fuggetta, L. Lavazza, A.L. Wolf, A conceptual basis for feature engineering, The Journal of Systems and Software 49 (1999).
[48] D.C. Fallside, (Ed.), XML Schema Part 0: Primer, W3C, 2 May 2001. See http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/primer.html.