Journal of Universal Computer Science, vol. 18, no. 19 (2012), 2669-2685
submitted: 30/3/12, accepted: 6/9/12, appeared: 1/11/12 © J.UCS

Metamodeling the Structure and Interaction Behavior of Cooperative Component-based User Interfaces

Luis Iribarne, Nicolás Padilla, Javier Criado
(Applied Computing Group, University of Almeria, Spain
{luis.iribarne, npadilla, javi.criado}@ual.es)

Cristina Vicente-Chicote
(Department of Information Technology and Communications
Technical University of Cartagena, Spain
cristina.vicente@upct.es)

Abstract: In *Web-based Cooperative Information Systems* (WCIS), user groups with different roles cooperate through specialized interfaces. Cooperative interaction and user interface structures are usually rather complicated, and modeling has an important part in them. *Model-Driven Engineering* (MDE) is a software engineering discipline which assists engineers in abstracting system implementations by means of models and metamodels. This article describes an interactive, structural metamodel for user interfaces based on component architectures as a way to abstract, model, simplify and facilitate implementation. The paper also presents a case study based on an *Environmental Management Information Systems* (EMIS), where three actors (a politician, a GIS expert, and a technician) cooperate in assessing natural disasters.

Key Words: MDE, component-based development, user interfaces, user interaction, cooperative systems
Category: D.2, H.4, H.5

1 Introduction

In a more open and changing world, where information globalization and the knowledge society are spread on the Internet, modern *Web-based Cooperative Information Systems* (WCIS) must be flexible and easily adaptable and extendable. They must also be accessible and manipulable at runtime by different people or groups of people with common interests located in different places. There has recently been special interest in information globalization by providing systems with a common vocabulary through ontologies and Web semantics. Much attention has also focused on standardizing the way in which information is retrieved from the Web using powerful search engines based on ontologies and intelligent software agents. Nevertheless, WCIS user interfaces (as well as the knowledge they manage) are still being built based on traditional software development paradigms without taking into account the main criteria of globalization: they have to be distributed, open and changeable.

In this scenario, our research interest lies in providing a solution for cooperative user interfaces that operate in Web-based collaborative information systems.

There are many different reasons. Firstly, Web-based information systems are the most widespread and commonly used systems in distributed **social interaction** (for instance, social networks). Secondly, they allow non-compiled Web user interfaces, easily interchangeable at runtime. Thirdly, and particularly, our methodological proposal gives a *Component-Based Development* (CBD) solution to cooperative component-based gadget/widget-type user interfaces. iGoogle[1] *gadgets* are a good example of interface-components, and *Environmental Management Information Systems* (EMIS) [El-Gayar and Fritz, 2006] [Iribarne, 2010] are a good example of social interaction. For instance, a wide range of final users and actors (such as politicians, technicians or administrators) cooperate with each other and interact with the system for decision-making, problem-solving, etc. In this kind of system, user groups (who often have different roles) cooperate through distributed user interfaces, where interaction between different elements involved in the system (e.g., actors, roles, tasks, interaction rules, etc.) is usually highly complex. Due to the variety of *social interaction*, interfaces must adapt to the needs of users and/or groups of users who cooperate. Cooperative user interfaces must be able to be dynamically regenerate at runtime depending on the type (individual or collective) and the purpose (management, technical purpose, etc.) of interaction.

Furthermore, our methodology pursues evolutionary user interfaces: changeable and adaptable to user needs at runtime[2]. Such evolution is caused by cooperative interaction between users (and/or groups) and the user interface (UI). As a solution to this approach (i.e., cooperative, evolutionary Web component-based user interfaces), our proposal is inspired by *Model-Driven Engineering* (MDE) principles [Schmidt, 2006], especially runtime models, model evolution and model transformation. It uses models and metamodels to abstract the dynamic behavior of user interfaces and user interaction. *Interaction* is one of the metamodels used by the methodology, where the elements of the cooperative user interface are defined at a high level (i.e., mainly groups, actors, roles, choreographies, tasks and interface-components).

But this methodology is appropriate only for certain types of user-interfaces: (a) **Component-based interfaces**. We consider the UI a collection of interface-components with dependencies (functional, interaction, visual or temporal dependences, among others). An example of component interface is the iGoogle interface, made up of interface portions (or "gadgets") that together form the UI; (b) **COTS (***commercial off-the-shelf***) UI components**: commercial UI components developed by third-parties, available in public repositories and accessible by *traders* [ISO, 2004] [Iribarne *et al.*, 2004] for UI architecture configuration. Here, the UI is considered component architecture; (c) Interfaces should be **self-**

[1] `http://code.google.com/apis/gadgets/`
[2] `http://www.ual.es/acg/soleres/jism`

reconfigurable. The UI should be able to adapt itself to the user. Our aim is therefore not to work with complex UI or interface-components. We use only WIMP interfaces, simple UI made up of graphical elements such as *Windows, Icons, Menus and Pointers* (WIMP) [Almendros and Iribarne, 2008]; and finally, (d) our methodology is suitable for **WIS interfaces**, i.e., Web-based information system interfaces. WIS user interfaces do not need to be compiled environments, which justifies even more specifically the suitability of this solution to these (Web) interfaces.

The rest of the article is structured as follows. Next section introduces background research work used by our approach. Section 3 describes an example as a guideline used throughout the article. Section 4 describes the interaction and structural metamodels, and an example of their use. Section 5 describes some related studies, and finally, some conclusions and future work are presented.

2 Background

The work presented in this paper is based on previous research work for automatic composition of user interfaces [Iribarne *et al.*, 2010]. In this sense, the proposed methodology is based on an model-driven engineering (MDE) approach to **model evolution** [Mens, 2008] by considering the interface architectural models able to evolve at **runtime** [Blair *et al.*, 2009]. We do this in two stages [Criado *et al.*, 2010]: (a) *model transformation* and (b) *regeneration* (by trading). The starting user interface is treated like a set of models. A model is an instance of a metamodel, which sets the rules and elements describing the system. Our system is built on the basis of two metamodels (Figure 1): the architectural metamodel (AMM), and the runtime component metamodel ($RTCMM$). The former defines the component architecture by describing component structure and behavior, while the latter models the user interface element composition and their concrete component references.

The architectural metamodel is divided into three subsets: the structural metamodel (SM), the visual metamodel (VM) and the interaction metamodel (IM). The former metamodel describes composition dependencies between components through connection ports (*i.e.*, provided and required interfaces). The visual metamodel models visual component behavior (open, close, show, hide components, etc.) by means of a state machine. The interaction metamodel models user-interaction behavior and describes the structure of interaction tasks that users may execute in the system (roles, tasks associated with those roles, choreography, etc.). The architectural model is used as input for the transformation process, and the transformer implements evolution. As input, it uses a set of rules that define transformer behavior, and the current architectural model (AM_i), including the interaction model (IM_i). As output, the transformer creates a new architectural model (AM_j) with its corresponding interaction models.

The transformation process operates when certain events occur in the system. Such events report on changes that have been made (for instance, interaction between user interfaces, time interval fulfillment, interaction with a component, etc.), affecting the component architecture.

The runtime component models are regenerated from the new architectural models obtained as previously stated. In this vein, a trading service (*trader*) [ISO, 2004][Iribarne *et al.*, 2004] calculates the best configuration for satisfying the architectural requirements, starting from abstract component requirements and a set of concrete components in repositories linked to the trader. The result is a runtime component model ($RTCM$) that will be processed to show the final user interface. The regeneration process and the runtime component models are out of the scope of this paper.

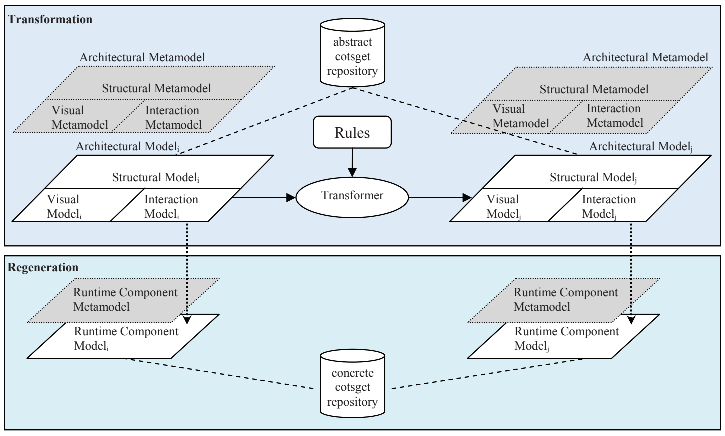

Figure 1: A model transformation for user-interface evolution

This article only explains the interaction (IM) and structural (SM) metamodels in the model evolution methodology (interface), using a case study as an example of interaction based on an *Environmental Management Information System* (EMIS), in which three actors (a politician, a GIS expert, and a technician) cooperate in assessing natural disasters. The interaction metamodels include groups of actors and their interaction choreographies (protocol) as defined by activities and tasks. In the structural metamodel, the user interface is defined by components, ports and connectors to describe user interface component architectures. More information about how the model evolution and transformation are interrelated with their structure and interaction model, and how the run-time component model (RTCM) is generated can be found at [Criado *et al.*, 2010] and [Iribarne *et al.*, 2010].

3 A simple running example

In this section we examine a simple example which we employ as a guide throughout the article to assist in explaining the behavior of the interaction and structural metamodels. User actions in a cooperative task are identified. The case study is related to a typical cooperative decision-making task in an EMIS for assessing natural disasters. This cooperative task assesses damage caused by a catastrophe in a particular land area (for instance, a wooded area). Three users with three different roles take part in this task, a politician, a GIS expert, and an evaluator.

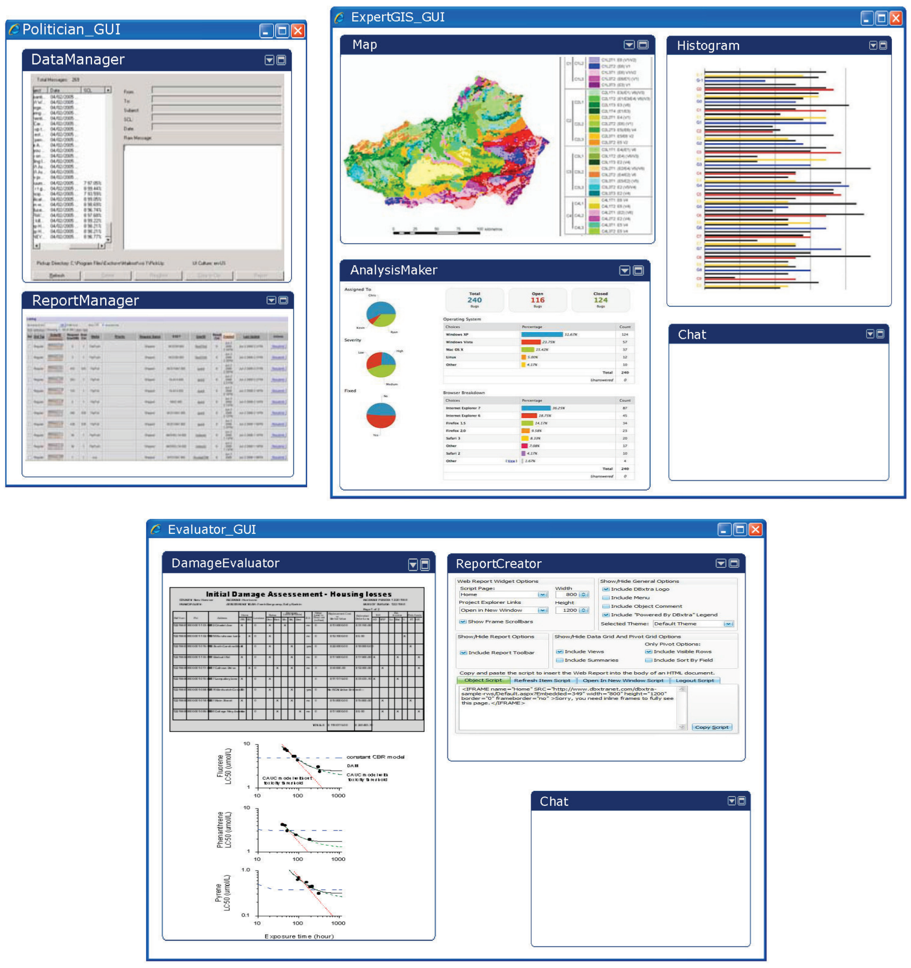

Figure 2: User interfaces handled by actors

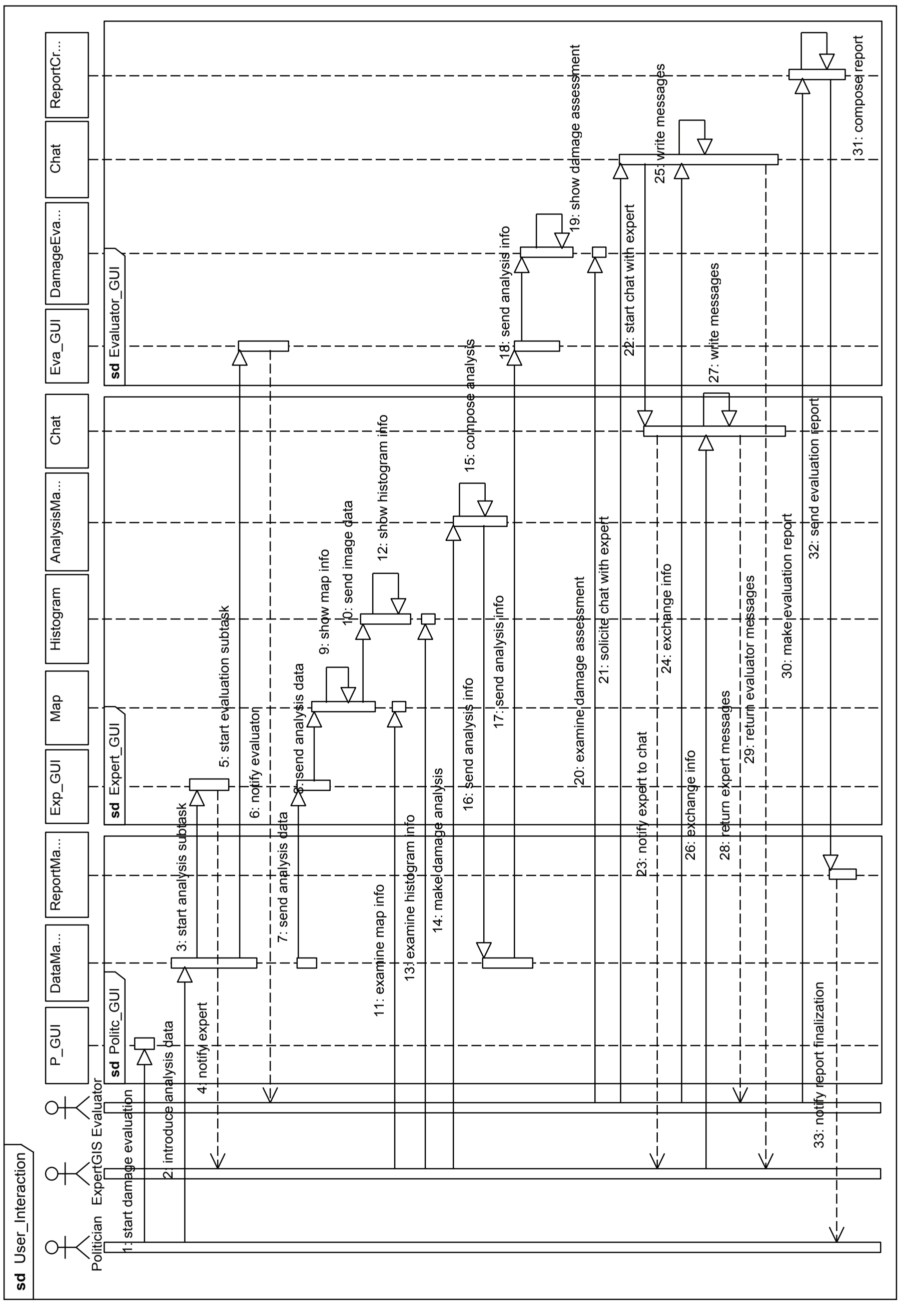

Figure 3: User interaction sequence diagram (user-interaction view)

An example of the three user interfaces handled by each of the three actors (politician, expert and evaluator, in this order) is shown in Figure 2. The politician's interface (on the left) has two components, an information manager (*DataManager*) and a report generator (*ReportManager*). The Expert's interface has four interface components, a geographic map viewer, a histogram viewer, an analysis generator and online chat (used to communicate with the evaluator for decision-making). The Evaluator's interface has three more interface components: a report generator, a damage evaluator and online chat. A flow diagram of the interaction between users and between them and the user interface components is shown in the Figure 3.

Briefly, the interaction sequence takes place in four steps. Firstly, there is a politician (`PoliticianRole`) who is interested in making a damage assessment and, therefore, he/she is the only user who can initiate the cooperative task. To make the impact study, the politician needs two assessments, one by a GIS expert who makes a technical evaluation, and the other an economic assessment based on the technical evaluation. Secondly, there is a GIS technician (`ExpertGISRole`) who is in charge of analyzing the affected areas in order to classify the types of soil, damaged infrastructure, the size of each area affected and so on. Thirdly, there is an administrator (`EvaluatorRole`) who makes an economic estimate of the affected soil, damaged infrastructure, etc. based on the information provided by the expert. Finally, the politician makes his final report based on the information prepared and coordinated in the cooperative process described above.

4 The interaction and structural metamodels

This section describes the cooperative user interface interaction and structural metamodels. Figure 4 shows the proposal architectural metamodel (see Figure 1 again) with its three parts: (a) Interaction metamodel section (IM), (b) Structural metamodel section (SM), and (c) Visual metamodel section (VM). The interaction models groups of actors and their interaction protocols through activities and tasks. The user interface structure is modeled using components, ports and connectors. The visual metamodel is described by a state machine, which is not explained here as it is out of the scope of this article. The interaction and structural metamodel sections are described separately below.

4.1 Interaction metamodel

The interaction metamodel conceptually describes the structure of the cooperative system, based on roles (`Role`) and groups (`Group`) of actors. The actors interact with user interface components by means of the structural metamodel `Component` concept (defined below).

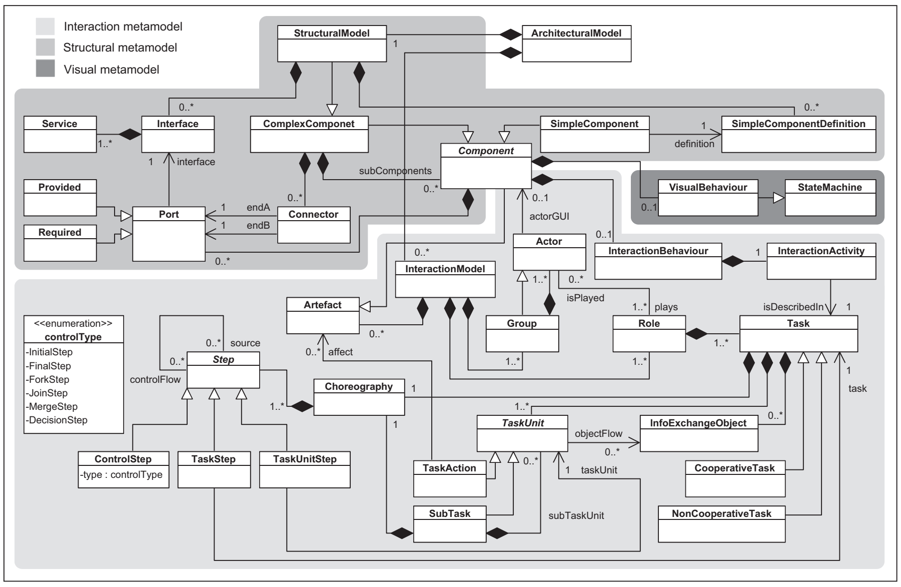

Figure 4: The architectural metamodel

Each actor (`Actor`) has at least one associated role. Each role is made up of a set of *tasks* which make it possible to identify the *activities* carried out by actors who have the same system role. Any task can be interrupted by another one at a specific time. There are two types of tasks, `CooperativeTask` and `NonCooperativeTask`. Both are modeled similarly, considering that cooperative tasks have some conceptual and implementation restrictions, such as that at least two actors may take part (with the same or different role). Each task in turn, is made up of task units (`TaskUnit`). A task unit can be a subtask or an action. A `SubTask` is a set of task units (actions or new subtasks). The `Action` is an atomic task, so, it cannot be decomposed into different actions. All these actions are related to the actors who use such actions and to the artifacts which the actors interact with. The artifacts used in our system are the interface components.

Each task and subtask always has a choreography associated. The choreography models the steps necessary for task or subtask execution. There are three different steps. Firstly, `TaskUnitStep` is used to model invocation of a task unit, and consequently, it relates subtasks or actions within the same task or in different tasks. Secondly, `TaskStep` is used to model invocation of a new task, and lastly, `ControlStep` is used to add control flow capacities, of which there are several. On one hand, `DecisionStep` is used to implement a selection of steps among a group of possible steps. `MergeStep` joins control flows (equivalent to OR in logic), `ForkStep` creates several concurrent control flows starting from only one control flow, and `JoinStep` joins the control flows that are dependent on each other (equivalent to AND in logic). Finally, `InitialStep` and `FinalStep` delimit the sequence of steps to be followed in the choreography. Both `TaskUnitStep` and `TaskStep` can use the `InformationExchangeObject` concept. This object contains the information exchanged between activities.

We have also set up OCL constraints to improve interaction model construction. The main constraints specifically refer to the definition of choreography steps and their relationships through `ControlFlow` and `Source` roles from the reflexive association of the `Step` concept.

Rule #1 (*Context Step*): The following restriction operates on the "Step" concepts and means that one step cannot be connected to another.

```
inv: self.controlFlow->forAll(c | c.id <> self.id)                    (1)
```

Rule #2 (*Context ControlStep*): This other restriction operates on "ControlStep" concepts and means that a first step can only be connected to `ForkStep`, `TaskStep` or `TaskUnitStep`:

```
inv: self.type = controlType::InitialStep implies(                   (2)
    self.controlFlow->forAll(c | c.oclAsType(ControlStep).type =
       controlType::ForkStep or c.oclIsTypeOf(TaskStep) or
       c.oclIsTypeOf(TaskUnitStep) ) )
```

Rule #3 (*Context ControStep*): Finally, the following restriction also functions on concepts of the "ControlStep" type and means that a `ForkStep` has an input connection and two or more outputs.

```
inv: self.type = controlType::ForkStep implies(                      (3)
    (self.source->size()=1) and (self.controlFlow->size() >=2) )
```

Figure 5 shows a diagram describing the relationships between users as well as their activities for cooperative task execution. The figure shows an instance (or model) of the interaction metamodel as a guiding example. To draw the model we have used a representation adapted from the UML activities diagram. To help read it, each symbol has been stereotyped with the corresponding concept (element) in the metamodel. The activity starts as soon as the politician starts the `DamageEvaluationTask` task. As shown above, all tasks and subtasks have a choreography, begin with the `InitialStep` control-flow, and finish with `FinalStep`.

The `DamageEvaluationTask` choreography includes three steps (not counting `InitialStep` and `FinalStep`). On one hand, a `TaskUnitStep` enables the politician to introduce the basic data necessary for assessment, providing some information about the study area, infrastructure, etc. Next is a `ForkStep`. This

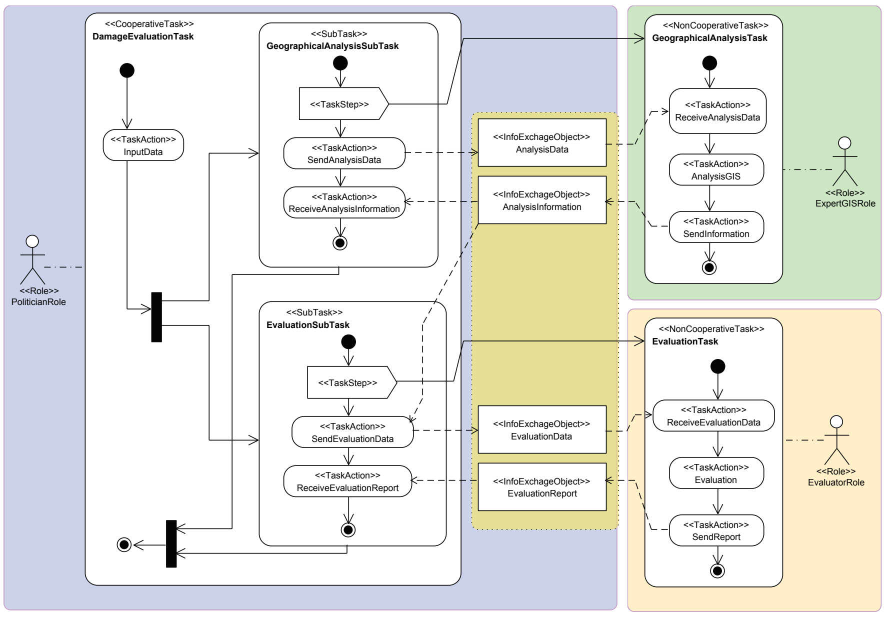

Figure 5: An instance scenario of the interaction metamodel

step allows two subtasks to be initiated, the `GeographicalAnalysisSubtask`, which identifies appropriate actions for geographical analysis of the study area, and the `EvaluationSubtask`, which identifies actions for damage assessment. By initiating both subtasks, the two users affected are kept informed in a cooperative task. Both have their own choreography as described below.

Finally, `JoinStep` synchronizes the two subtasks initiated in the previous step, and until both subtasks are executed, the following step in the choreography, which finishes with the cooperative task in our example, cannot be carried out. The choreography of the `GeographicalAnalysisSubtask` has three steps (again not counting `InitialStep` and `FinalStep`). The first is execution of the `GeographicalAnalysisTask`, which is done by a user in the `GISExpert` role. This user has COTS gets components (not described here) that allow him to manipulate or visualize maps to carry out his activity. The second step enables the expert to send data (`AnalysisData`) so he can begin his activity. Finally, the subtask choreography ends by executing `ReceiveAnalysisInformation`, allowing the user to receive the analysis made by the GIS expert. As described below, this information is necessary for the `EvaluationSubtask`.

The choreography of the `EvaluationSubtask` also has three steps. The first one is used for the `EvaluationTask`, which pertains to the `EvaluatorRole`. The second one sends the information necessary to make the appropriate assessment.

The third receives the analysis made by the expert. The subtask finishes when this action has been carried out. Finally, the model shows the specific choreographies of the non-cooperative `GeographicalAnalysisTask` and `EvaluationTask` tasks. Without going into further detail, the first task is carried out by the `ExpertGISRole` user to make the geographical analysis of the area affected by the catastrophe. The second one is carried out by an `EvaluatorRole` user to make a damage assessment from the information provided by the GIS expert.

4.2 Structural metamodel

The structural metamodel which describes the elements of a user interface component architecture following the "*component/port/connector*" model is shown in Figure 4. The main concept of the metamodel is the `Component` element, which can be simple (`SimpleComponent`) or complex (`ComplexComponent`). The functionality of a component is defined by its ports (`Port`), which are interfaces from which the services are described. There are two types of ports, `Provided` and `Required`. The first define the services the component provides (offers), and the second the services which the component requires to be able to function (together). The dependencies between components are set by the `Connector` concept. A connector joins two components by two dual ports (provided/required). A complex component (`ComplexComponent`) may be comprised of two or more components (which in turn may be complex or simple). Complex components are treated the same way as a simple component (with ports and connectors). Furthermore, a component may have a defined interaction behavior in the interaction model associated with it through the `InteractionBehaviour` concept.

Figure 6 shows an instance of the structural metamodel for the guide example. This figure shows the three user interfaces for the three actors in the example (seen above in Figure 2), one for the politician (at the top), another for the GIS expert (on the left) and another for the evaluator (on the right). The diagram shows the internal dependencies between components in each user interface and also between interface components. A UML component diagram was used to draw the model of the instance in the figure, using a graphical notation for the "capsule/port/connector" model. The components are drawn with a box or capsule and labeled with the stereotype `<<component>>` which requires the metamodel `Component` concept. At this level, only the name of the component is shown inside a capsule (the box). For example, `/datM:DataManager` means that `datM` is an instance of the base component `DataManager`. A **port** mediates the interaction of the capsule with the outside world. Ports carry out protocols, which indicate the order in which the messages are sent between connected ports. For instance, the `+datMProv:MapData` notation is a port called `datMProv` and a protocol called `MapData`. The `MapData~` notation represents the dual protocol. Ports are provided with a mechanism for a capsule to specify the input

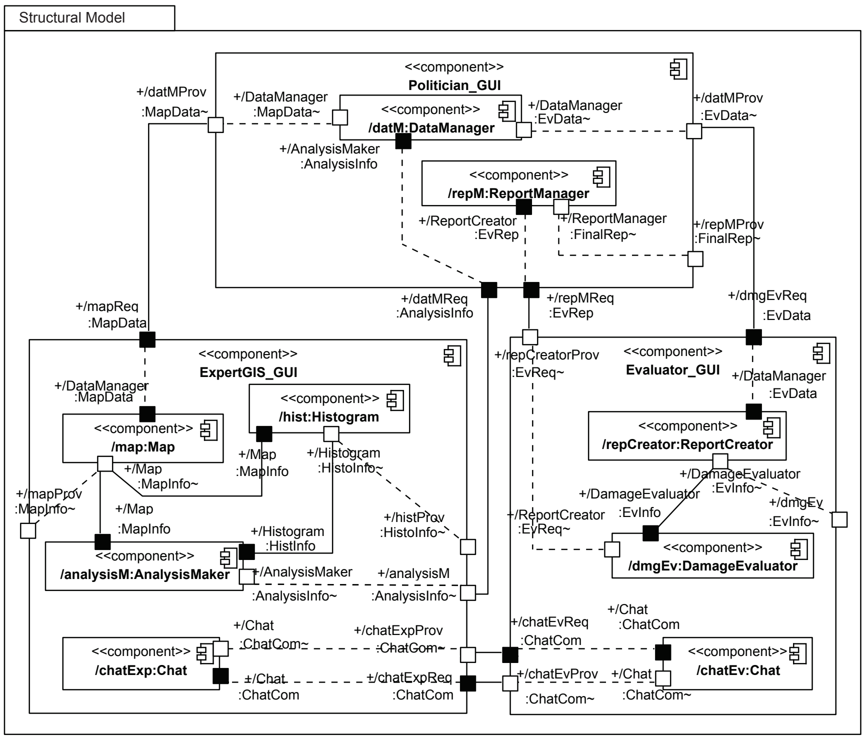

Figure 6: An instance of the structural metamodel with three user interfaces

and output interfaces. In our approach, the output interface notation (the small white box) is used to describe the component interfaces provided, and we use the input interface notation (the small black box) to describe the component interfaces required. Finally, **connectors** capture the key communication relationships between capsules. They are represented by means of lines that join two dual ports (provided/required, white/black).

For component composition, the ports provided (white-box) by the resulting container "parent" component would be all those that internally provide their contained components, and the required ports (black-box) would be those required that are not internally covered by those provided. For example, the *Evaluator* interface is a complex component made up of another three components. As may be observed, the services provided by parent components (the interface) are the same ones that are provided by the child components (with a different name). In such cases, the connector between the ports is the "use" type (shown in the diagram by a dashed line) denoting that one output port "*uses the behavior of another internal one.*" Something similar occurs in the case of ports required by a complex component. Note that the required port, `+DamageEvaluator:Evinfo` of the `/dmgEv:DamageEvaluator` component is already covered by the port offered by the `/repCreator:ReportCreator` component through its port providing `+DamageEvaluator:EvInfo`.

As in the interaction metamodel, it has also been necessary to include OCL rules for some of the structural metamodel elements to improve the semantics not established by the metamodel itself. Thus, for example, it was necessary to include a rule to preset the types of connectors between ports: dual connectors (provided/required) and use connectors (*provided/provided* or *required/required*).

Rule #4 (*Context Connector*): The ends that connect a *Connector* are different types (*Provided-Required* or *Required-Provided*) when they connect components from the same parent, and they are the same type (*Provided-Provided* or *Required-Required*) when they connect parent components with children and vice versa.

```
inv: self.endA.oclIsTypeOf(Provided) and                            (4)
   self.endB.oclIsTypeOf(Provided) implies (
      (self.endA.parent.parent.parent = self.endB.parent.parent)
      or (self.endA.parent.parent = self.endB.parent.parent.parent) )

inv: self.endA.oclIsTypeOf(Required) and
   self.endB.oclIsTypeOf(Required) implies (
      (self.endA.parent.parent.parent = self.endB.parent.parent)
      or (self.endA.parent.parent = self.endB.parent.parent.parent) )

inv: self.endA.oclIsTypeOf(Provided) and
   self.endB.oclIsTypeOf(Required) implies (
      self.endA.parent.parent = self.endB.parent.parent )

inv: self.endA.oclIsTypeOf(Required) and
   self.endB.oclIsTypeOf(Provided) implies (
      self.endA.parent.parent = self.endB.parent.parent )
```

Rule #5 (*Context Connector*): The interfaces pertaining to ports associated by a connector have to be the same type.

```
inv: self.endA.port_interface = self.endB.port_interface            (5)
```

Rule #6 (*Context Component*): The root element "StructuralModel" is the only component that can be defined without any port; the rest have to have at least a "Provided" port.

```
inv: self.oclIsTypeOf(StructuralModel) implies                        (6)
   self.ports->size()>=0

inv: not(self.oclIsTypeOf(StructuralModel)) implies (
   (self.ports->size()>=1)
   and (self.ports->exists(p | p.oclIsTypeOf(Provided))) )
```

5 Related Work

In Cooperative Information Systems (CIS), **models** play an important role, especially in the user-interface (UI) field. In this type of system, where groups of users (with different roles) cooperate through distributed UI, and interaction between different elements involved in the system (e.g., actors, roles, tasks, interaction rules, etc.) is usually highly complex, *Collaborative Software Engineering* (CSE) [Mistrik *et al.*, 2010] and *Model-Driven Engineering* (MDE) could represent a solution for modeling UI [Obrenovic and Starcevic, 2005] and cooperative interaction [Bourguin *et al.*, 2001]. There are many model-based proposals for modeling UI in the literature (e.g., IDEAS, OVID, WISDOM, UMLi, etc.); see [Pérez-Medina *et al.*, 2010] for a survey. The use of models to represent UI assists designers in their construction, conceptualization and visualization [Clerck *et al.*, 2005]. Some references use an MDE perspective for Web-based UI, as in [Chavarriaga and Macia, 2009] and [Angelaccio *et al.*, 2009], although they do not consider cooperative interaction models. Other proposals, such as in [Guerrero *et al.*, 2008] present a metamodel for designing the various UI in a workflow information system which integrates some different interaction elements, such as process, task, domain, job, among others. However, that proposal does not define an interaction metamodel for cooperative Web UI either.

On the other hand, there are model-based approaches for user interface development depending on interaction models. For example, in [Engel 2010] the author describes a task model-based framework for the automatic user interface creation and code generation. In [Bodgan *et al.*, 2008] authors present a interaction metamodel based on human communication called discourse metamodel. Furthermore, the discourse models are used to facilitate the automatic user interface generation and their associated behaviour. These proposals are different from ours because we focus on task interaction between actors and components with the goal of modeling this information about the cooperative system, which is used as part of the adapting and evolutionary methodology.

Component-Based Software Development (CBSD) approaches for the design and implementation of GUIs are also increasing quickly. Some articles present a combined MDE and CBSD approach for modeling both structure and behavior of component-based software architectures [Alonso *et al.*, 2008], which is what we have done for the development of our GUI architectural models. MDE also plays an important role in collaborative systems. In [Gallardo *et al.*, 2008] the authors propose an awareness metamodel that conceptualizes collaborative systems for modeling activities. The proposal distinguishes five metamodel views: (a) work group view, (b) actions view, (c) workspace view, (d) domain view, and (e) awareness mechanisms view. Cooperation among users takes place in the "workspace" view, which represents the user interface. [Hawryszkiewycz, 2005] proposes a collaborative metamodel for defining collaborative work practices. Nevertheless, none of the aforesaid proposals considers an interaction model for cooperative interfaces or choreographies among groups of users. In our case, we model them through state machines defined in the metamodel itself.

6 Conclusions and future work

Globalization of information and the knowledge society involve the use of varied (and sometimes complicated) social interaction which requires more collaborative Information Systems. *Environmental Management Information Systems* (EMIS) [El-Gayar and Fritz, 2006] [Iribarne, 2010] are a good example of social interaction, in which a wide range of final users and actors (such as politicians, technicians and administrators) cooperate with each other and interact with the system for decision-making, problem-solving, etc. In this type of system, groups of users (who often have different roles) cooperate through distributed user interfaces, where interaction between different elements involved in the system (e.g., actors, roles, tasks, interaction rules, etc.) is usually highly complex. Due to the variety of social interaction, interfaces must adapt to the needs of users and/or groups of users who cooperate. Cooperative user interfaces must be able to dynamically regenerate at runtime depending on the type of interaction (individual or collective) and the purpose of interaction (management, technical purpose, etc.). However, cooperative user interfaces (especially in *Web-based Cooperative Information Systems* (WCIS), such as some EMIS) (as well as the knowledge they manage) are still being built based on traditional software development paradigms without taking into account the main criteria of globalization: they have to be distributed, open and changeable. This implies that WCIS user-interfaces should be modeled according to the type of cooperative interaction, purpose (political, management, technical purpose, etc.) and structure.

In this article we present interaction and structural metamodels as part of an evolutionary model methodology for cooperative user interfaces. This proposal is inspired by basic principles of *Model-Driven Engineering* (MDE), particularly runtime models, model evolution and model transformation. The proposed interaction metamodel uses six basic concepts: groups, actors, rules, choreographies, tasks and components, and the structural metamodel is based on a component/port/connector" model. We also present an interaction scenario for decision-making in environmental impact assessment, common in GIS (*Geographical Information Systems*), to explain the main concepts of both metamodels and some instances obtained from them. The example scenario models the interaction of three users with three different roles (a politician, an expert and an evaluator) in a cooperative task. The interaction and structural metamodels, and the example described in this paper are a part of the SOLERES system, an *Environmental Management Information System* (EMIS) [Iribarne, 2010].

In our future work, we want to develop a graphical tool using the *Eclipse Graphical Modeling Framework* (GMF, `www.eclipse.org/gmf/`) for easy creation of new scenarios, such as instances (models) of the interaction metamodel. Our models are currently written directly in XMI and manually drawn as activity and object diagrams using Visual Paradigm for Eclipse. We are also interested in studying possible change detection and variability in the interaction and structural metamodels by means of automated co-evolution mechanisms and metamodel adaptation [Cicchetti *et al.*, 2008] [Wachsmuth, 2007]. We are presently working on development of a simulation tool for reproducing user interaction behavior with user interface components by generating random or programmed events to check the models and metamodels.

Acknowledgments

This work was funded by the EU (ERDF), the Spanish Ministry of Science and Innovation (MICINN) under the TIN2010-15588 and TRA2009-0309 Projects (`http://www.ual.es/acg`), the JUNTA de ANDALUCÍA (proyectos de excelencia) under grant the TIC-06114 project, and also by Ingenieros Alborada, (`http://www.ingenierosalborada.es`).

References

[Almendros and Iribarne, 2008] Almendros, J., Iribarne, L.: "An extension of UML for the modeling of WIMP user interfaces"; J. Vis. Lang.&Comp., 19(6):695–720, 2008.

[Alonso *et al.*, 2008] Alonso, D., Vicente-Chicote, C., Barais, O.: "V3Studio: A component-based architecture modeling language"; In 15th IEEE Int. Conf. and Work. on the Eng. of Comp. Based Systems, pages 346–355. IEEE, 2008.

[Angelaccio *et al.*, 2009] Angelaccio, M., Krek, A., D'Ambrogio A.: "A model-driven approach for designing adaptive Web GIS interfaces"; LNGC, pp 137–148, 2009.

[Blair *et al.*, 2009] Blair, G., Bencomo, N., France, R.B (eds.): "Models@Run.Time"; Special Issue, Computer, IEEE Computer Society, 2009.

[Bodgan *et al.*, 2008] Bogdan, C., Falb, J., Kaindl, H., Kavaldjian, S., Popp, R., Horacek, H., Arnautovic, E., Szep, A.: "Generating an abstract user interface from a discourse model inspired by human communication"; In 41st Hawaii Int. Conf. on System Sci., pages 36–45. IEEE, 2008.

[Bourguin *et al.*, 2001] Bourguin, G., Derycke, J.C., Tarby, J.C.: "Beyond the interfaces, co-evolution inside interactive systems: A proposal founded on the activity theory"; In Proc. of the Human Computer Interaction 2001, Springer, 2001.

[Chavarriaga and Macia, 2009] Chavarriaga, E., Macia, J.A.: “A model-driven approach to building modern semantic Web-based user interfaces”; Advan. in Eng. Soft. 40, 1329–1334, 2009.

[Cicchetti *et al.*, 2008] Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: “Automating co-evolution in model-driven engineering”; EDOC, pp. 222-231, 2008.

[Clerck *et al.*, 2005] Clerck, T., Luyten, K., Coninx, K.: “DynaMo-AID: A design process and a runtime architecture for dynamic model-based user interface development”; Eng. Human Computer Inter. and Interactive Systems, pages 7795, 2005.

[Criado *et al.*, 2010] Criado, J., Vicente-Chicote, C., Padilla, N., Iribarne, L.: “A Model-driven approach to graphical user interface runtime adaptation”; Models@Run.Time, CEUR-WS Vol 641, pages 49-59, 2010.

[El-Gayar and Fritz, 2006] El-Gayar, O., Fritz, B.D.: “Environmental management information systems (EMIS) for sustainable development: A conceptual overview”; Comm. of the Assoc. for Inf. Syst. 17(1):34, 2006.

[Engel 2010] Engel, J.: “A model-and pattern-based approach for development of user interfaces of interactive systems”; In 2nd ACM Symp. on Eng. Interactive Comp. Systems, pages 337–340. ACM, 2010.

[Gallardo *et al.*, 2008] Gallardo, J., Crescencio, B., Redondo, M.A.: “Developing collaborative modeling systems following a model-driven engineering approach”; OTM 2008 Workshops, LNCS 5333, pp. 442–451, 2008.

[Guerrero *et al.*, 2008] Guerrero, J., Lemaigre, C., Gonzalez J.M., Vanderdonckt, J.: “Model-driven approach to design user interfaces for workflow information systems”; Journal of Universal Computer Science 14(19):3160–3173, 2008.

[Hawryszkiewycz, 2005] Hawryszkiewycz, I.T.: “A metamodel for modeling collaborative systems”; Jour. of Comp. Inf. Systems, 5(3):63–72, 2005.

[Iribarne *et al.*, 2004] Iribarne, L., Troya, J.M., Vallecillo, A.: “A trading service for COTS components”; The Computer Journal. 4, 3, pp. 342–357, 2004.

[Iribarne *et al.*, 2010] Iribarne, L., Padilla, N., Criado, J., Asensio, J.A., Ayala, R.: “A Model Transformation Approach for Automatic Composition of COTS User Interfaces in Web-Based Information Systems”; Information Systems Management. 27, 3, pp. 207–216, 2010.

[Iribarne, 2010] Iribarne, L.: “SOLERES project: A spatio-temporal information system for the enviromental management based on neural-networks, agents and software components”; TR, jspTIN2010; `http://www.ia.urjc.es/jspTIN2010/`.

[ISO, 2004] ISO: “Information Technology — Open Distributed Processing — Trading Function: Specification”; ISO/IEC 13235-1, ITU-T X.950.

[Mens, 2008] Mens, T.: “Introduction and roadmap: History and challenges of software evolution”; Software Evolution, pp. 1–11, Springer, 2008.

[Mistrik *et al.*, 2010] Mistrik, I., Grundy, J., Hoek, A., Whitehead, J.: “Collaborative software engineering”; Springer book, ISBN: 978-3-642-10293-6, 2010.

[Obrenovic and Starcevic, 2005] Obrenovic, Z., Starcevic, D.: “Model-driven development of user interfaces: Promises and challenges”; Eurocon (1-2):1259–1262, 2005.

[Pérez-Medina *et al.*, 2010] Pérez-Medina, J.L., Dupuy-Chessa, S., Front, A.: “A survey of model driven engineering tools for user interface design. Task models and diagrams for user interface design”; LNCS 4849, pp. 84–97, 2010.

[Schmidt, 2006] Schmidt, D.: “Model-driven engineering”; Comp. 39(2):25–31, 2006.

[Wachsmuth, 2007] Wachsmuth, G.: “Metamodel adaptation and model co-adaptation; ECOOP 2007, LNCS 4609, pp. 600–624, 2007.