An Interaction Meta-model for Cooperative Component-Based User Interfaces

Luis Iribarne[1], Nicolas Padilla[1], Javier Criado[1], and Cristina Vicente-Chicote[2]

[1] Applied Computing Group
University of Almeria, 04120 Almeria, Spain
{luis.iribarne,npadilla,javi.criado}@ual.es
[2] Department of Information Technology and Communications,
Technical University of Cartagena, Spain
cristina.vicente@upct.es

Abstract. *Model Driven Engineering* (MDE) aims to help software developers to abstract the system implementations by means of models and meta-models. In *Web-based Collaborative Information Systems* (WCIS) modelling plays an important role, especially in the user-interface field. In this kind of systems, where groups of users (with different roles) cooperate through distributed user interfaces, and the complexity of interaction between different elements involved in the system (e.g., actors, roles, tasks, interaction rules, etc.) is usually high, MDE could represent a good solution to model evolvable user interfaces. This paper describes a proposal for an interaction meta-model, as a part of a model-evolution methodology for cooperative *Graphical User Interfaces* (GUI) through *Component-Based Development* (CBD) approaches. 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 for assessing natural disasters.

Keywords: MDE, collaborative systems, user interfaces, interaction.

1 Introduction

Globalization of the information and of the knowledge society implies the use of a varied (and sometimes complicated) social interaction which requires more collaborative Information Systems. *Environmental Management Information Systems* (EMIS) [1] [2] are a good example of social interaction. 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, problems resolution, etc. In this kind of systems, groups of users (who often have different roles) cooperate through distributed user interfaces, where the complexity of interaction between different elements involved in the system (e.g., actors, roles, tasks, interaction rules, etc.) is usually high. Due to the variety of social interaction, interfaces must adapt themselves to the needs of users and/or groups of users who cooperate. Cooperative user interfaces must be able to be dynamically regenerated at runtime depending on the type of interaction (individual or collective) and the purpose of interaction (management, technical purpose, etc.).

In this scenario, our research interest lies in giving a solution to 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 used systems in distributed social interaction (for instance, social networks). Secondly, they allow us to have 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 user interfaces of gadgets/widgets-type. iGoogle[1] gadgets are a good example of interface-component.

Furthermore, our methodology pursues evolutive user interfaces: changeable and adaptable to the user needs at runtime. Such evolution is caused by the cooperative interaction between users (and/or groups) and the user interface (UI). As a solution to this approach (i.e., cooperative, evolutive Web component-based user interfaces), our proposal is inspired on principles of *Model-Driven Engineering* (MDE) [3], especially runtime models, model evolution and model transformation issues. Therefore, it uses models and metamodels to abstract the dynamic behaviour of user-interfaces and the interaction of users. *Interaction* is one of the metamodels used by methodology, where the elements of the cooperative user interface are defined at high level (i.e., mainly groups, actors, roles, choreographies, tasks and interface-components). This article is focussed on the definition of cooperative interaction meta-model for evolutive user interfaces.

The remainder of the article is organised as follows. Section 2 describes some related work. Section 3 defines model evolution for runtime interfaces. Section 4 shows a cooperative interaction metamodel. Section 5 describes a case study of cooperative interaction for assessing natural disasters as an instance of the metamodel. Finally, some conclusions and future work are presented.

2 Related Work

In **Collaborative** Information Systems (CIS) **models** play an important role, especially in the UI field. In this kind of systems, where groups of users (with different roles) cooperate through distributed **user interfaces**, and the complexity of interaction between different elements involved in the system (e.g., actors, roles, tasks, interaction rules, etc.) is usually high, Collaborative Software Engineering (CSE) [4] and **MDE** could represent a good solution to model (evolvable) user interfaces [5] and cooperative interaction [6].

In the literature there are many model-based proposals for modelling user-interfaces (e.g., IDEAS, OVID, WISDOM, UMLi, etc.); see [7] for a survey. Some of them use an MDE perspective for Web-based user interfaces, as in [10] and [11], though they do not consider cooperative interaction models. Other proposals as in [8] present a metamodel for designing the various user-interfaces

[1] `http://code.google.com/apis/gadgets/`

of a workflow information system, which integrates some different interaction elements, such as process, task, domain, job, among others. This proposal does not define an interaction metamodel for cooperative Web user interfaces either.

From the point of view of collaborative systems, MDE plays an important role too. In [9] authors propose an awareness meta-model that conceptualizes collaborative systems to carry out modeling activities. The proposal distinguishes between five meta-model views: (a) work group view, (b) actions view, (c) workspace view, (d) domain view, and (e) awareness mechanisms view. Cooperation between users is carried out through the "workspace" view, which represents the user interface. In [12] authors present a model-driven approach to construction of web-based collaborative environments. In [13] they propose a collaborative metamodel to define 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.

3 User-Interface Model-Evolution

As previously advanced, the interaction metamodel is a part of a methodology based on MDE model evolution to regenerate user interfaces at runtime. Some features of such methodology[2] will be explained here to provide the reader with a context, before describing the interaction meta-model in the following section.

This methodology is appropriate only for certain types of UI: (a) **Component-**based interfaces. We consider UI as a collection of interface-components with dependences (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 Of-the-Shelf***) UI components**: commercial UI components developed by third-parties, available in public repositories and accessible by *traders* [14] [15] for UI architecture configuration. Here, the UI is considered as a component architecture. We called components as "cotsgets" (COTS and gadgets/widgets); (c) Interfaces should be **self-reconfigurable**. The UI should be able to adapt itself to the user. For this purpose we do not aim to work with complex UI or interface-components ("cotsgets"). We just take into account WIMP interfaces, simple UI made up of graphical elements such as *Windows, Icons, Menus and Pointers* (WIMP) [16]; (d) Finally, our methodology is suitable for **WIS interfaces**, 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.

As a solution to the interface evolution process, our methodology is based on an MDE approach of model evolution [17] by considering the interface architecture as models capable of evolving at runtime [18]. We solve the model evolution in two phases: (a) *model transformation* and (b) regeneration (by means of *trading*). We consider a starting UI as a set of models. A model is an instance of a

[2] `http://www.ual.es/acg/soleres/jism`

meta-model, which establishes the rules and elements that describe our system through a model. Our system is built on the basis of two meta-models (Figure 1): the architecture metamodel (AMM), and the runtime component metamodel ($RTCMM$). AMM defines the component architecture by describing the structure and behaviour of components. This metamodel is divided into three subsets: the structural metamodel (SM), the visual metamodel (VM) and the interaction metamodel (IM). The first one models composition dependences between components through connection ports (i.e., provided and required interfaces). The visual metamodel models the components behaviour from a visual point of view (open, close, show, hide components, etc.) by means of a state machine. The interaction metamodel models the user-interaction behaviour and describes the structure of interaction tasks that users may fulfil in the system (roles, tasks associated with those roles, choreography, etc.). The architecture model is used as an input of transformation process. The transformer implements the evolution. As an input, it uses a set of rules that define the transformer behaviour, the current architecture model (AM_i), including the interaction model (IM_i).

As an output, the transformer creates a new architecture model (AM_j) with its corresponding interaction models. The transformation process is invoked ("operates") when certain events occur in the system. Such events inform us that some changes have been made (for instance, interaction between user interfaces, time interval fulfilment, etc.), affecting the component architecture. Therefore, our transformer executes a model-to-model transformation (M2M) of MISO type (*Multiple Input and Single Output*). Finally, a trading service (trader) [14][15] calculates the best configuration to satisfy the architecture requirements, starting from abstract component requirements and a set of concrete components in repositories linked to the trader; as a result, we get a runtime component model ($RTCM$) that will be shown to the user.

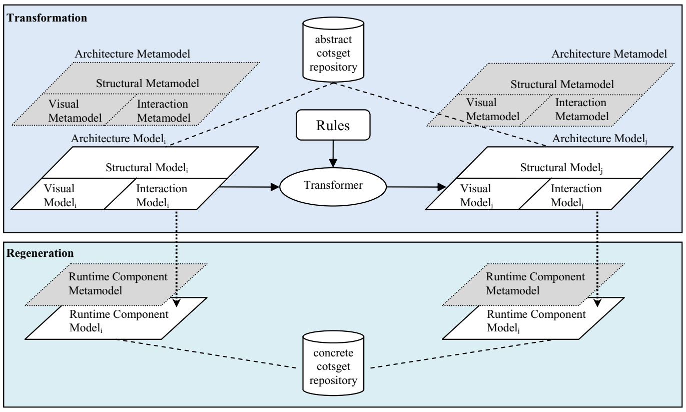

Fig. 1. A MISO model transformation for user interface evolution

4 Interaction Meta-model

Here we present the cooperative interaction meta-model. Figure 2 shows the architecture metamodel subset that describes the interaction (i.e., interaction meta-model). It conceptually describes the structure of the cooperative system, based on roles (`Role`) and groups (`Group`) of actors. Each actor has at least one role associated. Each role is made up of a set of *tasks* and thus we can identify the *activities* carried out by actors who belong to the same system role. Any task can be interrupted by another one at a specific time. We can distinguish between two types of tasks: `CooperativeTask` and `NonCooperativeTask`. Both are modelled similarly, taking into account that cooperative tasks have some conceptual and implementation restrictions such as the fact that at least two actors may take part (with the same or different role). In turn, each task 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 the atomic unit of a task, so, it cannot be decomposed into different actions. All these actions are related to the actors who use such actions and to the artefacts which the actors interact with. The artefacts used in our system are the "cotsgets" components.

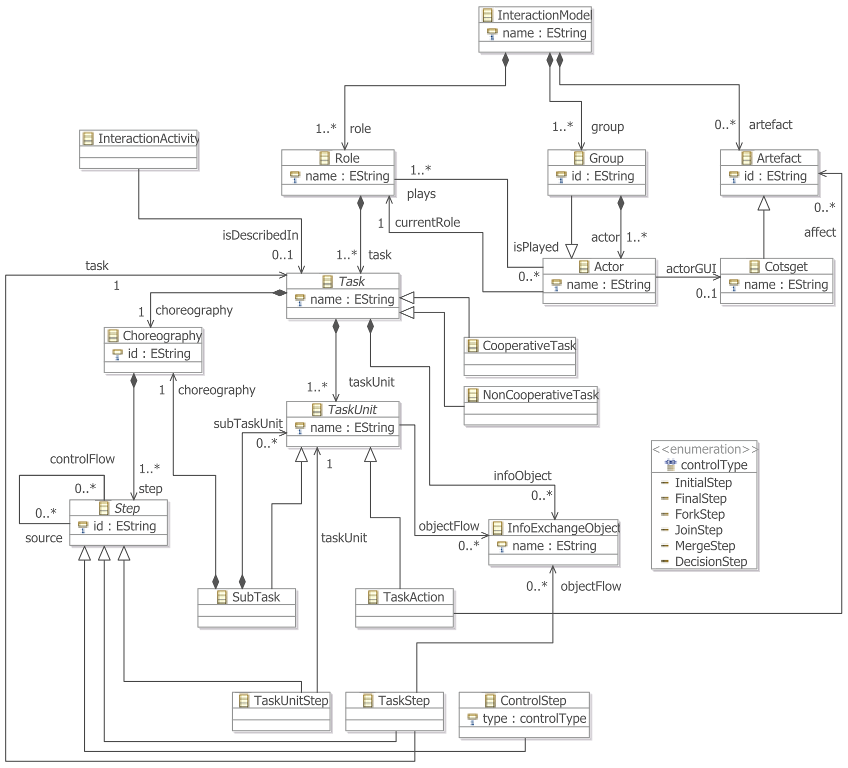

Fig. 2. The interaction meta-model

Each task and subtask always has a choreography associated. A choreography models the steps necessary for task or subtask execution. There are three different steps. Firstly, `TaskUnitStep` is used to model the invocation of a task unit, and consequently, it relates subtasks or actions within the same task or different tasks. Secondly, `TaskStep` is used to model the invocation of a new task. Lastly, `ControlStep` is used to add control flow capacities. There are different control capacities. On the one hand, `DecisionStep` is used to implement a selection of steps among a group of possible steps. On the other hand, `MergeStep` joins control flows (equivalent to a logic OR). `ForkStep` creates several concurrent control flows starting from only one control flow. `JoinStep` joins the control flows that are dependent on each other (equivalent to a logic AND). Finally, there are `InitialStep` and `FinalStep` that delimit the sequence of steps to be followed in a choreography. Both `TaskUnitStep` and `TaskStep` can use the `InformationExchangeObject` concept. This object contains the information exchanged from an activity to another one. Lastly, `InteractionActivity` (on the top left of the figure) and Cotsget (on the top right) represent joinpoints with the other two subsets of the architecture metamodel (*AM*) (i.e., Visual Metamodel and Structural Metamodel).

We've also established a set of OCL constraints in order to improve the interaction model construction. Table 1 shows a sublist of four constraints which specifically refer to the definition of choreography steps and their relationships through `ControlFlow` and `Source` roles of the reflexive association of the `Step` concept. Constraints of Table 1 mean the following. A step cannot be connected by itself (constraint #1). An initial step can only be connected to `ForkStep`, `TaskStep` or `TaskUnitStep` (constraint #2). A `ForkStep` has one incoming connection and two or more outgoing connections (constraint #3). A non cooperative task doesn't have any `TaskStep` either `TaskUnitStep` connected with a task or task unit from a different actor (constraint #4). It implies the task choreography and all subtask choreographies of the task.

Table 1. Some OCL restrictions on the interaction meta-model

Context	**OCL Expression**
Step	`inv: self.controlFlow->forAll(c | c.id <> self.id)`
Control Step	`inv: self.type = controlType::InitialStep implies( self.controlFlow->forAll(c | c.oclAsType(ControlStep).type = controlType::ForkStep or c.oclIsTypeOf(TaskStep) or c.oclIsTypeOf(TaskUnitStep)))`
Control Step	`inv: self.type = controlType::ForkStep implies( (self.source->size()=1) and (self.controlFlow->size() >=2) )`
Task	`inv: self.oclIsTypeOf(NonCooperativeTask) implies((self.choreography.step->forAll(s|s.oclIsTypeOf(TaskStep)). oclAsType(TaskStep).task.parent.isPlayed.name = self.parent.isPlayed.name) and (self.choreography.step->forAll(s|s.oclIsTypeOf(TaskUnitStep)). oclAsType(TaskUnitStep).taskUnit.parent.parent.isPlayed.name = self.parent.isPlayed.name) and (self.taskUnit->forAll(st|st.oclIsTypeOf(SubTask)).oclAsType(SubTask).choreography. step->forAll(s|s.oclIsTypeOf(TaskStep)).oclAsType(TaskStep).task.parent.isPlayed.name = self.parent.isPlayed.name) and (self.taskUnit -> forAll(st|st.oclIsTypeOf(SubTask)).oclAsType(SubTask).choreography. step->forAll(s|s.oclIsTypeOf(TaskUnitStep)).oclAsType(TaskUnitStep).taskUnit.parent. parent.isPlayed.name = self.parent.isPlayed.name))`

5 A Case Study

In this section we will examine a case study for a better understanding of the metamodel described above. We will identify the actions that users should carry out in a cooperative task. The case study is related to a typical cooperative task in EMIS for decision-making when assessing natural disasters. This cooperative task allows us to assess damages caused by a catastrophe in a particular area of land (for instance, a flooded area). Three users with three different roles take part in such a task. Firstly, there is a politician (`PoliticianRole`) who is interested in conducting a damage assessment and, therefore, he is the only user who can initiate the cooperative task as he is responsible for it. Secondly, there is a GIS technician (`ExpertGISRole`) who is in charge of analysing the affected areas in order to classify the types of soil, damaged infraestructure, extensions of each affected area and so on. Thirdly, there is an administrator (`EvaluatorRole`) who carries out an economic estimate of the affected soils, damaged infraestructure, etc. starting from the information provided by the GIS expert.

Figure 3 shows a diagram describing the relationships between users as well as their activities for cooperative task execution. The activity starts as soon as the politician starts the task `DamageEvaluationTask`. As indicated above, all tasks and subtasks have a choreography and begin with the control-flow `InitialStep` and finish with the `FinalStep`.

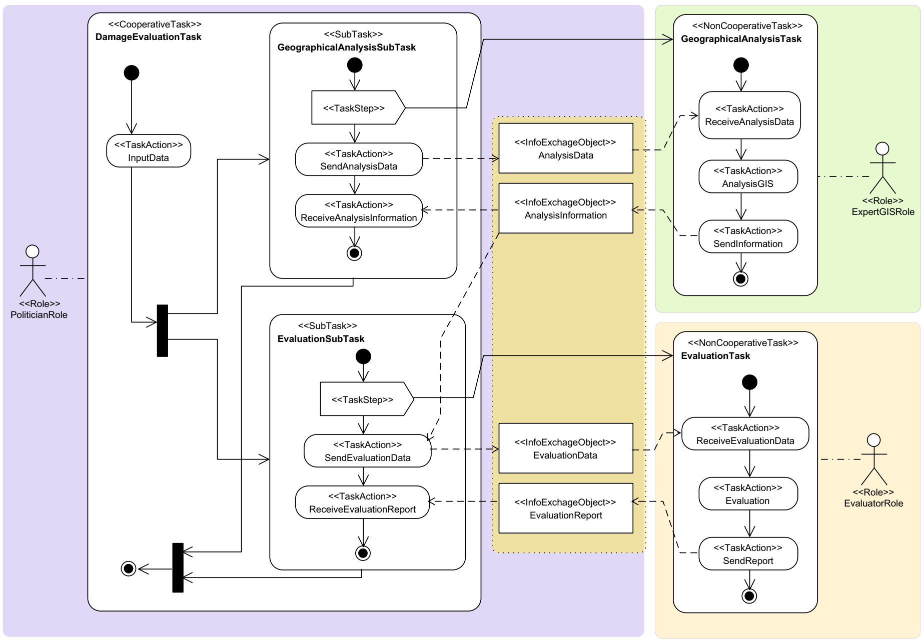

Fig. 3. A cooperative interaction scenario as an instance of the metamodel

The DamageEvaluationTask choreography includes three steps (regardless of InitialStep and FinalStep). On the one hand, there is a TaskUnitStep used so that the politician can carry out an action to introduce basic data necessary for assessment, providing some information about the study area, infraestructure, etc. Then, a ForkStep is carried out. This step allows us to initiate two subtasks: GeographicalAnalysisSubtask, which identifies the appropriate actions for the geographical analysis of the study area, and the EvaluationSubtask, which identifies actions for damage assessment. Initiating both subtasks makes the two affected users are informed in a cooperative task. Both have their own choreography which will be described next. Finally, there is the JoinStep, which allows to synchronize the two subtasks initiated in the previous step; as long as both subtasks are not executed, it is not possible to carry out the following step in the choreography, which will finish with the cooperative task in our example. The choreography of the GeographicalAnalysisSubtask carries out three steps (regardless of InitialStep and FinalStep again). The first is the execution of the GeographicalAnalysisTask, which will be fulfilled by a user of the GISExpert role. This user has cotgets components (not described here) that allow him to manipulate or visualize maps in order to carry out his activity. The second step enables to send data (AnalysisData) so that the expert can initiate his activity. Finally, the subtask choreography ends by executing the ReceiveAnalysisInformation action allowing the user to receive the analysis made by the GIS expert. As we will describe next, this information is necessary for the EvaluationSubtask.

The choreography of the EvaluationSubtask also provides three steps. The first one is used to fulfil the EvaluationTask that belongs to the EvaluatorRole. The second one sends the information necessary to make the appropriate assessment. The third one carries out the action of receiving the analysis made by the expert. Once this action is carried out, the subtask will finish. Finally, the previous Figure 3 shows the specific choreographies of the non-cooperative tasks GeographicalAnalysisTask and EvaluationTask. Without going into further detail, the first task is carried out by the user of the ExpertGISRole to make the geographical analysis of the area affected by the catastrophe. The second one is carried out by a user of the EvaluatorRole to conduct a damage assessment from the information provided by the GIS expert.

6 Conclusions and Future Work

In a more open and changing world, where information globalization and knowledge society are spreading in Internet, the modern Web-based cooperative information systems must be flexible and ready to be easily adaptable and extendable; they must also be accessible and manipulable at runtime by different people or groups of people with common interests and located at different places. There has recently been a special interest in the information globalization by providing the systems with a common vocabulary through ontologies and semantics for the Web. There has also been much attention focussed on the standardized way in which the information is retrieved in the Web using powerful search engines based on ontologies and intelligent software agents. Nevertheless, WCIS user interfaces are still being built based on traditional software development paradigms without taking into account (as well as the knowledge they manage) the main criterion of the globalization: they have to be distributed, open and changeable. This implies that UIs of a WCIS can self-generate themselves at runtime depending on the type of cooperative interaction and its purpose (political, management, technical purpose, etc.). Due to the inherent complexity of user interfaces (from a functional, visual and interactive point of view), our research is determined by the following premises: (a) User interfaces are simple and made up of WIMP-type elements (Windows, Icons, Menus and Pointers), (b) User interfaces are used for Web-based collaborative information systems, (c) User interfaces are formed on the basis of the composition of interface portions (widgets/gadgets-type). The interaction of user (groups) with the interface portions in their user interfaces makes them change at runtime. This viewpoint allows us to consider evolutive user interfaces that are somehow dependent on the interaction between the groups of users who are involved and collaborate in the system (WCIS).

In this article we present an interaction metamodel that is used as part of an evolutive model methodology for cooperative user interfaces. This proposal is inspired in basic principles of *Model-Driven Engineering* (MDE), particularly runtime models, model evolution and model transformation issues. The proposed interaction metamodel basically uses six concepts: groups, actors, rules, choreographies, tasks and cotsgets. *Cotsgets* are widget/gadget-type interface-components that together form the user interface. We also present an interaction scenario for decision-making in environmental impact assessment, usual in GIS (*Geographical Information Systems*). The example scenario models the interaction of a cooperative task between three users with three different rules (a politician, an expert and an evaluator). The interaction metamodel and the example described in this paper are a part of the SOLERES system, an *Environmental Management Information System* (EMIS) [2].

As future work we'd like to develop a graphical tool using the *Eclipse Graphical Modeling Framework* (GMF[3]) in order to easily draw new scenarios such as instances (models) of the interaction metamodel. Nowadays, models are directly written in XMI and manually drawn as activity and object diagrams by using Visual Paradigm for Eclipse. We are also interested in to study possible change detection in the interaction metamodel by means of automated co-evolution mechanisms and metamodel adaptation [19] [20].

Acknowledgment. This work has been supported by the EU (FEDER) and the Spanish Ministry MICINN under grant of the TIN2007-61497, TIN2010-15588, and TRA2009-0309 projects, `http://www.ual.es/acg/soleres`.

[3] `www.eclipse.org/gmf/`

References

1. 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)
2. SOLERES project: A spatio-temporal Information System for the Enviromental Management based on Neural-Networks, Agents and Software Components. University of Almeria, `http://www.ual.es/acg/soleres`
3. Schmidt, D.C.: Model-Driven Engineering. Computer 39(2), 25–31 (2006)
4. Mistrik, I., Grundy, J., Hoek, A., Whitehead, J.: Collaborative Software Engineering. Springer book, Heidelberg (2010) ISBN: 978-3-642-10293-6
5. Obrenovic, Z., Starcevic, D.: Model-driven development of user interfaces: Promises and challenges. In: EUROCON 2005, vol. 1(2), pp. 1259–1262 (2005)
6. 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, Berlin (2001)
7. Pérez-Medina, J.L., Dupuy-Chessa, S., Front, A.: A Survey of Model Driven Engineering Tools for User Interface Design. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 84–97. Springer, Heidelberg (2007)
8. 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)
9. Gallardo, J., Crescencio, B., Redondo, M.A.: Developing Collaborative Modeling Systems Following a Model-Driven Engineering Approach. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 442–451. Springer, Heidelberg (2008)
10. Chavarriaga, E., Macia, J.A.: A model-driven approach to building modern Semantic Web-Based User Interfaces. Advan. Eng. Soft. 40, 1329–1334 (2009)
11. Angelaccio, M., Krek, A., D'Ambrogio, A.: A Model-Driven Approach for Designing Adaptive Web GIS Interfaces. LNGC, pp. 137–148. Springer, Heidelberg (2009)
12. Levytskyy, A., Vangheluwe, H., Rothkrantz, L., Koppelaar, H.: MDE and customization of modeling and simulation web applications. Simulation Modelling Practice and Theory 17, 408–429 (2009)
13. Hawryszkiewycz, I.T.: A metamodel for modeling collaborative systems. Journal of Computer Information Systems 5(3), 63–72 (2005)
14. I.S.O,Information Technology — Open Distributed Processing — Trading Function: Specification. ISO/IEC 13235-1, ITU-T X.950
15. Iribarne, L., Troya, J.M., Vallecillo, A.: A Trading Service for COTS Components. The Computer Journal 4(3), 342–357 (2004)
16. Almendros, J., Iribarne, L.: An Extension of UML for the modeling of WIMP user interfaces. J. of Visual Lang. and Computing 19(6), 695–720 (2008)
17. Mens, T.: Introduction and Roadmap: History and Challenges of Software Evolution, pp. 1–11. Springer, Heidelberg (2008)
18. Blair, G., Bencomo, N., France, R.B. (eds.): Models@Run.Time. Special Issue, Computer. IEEE Computer Society, Los Alamitos (2009)
19. Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: Automating Co-evolution in Model-Driven Engineering. In: 12th Int. IEEE EDOC, pp. 222–231 (2008)
20. Wachsmuth, G.: Metamodel Adaptation and Model Co-adaptation. In: Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 600–624. Springer, Heidelberg (2007)