An Interface Agent for the Management of COTS-based User Interfaces

Jose F. Sobrino, Javier Criado, Jesus Vallecillos, Nicolas Padilla and Luis Iribarne
Applied Computer Group, University of Almeria, Almeria, Spain
{jfsobrino, javi.criado, jesus.vallecillos, npadilla, luis.iribarne}@ual.es

Keywords: Interface Agents, Component, User Interfaces, Widgets, Interaction.

Abstract: The great development of the knowledge society on the Internet requires that Web information systems are adapted at runtime to user groups with common interests. Interface agents help us to observe and learn from user preferences making interfaces adaptable to user working habits. We propose an interface agent which works on Web interface based on COTS components, adapting the interface to the user needs or preferences. Our agent runs two main behaviors: observation behavior which analyses the user interaction on the interface and a second behavior which runs the adaptation actions to adapt the user interface at runtime.

1 INTRODUCTION

Due to the rapid expansion of the information and knowledge society on the Internet, Web-based Information Systems (*WIS*) must be prepared to be easily adaptable, extensible, accessible and manageable at runtime by different people with common interests. In this type of system is important to have interfaces that facilitate human-computer interaction, promoting the dynamism design of components adapted to the users habits and the development of their work. Interface agents have become a technology widely used in the development of interfaces adapted to the user needs. Such agents have the ability to observe and learn from the preferences and work habits to provide an UI adaptation. Our interface agent works on Web-UI based on *Commercial Off-The Shelf* (COTS) interface components of *widgets*-type. These interfaces offer a great versatility and adaptability making the adaptation an easy process for the interface agent.

Our research proposes an interface agent that *looking over the shoulder* of the user (Maes, 1994), collecting each event or action that user performs on the components and executing changes in the interface model. Each action performed on a component by user is interpreted and classified by the interface agent, deciding whether the action changes the model. These changes involve an adaptation process. The adaptation process is not performed by interface agent, but this is done within a *adaptation engine* described in (Rodriguez-Gracia et al., 2012a) and (Criado et al., 2012). This article does not detail this engine, limiting us to use its input/output. After obtaining the adaptation actions that provides the adaptation engine, our interface agent makes the changes on the interface. This type of interface agent that helps to adapt and evolve interfaces is very useful, since it shows to user a friendlier interface when performs his tasks. The use of *widgets* components gives us a lot of dynamism in interface design due to encapsulation of functionalities and properties.

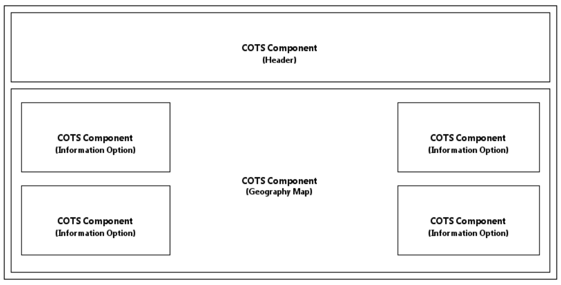
The rest of the paper is structured as follows. Section 2 describes the type of user interface where our interface agent works. Section 3 details the two main behavior of user interface. Section 4 gives some related work. Finally, in Section 5 describes some conclusions and future works.

2 PREVIOUS SCENARIO

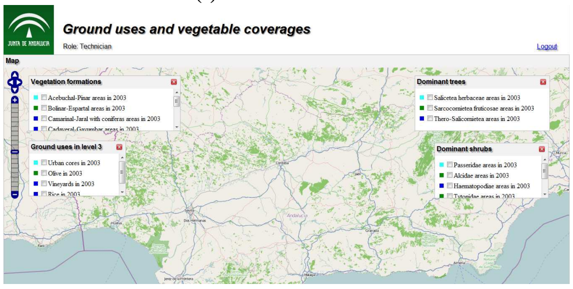
In a world more and more interconnected, the Web interfaces should be flexible and are prepared for an easy adaptation to the new type of users that work on the Internet. However today, Web interfaces are being built statically based on traditional Web development paradigms. We propose to build a Web interface with self-contained visual components, interchangeable and modifiable, varying its appearance or properties at runtime, depending of the user interaction on the components. We assume that there is an UI components market (Heineman and Councill, 2001) (Lau, 2004) that we use as source of our components. These components are stored in public repositories, being prepared to be assembled and put into operation. These interfaces are increasingly proliferating in interface types *smarts* for phones and smart TV, which are other particular interface where our approach can also be applied.

Our Web interface bases on the composition at runtime of *widgets*-type interface components, offering the possibility of reorganize the composition of the interface at runtime without losing functionality. The components are organized at the interface with a predetermined pattern that will be adapted with the user interaction. Figure 1(a) shows a interface model that represents this concept. Each component of the model can change its visual appearance or disposition within the interface, or even to be replaced by another component with different characteristics.

In Figure 1(b), we show a example of Web interface for the environmental management about a project of the regional government of Andalusia (Spain) in order to get a Web intelligent agent, being our interface agent a first step forward. In this Web interface prototype we can see six components. The first that we can see is the `Header` that is responsible for controlling the access to the system. The `Geography Map` visualizes the maps and the other four components correspond to different checkboxs to show data on `Geography Map`. Once detailed the scenario where the interface agent works, we detail the necessary behaviors to perform the interface adaptation.



(a) Model User Interface


JUNTA DE ANDALUCIA
Ground uses and vegetable coverages
Role: Technician
Logout
Map
Vegetation formations
Acebuchal-Pinar areas in 2003
Bolinar-Espartal areas in 2003
Camarinal-Jaral with coniferas areas in 2003
Cadaveral-Gavumbar areas in 2003
Ground uses in level 3
Urban cores in 2003
Olive in 2003
Vineyards in 2003
Rice in 2003
Dominant trees
Salicetea herbaceae areas in 2003
Sarcocornietea fruticosae areas in 2003
Thero-Salicornietea areas in 2003
Dominant shrubs
Passeridae areas in 2003
Alcidae areas in 2003
Haematopodiae areas in 2003
Tytonidae areas in 2003

(b) Real Prototipe User Interface

Figure 1: User Interface.

3 COTS-BASED INTERFACE AGENT MODEL

In our system, the interface agent is responsible for observing the user interactions with the interface and implementing the necessary changes to adapt the interface to the user needs. In Figure 2, we can see the user interaction with the interface components performing the tasks assigned. Our interface agent *observes* the actions sequences that executes the user on the components (Section 3.1) and transforms them in an *observer model* (Section 3.2). This model is the input of the *adaptation module*, which makes the appropriate adjustments depending on the user actions. The result is a set of *adaptation actions* (Section 3.3) that the interface agent executes on the interface.

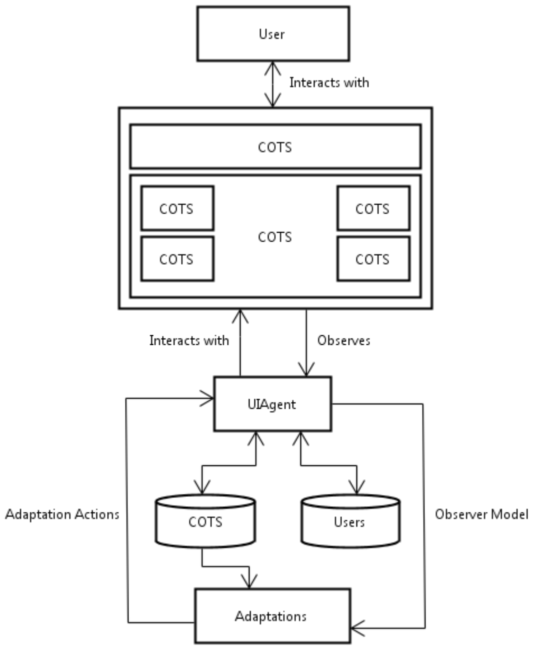


Figure 2: Interface Agent.

We will explain with more detail the UI adaptation at runtime in the Figure 3. The interface agent observes the user interaction with the `Observe Interaction`. In the next step, `Manage Interaction` compiles all the user actions creating sequences of actions. These two steps are performed continuously, storing these action sequences in the `Functional History` (FH). The`Identify Plans` module classifies the user actions stored in FH, deleting the actions which do not represent changes in the model. `Create Observer Model` is responsible for transforming the actions sequences performed by the user in an observer model for the adaptation module. When the adaptation module ends,

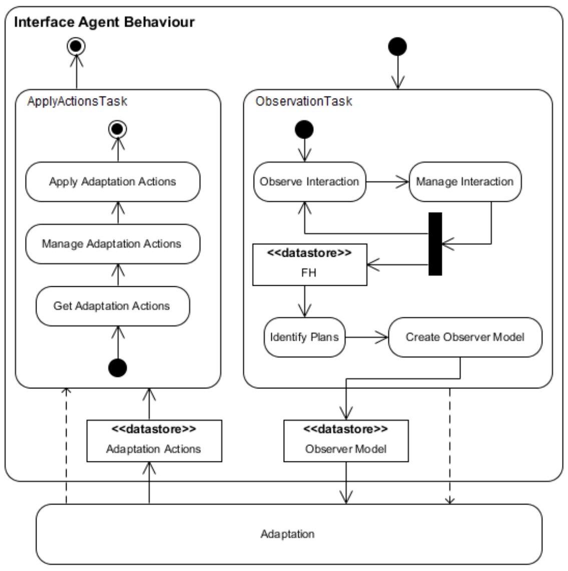


Figure 3: Interface Agent Behavior.

`Get Adaptation Actions` stores the adaptation actions and the `Manage Adaptation Actions` completes the required fields in order to execute the actions in the interface. The `Apply Adaptation Actions` executes the actions and adapts the interface components to the user actions. In the following sections we will explain each step with more detail.

3.1 User Interaction

Our interface agent operates as show in Figure 4, collects each user interaction. We define a ***input*** as a tuple $\langle e, obj, comp \rangle$ where *e* is the observed event, *obj* is the object on which was made the event and *comp* is the component. These inputs are analyzed in *observation module* and are sent to *intention detection module*. This module creates the user intention based on the input information considering the component and the event made. An ***intention*** is represented by a tuple $\langle e, obj, comp, int \rangle$, where *e, obj, comp* are elements inherited to observation module and *int* is the intention may be created of inputs. Each intention is sent to *create action module* which determines the action based on the user intention and stores this action in the ***Functional History*** (FH). This module defines an ***action*** as a pair $\langle n, comp \rangle$ where *n* is the name of the action and *comp* is the component which performs the action. The process returns cyclically to observation module completing a cycle.

A possible example would be to execute the necessary actions to display the information in an area of the *Geography Map* component (sequence (1)).

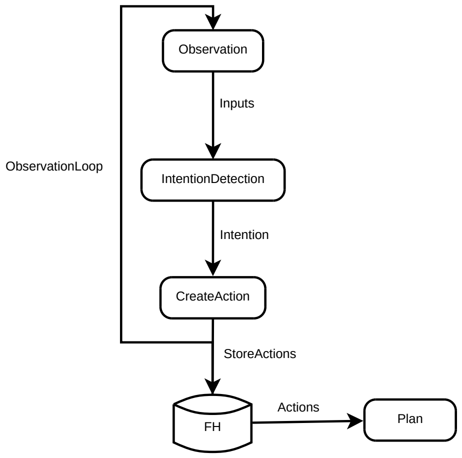


Figure 4: Interaction Basic Process.

$$\begin{aligned} \textbf{input} &= \langle Click, Layer, Map \rangle \\ \textbf{input} &= \langle Move, Layer, Map \rangle \\ \textbf{input} &= \langle DblClick, Layer, Map \rangle \end{aligned} \tag{1}$$

We suppose that the user moves the mouse to the position where wants to consult, making *click* at an upper point of sector and *moving* to the other point selecting an area. Then the user makes *dobleclick* in the center of the selected area showing the necessary information for the user. Then the intention detection module examines each input and builds the following intentions:

$$\begin{aligned} \textbf{intention} &= \langle Click, Layer, Map, Click \rangle \\ \textbf{intention} &= \langle Move, Layer, Map, Select \rangle \\ \textbf{intention} &= \langle DblClick, Layer, Map, Action \rangle \end{aligned} \tag{2}$$

These intentions are studied and turned in actions:

$$\begin{aligned} \textbf{action} &= \langle ClickLayer, Map \rangle \\ \textbf{action} &= \langle SelectLayer, Map \rangle \\ \textbf{action} &= \langle ShowInfo, Map \rangle \end{aligned} \tag{3}$$

These actions are orderly stored in the FH. A ***plan*** is represented by a tuple $\langle p, comp, A_p \rangle$ where *p* is the name of the plan, *comp* is the name of the associated component and A_p is an ordered set of necessary actions to carry out the plan. Each component has associated a set of plans which are defined during the development of the components in period of software engineering. The sequence (3) belongs to a plan associated to the *Geography Map* component. A sequence of actions is determined by one of the following cases: the collection of actions appears in the plan, the collection of actions corresponds to the entire plan or an action does not appear any time within the plan.

The system must allow to segment and classify all actions included in FH. The partition algorithm is described in the Table 1.

Table 1: Partition Algorithm Pseudocode.

```
process PartitionActions
1: for all $a$ in $FH$ do
2:   if $a.comp$ is contained in $COTSGUI$ then
3:     for all $P_j$ in $COTS.PL$ do
4:       if $a = P_j[1]$ then
5:         ADD $a$ in $P_t$
6:         if $P_j = P_t$ then
7:           return $P_t$
8:         else
9:           REMOVE $a$ to $P_j$
10:           $PartitionActionIn(FH, P_j, P_t)$
11:           if $P_j = P_t$ then
12:             return $P_t$
13:           else
14:             return $P_t = \emptyset$
15:           end if
16:         end if
17:       end if
18:     end for
19:   end if
20: end for
```

```
process PartitionActionIn
1: for all $a_i$ in $FH$ do
2:   for all $a_j$ in $P_j$ do
3:     if $a_i = a_j$ then
4:       ADD $a_i$ in $P_t$
5:     else
6:       return $P_t = \emptyset$
7:     end if
8:   end for
9:   if $P_j = P_t$ then
10:     return $P_t$
11:   else
12:     return $P_t = \emptyset$
13:   end if
14: end for
```

If we examine our algorithm, each action a is obtained from the *FH* queue and is never considered any more. Then it checks the component where the action has been executed in the set of components (*COTS-GUI*) from GUI. This component (*COTS*) returns all associated plans (*COTS.PL*) and it checks if it corresponds to the first actions sequence that forms a plan of actions (P_j). If the action a corresponds to the first action of a plan, this action is added to a temporal plan P_t. If the plan has an action of sequence, the algorithm finishes returning the executed plan. If the plan has two actions or more, the algorithm uses the *Partition-ActionIn* algorithm. This algorithm tries to complete the actions sequence included in the selected plan. That is, the algorithm goes straight on checking the actions obtained from *FH* until completing the plan. If at any time, the action obtained from *FH* does not match with the action to be executed, then the algorithm finishes returning an empty plan. If the actions sequence obtained from *FH* matches perfectly with the sequence of the plan, the algorithm returns the executed plan.

For example, we display the information in a sector of *Geography Map* component. We define an action a_{ij} where i is the number associated with the component plan and j is the sequence of actions of each plan. We suppose that in FH stores the following collection of actions $\{a_{11}, a_{21}, a_{71}, a_{22}, a_{23}, a_{72}, a_{31}, a_{32}, a_{33}\}$, where $\{a_{31}, a_{32}, a_{33}\}$ are the actions of example:

$$\begin{aligned}\mathbf{a_{31}} &= \langle ClickLayer, Map\rangle\\ \mathbf{a_{32}} &= \langle SelectLayer, Map\rangle\\ \mathbf{a_{33}} &= \langle ShowInfo, Map\rangle\end{aligned} \tag{4}$$

We suppose that all actions have been carried out on a component which only has three associated plans (PL_1, PL_2, PL_3) where PL_3 is the corresponding to the previous example. When the algorithm is performed, the action $\{a_{11}\}$ is associated to PL_1. The sequence $\{a_{21}, a_{71}\}$ does not match the PL_2 since $\{a_{71}\}$ is not an action of the selected component. $\{a_{22}, a_{23}\}$ does not correspond with the beginning of a plan component. $\{a_{31}, a_{32}, a_{33}\}$ corresponds to PL_3. PL_1, PL_3 are plans component. $\{a_{72}\}$ is an action that does not belong to any plans. Using the partition algorithm, we get an effective segmentation. The sequence $\{a_{72}\}$ would be checked if it belongs to some other plans, being discarded at the end. Once the interface agent has identified a plan, it generates a model that is sent to the *adaptation engine*. This process is described below.

3.2 Observer Model

In order to develop a right adaptation, it is also necessary to get other elements of the UI proposed by (Criado et al., 2012) and observing system requirements that are described in (Troya et al., 2010). All these variables make up our observer meta-models showed in Figure 5, which shows the meta-model of the relationship between observed actions and monitored context variables. This meta-model is required by the adaptation module. This defines three types of observers: `ComponentObserver`, `ObserverObserver` and `ContextObserver`. The first type monitors variables of the components on the system. The second type provides information about other observers, and the third is responsible for observing the context variables. We add a new *ComponentObserver* completing the observation of the user interaction (inputs), which is represented by `PlanObserver`. The observation of these types of *observer* gives a complete view of everything that happens in the graphic user interface.

This article will not detail the adaptation operation. For a more detailed we suggest the reading of (Criado et al., 2012) and (Rodriguez-Gracia et al.,

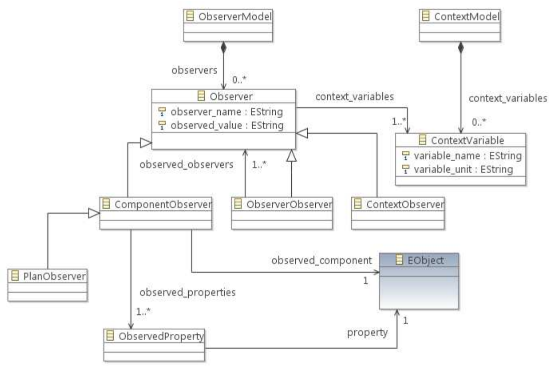


Figure 5: Observer Meta-model.

2012b). We will send created observer model to *adaptation module* and will return a set of adaptation actions that the interface agent interprets and executes in the Web interface. In the next section we will detail these adaptation actions.

3.3 Adaptation Actions

When the *adaptation* module finishes, the interface agent receives the adaptation actions that uses to carry out the adaptation of the UI. The Figure 6 shows the adaptation actions meta-model.

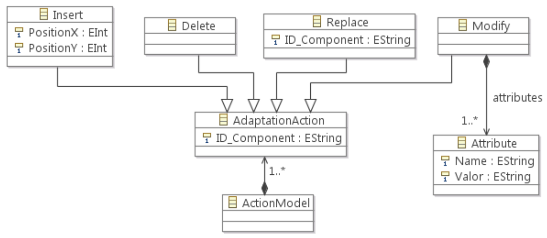


Figure 6: Adaptation Actions Meta-model.

As we can see there are four types of adaptation actions, *Insert*, *Delete*, *Replace* and *Modify*. The adaptation action element has as main attribute the component ID which executes the action. Each of one of these adaptation actions are performed under a certain order. It also checks that all adaptation actions complete each adaptation action with specific attributes of Web environment. Each of one of the adaptation actions are converted by the interface agent to Web specific language in order to be executed directly on the interface.

In the table 2 we can see an adaptation action in *XML* that modifies the *Geography Map* component. Each component is identified by its ID. This operation changes the frame size of the component executing the new conditions of the user action. It is executed directly by the interface agent at runtime, modifying the visual appearance of the UI without the intervention of the user.

Table 2: Adaptation Actions.

Action *ModifyAction*
< *widget id* = ”*map*01258” *operation* = ”*modify*” >
< *closePosition value* = ”*relative*”/ >
< *closeTop value* = ” − 22*px*”/ >
< *closeLeft value* = ”208*px*”/ >
< *height value* = ”100*px*”/ >
< *iframeHeight value* = ”100*px*”/ >
< *iframeWidth value* = ”100*px*”/ >
< *width value* = ”230*px*”/ >
< /*widget* >

4 RELATED WORK

Some approaches take into account variables associated with the actions as (Maes, 1994) (Brown et al., 1998). The main problem with this approach is that it ignores the user tasks and that the values of variables are pre-fixed. There are studies that attempt to predict the user intentions with non-probabilistic methods using logical methods as (Kautz et al., 1991) (Goultiaeva, 2006). This approach can not decide what measure the evidence supports the hypothesis particular user intention. Other works like (Mott et al., 2006) (Charniak and Goldman, 1991) use n-gram models and Bayesian networks to probabilistically improve the prediction of user intentions.

Another approach to interface agents to consider is to control the design and creation of UI. SurfAgent (Somlo and Howe, 2003) is an information agent that builds a user profile by using examples provided by the user documentation. (Li, 2009) is a method of constructing UI based on a model agent type *Model-View-Controller* showing the pattern design and development method of UI based on agents an information retrieval system. (da Silva et al., 2000) describes the solutions offered by a mobile agent system (AgentSpace) showing two complementary ways to create UI with the mobile agent. (Arias and Daltrini, 1996) shows a framework for the conceptual and detailed design of the UI that helps people involved in the task of designing the UI, helping to create a custom interface. All these papers are focused on the design and control of interfaces, but do not take into account the user interaction.

5 CONCLUSIONS AND FUTURE WORK

Our paper presents an interface agent that observes to the user, collecting all events and actions performed on the *widget*-type interface component, and promotes changes in the interface model, making that the interface evolves and adapts to the characteristics and needs of the user work. We have shown the process to obtain the user actions in form of plans and how the interface agent creates an observer model. This observer model is sent to the adaptation engine that generates a set of adaptation actions to change the interface. Our interface agent completes and executes these adaptation actions transparently, making that the interface is adapted to the actions performed by the user. Our interface agent combines capabilities and other features of interface agents, allowing to obtain the user intention based on the interaction among components, and at the same time is capable of changing the visual appearance of the interface as a result of the user interaction.

Finally, we are working to provide the system with social features through user groups, adding to the system abilities to work with multiple users in a cooperative manner, showing different interfaces to different users who are doing work cooperatively.

ACKNOWLEDGEMENTS

This work has been supported by the project JUNTA ANDALUCIA (proyecto de excelencia) TIC-6114, and the EU (FEDER) and the Spanish Ministry MINECO under grant of the projects TIN2010-15588 and TRA2009-0309 and Ingenieros Alborada IDi.

REFERENCES

Arias, C. and Daltrini, B. (1996). A multi-agent environment for user interface design. In *EUROMICRO 96. 'Beyond 2000: Hardware and Software Design Strategies'., Proceedings of the 22nd EUROMICRO Conference*, pages 242 –247.

Brown, S. M., Santos, Jr., E., Banks, S. B., and Oxley, M. E. (1998). Using explicit requirements and metrics for interface agent user model correction. In *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, pages 1–7, New York, NY, USA. ACM.

Charniak, E. and Goldman, R. (1991). A probabilistic model of plan recognition. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 1*, AAAI'91, pages 160–165. AAAI Press.

Criado, J., Iribarne, L., Padilla, N., Troya, J., and Vallecillo, A. (2012). An mde approach for runtime monitoring and adapting component-based systems: Application to wimp user interface architectures. In *38th Euromicro Conference on Software Engineering and Advanced Applications*.

da Silva, A., da Silva, M., and Romao, A. (2000). Web-based agent applications: User interfaces and mobile agents. volume 1774 of *Lecture Notes In Computer Science*, pages 135–153. Springer-verlag Berlin. 7th International Conference On Intelligence In Services And Networks (is&n 2000), Athens, Greece, Feb 23-25, 2000.

Goultiaeva, A. (2006). Incremental plan recognition in an agent programming framework. In *In Cognitive Robotics Workshop*, pages 83–90.

Heineman, G. T. and Councill, W. T. (2001). *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Kautz, H. A., Kautz, H. A., Pelavin, R. N., Tenenberg, J. D., and Kaufmann, M. (1991). A formal theory of plan recognition and its implementation. In *Reasoning about Plans*, pages 69–125. Morgan Kaufmann.

Lau, K.-K. (2004). *Component-Based Software Development: Case Studies*. World Scientific Press.

Li, Y. (2009). Intelligent user interface design based on agent technology. In Tran, D. and Zhou, S., editors, *2009 WRI World Congress on Software Engineering, Vol 1, Proceedings*, pages 226–229. World Res Inst, IEEE Computer Soc. World Congress on Software Engineering, Xiamen, China, May 19-21, 2009.

Maes, P. (1994). Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40.

Mott, B., Lee, S., and Lester, J. (2006). Probabilistic goal recognition in interactive narrative environments. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 187–192. AAAI Press.

Rodriguez-Gracia, D., Criado, J., Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2012a). Composing model transformations at runtime: an approach for adapting component-based user interfaces. In *ICSOFT 2012*, pages 261 – 226.

Rodriguez-Gracia, D., Criado, J., Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2012b). Runtime adaptation of architectural models: an approach for adapting user interfaces. In *LNCS 7602*, pages 16 – 30.

Somlo, G. L. and Howe, A. E. (2003). Using web helper agent profiles in query generation. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 812–818, New York, NY, USA. ACM.

Troya, J., Rivera, J. E., and Vallecillo, A. (2010). On the specification of non-functional properties of systems by observation. In *Proceedings of the 2009 international conference on Models in Software Engineering*, MODELS'09, pages 296–309, Berlin, Heidelberg. Springer-Verlag.