Interface Design through Knowledge-Based Systems: An Approach Centered on Explanations from Problem-Solving Models

Andréia Libório, Elizabeth Furtado, Ismael Rocha, Vasco Furtado
Master's Program in Applied Information Sciences
University of Fortaleza - UNIFOR
Av. Washington Soares, 1321, Edson Queiroz CEP 60.811.905 Fortaleza-Ceará-Brazil
(55) (85) 3273-3382
Andreia_Liborio@yahoo.com.br , Elizabet@unifor.br, Ismael_Rocha@yahoo.com, Vasco@unifor.br

ABSTRACT
The process of generating user interfaces is complex and demands a great deal of effort from the specialist, because there are several possible combinations and uncertainties regarding any one option. We believe that the modeling of HCI concepts and of the knowledge of all parties involved is of great importance, as well as the reuse of this modeling to automate and optimize such process. Therefore, we are proposing a KBS (Knowledge-Based System) that represents the knowledge of the interface designers. This System possesses the feature of interactivity, so that the user can have an explanation of the results produced, thus improving the understanding and acceptance of the proposed interface. Furthermore, the system allows for its knowledge base to be modified. We applied the proposed method to generate an abstract user interface for a system of email control and for a system of simulation of criminal activity.

Keywords
Interface Generation, Abstract Interface, Knowledge-Based System, Problem-Solving Method

INTRODUCTION
The design and implementation of a Human/Computer Interface (HCI), in addition to being of fundamental importance to the quality of the software, is one of the most time-consuming stages in the development process. Designers have attempted to streamline these stages with the creation of automatic interface generators. The aim is basically to define programs that perform a mapping between models that represent tasks carried out by a user in a model of interfaces. In several works [1], [16], [8], interface generation goes through an intermediate stage of generating an abstract interface, which is the description of the components that will make up the final interface—without its characterization in terms of interactive objects such as buttons and windows, and without being concerned about the features of the device whereon the interface will be shown. The abstract interface aims at facilitating the dialogue between the interface designer and the domain user during the process of system analysis that is being developed. It allows for the schematic viewing of what components will be shown on the software interface and, in a generic way, how this will be done.

The process of mapping out the tasks model and the data model on an interface (whether abstract or not) is not trivial [8], [3]. These initiatives have been shown to be insufficient, since the main component involved in this mapping, i.e. the designer's knowledge, is improperly addressed. Our argument is that it is necessary to create an explicit structure of acquisition and representation of the knowledge of one (or several) specialist(s) in designing interfaces to compose a Knowledge-Based System (KBS). Moreover, we consider that the process of interface generation is cyclical, and a revision of the entry models and primarily of the designer's knowledge can be carried out constantly. One cannot believe that it is possible to feed, only once, a model mapping system that will generate a complete and ready-to-use interface. Even if the KBS has rules of unquestionable quality, a cyclical process of validation is necessary, since the HCI models (tasks model and data model), which are the basic inputs of the generation process, often require constant revalidations.

Under this perspective, we consider that the KBS must provide a capacity for interaction with the designer and with the user of the system, aiming to facilitate the understanding of the process of automatic generation and allowing for the revalidation of the inputs in this process. In order to attain these objectives, the KBS must provide explanations of why a certain interface was proposed for the entry models. This explanation must make transparent not only the basic domain rules, but also the strategy of reasoning used by the interface specialist. What is really desired is that the KBS, by providing structured explanations of the process of interface generation, facilitate the validation of the tasks model, of the domain model, and ultimately of the very rules used in the KBS. The current state of the art in interface generation reveals deficiencies for it to be used in the way that we specify. One of the first problems comes from the fact that the existing works consider that the specialist's knowledge is represented exclusively by the "if-then" type of production rules, which is not appropriate for representing the specialist's reasoning strategy. Additionally, the need for interaction between the designer and the system, which is characterized as essential in this context, it is not explored.

Aiming to solve the deficiencies identified, we describe in this article a solution based on the representation of the reasoning of an interface designer starting with the assumption that he/she performs a design activity following a propose-and-revise type of solution method. We will describe how the solution was designed and implemented through the UPML framework and by using the concept of Interactive Knowledge-Based System— KBSi [10]. In this implementation, we used the components for explanation defined in this work, which permits an explanation of all the steps of reasoning of the KBS at different levels of abstraction, thus facilitating an understanding of how the interface was designed. A user-friendly interface provides explanations on the process of interface generation and allows revalidations of the task model, the domain model, and the knowledge base.

This article is structured in the following way: first we will present the state of the art in Knowledge Engineering regarding the construction of KBSs, and then we will relate some of the existing efforts in the field of HCI that approach this issue of user interface generation. Afterwards, we will describe the proposed method and the stages of development of the KBS. We will show how our focus was validated with a description of its application for the generation of different abstract interfaces in two real domains: a system of email control and a tutorial system based on simulations for the realm of public safety.

STATE OF THE ART

Below we will describe important concepts and referenced works in the disciplines of Artificial Intelligence and HCI in relation to the use of KBSs to generate user interfaces.

Development of Knowledge-Based Systems

KBSs are systems with characteristics that we associate with human intelligence and are developed to help people solve problems that involve an intensive use of knowledge, such as: control, planning, assessment, diagnostics, design, and decision-making, among others. Therefore, KBSs constitute an important tool for acquiring, retaining, and disseminating knowledge, since they represent—in a knowledge base—a model of the reasoning used by humans as well as domain concepts and relationships.

Knowledge engineering aims at studying methods and techniques for the development of KBSs. Work in this field has evolved such that the process of knowledge acquisition becomes a process of instantiation of reasoning models. This is being done through the concept of the Problem-Solving Method (PSM) whereby the steps of reasoning and the type of knowledge necessary to perform a task are represented. The function of the PSM is to make the reasoning process explicit in a KBS.

Some PSM libraries do exist; they seek to define and to structure the problem-solving models, whereby the construction of KBSs is facilitated. PSM libraries are described in [12] and [9]. An approach to model a PSM generically is the Unified Problem-Solving Method Description Language (UPML) [2], which describes the different software components of a KBS integrating two important lines of research in Knowledge Engineering: reusability of components and ontologies. The UPML framework supports the modeling of KBSs starting from reusable components, adapters, development guidelines, description language and tools. UPML architecture describes the different components of a KBS: (i) *Task* defines the problem to be solved by the KBS; (ii) *PSM* (problem-solving method) defines the KBS's process of reasoning; (iii) *Domain Model* describes the domain of knowledge involved in the KBS; (iv) *Ontologies* describes the terminology used in the other elements; (v) *Bridge* models the relationship between two UPML components; (vi) *Refiner* can be used to specialize a component.

Explanations in KBS

The knowledge base of a KBS is constructed according to the specialist's static vision of knowledge, which often diverges from the knowledge of the user, making it difficult to understand the system's reasoning strategies and functionalities. The difficulties that come up when KBSs are being utilized are prejudicial to the user's acceptance of the proposed solutions. Therefore, a KBS's capacity for explanation is considered fundamental to improving the user's acceptance. A KBS must have the capacity to explain "how" and "why" it arrived at a particular solution.

In a KBS, explanation allows the user to interact with the system by requesting and receiving explanations on the system's reasoning and actions. The explanations provide perception within the system's knowledge and facilitate an understanding of its functioning.

Most KBSs that have come on the scene have been designed to provide explanations in the form of "rules tracking." However, this approach to explanation is more useful for the developer to discover problems in the system's functioning ("bugs"), than to the "novice" users, since the explanations presented are overloaded with detailed information in the KBS's development language.

Interface Generation

Interface Generators allow the developer to have available and to organize the diverse elements of a graphical user interface. Some approaches include code generators that produce skeletons so that the developers can write the code that will be integrated into the interface. Some works are based on models. One of them is the interface-generating framework called "Cameleon Reference Framework" [1], which defines steps to develop user interfaces for context-sensitive interactive applications. A context is made up of: environments, platform and user. Four steps are identified during the process of interface generation: (i) Task and Concepts: describes several tasks to be executed and domain concepts; furthermore, represents how such concepts are required by the tasks; (ii) Abstract User Interface: a standard expression that generates domain concepts and functions such that its representation is independent of the available interactions on the target interface. The elements used in the logical Interface are abstractions of existing concrete objects.; (iii) Concrete User Interface: materializes an Abstract Interface into Objects of Interaction independent of the representation of the final interface; this defines both the layout of the objects as well as the navigation among screens; (iv) Final Interface: it is typically the concrete interface codified in some language, interpreted or compiled.

Also—and still following the line of interfaces generation based on models—another interesting proposal is the tool known as TERESA (Transformation Environment for Interactive Systems Representations) [16], which it is a semi-automatic environment for construction of interfaces based on transformations of models, useful for designers to construct and analyze their designs on different levels of abstraction and consequently to generate the concrete interface for specific types of platform. The method considered for this environment consists of the following steps: (i) modeling of high-level tasks of a multiplatform application; (ii) development of the system's tasks model for the different platforms considered;, (iii) translation of the system's tasks model to an abstract user interface; and (iv) generation of the final user interface. TERESA adopts the XML standard to represent the models and interfaces.

In [8] the TransformiXML environment was defined, which carries out mappings by means of the relationships between mathematical expressions and allows for the definition and application of transformation rules. These rules represent association between models to generate the user interface. This environment is divided in two components: (i) an Application to program the interface (TransformiXML API) that can be used by any system to apply rules of transformation in a non-interactive "batch" manner; (ii) a graphical user interface which serves as API data entry (TransformiXML GUI) in an interactive way. This environment adopts the UsiXML standard [7] for the HCI models and the rules that map out such models.

Also using the Interface Generation strategy based on the notion in the concept of the abstract interface and using guidelines proposed in [15] and [3]. TRIDENT [15] is a set of interactive tools that automatically generate a user interface for applications geared toward highly interactive businesses. It combines the use of tasks model with interface design guidelines. These guidelines are encapsulated using a decision tree technique. In [3] a new methodology of adaptive automatic interface construction is presented, which integrates human factors in the process of software development, thus creating a bridge between the two sciences—Software Engineering and Cognitive Sciences. Human factors relative to the behavior of the user and ergonomic recommendations are considered in such a way as to automatically deduce aspects relative to the human/computer interaction, such as to define a mechanism for adapting the interfaces constructed.

None of the aforementioned model the knowledge of the specialist for the activity of interface generation. [16], [3] and [8] consider part of this knowledge through the use of rules. In [13], the author considers the use of knowledge management in the process of interface conception and defends that the valorization and the efficient exploration of the knowledge involved in the process of interface conception depend essentially on the definition and adoption of methods that allow for the effective classification, representation, integration, and usage this knowledge. Therefore, to facilitate the interface designer's preparation for an adequate practice of Knowledge Management in the process of interface conception, and consequently to minimize the work load and cognitive effort in the performance of their activities, the author defined a methodology for the preparation of the actors in an interface design environment in order to classify, represent, integrate, and use the relevant knowledge of this context. However, was not constructed a system for automatic generate interface.

After assessing various works on HCI, we saw that interface generation is complex and demands a great deal of effort from the specialist. Therefore the modeling of the specialist's knowledge would be of extreme relevance to assist in such a process. We also observed that very few works addressing this issue utilize the resources of knowledge engineering. The ones that modeled knowledge had done so in a less-than-satisfactory way: firstly because they only modeled the interface design guidelines without taking the designer's experience into account, and secondly because of the fact that the representation was in the form of rules, thus excluding the representation of the specialist's reasoning. Therefore, we consider the construction of an interactive KBS, called a KBS_{HCI}, to automate the process of user interface construction, which will represent the knowledge of an interface designer, including his/her implied knowledge.

The KBS_{HCI} will be described below, showing its architecture, features, advantages, and technologies utilized.

KBS_{HCI}: AN INTERACTIVE KBS FOR INTERFACE CONSTRUCTION

For the construction of the KBS_{HCI}, we based our efforts on the UPKi methodology [10] that defines who does "what," "when," and "how" in a process of producing a high-quality KBS that meets the user's needs. The description of the KBS_{HCI} will be made by following the phases that make up this methodology.

Conception Phase

In order to initiate the construction of the KBS that we are proposing, in the first phase we delimited the scope, surveyed the requirements, and performed a feasibility analysis for the KBS.

The basic requirements elicited in the conception phase indicated two relevant aspects: (i) we saw that the process of generating interfaces must be an iterative process, passing through diverse stages so that the designer may assess the several proposals that may be made until finding the most appropriate interface. This brought to us the idea that the KBS_{HCI} should allow forms of interaction, such as—for example—explanation and cooperation; (ii) we identified that interface generation is based on HCI models that describe the activities performed by a person in the development of his/her work duties as well as concepts of the domain of the application to be constructed. Based on the activities and concepts represented in these models, we can identify which objects will make up the interface; (iii) we observed that the interface generated must fulfill the requirements of designers, which reflect interface design construction guidelines as well as the designers' experience.

We then defined the objective of the KBS_{HCI}: to design the user interface based on HCI models and on requirements demanded by designers.

Elaboration Phase

In this phase we identified technical and functional aspects, elaborated the architecture of the KBS_{HCI}, and accomplished knowledge acquisition and modeling.

In the first place, we sought to identify which type of task characterized the activity of constructing an interface. We verified that this dealt with a design task where a developer performs an activity of solution proposal considering a set of preferences and constraints until the interface is generated.

In order to fulfill the requirement of interactivity: (i) in the implementation of the KBS_{HCI}, we used standards for explanation considered in [10], which allow us to explain all of the steps of reasoning of the KBS at different levels of abstraction, thus facilitating the understanding of how the interface was designed and improving acceptance by user; (ii) we provided resources for the design to modify the KBS knowledge base.

Figure 1 illustrates the architecture of the KBS_{HCI} with the components and agents involved.

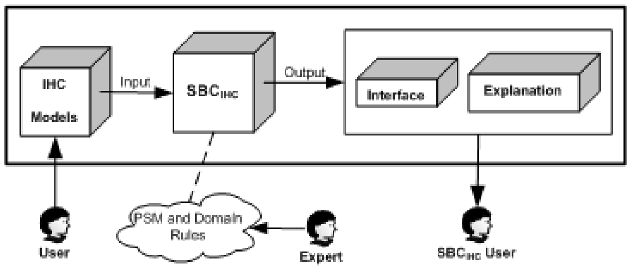


Figure 1. Architecture of the $KBS_{HCI.}$

The KBS_{HCI} receives the HCI models (Tasks and Data) as input constructed by a user familiar with the application domain. The KBS_{HCI} possesses the knowledge of how to produce an interface; it is composed of the reasoning strategy (PSM) and the Domain rules. The output produced is the designed Interface and a module of explanation that will show "how" and "why" the design was conceived. This result is presented to the KBS_{HCI} user.

After defining the architecture, we carried out the KBS_{HCI} knowledge acquisition, which lead us to define a PSM in UPML that implements a strategy of propose-and-revise. We adopted the method defined in [12]. Figure 2 graphically represents the knowledge of the KBS_{HCI} with the following elements: (i) the "Design" task that represents the problem to be involved; (ii) the Propose-and-Revise PSM, which is composed of the subtasks Specify, Propose, Verify, Critique, Select, and Modify; (iii) the domain model that is composed of the HCI base knowledge; (iv) the ontologies Design, Propose and Revise HCI correspond to the ontologies of the task, the PSM, and the domain. It is important to verify that both the "Propose and Revise" method and the "Design" task can be reused in other domains, being necessary only to map out the concepts used in the PSM and the task with the domain concepts. In this case, the ontology of the task is composed of the concepts of components, specifications, preferences, constraints, and fixes, and in the domain ontology there is the tasks model, the data model, and the interface design guidelines. The task-domain bridge is responsible for establishing this mapping.

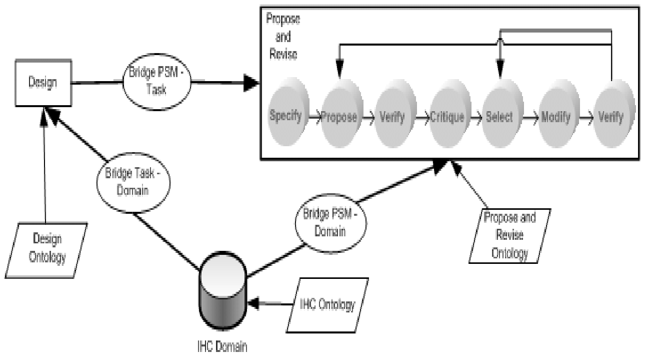


Figure 2. Model of the KBS_{HCI} Knowledge according to UPML architecture.

Construction Phase

Below we detail the implementation of the KBS_{HCI}. We will describe the main screen and we will outline the technologies and tools utilized in this phase.

The main screen of the KBS_{HCI} is illustrated in Figure 3. It allows the execution of its functioning to be shown step-by-step. It is divided into two parts: the left side is the graphic space for viewing the interface being generated, and the right side is for viewing the explanation of the reasoning implemented. The Explanation section shows why a specific object was inserted on the interface, and is divided into two frames: The first one presents the overall level of the explanation in the form of sentences showing the entire process of reasoning, and the second one details the explanation of the sentence's existing questionings.

Another option offered is that of modifying the knowledge base of the KBS_{HCI} through the options Specifications, Preferences, Constraints, and Fixes.

There are also the "Start," "Stop," "<<," and ">>" buttons, which control the functioning of the execution and allow one to run the KBS step by step, graphically showing the objects being inserted on the screen.

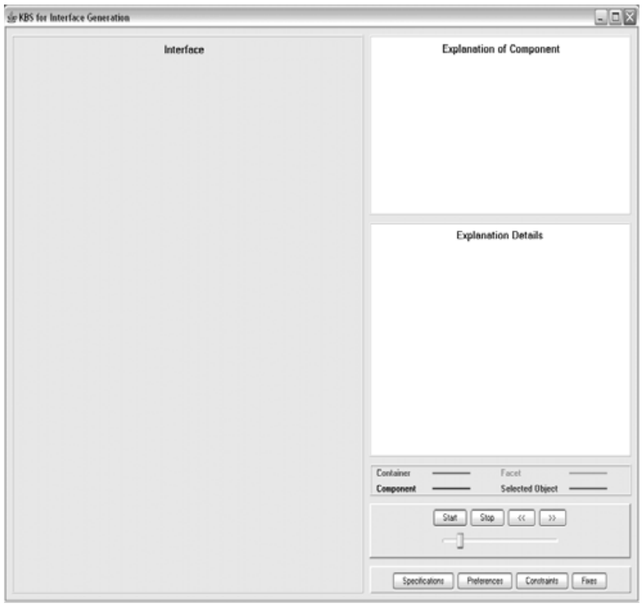


Figure 3. KBS Interface

The explanation module is one of the outputs of the KBS_{HCI}. It is a set of Java classes that allows the generation of the explanation of the functioning through the proof tree, which is the structure that stores the KBS's reasoning steps. This tree is generated during the execution of the PSM through the use of the "ProofGeneration" class which implements the explanation standard.

Several technologies were used to support the construction of the KBS: (i) Protégé was the ontologies editor used to edit domain concepts (tasks model, specifications, preferences, constraints, and fixes); (ii) CTTE was the tool used to draw the tasks model; (iii) we adopted the UsiXML standard defined in [8]—which is a description language that represents the user interface—to represent the tasks model and some HCI concepts; (iv) we adopted UPML architecture to represent the knowledge, since it structures knowledge in such a way as to allow the reuse of some portions and facilitates the explanation of the KBS_{HCI}'s functioning.

APPLICATION / CASE STUDY

Below we will demonstrate how the KBS_{HCI} was used. We used the KBS to generate the abstract interface for two systems: the first one is a system of email control and the second one is a tutorial system that simulates criminal activity in a particular region. We describe the features of the interfaces that we wished to design with the use of the $\text{KBS}_{\text{HCI,}}$ we report on how the models were instanced, and we show the designed interfaces. In the first example, we outline how the explanation of the KBS's functioning works, showing an example of a component that violated a constraint and the repair that was performed. Secondly, we show a situation in which there is a need to refine the tasks model during the validation of the abstract interface constructed.

Generation of an Abstract Interface for Email Control System

Description of the Interface to be Designed

We selected a system of sending/receiving and manipulation of email. One of the characteristics of this application is the necessity of being accessed from several different types of devices, that is, the user can operationalize his/her mailbox from a desktop computer, a palmtop, a cell phone, etc. To contemplate this necessity, some of the works consulted [1, 8, 16, 15] use the concept of abstract interface that is a representation of the interface elements (abstract spaces and objects) and of the navigation between the abstract spaces, independent of the mode of interaction (e.g.: graphical, vocal, video, etc.) and of the device. It is constructed during a stage prior to the concrete interface. An abstract interface is composed of abstract objects of the following facets: object of input, object of output, object of control, and object of navigation. Such components represent a typology of concrete objects (buttons, text boxes, lists, etc.). The objective is for the interface to be adapted to the different user types, executing the same tasks, using specific devices in various physical environments. Therefore, we wished to design the abstract interface of the system of email control, starting from the tasks model and data model previously defined by the designer.

Instantiation of the Models

The interface designer constructed the tasks model in the CTTE graphical tool (Figure 4). Afterwards, the tasks model and data model were instanced on the ontology corresponding to the UsiXML standard [7] in the Protégé tool [4.]. Immediately thereafter, we generated Java objects from the classes of the KBS domain based on the Protégé ontology.

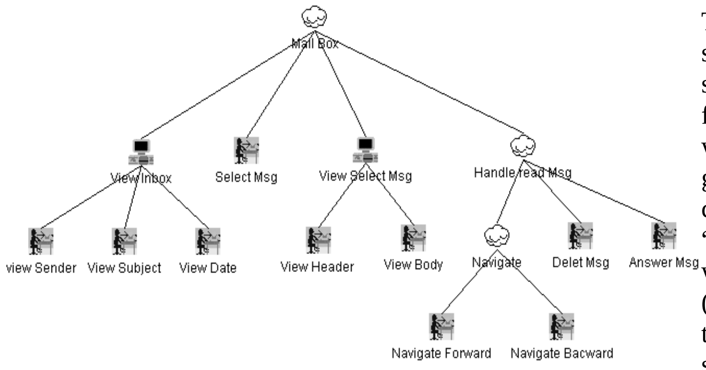
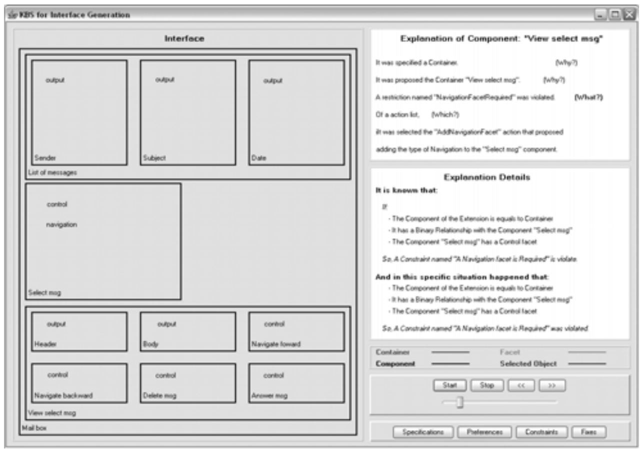


Figure 4. Tasks Model in the CTTE tool.

Designed Interface

Figure 5 presents the interface designed for the system of email control. It is made up of 13 components (Mailbox, List of Messages, Sender, Subject, Date, Select Message, View Selected Message, Header, Body, Navigation Forward, Navigation Backward, Delete Message, Reply to Message). These components are called abstract objects and may have the characteristics of a "container," i.e., it is a space that groups other objects.



Explanation of Component: "View select msg"

It was specified a Container. (Why?)

It was proposed the Container "View select msg". (Why?)

A restriction named "NavigationFacetRequired" was violated. **(What?)**

Of a action list. (Which?)

It was selected the "AddNavigationFacet" action that proposed adding the type of Navigation to the "Select msg" component.

Explanation Details

It is known that:

If

- The Component of the Extension is equals to Container
- It has a Binary Relationship with the Component "Select msg"
- The Component "Select msg" has a Control facet

So, A Constraint named "A Navigation facet is Required" is violate.

And in this specific situation happened that:

- The Component of the Extension is equals to Container
- It has a Binary Relationship with the Component "Select msg"
- The Component "Select msg" has a Control facet

So, A Constraint named "A Navigation facet is Required" was violated

Figure 5. Designed interface with example of violated constraint

In the example, the main container—"mailbox"—contains all of the other interface objects within itself. The "List of Messages" object groups the components Sender, Subject, and Date. Each object may have more than one facet, for example: "Select Message" has the Navigation and Control facets.

In figure 5, we show an example of a component that was inserted onto the interface and at that moment violated a constraint. We will relate how this happened and how the system made the repair.

Example of a Constraint Violated

After the interface had been designed, the "View Select Msg" component was chosen to obtain the explanation of how and why it was inserted onto the interface. The upper frame of Figure 5 shows the abstract level of the explanation in the form of sentences.

The first sentence "A component of the Control facet was specified (Why?)" represents the *Specify* subtask that specified the necessity of having an object with the Control facet in the interface. There is also the "Why?" link, whereby it is possible to view the response to the question generated by the *Specify* subtask and to discover why a control component was specified. The second sentence "The SelectMessage (Why?) component was proposed which violated a NavigationFacetRequired constraint (What?)" represents the subtasks *propose and verify*. In this example, the "what?" link was expanded. Its detail is shown in the figure below, where the violated constraint is described. The constraint states: if the task has a control facet, and if the task is of an action list and if the task possesses a binary relationship with the previous task and if the previous task closed the container, then the object to be generated by this task must have a Navigation facet.

Abstract Interface Generation for Crime Simulation Tutorial System

We applied the KBS_{HCI} to generate the interface of a module from the ExpertCop system. This is a geosimulator which aims to support education through the simulation of an urban region, where various crimes occur during a interval of time. It allows the user to perform a dynamic allocation of police resources that will be used to prevent the crimes from occurring. We selected the module responsible for allocating the teams of police officers in a certain area of the map.

Instantiation of the Models

The Interface Specialist elaborated the tasks model as shown in Figure 6.

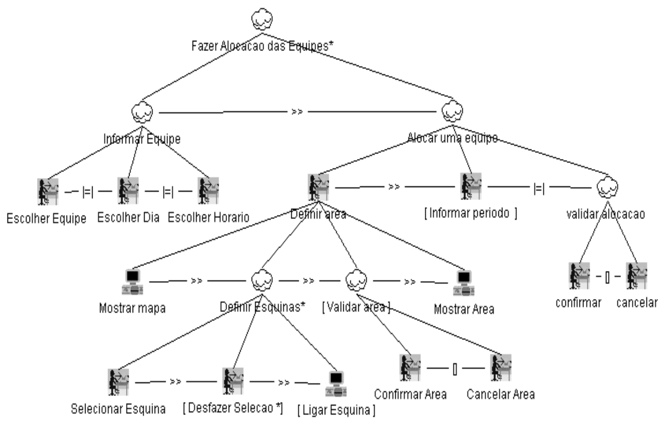


Figure 6. ExpertCop Tasks Model.

Afterwards, this tasks model was instanced on the ontology of the USIXML standard [7] through the Protégé tool [4] so that the KBS's Java objects, referring to the domain, could be generated.

In this model, the user chooses a team of police officers to be allocated to a specific area to be defined. Upon defining this area, the user selects the street corners on the map with the possibility of undoing the lines or canceling the operation before confirming. If the process of defining the area is confirmed, it informs the period and confirms the allocation.

Designed Interface

After the execution of the KBS, the abstract interface illustrated in Figure 7 was generated. It is made up of 19 abstract objects, as follows: Team Allocation, Input Team, Select Team, Select Day, Select Time, Team Allocation, Define Area, View Msg, View Corner, Undo, Link Corner, Confirm Area, Cancel Area, Input Interval, Begin Time, End Time, Confirm, Confirm, Cancel.

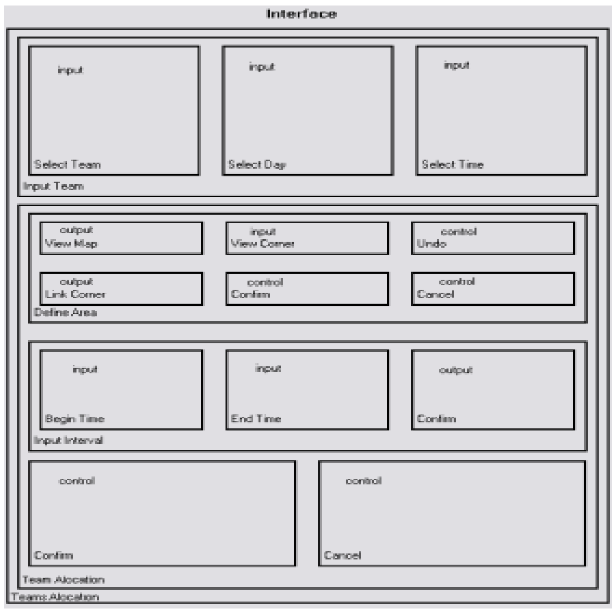


Figure 7. Abstract interface designed for the ExpertCop System.

Revising the Interface Generated

After the generation of the interface in this last example, presented in Figure 7, the designer asked "why?" regarding the insertion of the abstract object "Cancel Area" that is found within the object "Define Area" and what purpose this object serves. The KBS provided the following explanations: (i) the first question was identified by the KBS through the execution of a specification rule that allowed for the identification of the task associated with this object of why it was proposed. It clarified that there was a specification rule stating: If the task is "Interactive" and has the action of "Modifying" an element, then create an abstract object with an "Enter" facet. In this system's tasks model, the task "Cancel Area" met these conditions of the specification, therefore the creation of the "Cancel Area" object was proposed; (ii) What is this object for? "Cancel Area" serves to undo a set of selected corners, the explanation corresponds to the objective of the "Cancel Area" task associated with the abstract object. This objective is described in the tasks model. Therefore, it was observed that the "Cancel Area" task would be equivalent the repetitive execution of the task "Undo Selection". Another problem identified in this interface was the existence of objects with similar meanings (Confirm Area, Cancel Area, Confirm, and Cancel), since the non-necessity of the object "Cancel Area" was perceived started to reflect on the necessity of the object "Confirm Area", it was confusing for the user. In this way, the designer decided to modify the tasks model by removing the tasks "Confirm Area" and "Cancel Area". This directly influenced the abstract interface by diminishing two abstract objects: "Confirm Area" and "Cancel Area."

The designer resolved this situation by modifying the tasks model, but he/she could have opted to modify some rule of the knowledge base. For this purpose, the KBS_{HCI} offers options through the buttons *Specifications*, *Preferences*, *Constraints* and *Fixes*.

DISCUSSION

Some works that approach interface generation follow the line of thought that the user must participate in the process together with the designer. Our proposal allows this idea to be implemented and facilitates the user's understanding of how interface was generated, because the KBS_{HCI} has the resource of explaining what was designed. The possibility of modifying the knowledge base can also be useful for adjustments to be made according to the user's needs.

We expect the specialist's knowledge built into the system (knowledge base) to be increasingly perfected; as the system continues being utilized, the designer keeps adjusting the knowledge in a way such that the KBS_{HCI} will reflect the knowledge of an experienced designer.

We believe that the KBS's interactivity resource could be quite useful for the process of educating the interface designer's, that is, the KBS_{HCI} will be able to function as a tutorial to assist beginner designers.

We verified that the proposal presented in this article is particularly useful in situations where the context of interactive software use that is being developed is not very well defined. Web systems are a good example of this. Another example where this occurs is in the development of software in the environment of academic research. We were able to conduct an experiment on this, since we lived the exploratory characteristics of this context during the process of generating the ExpertCop interface. The problem of allocation of police officers and simulation of criminal activity was innovative and was not very well defined. Consequently, one could not have a very accurate idea of how the user's tasks would be. In this way, the first interfaces generated by the KBS_{HCI} proved to fall well short of what was desired. This served as a subsidy to the designer so that he/she could revise the task model and data model and then generate new interfaces based on the redefined models. This process was repeated several times and evidenced the cyclical character of the design and primarily the importance of the KBS_{HCI}'s being explicative. The explanations were geared toward the revalidations of the models.

CONCLUSION

In this article, we apply resources of Artificial Intelligence in order to automate and optimize the process of user interface generation. Therefore, we propose the construction of a KBS that represents the interface specialist's knowledge, so that the user can have an explanation of the result produced. For this, we represent knowledge according to UPML architecture and reutilize the patterns of interaction. We adopt the USIXML standard to represent the HCI models and adopt the Cameleon Framework [1] for generating the interface, which defines stages for the development of interfaces, one of which being the concept of the abstract interface. We apply the KBS to generate the abstract user interface for a system of email control and for a tutorial system that simulates criminal activity. We intend to apply the KBS to generate the concrete interface of these systems.

REFERENCES

1. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15,3 (2003), 289-308.
2. Fensel, D. et al., The Unified Problem-Solving Method Development Language UPML. Knowledge and Information Systems, An International Journal, 5, 83-127, 2003.
3. Furtado E., Mise en Oeuvre d'une Méthode de Conception d'Interfaces Adaptatives pour des Systèmes de Supervision à partir des Specifications Conceptuelles, Doctorate in Information Science, Université d'Aix-Marseille III, France, 1997.
4. Gennari, J.H. et all, The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. SMI Report Number: SMI-2002-0943, Stanford, 2002.
5. Irandoust, H., Attitudes for Achieving User Acceptance: Explaining, Arguing, Critiquing. Defence Research and Development, Valcartier, Canada, 2001.
6. Kay, J. User Modeling for Adaptation. User Interfaces for All – Concepts, Methods and Tools, LEA Publishers. London. 271-294, 2001.
7. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M. and Trevisan, D. USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. Proc. of the AVI'2004 Gallipoli, 2004.
8. Limbourg, Q., Vanderdonckt, J. Addressing the Mapping Problem in User Interface Designs with UsiXML. Proc. TAMODIA, Prague, 2004.
9. Motta, E. Reusable Components for Knowledge Modeling, Ph.D. Thesis, Knowledge Media Institute, The Open University, UK, 1997.
10. Pinheiro, V. et all, A Unified Architecture to Develop Interactive Knowledge Based Systems. In Proceedings of 17th Brazilian Symposium of Artificial Intelligence (SBIA 2004), São Luís, MA, Brazil, October 2004.
11. Savidis, A., Akoumianakis, D., Stephanidis, C., The Unified User Interface Design Method. User Interfaces for All – Concepts, Methods and Tools, LEA Publishers. London. 417-440, 2001.
12. Schreiber et al., Knowledge Engineering and Management: The CommonKADS Methodology. The MIT Press. Cambridge, MA, 2000.
13. Suárez, P. Gestão do Conhecimento no Processo de Concepção de IHC e uma Nova Abordagem para a Obtenção de uma Especificação Conceitual da Interação, Master's Thesis. UFCG, 2004.
14. Thevenin, D., Coutaz, J. and Calvary, G. A Reference Framework for the Development of Plastic User Interfaces. Cap.3 de Multiple User Interfaces (2004), 29-51.
15. Vanderdonckt, J. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection, 1993.
16. http://giove.cnuce.cnr.it/teresa.html