Patry G. & Girard P. From Adaptable Interfaces to Model-Based Interface Development: The GIPSE Project. ERCIM Workshop on User Interfaces for All (UI4ALL'97), Ed. N. Carbonell, Pub. INRIA Lorraine, Obernai, France, 3-4 november 1997, pp. 127-133.

From Adaptable Interfaces to Model-Based Interface Development: The GIPSE Project

G. Patry, P. Girard

Laboratoire d'Informatique Scientifique et Industrielle
École Nationale Supérieure de Mécanique et d'Aérotechnique
Site du Futuroscope - BP 109 - 86960 Futuroscope Cedex - France
email: {patry, girard}@ensma.fr

Abstract: Developping specialized versions of applications is usually made by computer experts. The GIPSE system has been designed to allow end-users to specialise their application by creating new functions. As it allows this task by Programming by Demonstration techniques, without any use of textual programming language, GIPSE can be used by non computer literates.

Starting from GIPSE, we propose to rely two approaches which, at a first glance, seem very different: model-based interface development and user-adaptable applications. We argue that these two approaches may converge, bringing user enpowerment to formal approaches, and conversely, bringing formal insurance to end-user developments.

Introduction

With the advent of personal computing, computer programming trend is to design applications for the widest possible user groups. They try to address needs of a large population with very different requirements. The same drawing package may be intended to be used for designing the outline of a mechanical structure, the appearance of a building, or the disposal of a flat. To accommodate such different purposes, these applications usually provide a large number of low level functions that can be combined by end users to fulfil their needs. This allows multiple uses, but is also very tedious when the same kind of operations needs to be done over and over.

The removal of these drawbacks may be done by using the Programming by Demonstration (PbD) paradigm that can be summarized as follow: while the user uses a PbD system, this system generalizes the interactive session (the example) to compute a program. For example, achieving the same task on two objects may be generalized on a whole set of objects. Many systems have been developped over this idea [Cypher 1993]. In the CAD area, our laboratory has been involved in major projects that have led to powerful PbD systems [Pierra, Potier, & Girard 1996].

At the same time, interface development drastically changed. Theories and models have been developped. Computer-Aided Design of User Interface (CADUI) became a great center of interest. Model-Based systems [Puerta 1997] seem to be now one of the best approaches to build powerful interactive applications: from one or different models (user-task model, dialogue model, presentation model) model-based systems construct powerful interfaces that may be connected to functional cores in order to build the whole interactive application. The major strength of these systems is the ability to check the formal description of the models.

Model-based systems allow a user-centered approach of interface design. Nevertheless, two major drawbacks may be pointed out: (1) they require that their users learn abstract models and languages, and (2) they only work in one way, from models to the generated application. When generated, the system has to be tested, and any modification requires a modification of the models to generate a new system.

The goal of this position paper is to propose a solution to these problems. Using PbD techniques, we are now able to design a system that allows much more than a simple adaptability of interfaces: it allows a real interactive definition of new applications. And because of the underlying models we use, it permits reverse engineering from the interactive

application -on which no formal evaluation may be done- to the model level -which allows formal evaluation.

This paper is organized as follows: the first section gives a short description of the underlying models of our approach, called the the H^4 Model. Section two describes the GIPSE project, and section 3 relates our approach to other works.

1. The H^4 Model

Presented by L. Guittet [Guittet 1995], the H^4 model has been designed to be a suitable model for CAD systems. These systems are well-known to possess several characteristics : using the taxonomy proposed in [Pierra 1995], they can be described as supporting multi-object and structured tasks, reflecting a goal/subgoal hierarchy [Norman 1986]. Their conceptual objects are highly structured (the user may access several levels of objects, such as sets of entities, or only one component of a given set), and relational (the state of one object may depend on the state of another domain's object).

The H^4 model belongs to the ARCH family [Bass, et al. 1992]. It's dialogue controller is composed of a hierarchic set of agents, named interactors, relatively to the theory of Interactors ([Duke & Harrison 1993 ; Faconti & Paterno' 1990 ; Paternó & Faconti 1994]). Each of these corresponds to a task level, and is defined by the nature of the information it accepts as input. They produce output informations that are the result of the task. Common interactors in drawing applications include *designation* (picking graphical entities), *information* (giving information such as center or extremity of entities), *calculus* (projection, distance), and *creation*. Communications between interactors are managed by a **Monitor**. Interactors may accept information from any lower interactor in the hierarchy, via the Monitor. The communication unit is a typed element, named **Token**, that may either be commands (order to initiate actions) or data (parameters of actions). Tokens are transmitted by the monitor to the next interactor waiting for such a token, conforming with the interactor hierarchy.

The H^4 model describes the dynamic behaviour of an application as an advanced automaton. Each Interactor has a current state and accepts or refuses tokens that are proposed by the Monitor. Acceptation fires a transition and allows the storing of the token in a register. Transitions may have actions as attributes. Such actions are called when their associated transition is triggered, using the register content 's as parameters.

The H^4 model has been implemented in an application generator that allows the complete generation of interactive systems from formal descriptions of both the user-task model and the dialogue model. It is, in fact, a model-based system.

2. The GIPSE System

GIPSE (**G**enerateur d'**I**nterface par **P**rogrammation **S**ur **E**xemple, for Programming by Example Interface Generator) is a major enhancement of the above generator. Moreover the above functionalities, it is a run-time environment allowing the extension of the application's kernel by the way of adding new functions, as well as new kinds of entities that can be manipulated by the system as native entities. The motivation behind our system is to allow end users to customise their generic applications into task-specific ones. Some applications already allow this, by the way of simple macro recorder facilities (Microsoft Word™ or Excel™ are good examples here), but most of complex tasks, involving loops or conditional statements, must be expressed in an interpreted language that must be already masterised by users. In contrast, the goal of GIPSE is to create, implement and integrate whole new functionnalities to applications without actually coding them.

The goals of GIPSE can be stated as following:

- GIPSE should allow the extension of existing applications.
- The intended targets are either developers who want to specialise their generic system, either CHI experts who want to design interactive applications from some functional core, or end-users who want to extend/customise their application. None should have to be a computer specialist to do this.
- Dialogue extension should be specified by means of Programmation by Demonstration (PbD) rather than conventional textual programming.

As it has been underlined, GIPSE implements the H^4 model. It is mainly structured in two components. The first one is responsible for the dynamic behaviour of the application, while the other is responsible for the creation and execution of actions recorded by the user.

2.1. Dynamic behaviour

In order to allow indifferent use of regular (hard coded) and extended (macro) actions by the user, the dialogue controller responsible for the dynamic behaviour of the application has itself to be dynamic and extensible. As it is unknown at compile time which actions will (or will not) be included in the application, and the exact stucturation of the actions into interactors, the dialogue controller has to allow adjunction and removal of dialogue threads (that is, successions of command and parameters leading to one action call). The GIPSE system is based on a dynamic version of the H^4 model. Monitor and Interactors are loaded/created at run time via a textual description of their structure and representation. They are maintained in the system as a set of interpreted data rather than a set of functional calls.

The dynamic nature of the controller force the existence of a general dispatcher, responsible for the actual call of the function when requested by an interactor. Interactors call the dispatcher by the way of a message identifying the action to be performed and the parameters. The dispatcher sends back the response of the action with a report indicating success or failure. From the Interactor's point of view, there is no difference between regular actions and extended ones, both being described as triplets (action reference, parameters, return) and sent to the dispatcher.

The effort made for differentiating macro-actions from hard-wired ones is put on the dispatcher. Separation is done by the way of a database (cf Fig. 1) created during the initialisation of the application: during application's launch, regular actions send their description, while a macro component gives the description for all recorded macros. Later, when the dispatcher gets a message referencing an hard-wired action, it calls the corresponding function in the domain adaptator component, passing the arguments the function needs. When the action is a macro, a dialogue is established between the dispatcher and the Macro-recorder, resulting in the sequential execution of actions given by the latter to the former.

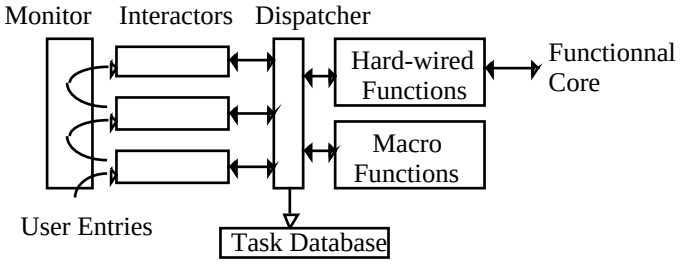

Fig. 1 : the GIPSE System architecture

Patry G. & Girard P. From Adaptable Interfaces to Model-Based Interface Development: The GIPSE Project. ERCIM Workshop on User Interfaces for All (UI4ALL'97), Ed. N. Carbonell, Pub. INRIA Lorraine, Obernai, France, 3-4 november 1997, pp. 127-133.

2.2. Adding new functionalities

Our experience with PbD in CAD allowed us to easily include a powerful PbD system in GIPSE. Once any PbD function is recorded, it can be integrated seamlessly in the application. This is done by an *Integrate* command, that prompts (1) for the action to be integrated, (2) for the nature of the action, in terms of task level (thereby identifying an interactor), and (3) for the representation associated with the action. This representation is composed of a label to be displayed on a menu button (which position depends on the chosen interactor), and a help line. As an alternative to (2) it is possible to create a new interactor. In this case, the user is prompted for the name of the new component and its position in the hierarchy. From this point, the new component(s) will behave as if it was created at initialisation time.

Similarly, but with the opposite effect, it is possible to remove any action from any interactor. This may be necessary if the goal of the user is not to expand his/her application, but to specialize it. In this case, removing some functions using no-more-to-be-used parameters or performing no-more-to-be-used actins is essential. This is done with a *Disintegrate* command that prompts either for the action or the command to be removed. The removal is done at the dialogue controller level and actions still exist at their own level (macro for an PBD-Macro, Domain Adaptator level for hard-wired ones). Any command/action uniquely associated with the action/command to be removed will also be removed.

It should be noticed that any action can be (Dis)integrated from the controller. This operation is done at the dialogue controller level, whereas identification of the nature of an action is done by the dispatcher.

2.3. Adding new kinds of objects

The couple made by the dynamic implementation of H^4 model and a macro-recorder allows the adjunction and removal of functionalities to a preprogrammed kernel. Using extended actions to define new kinds of interaction objects makes up for the second step towards a complete CADUI. Such a goal can be achieved by two steps. First it must be possible to create new categories of model objects, new types. Second, these types, to be useful, should also be usable as parameters for new functions.

Any PBD-program creating objects in the application model can be used as the basis for the definition of a new type. The new kind of object is defined as a named group of every object created as a result of the execution of the PBD program. This PBD program can be considered as the type definition, while instances of this type are the result of this action with effective associated parameters. Defining new kinds of objects also involves the automated definition of some functionalities that are associated with this type, such as designation or geometric transformations. They can be inherited from the components of the object: translation of such an object, for example, is a set iteration of the translation on all its components.

New object typing implies the use in the application model of a special set that we call a extended set (x-set for short). Such a set, once created, has a unmodifiable name. Instances of the type are multiples x-set using the same name. Any PBD program resulting in an instance of a new type (that is, used as type definition) begins with creating a x-set with the type as name. During its execution, it add any created object to this set. These objects still exist on their own (independently from the typed object), and may even be part of a user defined set.

The use of new types as function parameters is straightforward. As the parameters are graphically designed, the developer uses the designation function to provide an object of the type of interest. The Macro recorder will then record an x-set *along with the name of that particular x-set*, as the formal parameter of the action being recorded. From a (macro's) developer's point of view, there is no difference between the use of a standard type or a new

one in creation of a new action. In fact, if the developer is not the end user, the latter may not know that there is different kind of objects and actions. If the commands of the macro system are removed from the application, he/she may not even know that there is a macro system.

3. Related works

GIPSE is a development and run-time environment based on programming by demonstration technique. The development part is expressed in terms of actions and action-tasks levels. In the following, we summarise the main differences between our work and previous works in the related domains.

GIPSE may be assimilated to task-oriented systems. While the creation of new functionalities is straightforward, it does not eliminate the needs for analysing the intent of these actions: how will end-users use them? What tasks are to be used as subtasks? What is the task level for this action? … There are the questions the developer must answer while extending his application.

The absence of more advanced descriptions, such as presentation model or user model, as well as the absence of advanced tools (design critics for example), would make GIPSE a low-end Model Based system [Szekely 1996 ; Wilson & Jonhson 1996] compared to existing ones such as Janus [Balzert 1995 ; Balzert, et al. 1996], Humanoid [Szekely, et al. 1995], Trident [Bodart, et al. 1995] …

Nevertheless, the goal and method of GIPSE differ from the ones of these systems. GIPSE has been designed as a specialisation/extension system from an existing, programmed, kernel, whereas these systems have been designed with the goal of easing the whole development process. They generate the code of an application (MasterMind [Szekely et al. 1995]) or a file description that can be used by existing UIMS generators (FUSE [Lonczewski & Schreiber 1996]), via a formal description of various aspects of the application. In contrast, GIPSE does not need a formal textual description of the task to be added to the application nor it does generate compilable code. GIPSE allows the modification, during execution of the target application, of the existing dialogue structure and components. These modifications are then stored as description of the new dialogue, and restored the next time the application is run. GIPSE is interpreting the task model of the application at run time, much like ITS [Wiecha, Bennet, & al 1989 ; Wiecha, et al. 1990] and Humanoid do for their model.

What is completely new in GIPSE is the dual access between resulting applications and underlying models: we can generate applications from models, and we can see, and best, evaluate, the models that are generated by customized applications as well as entirely new applications coming from interactive design.

Conclusion

In this paper, we have described a system using Programming by Demonstration to allow end-users to customize their applications. More, without any programming knowledge, end-users are able to create powerful applications from existing ones.

Because of the undelying models of our system, we think that it is possible to rely two approaches, which seem, at a first glance, very different: the model-based interface development approach and user-adaptable applications. Despite their obvious difference, we argue that these two approaches may converge, bringing user enpowerment to formal approaches, and conversely, bringing formal insurance to end-user developments.

Bibliography

[Balzert 1995] Balzert H. From OOA to GUI : The JANUS-System. *InterAct95,* 1995. p. 319-324.

[Balzert et al. 1996] Balzert H., Hofmann F., Kruschinski V., & Niemann C. The JANUS Application Development Environment-Generating more then the User Interface. *CADUI'96, Namur*, 1996. p. 183-206.

[Bass et al. 1992] Bass L., Faneuf R., Little R., Mayer N., Pellegrino B., Reed S., Seacord R., Sheppard S., & Szczur M.R. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 1992. vol. 24, n° 1, p. 32-37.

[Bodart et al. 1995] Bodart F., Hennebert A.-M., Leheureux J.-M., Provot I., B. Sacré, & Vanderdonckt J. Towards a systematic building of software Architectures : the Trident Methodological Guide. *DSV-IS'95,* 1995. p. 262-278.

[Cypher 1993] Cypher A. *Watch What I Do: Programming by Demonstration*. Cambridge, Massachusetts : The MIT Press, 1993. p. 604.

[Duke & Harrison 1993] Duke D.J. & Harrison M.D. *Abstract Interaction Objects.* Computer Graphics Forum. 1993. vol. 12,n° 3, p. 25-36.

[Faconti & Paterno' 1990] Faconti G.P. & Paterno' F. An Approach to the Formal Specification of the Components of an Interaction. *European Computer Graphics Conference and Exhibition, Montreux, Switzerland*, 3-7, September 1990. p. 481-494.

[Guittet 1995] Guittet L. *Contribution à l'Ingéniérie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO*. Thèse de Doctorat : Université de Poitiers, Poitiers, 1995.

[Lonczewski & Schreiber 1996] Lonczewski F. & Schreiber S. The FUSE-System: an integrated User Interface Design Environment,. *CADUI'96, Namur*, 5-7 Juin 1996 1996. p. pp. 37-56.

[Norman 1986] Norman D. *User Centered System Design*. Lawrence Erlbaum Associates, 1986.

[Paternó & Faconti 1994] Paternó F. & Faconti G.P. *A semantics-based approach for the design and implementation of interaction objects.* Computer Graphics Forum. 1994. vol. 13,n° 3, p. 195-204.

[Pierra 1995] Pierra G. Towards a taxonomy for interactive graphics systems. *Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Bonas*, June 7-9 1995. p. 362-370.

[Pierra, Potier, & Girard 1996] Pierra G., Potier J.-C., & Girard P. *The EBP system : Example Based Programming for Parametric Design*. Modelling and Graphics in Science and Technology, Springer-Verlag, 1996.

[Puerta 1997] Puerta A.R. *A Model-Based Interface Development Environment.* IEEE Software. July-August 1997. vol. 14,n° 4, p. 40-47.

[Szekely 1996] Szekely P. Retrospective and challenge for Model Based Interface Development. *CADUI'96, Namur*, 1996. p. xxi-xliv.

[Szekely et al. 1995] Szekely P., Sukaviriya P., Castells P., Muthukumarasamy J., & E. Salcher. Declarative interface models for user interface construction tools : the MASTERMIND approach. *ECHCI95,* 1995. p. 120-150.

[Wiecha, Bennet, & al 1989] Wiecha C., Bennet W., & al e. Generating Higly Interactive User Interfaces. *CHI'89, Austin, USA*, 30 April-4 May 1989 1989. p. 277-282.

[Wiecha et al. 1990] Wiecha C., Bennet W., Boies S., Gould J., & Greene S. *ITS: a tool for rapidly developing interactive applications.* ACM Transactions on Information Systems. July 1990. vol. 8,n° 3, p. 204-236.

[Wilson & Jonhson 1996] Wilson S. & Jonhson P. Bridging the Generation Gap : From Task to User Interface Designs. *CADUI'96, Namur*, 1996. p. 77-94.