JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, no. 3, March-April 2004

Patterns and Objects for User Interface Construction

Alecia Eleonora Acosta and Nancy Zambrano, Central University of Venezuela, Venezuela

Abstract

Nowadays, computers play a very important role, that is to say, as a communication tool between people. This introduces the interface between human and machines as a key player, therefore the importance of these interfaces. The existing software development processes recognize this importance but do not establish precise guidelines for the construction of the user interface as an activity within the system life cycle. This article describes a method for constructing user interfaces based upon interaction patterns. This method can be incorporated to an object-oriented software development process which fulfills certain characteristics. Interaction patterns describe interface design solutions favoring the development of a user interface prototype.

1 INTRODUCTION

This paper presents a method for constructing user interfaces based upon interaction patterns. This method is intended to reduce time spent in developing user interface. This method can be introduced into an object-oriented software development process that embodies the remaining stages of software life cycle, which benefits the integration of the areas of Software Engineering and Human-Computer Interaction.

The method is engraved within the context of software development based upon prototyping and supported by the reuse of components: interaction patterns. An interaction pattern captures essential information relative to a recurrent problem, shows a successful solution and describes the context of this solution. A pattern is written to communicate the experience and to allow designers to reuse it.

Among other things, these interaction patterns are used to generalize a solution regarding interaction designs, to record knowledge or experience and to reuse it, and to facilitate communication among people involved in software development, mainly from different areas (software engineer, interface experts, problem domain expert, user, etc.). Section 2 of this article deals with the essential aspects related to interaction patterns. Section 3 describes the user interface prototype construction method, where interactions

Cite this article as follows: Alecia E. Acosta, Nancy Zambrano: “Patterns and Objects for User Interface Construction”, in *Journal of Object Technology*, vol. 3, no. 3, March-April 2004, pp. 75-90. http://www.jot.fm/issues/issue_2004_03/article1

patterns are used; and the third section will explain the integration of this method within a software development process.

2 GENERAL ASPECTS OF INTERACTION PATTERNS

Knowledge reuse is a technique used by experts of any area; when solving a problem, they make use of their own experience - as well as of others- to verify whether they have successfully solved a similar problem and then apply that solution with the required modifications (if any).

Generally speaking, experienced designers almost never solve every new problem since the beginning, they reuse solutions that have been successfully applied before and that are part of their expertise. This expert knowledge is translated into components that are going to be used by others to solve similar problems. In the area of Software Engineering a way of recording this knowledge is by using software design patterns [Gamma97] and this practice allows designers to reduce development costs and time. This same technique is applied to Human-Computer Interaction, where the experience of interface design is recorded in interface design patterns – or interaction patterns-which are reusable components that describe a successful solution within a given context to a recurrent problem regarding user interface design.

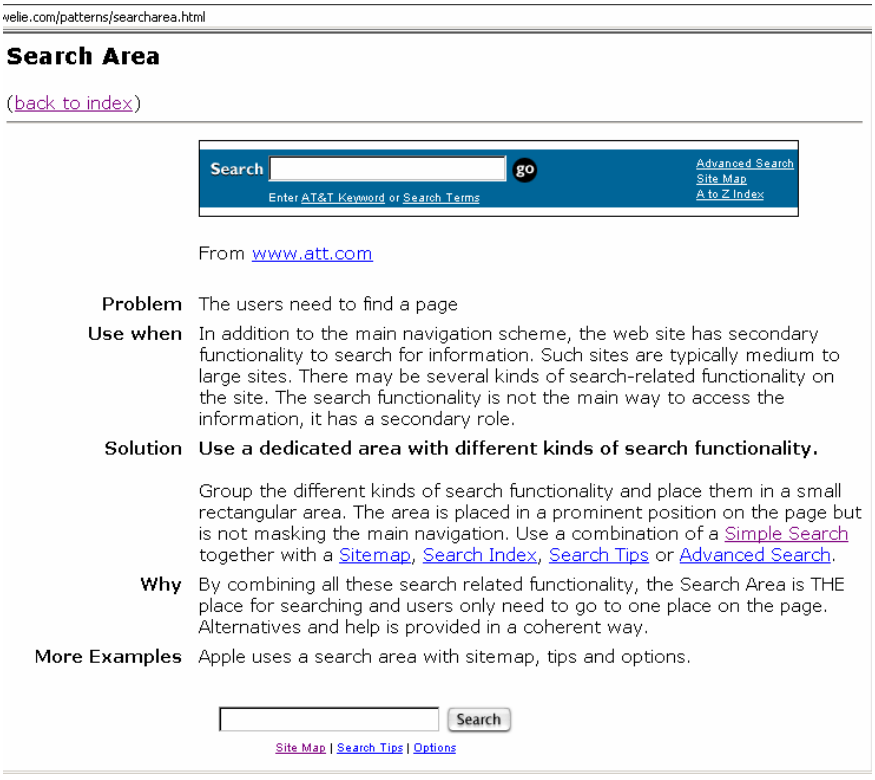
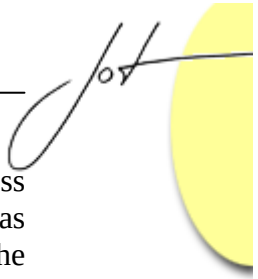


Figure 1. An interaction pattern, Author: Marjtin van Welie
URL: http://www.welie.com/patrones/searcharea.html [Welie03]



Software design patterns and interaction patterns, even though they express solutions, are two very different concepts. Interaction patterns are closer, as far as objectives are concerned, to the original pattern concept developed by Alexander in the field of architecture [Alexander77], in the sense that they are oriented towards the creation of comfortable environment for end users.

An interaction pattern works as a communication tool between people working in a software development team (generally a multidisciplinary team made up by specialist in user interface design, software domain specialists and users, etc.). Figure 1 shows a part of an interaction pattern from Martijn van Welie's collection, available in http://www.welie.com/patterns [Welie03].

Pattern Structure

In literature we can find several ways to write an interaction pattern, all of them having basic components in common even thought names can change, for example [Mahemoff98], [Tidwell03], [Usability03]. Currently there is no standard structure for writing an interaction pattern. In this paper we propose a pattern structure using a metapattern, this is to say, a pattern that allows describing patterns. Figure 2 represents this metapattern.

Name, author Classification and range.	Name: states the central idea. Author: the name of the creator of the pattern. Classification: states the pattern type. Range: states the qualification of the pattern.
Problem	Describes the problem to be solved from the user point of view.
Solution	Describes, in a descriptive and graphic form, the solution of the problem.
Context	Presents the conditions under which this pattern is used.
Forces	Points out the conflicts that could restrain the solution.
Usability	Describes the impact of the use of the pattern from the usability point of view.
Consequences	Describes the result of applying the pattern.
Examples/ Counterexamples	Shows examples and counterexamples of the proposed solution.
Related Patterns	Enumerates other patterns related to this pattern.

Figure 2. Metapattern. Components of an interaction pattern

This metapattern is aimed at defining a notation that can be easily understood by the complete development team. It is not always necessary to describe each component for all patterns, with the exception of obligatory ones as: name, problem, solution, context, and usability.

Figure 3 shows an interaction pattern from van Welie's collection [Welie03], expressed using the metapattern of Figure 2.

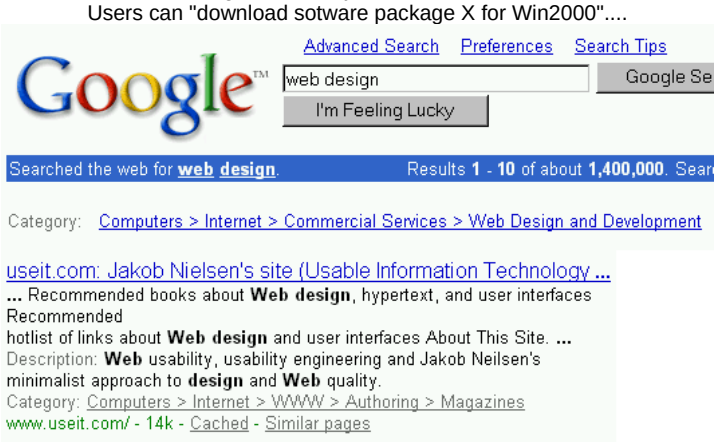
Name	Simple Search
Author	Martijn van Welie
Problem	The users need to find an item or specific information.
Usability	Action minimization
Context	Any web site that already has primary navigation. User may want to search for an item in a category. User might want to further specify a query.
Forces	By using this setup the whole search becomes a sentence that reads like the search query.
Solution	**Offer a search** * The search interface Offer search functionality consisting of a search label, a keyword field, a filter if applicable and a "go" button. Pressing the return key has the same function as selecting the "go" button. Also provide Search Tips and examples in a seperate page. A link to that page is placed next to the search functionality. The edit box for the search term is large enough to accomodate 3 typical user queries (typically around 20 characters). If the number of filters is more than 2, use a combobox for filterselection, otherwise a radiobutton. Search -- editbox -- for/in -- filter -- Go button or just... -- editbox -- Go button * Presenting search results The search results are presented on a new page with a clear label containing at least "Searchresults" or similar. The search function is repeated in the top-part of the page with the entered keywords, so that the users know what the keywords were. The number of "hits" is reported and the list of search results is organized; sorted or rated with the best matches at the top. When there are more than 10 results use a Paging mechanism. Each search result shows a link to the item itself and a snibbet of text to explain the item. Preferably that would a summary or abstract but can also be the first lines of text of the resulting item. The structure of a "result" typically shows: 1. Page Title 2. Description 3. Categorization 4. URL, Size, Date * Keyword matching If more than one search term is used the search engine must handle them as follows: if no special separators are used (not including the space), the search is interpreted as an OR function, the results that match both terms are listed first. If special separators are used the search engine must be able to handle more than one convention. For example, sometimes the "AND/OR" separators are used but using a "+" or a "-", include and exclude, must also be handled correctly. The engine must also be able to handle spelling mistakes of at least one character.
Example	From www.tucows.com  In this example from tucows, the designers actually were able to make the search read like a sentence. Users can "download sotware package X for Win2000"....  This example from Google shows how each result is presented
Related Patterns	Consider the Paging or List Browser patterns for navigating through a large set of results. However, users will not check more than one or two pages of search results; instead they will edit the search terms again.

Figure 3. Simple Search Pattern expressed by the Metapattern

Taxonomy of interaction patterns

Taxonomy permits classifying patterns as to facilitate its creation and usage. Patterns can be classified based on certain characteristics; some authors have presented different classifications [Coram03] [Mahemoff98].

This article defines the following interaction pattern types and their corresponding symbols:

-  Domain Patterns: describe aspects of the interface that are related to a specific software domain (for example, Electronic Commerce, Virtual Courses, etc.).
-  System Patterns: capture aspects of user interface concerning a specific application, for example usability attributes that must be guaranteed, purpose of the system, etc.
-  Task Patterns: describe interface aspects referenced to the interaction with software functionalities.
-  Complex Element Patterns: state interaction styles between user and the interface made up by interface elements.
-  Single Element Patterns: describe atomic elements of user interface.
-  User Patterns: describe user profiles relevant for the application.

Interaction pattern organization

Isolated patterns are not very important. The link between patterns is as important as the pattern itself, and its organization is the basis for defining collection of patterns. It is possible to have a pattern with a higher level than other (for example, domain and simple element patterns) and generate a hierarchy among them. Also, it is possible to relate patterns from the same level. Last but not least, it is possible to establish a multiple link structure where patterns reference other patterns. The notion of a pattern implies the creation and/or manipulation of pattern groups related among each others, describing a design of a complex interactive system.

After determining a structure to describe a pattern and defining interaction pattern taxonomy, it is necessary to establish how these patterns will be organized within a collection, as to facilitate its creation, reuse and maintenance.

Patterns will be organized in a collection depending on how they will be used on the practice; this means that the final goal of any organization of patterns is to support an iterative design process where links among patterns would lead the designer towards the next logical step in the process of user interface construction.

One of the workshop INTERACT99 results, presented on [Borchers99], is the usefulness of interface pattern languages, having established that the goal of pattern language is to share successful interaction design solutions among professionals and to supply a common language available to any one involved in the analysis, design,

development, evaluation and use of interactive systems. There are also other experiences on the definition of pattern languages proposed by [Bayle97], [Casaday97], [Erickson01] and [Coram03].

An interaction pattern language is a collection of structured patterns which serves designers as guidelines for constructing a user interface. A pattern at one level can reference a great number of other patterns located at a lower level in order to complete the solution. When a pattern references other pattern it means that the first needs the solution described by the second to complete the solution. The patterns so connected provide an informal grammar for design: a pattern language in a certain domain.

The next definition of a pattern language, using the metapattern presented in Figure 2, is based on the definition provided by [Borchers01]:

1. A pattern language is a directed acyclic graph *PL*= (P, R) with nodes P = {P_1,..,P_n} and edges R = {R_1,…R_m}.
2. Each node $P \in$ P represents a *pattern*.
3. For two patterns *P* and *Q* $\in$ P it is said that *P* reference *Q* if and only if there is a directed edge R $\in$ R leading from P to Q.
4. The set of edges coming from a pattern P $\in$ P is called its *references* and the set of edges coming to a pattern is called its *environment*.
5. Each pattern P $\in$ P is a n-uple:
   P = (n, a, c, co, p, s, ctx, f_1,…f_n, e_1,..e_j, ce_1,…ce_m, p_1,…p_n) where:

n	name	*a*	author	*c*	classification
co	confiability	*p*	problem	*s*	solution
ctx	context	f_1,..,f_i	forces	*u*	usability
con	consequences	e_1,..,e_j	examples	ce_1,..,ce_m	counterexamples
		p_1,..,p_n	related patterns		

On advantages of using pattern languages is that development team speaks the same language, which establishes an organizational principle that facilitates the use of interaction patterns.

Figure 4 is an abstract and simplified example of pattern language, made up from different related pattern types. It can be observed a domain pattern *D* which references a system pattern *S*, the latter at the same time references task patterns *T1* and *T2*, and the representation of task *T2* in the user interface is a menu described through a complex element pattern called *Menu*.

The user interface construction method that will be explained in the next section is based upon the creation of an Interaction Pattern Language, which together with an Interface Object Model, is the starting point for the construction of a user interface prototype.

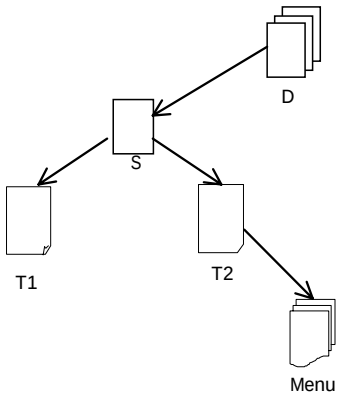


Figure 4. Example of interaction pattern language.

3 A USER INTERFACE CONSTRUCTION METHOD

User interface construction method (UIC) proposed here can be integrated to an object-oriented software development process. Moreover, this method is inscribed in the development models that incorporate prototyping since the first stages of the software life cycle; in this case we propose to construct a user interface prototype starting from interaction patterns and interface objects. The patterns are produced while applying the method or they are selected from a pattern repository.

The UIC method states a process for constructing user interface based upon interaction patterns and interface objects. The method is aimed at developing a user interface prototype that can be evaluated by the end user; in this way the prototype could evolve up to satisfy user requirements.

As it can be seen, this paper is not intended to create a new software development method, but any object-oriented process based on functionalities of the system, supported by different architectures, interactive and incremental, can be extended by the incorporation of the UIC method.

In order to construct the user interface prototype we started from known models, models defined in UML (Unified Modeling Language) [Rumbaugh99]. They are:

- Use Case Model which allows identifying functionalities of a system from the point of view of the user interaction sequences.
- Domain Object Model allows identifying objects pertaining to the domain of the system and the relationships among them.

Here it must be emphasized that the method only takes into consideration the aspects relative to user interface construction, and as a consequence it should be incorporated to a software development process which takes into account the remaining stages of software life cycle.

Artifacts used during a user interface construction

Figure 5 shows a diagram of the artifacts used and created during the user interface construction: Use Case Model, Domain Object Model, User Interface Model (UI) consisting of a Pattern Language and an Object Interface Model, and a User Interface Prototype.

It is worth reminding that a Use Case Model describes all user interaction sequences with the system that are required to carry out its functionalities, and each Use Case represents one unit of interaction between the user and the system [Sparks02] Therefore, a Use Case Model shows all possible scenarios from user interaction and represents the behavior of the system.

The Domain Object Model, as any other object oriented process, is created by identifying the objects located within the domain of the problem, its characteristics, (attributes and operations) and the existing relationships among them. This model is based on Use Case Model allowing to integrate these two visions of the system.

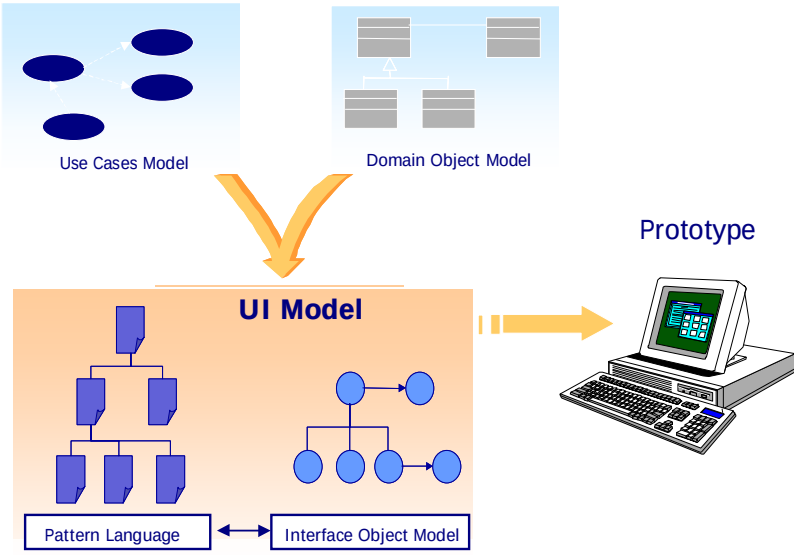


Figure 5. Artifacts of UIC method.

The User Interface Model can be built based on the two aforementioned models. This model embraces all the aspects related to the interface, that is to say, the representation of tasks that the user needs to perform while using the software. This model is formed by an Interaction Pattern Language and an Interface Model Object, described as follows:

- The Interaction Pattern Language is formed by a domain pattern which describes the domain to which the system that is being developed belongs to. A system can belong to one or more domains. Then, a domain pattern should reference a system pattern which describes its purpose, as well as usability aspects that must be taken into account for the user interface design. This system pattern will reference all resulting patterns from use cases analysis.

- The Interface Object Model is formed by objects which belong to the Domain Object Model and also are going to be represented in the user interface (not all domain objects must be represented in the interface). Moreover, Interface Object Model can include new objects as a result of interaction instruments associated to operations, defined as such in the Domain Model (for example, in an editor cutting operation is generally represented by a scissor object). In both cases the result is obtained by applying a process denominated reification, that is to say, conversing elements from one model to another model of lower abstraction level.

User Interface Model is the basis for constructing a User Interface Prototype. This is a high fidelity prototype which is drawn from a horizontal and evolutive prototyping, as to embrace the whole user interface.

User Interface Construction Process

Once we have established Use Case and Domain Object models (which are basically a part of the analysis activity of software development process) it is time to construct the User Interface Model and the User Interface Prototype, incorporating prototyping since the first stages of an interactive software life cycle.

Figure 6 shows an activity diagram reflecting user interface construction process. Next we describe the activities that take place during this process.

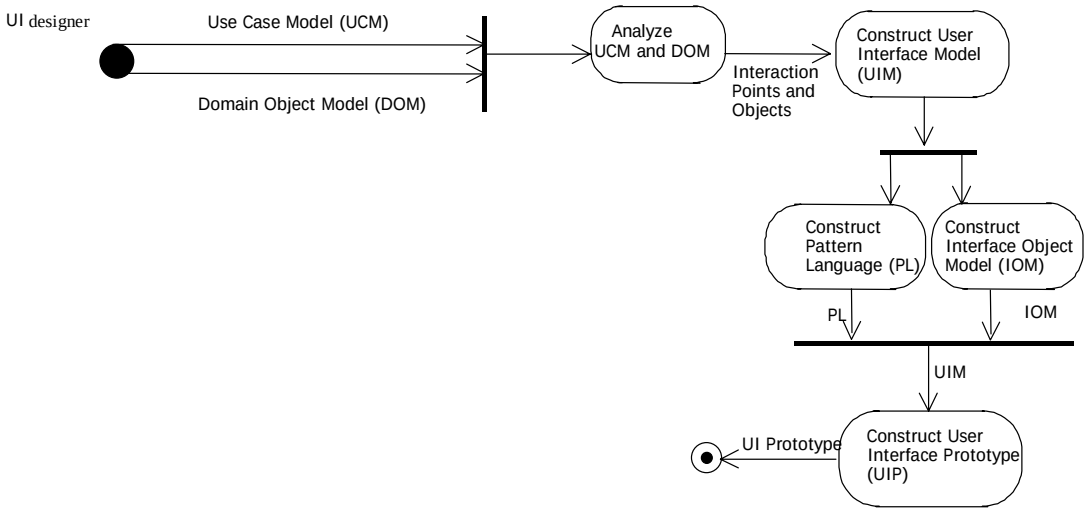


Figure 6. User interface construction process.

Constructing a User Interface Model implies constructing an Interaction Pattern Language and an Interface Object Model. It appears that these two activities are carried out one after the other but in the practice they can be carried out at the same time. The next part describes this model construction in two steps:

The **first step** is the creation of an Interaction Pattern Language from use cases analysis, this allows detecting interaction points that the user has with the software; each point (or points) is translated into a pattern that associates to it the interaction problem to

be solved, all this from the user point of view. These patterns can be completed by referencing to other interaction patterns. At the end we will have a set of related patterns that form the pattern language, which describes the user interface.

Once the interaction problem to be solved has been determined (a problem caused by one or several points of interaction), pattern construction consists in describing this *problem* as a component of the pattern and finding the appropriate *solution*. This solution can come from reusing an existing pattern or, on the other hand, creating a new pattern. If the solution is very complex, it should not be written in a single pattern, we just describe the essential elements and use other patterns to complete the solution; this will be reflected in the component: *related patterns*. In addition, another important factor to be underlined is that the activity of user interface prototype construction will depend on the way the solution is described. This means that if the solution is written in a descriptive and graphic way, the prototype construction will be easer because it will go from a low fidelity prototype to a high fidelity prototype.

Once the *solution* has been described, it is necessary to establish the *context* in which the proposed solution will be appropriated and the *forces* that will have an impact on that solution. Moreover, you must associate a mnemothecnic identification to the pattern (*name*) and an initial confidence (*range*) with a minimum value (in this case is 0).

In the case of interaction patterns it is important to describe the *usability* impact, which must reflect the usability aspects guaranteed by the pattern application. Also, the system answer should be described after the pattern is applied, which corresponds to the *consequences*, and last, we will illustrate a proposed solution using *examples* or *counterexamples* from existing software showing a good or bad usage of this pattern. Here is worth mentioning that the construction order of each pattern component is not rigorous, moreover, sometimes it would not be necessary to describe some of the components, except the compulsory ones.

The **second step** corresponds to Interface Object Model construction. In order to do so we take as basis the Domain Object Model and we apply reification of domain objects establishing its representation on user interface and creating new objects through reification of operations; in order to do this, it is necessary to define an interaction instrument [Beaudouin00]. An interaction instrument is simply a mediator between user and interface object. The number of objects from Interface Object Model could be greater than those from the Domain Object Model because objects in the Interface Object Model are drawn from reification of domain objects as well as from its operations.

Figure 7 shows an example of reification of a domain object denominated *document* and two of its operations: *cut* and *properties*. The left side of the figure shows the class Document with some of its attributes and operations. The right side shows the corresponding representation on the user interface of the object *document* as well as the interaction instruments associated to their operations. It is to be noted that on the first case a simple interface object is generated (*cut*) and the second case corresponds to a complete window (see *viewProperties*), which is described on a task pattern that belongs to a pattern language which describes the user interface of this software.

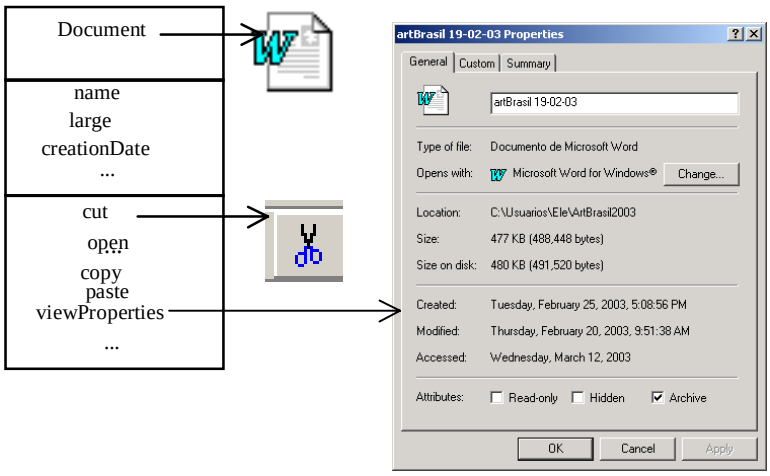


Figure 7. An example of object and operations reification.

The starting point for User Interface Prototype Construction is a User Interface Model analysis, that is, a Pattern Language and an Interface Object Model analysis which can be carried out at the same time.

Pattern Language analysis can present various cases depending on how the solution is expressed on interaction patterns. These cases are:

1. If solutions are only expressed on a descriptive way, in order to produce a prototype it is necessary to implement these solutions taking into consideration examples presented on patterns as well as objects pertaining to Interface Object Model.
2. If solutions contemplate windows layout, which are accomplished by using some tools different from those to be used to develop the end product (for example, diagram drawn by hand on paper), up to this point we have a low fidelity interface prototype. So, starting from this prototype, a high fidelity prototype will be constructed implementing solutions described on patterns and taking into account the learning obtained by the evaluation of low fidelity prototypes and incorporating objects pertaining to Interface Object Model.
3. If pattern solutions are graphically expressed, by means of an interface design made using the tools to be used for the development of end product, then we almost have ready a high fidelity prototype where we have incorporated objects pertaining to the Interface Object Model. What is left is to integrate and implement navigation between screens and the remaining details, so that this artifact could be executed and could evolve up to become the end product.

Here it must be emphasized that this work is aimed at constructing a high fidelity prototype, applying a horizontal and evolutionary prototyping, so that the prototype will evolve to become a final product.

Once the prototype has been constructed, it is necessary to continue developing the software following any object-oriented process. The next section will describe the insertion of UIC method within a Unified Process.

4 INCORPORATING THE METHOD TO AN OBJECT-ORIENTED SOFTWARE DEVELOPMENT PROCESS

One of the manifestations of the lack of integration between Software Engineering and Human-Computer Interaction is the absence of precise guidelines for the creation of user interface in software development processes. One result of this research is the proposal of a user interface construction method (UIC). It is also worth mentioning that there are other results regarding this integration which constitute investigation guidelines in Human-Computer Interaction, however at this moment there are no standards or proposal generally accepted by the community. Some of these results can be seen on the edition presented by Mark Van Harmelen, *Object Modeling and User Interface Design* [Harmelen01].

As stated before, UIC method can be incorporated to an object-oriented software development process, which is based upon functionalities, supported by architectures, iterative and incremental. When we say that it is based on functionalities, it means that the process starts by identifying the functionalities that the system will provide. Supported by architecture means that it allows constructing several models which express different perspectives of the system that is being developed; particularly it is required to model the functionalities by means of a Use Case diagram and modeling objects that make up the software domain by means of a Domain Object Model. On the other hand, the development process should be iterative and also feasible to develop in an incremental way as to become the final product.

Software Development Unified Process fulfils the aforementioned characteristics because, among other factors, it includes the construction of a Use Case Model and a Domain Object Model. Figure 8 shows activities as well as the stages belonging to the Unified Process as they are presented in [Jacobson99]; where the user interface construction is incorporated just as another activity within the software development process.

As it can be seen in Figure 8, the greatest effort of user interface construction is invested during the phases of Inception and Elaboration; this is because the prototype is built after defining Use Case and Domain Object models.

Once the UIC method has been integrated to the Unified Process, what it is left is a software development process characterized by the precise and clearly incorporation of user interface construction, using two of the UML models and introducing the concept of interaction pattern and interface object to a user interface construction.

One of the main advantages of using interaction patterns is the fact that you can successfully reuse solutions to recurrent problems related to user interface construction. In order to achieve its effectiveness it is convenient to use tools that facilitate the

handling of reusable components. In this way it is possible to storage patterns in a repository and to recover them to be used for constructing a particular user interface.

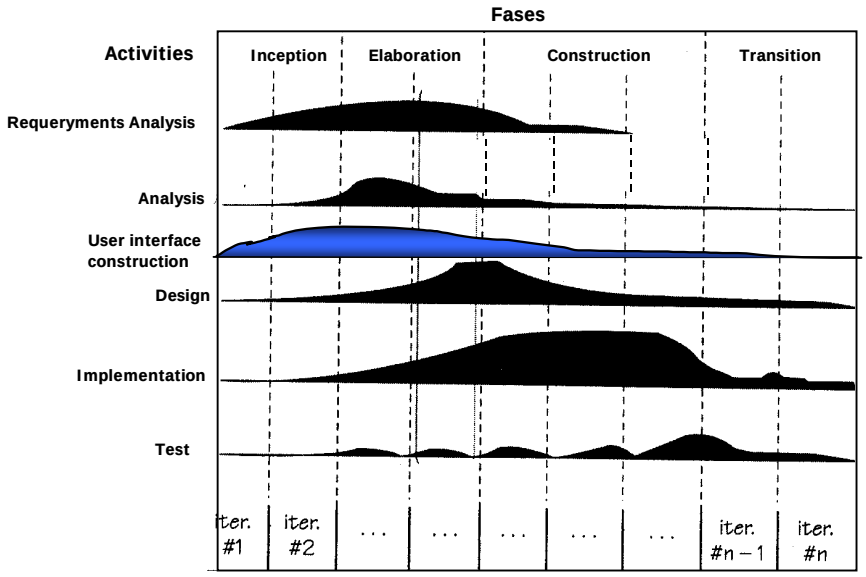


Figure 8. Unified Process and UI construction

5 CONCLUSION

This work shows interaction patterns as reusable components which, among other things, capture knowledge. These patterns are simple, adaptable and recoverable components, and combined with interface objects constitute the basis for defining a user interface construction method.

As far as interaction patterns are concerned, we propose a simple and precise structure to express these patterns which will help different professional groups to understand them; groups who could take part of the development of current user interfaces. Moreover, this work defines a taxonomy definition and a pattern organization that supports its creation and usage by interface design specialist as well as other specialists.

The user interface construction method represents a new step towards integration between Software Engineer and Human-Computer Interaction, establishing precise guidelines for user interface construction and incorporating this activity within system life cycle systems.

The user interface construction method has been experimentally validated at the first semesters of Computer Science at the Central University of Venezuela obtaining satisfactory results. In addition, it has been successfully applied in geological modeling software development as part of a master thesis [Reyes02]. These study cases allowed to simplify and clean the method showing its applicability and effectiveness.

ACKNOWLEDGEMENTS

The author of this paper would like to thank the Council of Scientific and Humanistic Development of the Central University of Venezuela (CDCH) for financing this research.

REFERENCES

[Alexander77] Alexander, Ch.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I. and Angel, S. "A Pattern Language: Towns, Buildings, Construction.", *Oxford University Press*, 1977.

[Bayle97] Bayle E, Bellamy R, Casaday G, Erickon T, Fincher S, Grinter B, Gross B, Ledher D, Marmolin H, Moore B, Pott C, Skousen G, Thomas J. "Putting it all together: Toward a Pattern Language for Interaction Design." *CHI97 WorkShop*, 1997.

[Beaudouin00] Beaudouin-Lafon, M. y Mackay, W. "Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces". *Proc. Advanced Visual Interfaces, AVI2000*, Italy, May 2000.

[Borchers01] Borchers, J. A Pattern Approach to Interaction Design. Wiley 2001.

[Borchers00] Borchers, J. "Breaking the interdisciplinary Limits of computer-human Interaction Design: A Pattern Approach.", *SIGCHI* vol 32 1, January 2000.

[Borchers99] Borchers, J.; Fincher, S.; Griffiths, R.; Pemberton, L and Siemon, E. "Usability Pattern Language: Creating a community." *Report of workshop at INTERACT'99* (Edinburgh, Scotland, Agosto 30-31, 1999).

[Carroll02] Carroll, John. *Human-Computer Interaction, the New Millenium*. Addison-Wesley. USA,2002.

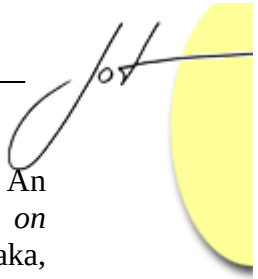
[Casaday97] Casaday, G. "Notes on a Pattern Lenguaje for Interactive Usability." *CHI'97* Electronic Publications: Late-Breaking/Short Talks URL: http://www.acm.org/sigchi/chi97/proceedign/short-talk/gca.html.

[Coram03] Coram, T. Y Lee, J. "Experiences – A Pattern Language for User Interface Design.", Visited June 2003. URL: http://www.maplefish.com/todd/papers/Experiences.html

[Erickson01] Erickson, T. "Lingua Franca for Design: Sacred Places and Pattern Languages." *DIS 2001*. URL: http://www.pliant.org/personal/Tom_Erickson/LinguaFranca_DIS2000.html

[Gamma97] Gamma, E.; Helm, R.; Johnson, R. y Vlissides, J. *Design Patterns. Element of Reusable Object-Oriented Software*. Addison-Wesley. USA, 1997.

[Jacobson99] Jacobson, I, Booch, G; Rumbaugt, J . *The Unified Software Development Process*. Addison Wesley, 1999.



[Mahemoff98] Mahemoff, M. Y Johnston, L. “Pattern Language for Usability: An Investigation of Alternative Approaches.” *Asia-Pacific Conference on Human-Computer Interaction* (APCHI’98) Proceedings, 25-31. Tanaka, 1998

[Nielsen93] Nielsen, J. *Usability Engineering*. Prentice-Hall. USA, 1993

[Reyes02] Reyes, A., Acosta, E, y Zambrano, N. “Patrones de Interacción: su uso en la construcción de la Interfaz de una Aplicación de Modelación de Medios Geológicos.” RT 2002-01. *Lecturas en Ciencias de la Computación*, ISSN 1316-6239. Escuela de Computación, U.C.V. Apartado 47002 Caracas, Venezuela 2002.

[Rumbaugh99] Rumbaugh , James; Booch, Grady and Jacobson, Ivar. *UML: Unified Modeling Language*. Version 1.3, June 1999. Rational Software Corporation http://www.rational.com

[Sparks02] Sparks, G. “An Introduction to UML. The Use Case Model. Enterprise Architect. UML Case Tool by Sparx Systems”. January, 2002. http://www.sparxsystems.com.au

[Tidwell03] Tidwell, Jenifer. “Common Ground”. Visited June 2003. URL: http://www.mit.edu/~jtidwell/common_ground.html

[Usability03] Usability Group University of Brighton. “The Brighton Usability Pattern Collection”. Visited June 2003. URL: http://www.it.bton.ac.uk/cil/usability/patterns/

[Harmelen01] Van Harmelen, M. Object Modeling and User Interface Design. Designing Interactive Systems. Addison-Wesley. USA, 2001

[Welie00] Van Welie, M. y Troetteberg, H. “Interaction Patterns in User Interfaces”, *7th Pattern Lenguaje of Programs Conference*, August 2000, Illinois, USA, 2000. URL: http://www.welie.com/patterns

[Welie03] Van Welie, Martijn. *Amsterdan Collection of Patterns*. Visited June 2003. URL: http://www.welie.com

About the authors

Alecia Eleonora Acosta is teacher and researcher at the Central University of Venezuela from 0ctober, 1988. She did a D.E.A d'Informatique at the Université Paris-Sud XI, France, September 1992. Next, she obtained her Master in Computer Sciences at the Central University of Venezuela in 1993. Currently she is working on her PhD. in Computing Sciences and doing research around user interface design, patterns, user interface design guidelines, web design, interface agents. She can be reached at eacosta@strix.ciens.ucv.ve

Nancy Zambrano Rivas is teacher and researcher at the Central University of Venezuela from 0ctober, 1976. She obtained her Master in Computer Sciences at the Central University of Venezuela in 1989. She obtained her PhD. in Computing Sciences, Informatique at the Laboratoire de Recherche en Informatique (LRI), Université Paris-Sud XI, 1995. She is researching around Software Engineering (light methods, heavy methods, object-oriented methods, Unified Process, software development methods based on transformations) and Human-Computer Interaction (interaction patterns, user interface construction methods). She can be reached at nzambran@strix.ciens.ucv.ve