From User Requirements to User Interfaces: A Methodological Approach*

Juan Sánchez Díaz, Oscar Pastor López, and Juan J. Fons

Department of Information Systems and Computation
Valencia University of Technology. Valencia, Spain
{jsanchez;opastor;jjfons}@dsic.upv.es

Abstract: When a software product is designed and implemented, it is very important to assure that the user requirements have been properly represented. To achieve this objective, a guided software production process is needed, starting from the initial requirements engineering activities and through to the resultant software product. In this paper, a methodological approach for generating user interfaces corresponding to the user requirements is introduced. By doing this, we go a step further in the process of properly embedding requirements engineering in to the software production process, because users can validate their requirements as early as possible, through the validation of the user interfaces generated as a software representation of their requirements. Also, these interfaces can be reused for further refinement as a useful starting point in the software development process.

1 Introduction

When conceiving a computer system, the first stage consists of understanding and showing the user´s needs in a suitable manner. This process is generically called requirements engineering and has been acknowledged as a crucial task within the development process ([2],[16]). It is widely known that errors which originate in this requirement stage can go undetected up to the operating step, thereby causing faults which have serious consequences, especially in critical systems.

Errors during the stage of elicitation of requirements are mainly caused by the gap which exists between the users and the development process. The users are presented with an abstract specification of the system, which is generally incomprehensible to them. This often makes the value of performing requirements engineering techniques unclear and originates well-known comments as “requirements engineering is fine, but we do not have time to deal with it!”. To face this problem, techniques for obtaining user interfaces which correspond to user requirements within a precise methodological approach are needed.

* The research reported in this paper has been partially supported by the Spanish Ministries of Education and Industry, CICYT- FEDER project number TIC1FD97-1102

A system's behaviour may be intuitively described through the use of scenarios. A scenario is defined as a partial description of a system's behaviour in a particular situation ([2]). This description's partial nature allows for a scenario to cover parts of a system's behaviour. This is an interesting characteristic, as different users can perceive the system in different ways.

Scenarios are a valuable tool for capturing and understanding requirements and for analysing interactions between man and machine ([17]). A standard requirements engineering process, one based on scenarios ([25]), has two main tasks. The first one generates specifications using the scenarios to describe the system's behaviour. The second one consists of validating the scenarios with the user by means of simulation or prototyping. Both tasks are tedious if not backed up by an automatic or semi–automatic tool.

The validation stage is tackled in some cases by using RAD tools (Rapid Application Development) ([15]). The process of defining and generating the user interface prototype is, in any case, a manual process, since each object must be explicitly created.

In order to overcome all these problems, we consider that it is necessary to define a methodological approach to derive user interfaces from user requirements. This is why we are interested in the validation of scenarios by means of automatically generating user interface application prototypes and their symbolic implementation.

In this paper, such a software production environment is introduced. As opposed to other proposals, we defend the idea of having a process with a high degree of automation where the generation of user interfaces corresponding to precise user requirements has a methodological guidance. Furthermore, a corresponding visual tool which allows us to almost completely automate the entire process has been implemented. An important contribution of the method is that it automatically generates an inter–form model of navigation which is based on the relationships *include* and *extend* specified among the use cases. The introduction of this navigation feature makes possible to use the generated interfaces in web environments.

In short, this paper presents both a methodological proposal and the associated support tool which backs it up, within the field of requirements engineering. They are based on the Unified Modelling Language (UML), extended by the introduction of Message Sequence Charts (MSC) ([12]). As we view MSCs as extended UML Sequence Diagrams by adding the needed stereotypes, the approach can be considered UML-compliant from the notational point of view.

A clear, precise iterative process allows us to derive user interface prototypes in a semi-automatic way from scenarios. Furthermore, a formal specification of the system is generated and represented through state transition diagrams. These diagrams describe the dynamic behaviour of both the interface and control objects associated to each use case or each MSC. The method has four main steps: the first two steps require analyst assistance to some degree, whereas the last two steps make the process of scenario validation fully automated by means of prototyping

The work is structured as follows: the second section introduces the UML models which we use in our proposal, together with the example that will illustrate the process of prototype generation. Section 3 contains a detailed discussion of the activities and processes which our approach entailed. Related work is discussed in section 4. Lastly, future lines of work and conclusions will be presented. Images captured on the CASE tool implemented for the method are include throughout the paper, to make the process easier to understand.

2 The Unified Modelling Language

The Unified Modelling language (UML) ([4]) is widely accepted as a standard for the modelling of object oriented systems. Among the different views or models proposed in UML, our approach is based on the use case model, the state transition diagrams and a variant of the sequence diagrams: the MSC. We assume the existence of an initial class diagram model that is used in the building stage of the MSC to show classes which emit or receive events. To illustrate the proposal, we shall use a simplified order management system as described in [20].

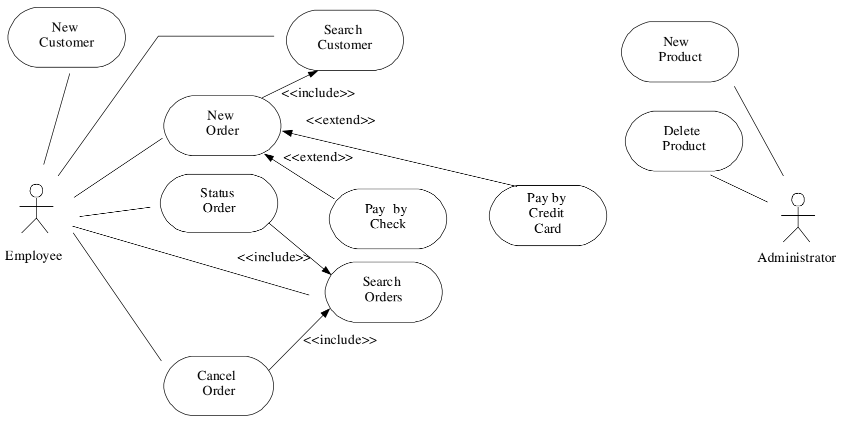

Fig. 1. Symplified use case model, order entry system.

Next, we are going to briefly introduce the main features of the UML models that we use, focusing on how we use them and which expressiveness has been added in order to understand why they are needed in our proposal.

2.1 The Use Case Model

The use case model contains the agents or actors which can interact with the system, represented as a use case. A use case, as introduced by I. Jacobson [13], constitutes a complete course of interaction that takes place between an actor and the system. Figure 1 shows a use case model (simplified due to shortage of space) for an order entry system. There are two actors: *employee* and *administrator*. The employee is in charge of adding new customers, introducing orders, finding out the order status (admitted, ready to send, sent) and lastly canceling them. The administrator is in charge of: adding new products and deleting products.

Two relationships can be defined: include and extend. The "include" relationship is employed when a flow of events can be textually inserted within another use case. On the other hand, the "extend" relationship shows an alternative execution flow. This can, under certain conditions, be considered as an inclusion.

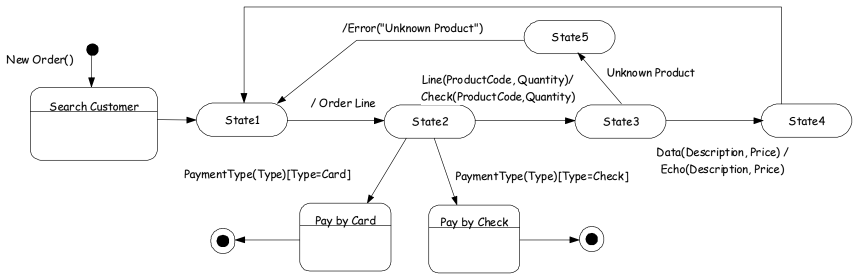
New Order()
Search Customer
State1
/ Order Line
State2
Line(ProductCode, Quantity)/
Check(ProductCode,Quantity)
State3
State4
State5
/Error("Unknown Product")
Unknown Product
Data(Description, Price) /
Echo(Description, Price)
PaymentType(Type)[Type=Card]
PaymentType(Type)[Type=Check]
Pay by Card
Pay by Check

Fig. 2. State transition diagram, user interface object in *New Order* MSC.

2.2 The MSC

The MSCs are widely used in the field of telecommunications to describe the exchange of information among system classes. In their simplest form, with no information on control or conditions, they are considered to be equivalent to the UML interaction diagrams. Figure 5 contains the notation used in our tool.

The vertical lines represent system classes or actors, whereas the sending of events or the exchange of information is shown as horizontal lines. Each event can have a number of arguments associated with it, and these can be basic types (integer, string, boolean, enumerated, real etc.) or class types. There is an additional notation that reflects the repetition of events, alternatives, exceptions, interruptions and conditions which can show what state the system is in. To keep the proposal within the UML umbrella, this extra-information can be seen as a UML sequence diagram extension based on the introduction of the corresponding stereotypes. The MSC can be broken down into levels, in such a way that the analyst can employ the desired degree of abstraction in each diagram. ITU ([12]) or Telelogic ([26]) may be consulted for further details.

2.3. State Transition Diagrams

The state transition diagrams ([9]) are normally used within the object oriented methodologies to describe the behaviour of the system classes. They are useful for representing the objects life-cycles of the objects, and they can also be used to describe the behaviour of the application user interface ([10]). In our approach, we shall employ them to describe the object of the user interface and the object of control that appears in each MSC diagram. The STD will be used in the process of user interface animation.

In Figure 2, a standard STD expressiveness (UML compliant) is used to describe the behaviour of the user interface object used in the *New Order* MSC. This diagram will be explained in detail in the next section. At the moment, we introduce it only to demostrate that we use the expressiveness provided by the STD of the UML.

3 Description of the Proposal

In this section we present precise method to guide the generation of user interfaces corresponding to the user requirements, according to the ideas introduced in section 1 and using the UML diagrams commented on the previous section. Figure 3 shows a schematic representation of the activities contained in the proposed method. As we have commented above, the first two activities, namely scenario representation and synthesis of use cases, are manual activities which the analyst must carry out. The last two, specification generation and generation of prototypes, are totally automatic.

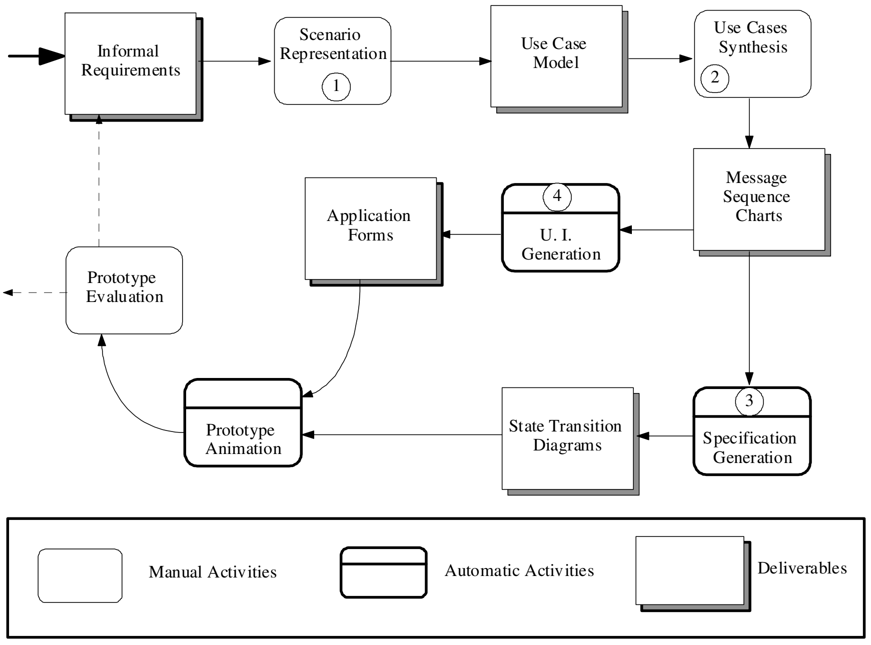

Fig. 3. Schematic representation of the method.

Figure 3 also fixes the order in which these activities should be performed. The process begins at the scenario representation stage where a use case model is created. The next stage consists of describing use cases by means of MSC. During the stage of specification generation, a state transition diagram (STD) for the class User Interface and another STD for the Control Object are automatically obtained from a given MSC. Lastly, the final stage consists of automatically generating the user interface prototypes as well. The method is iterative; in consequence, the prototyping is used to validate and enrich the initial requirements. We shall now proceed to explain each stage in detail.

3.1 Scenario Representation

During the scenario representation stage, the analyst has to construct the system's use case model. This process is structured in three layers or stages: *the initial model*, *the description model* and the *structured model* ([22],[23]).

The initial model shows the actors and use cases grouped together according to whether they carry out similar functions. Figure 4 shows the initial model of the order entry system.

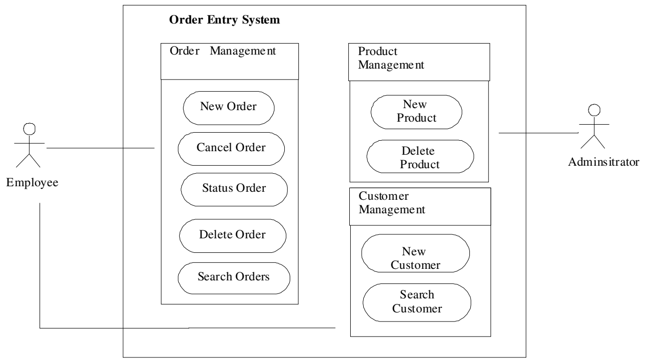

Fig. 4. Use case initial model.

We use the initial model for describing actors, non-structured uses cases and system services. For instance, the *order management* includes the use cases: new order, cancel order, status order, delete order and search orders. Once this is done, the description model contains the use case descriptions, using a natural language template. We shall use a variant on the template proposed by L. Constantine et al ([7]), which is composed of the following elements:

- Name: Identifies the use cases and should reflect its function or immediate purpose.
- Preconditions: the state of the system required for the use cases to be executed
- Post-conditions: the state the execution of the use cases leads the system to.
- Primary actor: This is the initiator actor who generates the starting stimulus to the system.
- Secondary actors: Other participating actors who communicate with the use case.
- Event flow description: it shows the events generated by the actors (user intentions) and the system commitments (system responsibilities).
- Extension points: Reflect alternatives, conditional o exceptional courses of interaction that can alter the main flow of events. We use two kinds of extension points: synchronous extensions and asynchronous extensions.
- Relationships: Identifies other use cases which are related in some way (include, extended) to the use case being described.

The template in Table 1 shows the description of the use case "search customer". The event flow is divided into two columns: user intentions and system responsibilities. This division allows us to identify when the actors request services, and when the system acts as information supplier. This is important for constructing the MSC in the next stage of the method as we will be seen below.

Table 1. Use case template, *Search Customer*.

Name: Search Customer	
Relations	
Include:	None
Extend:	None
Description	
Preconditions	None
Postconditions	None
Primary Actor	Employee
Secondary Actors	None
Event Flow	
User intentions	**System responsibilities**
1. The employee selects *Search Customer*	
2. The employee introduces Customer Name or Customer Code	
3. The employee selects "*Apply*"	
	4. The system searches customers by Name or Code
	5. The system displays the result of search
Asynchronous extensions	
6. The employee can select *Stop* at any point	
Synchronous extensions	
7. If there is no *Name* or *Code* then system displays a message error at point 3.	

Finally, the structured model graphically shows the use cases relationships *include* and *extend* (see Figure 1), producing the corresponding final graphical representation. With this step, the process of capturing user requirements is considered to be finished.

3.2 Synthesis of Use Cases

Once we have obtained the Use Case Model, we need to work with the involved use case information to undertake the process of designing a software system. To do this, use cases must be formally described: the formal definition of a use case is achieved by using a graphic, non-ambiguous language, such as MSC. In this phase, which is a manual one, the use case templates are used as help so that the analyst can detect the events sent by the actors and by the classes of problem domain In each MSC, besides the participating actors (initiator, supporting actors), one class for the user interface and one class that acts as control object are introduced, according to the initial Objectory proposal ([13]) and according to the UML approaches for a software production process ([14]).

The formalism that we use, as opposed to other well-known methods (for example, the trace diagrams of OMT [18]), allows us to show alternatives in the sending of events, iterations, exceptions, etc. Furthermore, the notation allows for a graphic representation of the dependence that exists among use cases. If a case A uses a case B, a rectangle with the name B is placed in A's MSC. This indicates that the diagram can extend to diagram B. If the given extension depends on a given condition, this extension relationship can be treated in the graphic level following the MSC semantics by properly representing the condition that must hold and calling the diagram produced by the extension.

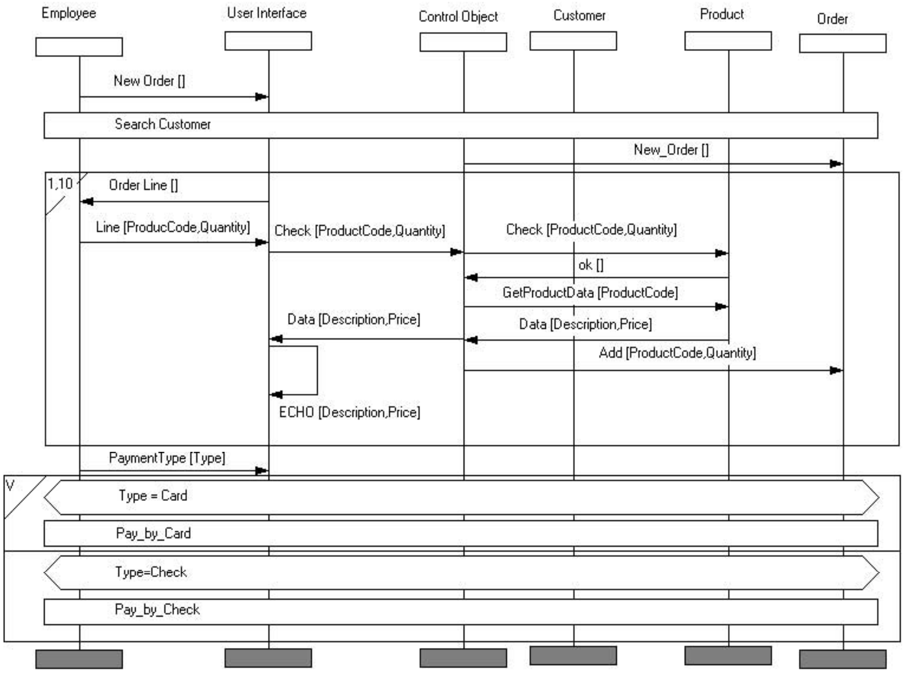

Fig. 5. MSC *New Order*.

Figure 5 show an "new order" MSC corresponding to the "new order" use case. This diagram shows the events which occur when an order is introduced into the system. The "include" relationship corresponding to the search customer use case appears in the diagram as a rectangle that is labeled with the name of the associated MSC diagram that contains the expansion (*search customer* in this case). The extension point attached to the payment by credit card use case is represented through the corresponding MSC condition (*type*=*card*) and the rectangle that refers to the MSC representation of the use case extension (*payment* by *credit card* in this case).

In the above figure, the actor *employee* initiates the MSC sending the event New Order. At this point the MSC search customer is "executed", and an empty order is created. The rectangle labelled with 1,10 shows the repetition of events: the system asks for an order line and the employee enters product codes and quantities. The user interface object checks the product code sending the event *CheckProduc(ProductCode, Quantity)* to the control object. The control object verifies the information with *Product*, returns the event *Data(Description, Price)* and adds a new order line to the current order. The employee enters the payment information (*PaymentType(Type)*) and finally, if the condition *Type=Card* holds, the MSC *Pay_by_Card* is "executed". It is important to note that this MSC shows a primary scenario of interaction without error conditions.

An important piece of data that must be introduced in this step is constituted by the labels that will appear in the user interface to identify relevant pieces of information. When following the flow of events specified in the MSC, a given piece of information enters or exits the user interface object, the analyst must specify the corresponding label, that will play a basic role in the process of generating the user interface.

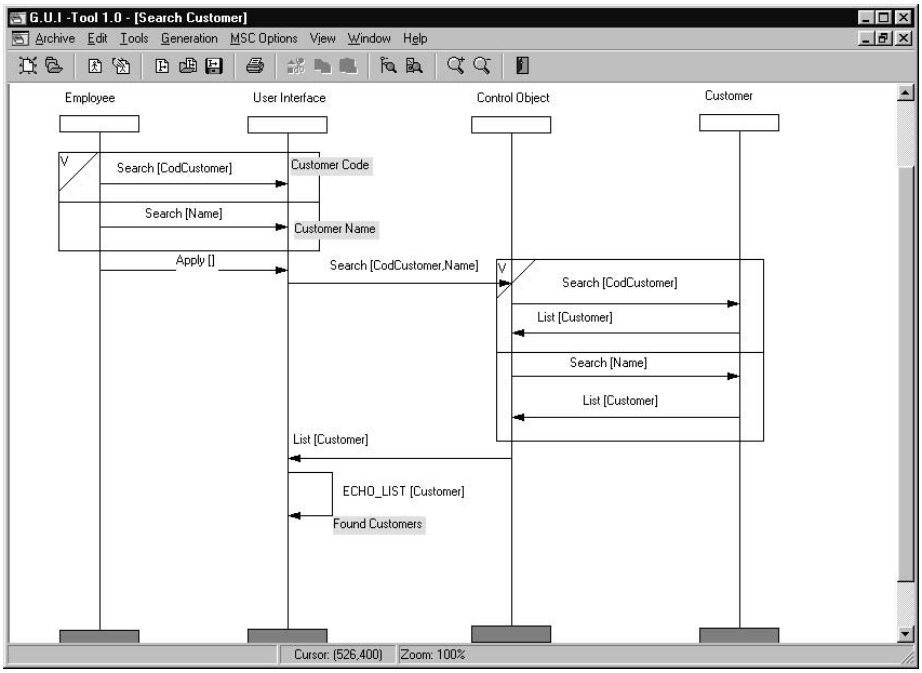

Fig. 6. Labelled MSC chart, *Search Customer*

Figure 6 shows the MSC corresponding to the use case "*search customer*". The analyst has placed 3 labels: "customer code", "customer name" and "founds customers". These correspond to two areas/fields of entry and one echo or area/field of exit. This information will literally appear in the form associated with the use case.

Apart from the labels, information about the type of the arguments of each event specified in the diagram must be introduced. The allowed types are the basic data-valued types (number, boolean, character, strings, enumerated) and the object-valued types corresponding to the system classes. The type of the event arguments together with the class attributes are used in the process of the interface generation, as will be seen to in the following sections.

3.3 Generation of the Specification

The specification is formed by a group of state transition diagrams that describe the system behaviour. One diagram for each user interface object and another for each control object that appears in a selected MSC are generated. To put this idea to work, in the tool, the analyst selects the MSCs for which he wants to generate the specification, and the resulting specification is stored in the tool´s repository. To make the implementation easier, we do not employ a graphic representation for the generated STDs; they are stored and viewed using a transition table.

In the process of generating transition tables, we use a variant of the algorithm proposed by Systa ([24]), the details of which can be seen in Sánchez ([21]). The process consists of associating pathways within a STD found in an MSC. Messages directed towards a particular object O, are considered events in the state transition diagram for O. Messages directed away from O are considered actions. References to another diagram in an MSC are transformed into super-states.

The figure 2 shows the STD for the user interface object in the use case *new order*. The states *search customer*, *pay by card* and *pay by check* are super-states. Each super-state has a flat state transition diagram associate to it. The path:

Search_Customer.(State1.State2.State3.State4.State1)$^{1..10}$*.Pay_By_Card* corresponds with the MSC or normal scenario of figure 5. The alternate paths (ie. *State3.State5*) correspond to an exceptional scenario.

3.4 User Interface Generation

The process of generating the user interface prototype to develop the final software product starts with the analysis of the use case structured model, or the equivalent, resultant MSCs obtained from them. For each actor that plays a role of service activator, a view model is obtained by generating a form that contains the set of forms that can be called up directly from the application menu. Each use case that an actor can execute is converted into a form. The model describing the way of navigating among these forms is obtained by analyzing the include and extend relationships.

Referring back to the example introduced in Figure 1, the following forms would be generated: new customer, search customer, order status, search order, cancel order, and new order. Notice that no forms for Pay by Check or Pay by Credit Card are generated in the view model: as they are the result of using "*include*" or "*extend*" relationships, they follow the following rules:

- If A (New Order) is one case that uses a case B (Search Customer), then there is a button placed in the form associated with A which allows navigation towards the form associated with B.
- If B (Payment by Check) is a case which extends a case A (New Order) and B cannot be directly executed, a button is placed in A which allows navigation towards B.

Let’s see how MSC services are converted into user interface components. For such a service, a form is generated. This form will contain the widgets needed to have the corresponding software representation at the interface level. To make this process of conversion clearer, in Table 2 we present the set of transformations that are done depending on which kind of information input/output do we have.

For instance, given a service $E(a_1,..a_n)$, if a_1 is an enumerated type (as specified in the corresponding MSC), depending on the size (greater or less than 4), a set of radio buttons or a list selection widget will be generated in the resulting user interface.

Table 2. User interfaces widgets.

Information Input: $E(a_1,..,a_n)$		
Argument type (a_i)	**Condition**	**Form Component**
Enumerated	Size =<4	Enabled radio buttons widget
	Size >4	Enabled List Selection widget
Boolean	-	Enabled Radio buttons widget
Character, Number, String	-	Enabled Text Edit widget
$a_1,.,a_n$ class dependent	-	Grid, an input entry for each attribute class
Information Output:		
Basic	-	Disabled Text Edit widget
Class type/ $a_1,.,a_n$ class dependant	-	Disabled Grid widget
Class type List	-	Disabled Grid widget with scroll bars
Error message	-	Message Box widget with confirmation button

The scheme set out in Table 2 is used by the tool to create the set of software components that constitutes the user interface. In fact, this is a closed group of rules that guides the generation process. The above table is based on heuristics and results found in the literature ([11]).

For the MSC shown in Figure 6, and in accordance with the previous table, the tool automatically generates a form with two fields of entry and with the labels “Customer Code” and “Customer Name”. As the system will produce an echo, and as this is made up of a list of objects, a grille/grid will be generated, where each column corresponds to Customer class attributes. The buttons *Apply* and *Exit* are always generated for every form, to represent the two basic options of committing an action or cancelling it in a standard way.

As the “*search customer”* use case is neither used nor extended by any other use case, there are no connecting buttons to allow navigation. What is possible is to reach “*search customer”* from the form associated with “*new order”*. This is the consequence of the “*include”* relationship declared between “*new order”* and “*search customer”* (see Fig.1).

Once a form has been generated according to our method, it can be modified using the target visual programming environment. It can also be animated by using symbolic execution, simulating the execution of the associated state transition diagram of the user interface object.

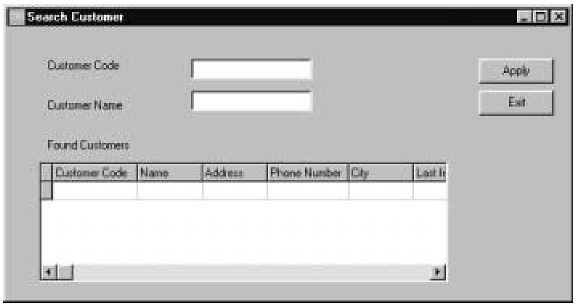

Fig. 7. Form associated to use cases Search Customer.

3.4.1 Actor View Model

The tool integrates the forms generated for the various use cases (MSCs) in a single application (see Fig. 8). The application comprises a menu where the options are the use cases that each actor can execute. For the example under study, this model shows the forms that the actors, employee and administrator, can activate.

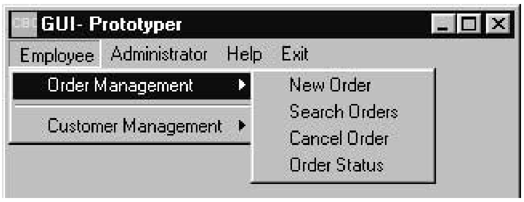

Fig. 8. Actor View Model.

For instance, the actor employee can execute the following services: *new order*, *search orders*, *cancel order* and *order status,* from the menu option *order management* and *search customer*, *new customer* from the menu option *customer management*. The actor administrator, on the other hand, can execute: *new product* and *cancel product*. We use the primary use case model to group the application menu options.

3.4.2 Interform Navigational Model

The inter-form navigation model represents the set of forms that an actor needs in order to complete all the required tasks, as well as the actor's potential options to go from one form to another.

In Fig. 9 the bold line represents the unconditional activation of a form, and the implicit return to the activation point after execution. This unconditional activation corresponds to the use case relationship "include". On the other hand, the activation conditions represents the extension points of a use case. By exploiting these relationships among the use cases or rather among the different MSCs, we automatically obtain the interform navigation model, from the structured use cases model.

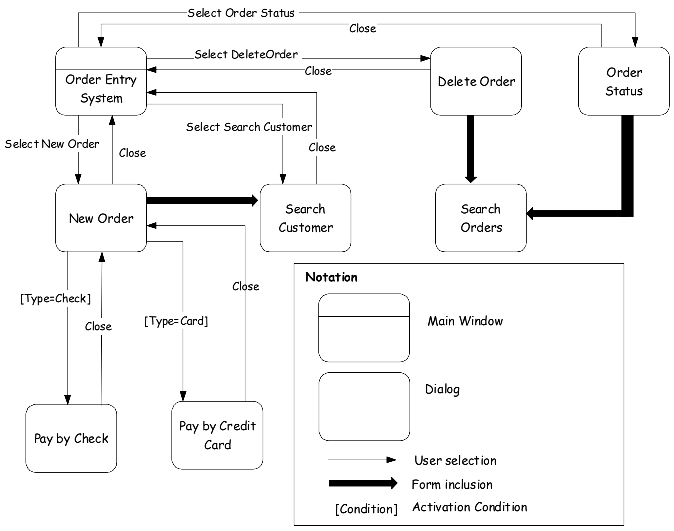

Fig. 9. Interform navigation model, actor employee.

4 Related Work

Within the field of user interface generation, there is a varied range of proposals, that go from those that use data models and generate only the static part of the interface to those that are scenario-based and support different degrees of automation (from partial to total).

If we take into account the proposals based on data models, it is normal to use a variant of the entity-relationship model, or rather an object model. However, if we consider those scenario-based models, they tend to use Petri nets or state transition diagrams as specification techniques. We shall now discuss some of the more relevant proposals in these two categories.

Worth citing from among the approaches based on data models which only generate the static part of the interface, are Janus and Trident. In Janus ([1]), an object model is built and a window is associated to each non-abstract class of the model. The analyst manually selects the methods and attributes which are relevant to that interface. The setting then generates a static view. Even if this approach is interesting, if we compare it with our method, it can be seen that the treatment is merely static, which is an important weak point that we have overcome.

Trident ([27]) uses a more sophisticated approach. Working from an entity – relationship model, an activity graph is drawn which connects interactive tasks with the system's functions and data. The graph is used as a means of entry to a process that selects the different units of presentation. As in the previous case, only the static part is generated, what again marks the main difference with our work.

In TADEUS ([19]), a stage generically called dialogue design is used, and this covers two tasks which must be carried out manually: the design of the navigational dialogue that describes the sequence of the different views, and the design of the processing dialogue which in turn describes the changes undergone by the objects within a dialogue view. The formalism used is based on the Petri nets. During the generation process, scripts are obtained which can be included in an interface management system. The manual Petri Net building process is, in our oppinion, the main drawback of the proposal: it is very hard to use Petri Nets for specifying the user interface properties.

Lastly, the most similar proposal to our own is SUIP ([8]) from Montreal University. They use collaboration diagrams, which they define by means of flat files which are enriched by information from the user interface. They focus on behavioural aspects as we do, but at a lower level of abstraction. We think that starting with use cases is more appropriate than using collaboration diagrams. Furthermore, in contrast to our proposal, it does not obtain a navigational model automatically. It is interesting for us to note that are currently modifying their proposal to use Petri nets instead of STD.

5 Conclusions and Future Work

A methodological approach to guide the process of generating user interfaces from the user requirements elicitation has been presented in this paper. As a main contribution, the approach uses a functional style to capture requirements using a Use-Case background, instead of starting from a merely static system view as other similar approaches do. This is an important point, because analysts can focus on the behavioural system aspects, which are represented by a Use Case Model in a natural way. This Use Case Model is the starting point for the user interface generation process.

Furthermore, the proposal has a high degree of automation, which allows us to provide a CASE tool to support the method. We have built a metamodel for the MSCs, this metamodel has been used to design a relational data base which acts as a repository for information. The tool is programmed in Delphi, and the data base in Interbase 5.1, but the underlying ideas could be implemented in any other conventional client-server software development environment.

The proposal has been successfully put to work to validate user requirements, by executing the user interface prototypes generated in an automated way following the four-step method presented in the paper.

Another aspect which is worth highlighting is that the method allows us to obtain the interform navigational model automatically. This is being used at the moment to generate web clients where the navigational capabilities are derived from the requirements specification, which is an interesting line of research. As a main objective which is underway at this moment, we want to focus on the appearance of the generated forms, with the final objective of being able to use them not only for prototyping purposes, but also as a ready-to-use final software product.

To reach this objective, we are presently trying to incorporate our scheme for the generation and definition of interfaces in the RSI (Requirements/Service/Interface) of M. Collins ([6]). This approach uses 3 categories of use cases: requirements, interface and services. The requirement use cases document business processes which can be automated. The interface model statically describes the user interfaces of the application, by only using drawings with their different components. Lastly, the service model describes what functions the system has to provide, regardless of the needs of one particular user interface. Our proposal would fit perfectly into the interface model, that is, the interface level can be described by means of an automatically generated prototype.

Finally, if some of the figures containing images caught by the tool are observed, a file labelled as "data" can be seen. Our idea is to define real data and to try to animate the prototype with this data, following the same line as those who validate object models based on STD ([5])

References

[1] Balzert, H.; "From OOA to GUIs: The Janus System". IEEE Software, 8(9), Febraury 1996, pp 43-47.

[2] Benner K.M.; Feather M.S.; Johnson W.L. "Utilizing Scenarios in the Software Development Process. En Information System Development Process, pp 117-134. Elsevier Science Publisher, 1993. Editores: N. Prakash, C. Rolland, y B. Percini.

[3] Bennett, D.W. "Designing Hard Software: The Essential Tasks". Manning Publications co, 1997.

[4] Booch G; Rumbaugh J; Jacobson I. "The unified modelling language". Addison-Wesley. 1999.

[5] Bridge Point. CASE TOOL, Project Technology. http://www.projtech.com

[6] Collins-Cope, M; "RSI- A Structured Approach to Use Cases and HCI Design". Personal Communication, Ratio Group Ltd. 1999.

[7] Constantine L.L; Lockwood L.A.D. "Software for Use: A practical Guide to the Models and Methods of Usage-Centered Design". Addison Wesley 1999.

[8] Elkoutbi M; Khriss I; Keller R; "Generating User Interface Prototypes from Scenarios". Proceedings of the Fourt IEEE International Symposium on Requeriments Engineering (RE'99). Limerick Ireland 1999.

[9] Harel D; "State Charts: a visual formalism for complex systems". Science of Computer Programming, 8(3), 231-274. 1987.

[10] Horrocks I. "Constructing the User Interface with Statecharts". Addison-Wesley, 1998.

[11] IBM, Systems Application Architecture: Common User Access- Guide to User Interface Design- Advanced Interface Design Reference, IBM, 1991.

[12] ITU: Recommendation Z. 120: Message Sequence Chart (MSC). ITU, Geneva, 1996.

[13] Jacobson I et al. "Object-Oriented Software Engineering: A use case driven approach". New-York ACM Press, 1992.

[14] Jacobson I; Booch G; Rumbaugh J. "The Unified Software Development Process". Addison-Wesley,1999.

[15] Kerr J; Hunter R; "Inside RAD". McGraw-Hill 1994.

[16] Kotonya, G; Sonmmerville, I. "Requirements Engineering: Process and Techniques". John Wiley & Sons. 1998.

[17] Nielsen J. "Scenarios in Discount Usability Engineering". Scenario-Based Design: Envisioning Work and Technology in System Development. John Wiley & Sons, 1995. pp 59-85.

[18] Rumbaugh J; et al. “Object Oriented Modeling”. Prentice Hall 1997.

[19] Schlungbaum E; Elwert T; “Modeling a Netscape-like browser using TADEUS”. CHI’96. Vancuver 1996. Pp 19-24

[20] Schneider G.; Winters J.P. “Applying Use Cases: A practical Guide”. Addison Wesley 1998.

[21] Sánchez Díaz J; “Generating State Transition Diagrams from Message Sequence Charts” Technical Report -DSIC,-1999. Valencia University of Technology (in Spanish).

[22] Sánchez J; Pelechano V; Pastor O; “Un entorno de generación de interfaces de usuario a partir de casos de uso”. Workshop On Requeriments Engineering’ 99. pp 106-116. Buenos Aires. Septiembre 1999.

[23] Sánchez J; Pelechano V; Insfrán E; “Un entorno de generación de prototipos de interfaces de usuario a partir de diagramas de interacción”. IDEAS 2000. Pp 145-155. Cancún (Mexico). Abril 2000.

[24] Systa T. “Automated Support for Constructing OMT Scenarios and State Diagrams”. Department of Computer Science. University of Tampere. Technical Report A-1997-8

[25] Somé S.S.; Dssouli R; Vaucher, J. "Toward an Automation of Requirements Engineering using Scenarios". Journal of Computing and Information, vol 2,1, 1996, pp 1110-1132,

[26] Telelogic. CASE TOOL Telelogic Tau 3.6.2. http://www.telelogic.com. 1999.

[27] Vanderdonckt J. “Knoledge-Based Systems for Automated User Interface Generation: the TRIDENT Experience”. RP-95-010. March 1995.