The Just-UI approach

Conceptual modelling of device independent user interfaces

MOLINA Pedro J.

CARE Technologies, Spain
pjmolina@care-t.com

MELIÁ Santiago

CARE Technologies, Spain
smelia@care-t.com

PASTOR Óscar

Technical University of Valencia, Spain
opastor@dsic.upv.es

Abstract : A Model for the Specification of Abstract User Interfaces based on Conceptual Patterns is proposed to enhance the captured semantic by an object-oriented analysis method. The model gathers both Presentation and Navigation issues. A graphical notation is also provided to make easier the specification tasks. This simple graphical notation allows that a non-analyst can understand and participate in the specification process.

Keywords : Model-based user interface specification, conceptual modelling, user interface requirements.

1 Introduction

In this new millennium, the volume of data produced in the Information Society is increasing continuously on a daily basis. In this *data ocean*, tools for searching and navigating among data are really needed as never before to extract valuable information. On the other hand, better applications in less time are demanded. Nevertheless, developing good user interfaces for such Information Systems is a non trivial task which requires effort to build and maintain them in order to assure a good quality product.

Commercial business applications based on Information Systems have similar interfaces, mainly based on forms. Just-UI tries to identify patterns in such user interfaces and abstract them to work in terms of the problem domain. The model is based on Conceptual Patterns to capture elemental user interface requirements such as: *how to search?, how to order?, what to see?, and what to do?* Answering these questions in an abstract specification allows us to express the requirements without fixing any target environment. Design aspects, deliberately, are not collected in this phase in order to maintain the specification free of any design issues.

This article is an extended version of the original paper (Molina, 2002) presented by the same authors at the CADUI'2002 Conference in Valenciennes, France. It starts with the motivation and introduction of the concepts used in the Just-UI approach. Next, a graphical diagram based on these concepts is also introduced. Following, the corresponding meta-model is also introduced. Afterwards, a case study is described in terms of the specification, analysis and implementation of the user interface. Later, generation issues are commented on. And finally, related work and conclusions are given.

2 The Just-UI approach

Traditional UML CASE tools such as Rational Rose (Quarani, 1998), Together (TogetherSoft, 2002) or Argo (Robbins, 1997) do not explicitly consider the specification of user interfaces. Therefore, User Interface Specification with these tools used to be complex and incomplete as it has been reported by Pinheiro (2000-a).

Collections of User Interface Patterns do exist. However, these collections are focused on design problems (Tidwell, 1999) & (Van Welie, 2000) and not on analysis problems. Indeed, surveys on MB-UIDEs (Szekely, 1996) & (Pinheiro, 2000-b) show that patterns at analysis levels are unexplored in classical MB-UIDEs.

The work here presented tries to bring User Interface Specification nearer to non analysts (users, graphical designers, etc.) with the ultimate intention of involving them in the specification process.

Our contribution is focused on Just-UI: a Model for the specification of User Interfaces linked to Conceptual Modelling and code generation techniques to automatically implement the user interface for business applications.

Constantine (1999) proposes a technique to gather User Interface requirements prototyping with users using paper and Post-Its. In this technique, each Post-It represents an interaction unit (e.g. a form or web page in the implementation). The user and analyst build together the navigation among interaction units drawing arrows. Just-UI follows the same intuitive approach to describe User Interfaces as it is used by Ruble (1997) and Constantine (1999). Indeed, Just-UI concepts can be collected in the same form: using paper and post-its working with the user.

The Conceptual Modelling stage is supported by an object-oriented method, the OO-Method (Pastor, 1997), to obtain a conceptual schema of an Information System. OO-Method is a method that provides UML (OMG, 1997-a & 1997-b) compliant diagrams and is based on the formal language OASIS (Pastor, 1992). The model captures classes, properties for classes: attributes, services (methods), static, and dynamic constraints, etc. following a classical object-oriented approach.

Just-UI extends OO-Method to capture User-Interface requirements to explore and navigate among data (information objects). The final goal is to collect interface requirements in an abstract way and to automatically generate the final user interface to different target environments such as Web, Windows, X11, UMTS, or PDAs. The user interface requirements captured by this approach are collected in the OO-Method Presentation Model.

The following subsections describe the patterns used. First of all, we will start introducing simple patterns. Afterwards, more complex patterns can be compounded using the simple ones to build bigger components and complete the specification.

2.1 Simple Patterns

Simple patterns are single components that can be asked directly to user or customer by an analyst. These simple concepts constitute the primitive *bricks* to build the user interface.

The concepts that are going to be introduced are:

- Filter (*How to search?*),
- Order Criterium (*How to order?*),
- Display Set (*What to show?*),
- Navigation (*What to explore?*), and
- Actions (*What to do?*).

2.1.1 Filter

A filter is a criterion that is useful for searching for objects. Fixing a class, the user needs to search for objects satisfying a given condition. E.g. searching players by name or surname in a Golf Tournament Tracking System (All the examples described correspond to the case of study described in section 3.). The analyst has identified the Player class in the problem domain. Now, he has to ask the user: *How do you need to search for Players?* Each answer from the user constitutes a filter criterion.

A filter for a class can be defined as a query method in such class. The filter has a unique name in the class and a list of parameters to fix the searching criteria. The filter method returns, as a result, a collection of objects. The filter is expressed in a formula using filter parameters, attributes of the class, constants, literals and operators. The formula always is evaluated to a boolean type. The semantic of the formula is the following: the result returned by the filter is a collection of objects that satisfies (makes true) the formula.

E.g. the formula for searching players by first name "Tiger" and last name "Woods" is:

```
FirstName="Tiger" and LastName="Woods"
```

The given example is a closed formula (nothing is needed to be asked to the user). Introducing filter parameters, the formula will be opened and more powerful (values for the variables will be requested to user before searching). E.g. the formula for searching players by first name and last name is:

```
FirstName=v_FirstName and LastName=v_LastName
```

Where *v_FirstName* and *v_LastName* are filter parameters that the user must provide.

Query By Example (QBE) techniques (Zloof, 1977) could be also applied, but previous experiences reveal us that, in general, users do not need a powerful search engine. On the contrary, users usually employ few filters for a given scenario and prefer specific filters tailored for the current tasks. In this way, usability is increased.

2.1.2 Order Criterium

After gathering search requirements, the analyst can inquire: *How must objects be ordered?, alphabetically?, by Name?* Again, each response given by our customer constitutes an Order Criterium Pattern.

Order Criterium can be specified in terms of a list of attributes defined in the conceptual schema and an ordering direction for each attribute: ascendant or descendent. The semantic associated with this pattern is: sort the collection of objects using the criteria expressed in the list. E.g. to sort Players by handicap (ascending) and number of rank (ascending), we have to express:

```
Handicap ASC, NumberRank ASC
```

The semantic is similar to the ORDER BY clause employed in an SQL statement: the first element in the list (in this case the attribute *Handicap*) is used to sort the objects in an ascending way (ASC). Then, the second element is used to sort objects that have identical value for the attribute expressed in the first element, and recursively sorting objects until reaching the end of the list.

2.1.3 Display Set

At this point, we have the objects filtered and ordered. But, what properties does the user need to see from a given object? The customer could answer for a Player: I need to see: *FirstName, LastName, Licence, NumberRank, Handicap, Address, and City.*

A list of visible attributes from the given class is enough to capture the initial user requirement. E.g. the Player Display Set example could be:

```
FirstName, LastName, Licence, NumberRank, Handicap,
Address, City
```

The list imposes an order among its elements. In this case, the order is a relevant characteristic. It is not the same requirement to show: *"FirstName, LastName"* than *"LastName, FirstName"*.

Once again, this information is extremely simple and useful to build the user interface.

Up to now, the user has filtered, ordered and listed the objects. At this point, the user has enough power in the user interface to search and select objects. In an object-action (noun-verb) paradigm the first step is to select the object to interact with. Once the object is selected, the second step can be proposed: What to do with such an object? Basically, two responses can be provided:

- To show information related to this object (*Navigation*).
- To execute methods to change the object's state (*Action*).

Note that *Navigation* is merely observation and it is innocuous by definition. Nevertheless, *Actions* may execute services or operations that could alter valuable data.

2.1.4 Navigation

Following with the analyst's questions to users, the next one to ask is: *When you have reached an enrolment, do you need to see any other related information?* E.g.: *The user is exploring an enrolment. The analyst has identified the need to navigate to additional information such as player or tournament.* Associated to this kind of navigation, the analyst is refining aggregation or inheritance relationships between classes in the conceptual schema. The analyst must specify the subset of objects to be reached in the target. E.g. it can be expressed using roles of relationships, or filters.

2.1.5 Actions

The second kind of semantic associated to an object is the possibility of changing the state of the current object. In conventional object-oriented methodologies, this is performed executing methods (also called services in OO-Method) defined in the class. Once again, in a given context, the analyst has to ask the user: *What actions the object can suffer*? The user can answer something like this: *In the enrolment scenario I need to work with enrolments for a player: I need to create, modify, and destroy enrolments. Additionally, I need to abandon the enrolment related to player.*

With this information, the analyst has enough clues to complete the conceptual schema with new services in classes and to organize a subset of services to offer users in a given scenario. This list (ordered subset) of visible services is the representation of what we call *Offered Actions Pattern.*

2.2 Building complex blocks

An *Interaction Unit* is a particular scenario in the user interface where users have to interact with them in order to carry out concrete tasks. The Presentation Units (Bodart, 1994-a) abstract these contexts as a set of AIOs (Vanderdonckt, 1993) (*Abstract Interface Objects*) discarding design issues. Interaction Units are Presentation Units dealing not only with presentation but also supporting behavioural and task constraints.

A reduced set of patterns focused on Interaction Units have been abstracted based on literature review (Ruble, 1997) & (Constantine, 1999), our experience exploring User Interfaces of commercial applications and developing business applications. These patterns are going to be referred to as Presentation Patterns. The Presentation Patterns are used to abstract interaction units with common behaviour. This approach provides different advantages:

- *Extensibility:* new Presentation Patterns can be identified and included in the model,
- *Homogeneity:* every interaction scenario is expressed with a Presentation Pattern, and
- *Design independence:* no design details are collected in the model.

At this stage, we have identified four types of Presentation Patterns:

- *Service* (interaction with service execution),
- *Instance* (interaction with an object),
- *Population* (interaction with a set of objects), and
- *Master-Detail* (composed interactions).

2.2.1 Service Interaction Unit

Service Interaction Unit is in charge of modelling a dialogue to deal with a service. The user must provide the arguments and launch the service. In the problem domain, the analyst identifies a service associated with a certain class: e.g. to create an Enrolment in the Golf Tournament Tracking System and declares it in the object model. In terms of specification, the Service Interaction Unit encapsulates the interaction unit to supply a service in the interface. In this context, the service specification can be completed asking the user: *What input data are needed for this service?, How must input data be grouped?, Are the input fields inter-related?, What kind of feedback do you need for each selected object?*, etc.

Many answers to these questions can be expressed using other auxiliary simple patterns:

- *Introduction* (constrains the introduction of values),
- *Defined Selection* (defines by enumeration the valid values),
- *Population Selection* (expresses how to select objects),

- *Dependency* (expresses dynamic interdependencies among objects),
- *Supplementary Information* (provides extra feedback for object identifiers),
- *Status Recovery* (recovers argument values from attributes), and
- *Grouping* (logical grouping of arguments).

For further information about these patterns refer to (Pastor, 2000) & (Molina, 2001-b).

2.2.2 Instance Interaction Unit

Its mission is to model the data presentation of an instance and support interaction with it. It is oriented to object manipulation. In terms of the user, the Instance Interaction Unit arises out of the necessity to observe single objects. In addition, the user may want to change the state of the object (by means of services) and to navigate to related objects. In the problem domain the analyst can identify a certain class, e.g. Player. Once the class is defined, he could ask the user the following questions: *Which attributes of the object do you wish to view? What actions can be done on the object? Which additional information (relationships) do you want to reach in the interface by navigation?*

The specification of the Instance Interaction Unit is defined using three of the simple patterns:

- A *Display Set* (what properties to show),
- *Actions* (what services could be executed for the instance), and
- *Navigation*: (the links to the inheritance and aggregation relationships which it is desirable to navigate to from this instance).

In the OVID approach proposed by Roberts (1998), the concept of *view* plays a similar role to the Instance Interaction Unit. However, a view is only expressed as a stereotype of a given class. The designer must decide what to show, how it will behave, and after that, implement it.

2.2.3 Population Interaction Unit

This pattern models interaction units focused on showing sets of instances of a class, together with its potential associated actions. It deals with the necessity of working with object collections. It is composed of five smaller patterns:

- *Filters* (how to search),
- *Order Criterion* (how to order),
- *Display Sets* (what to show),
- *Navigation* (how to navigate from an object), and
- *Actions* (what can be executed on objects).

As in the previous case, the analyst is working at the problem domain and he identifies a class with its attributes, services and relationships. Later, he can ask the user to identify these smaller components bricks.

2.2.4 Master/Detail Interaction Unit

The last pattern models more complex units of interaction dealing with master/slave presentations. This interaction unit is composed, as well, by simpler units of interaction. These units of interaction play a role in the composition. For this reason, the components are divided in two logical types; On one hand, the Master Component and on the other hand the Detail Components. These components are related by means of an aggregation relationship crossed in a given direction: from master to detail. Therefore, when a Master Component changes, the

Detail Components associated also do. In business applications, this pattern appears very frequently. E.g. an invoice and the possibility of introducing the lines of this invoice in the same scenario.

In order to specify them, the analyst has to detect and create the classes, the elemental components, and finally the Master Detail Interaction Unit.

Starting off from the Master Component the analyst has to identify which kind of pattern he needs for this purpose. Here, we can have two cases. The first is an Instance Interaction Unit, where the analyst forces to show only one object. The second case could be a Population Interaction Unit if the analyst wishes to show a collection of objects. The question for the customer is: *How many objects do you wish to display in the Master part?* Once identified, the interaction unit that is going to play the Master Component role, analyst could make the questions referred to such pattern (indicated in the previous points Instance or Population Interaction Units).

For each Detail Component, the analyst has to identify which interaction unit is going to constitute it. He could ask user about it: *How many details do you wish to use?* And for each one of them: *Which are the relationships between the master and the detail?, Would you like more than one step of depth level?*

The interaction units acting as Detail Component can be of several types: an Instance Interaction Unit if the role path is univalued, a Population Interaction Unit if the role path is multivalued or a Master/Detail Interaction Unit as well, which makes it possible to have, recursively, more depth levels in the master/detail.

Summarizing, the specification of the Master/Detail Interaction Unit is expressed by means of the following concepts:

- A Master Class,
- an Interaction Unit acting as a *Master*, and
- a Role Path (side of an aggregation relationship) visible of the Master class that acts as a detail and Interaction Unit like *Details*.

2.3 System Access

Finally, the user needs entry points to the interface in order to access the user interface. To deal with this type of requirement, a tree structure is used to organize the access to interaction units: the Hierarchical Action Tree *(HAT)* (Molina, 2001-a). This abstraction provides a tree for specifying how the user can access the system.

- Each intermediate node of the tree acts as a container with a label.
- Each leaf node contains a label and a link to an interaction unit.

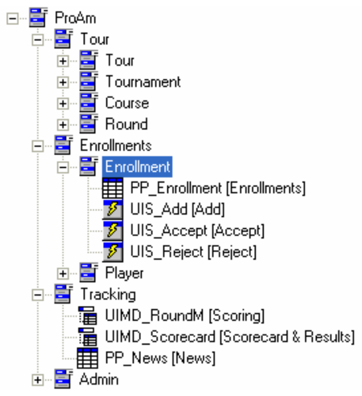


Figure 1. Example of HAT.

E.g. Figure 1 shows an example of HAT for the administrator of the Golf Tournament Tracking System.

This data is automatically mapped in the implementation phase, e.g. to menus (in windows environments) or anchors and links (in web environments). Using a tree structure is a good technique to support the *Gradual Approach Principle* (IBM, 1992). For further references, see (Molina, 2001-a).

2.4 The Pattern Language reviewed

The patterns just presented can be arranged in three levels:

- Level 1: The *first level* contains the HAT pattern, providing the access to the application.
- Level 2: The *second level* contains the Interaction Units. The user interface is discomposed in several scenarios to support user tasks.
- Level 3: Eventually, the *third level* is composed of patterns that add additional semantics for interaction units.

Figure 2 shows the levels and the corresponding interdependences among patterns.

2.4.1 Level 1: User Access

As mentioned previously, the analyst can start the specification building a HAT for building the access to the application. The analyst can be supported in this stage by Use Cases (Constantine, 1999), Task Analysis (Paternò, 2000) decomposition, or basing its construction on the pre-located Interaction Units.

2.4.2 Level 2: Interaction Units

Each Interaction Unit in the User Interface is described here. The analyst creates an Interaction Unit for each scenario needed in the User Interface. Depending of the task to be accomplished in the scenario, a different kind of Interaction Unit should be selected.

2.4.3 Level 3: Elemental Patterns

Finally, depending on the kind of Interaction Unit selected, a Interaction unit can be decorated using additional patterns to provide further semantic. Patterns in this third level are more specific and only can be applied to particular types of Interaction Units.

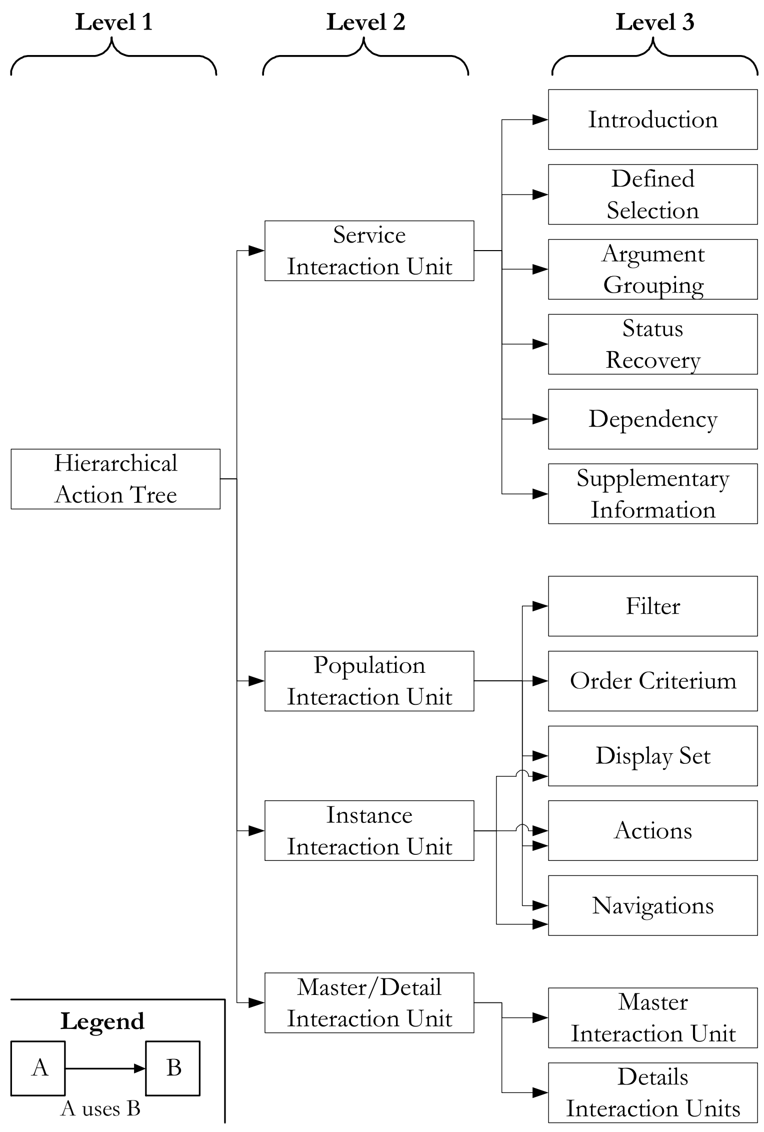


Figure 2. Pattern Language and use relationships.

This pattern language structured in three levels can be used to specify abstract user interfaces.

2.5 Navigational Diagram

Once the concepts have been introduced, a Navigational Diagram is going to be presented as a graphical representation of the stated concepts. Diagrams provide a more suitable workspace to deal with large projects and make it easier to understand and edit the user interface specification.

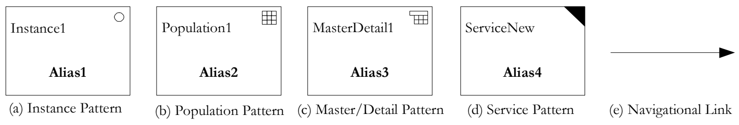


Figure 3. Navigational Diagram Graphical Primitives.

The proposed diagram is a directional graph where nodes constitute Interaction Units (represented as boxes) and arcs represent navigational links between a pair of Interaction Units (represented as arrows).

Figure 3 shows the representation of each concept. Interaction Units are represented as boxes with a small glyph in the right-upper corner. The glyphs employed are a circle for Instance IU, a grid for Population IU, a rectangle and a grid for the Master/Detail IU and a black triangle for Service IU.

An example of the Navigational Diagram is shown in Figure 13. The diagram shows the scenarios specified to be present in the User Interface and the navigational relationship among them.

2.5.1 Navigational Diagram Editor

The specification editing tasks could be very easy if a graphical editor for diagrams is available. A prototype of the editor (Just-UI/VISIO) has been built programming in the VISIO (Microsoft, 2002) environment (Figure 4). In Just-UI/VISIO, the left part of the window contains a component palette (called *Stencil* in VISIO terminology) and the right part of the window contains the drawing zone (work-area for diagrams). The user can select Interaction Units components on the palette and perform a *drag & drop* operation over the drawing zone. When this occurs, a modal window appears to ask for the properties of the new component, such as name, alias, class, etc. The navigational links between presentation components are displayed as arrows.

Diagrams can be saved, edited or printed using the powerful environment offered by VISIO.

When the specification is bigger enough, visibility facilities in the corresponding editor must be employed to keep the specification maintainable. E.g. as in a GIS (*Graphical Information System*) map, users could *select & clip* a region to show, hiding anything outside the region. User could hide or show details in the diagram. To show the part of the diagram related to a specific Interaction Unit, the given pattern and its related neighbourhood can be expanded by the user (expansion as it is done in the expansion of a branch in a tree widget). The user can follow and check navigational paths through expansion.

These facilities keep the editor usable, reducing the elements' overwhelm and maintaining the attention focused in the current work. These strategies follow the principle of *Gradual Approach* as stated in CUA (IBM, 1992).

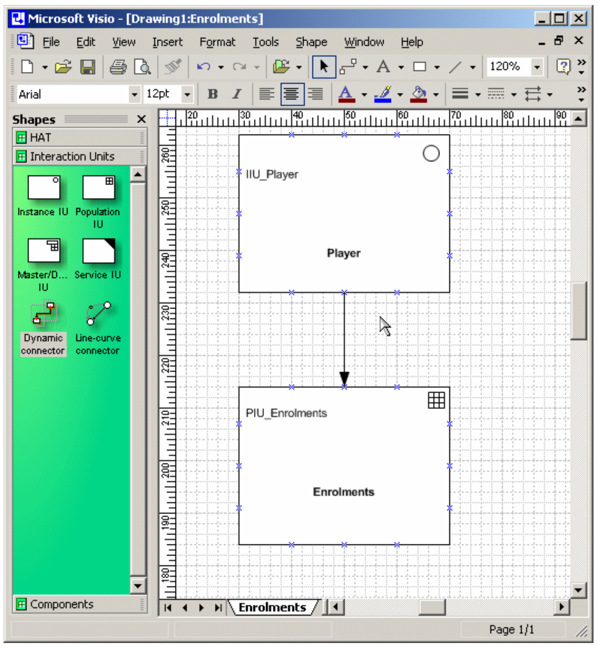


Figure 4. Just-UI environment prototype.

2.6 Just-UI Meta-model

The Just-UI Meta-model extends a classical object-oriented meta-model as the OO-Method meta-model (Pelechano, 1998) or the UML meta-model (OMG, 1997-a & 1997-b). As shown in Figure 5, the only prerequisites needed for applying Just-UI is that the concepts: `Class`, `Attribute`, `Service` (or method) and `Argument`; and its usual relationships were available in the kernel of the meta-model.

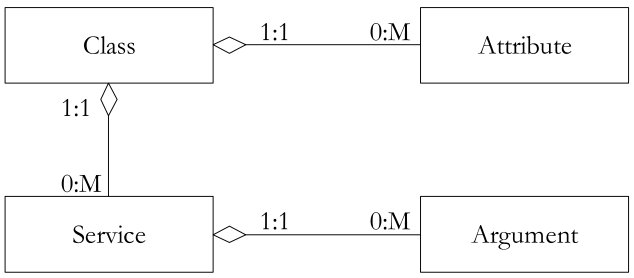


Figure 5. Basic meta-model kernel for an OO Model.

The extension mechanism is based on introducing news elements: the patterns and its supporting classes and relationships. The link between patterns and traditional concepts is achieved using two kinds of relationships (Figure 6):

- *Definition.* A pattern is always defined for a unique concept. However a concept can define different patterns at the same time. This kind of relationship is established at the creation time of the pattern. E.g. Filters, Display Sets, and Navigation are defined for a Class. Service Interaction Unit is defined for a Service.

- *Application.* A pattern can be applied to third concepts. In this sense, patterns can be reused in the model (applied in different scenarios). Application relationships are established by the analyst when building the model. E.g. Filter, Display Set, and Navigation are applied (or used) in a Population Interaction Unit.

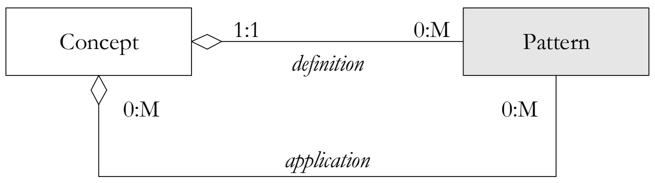


Figure 6. Pattern extension mechanism.

The concept of Interaction Unit is sub-typed in four subclasses (Figure 7). Each subtype redefines and extends properties and behaviour.

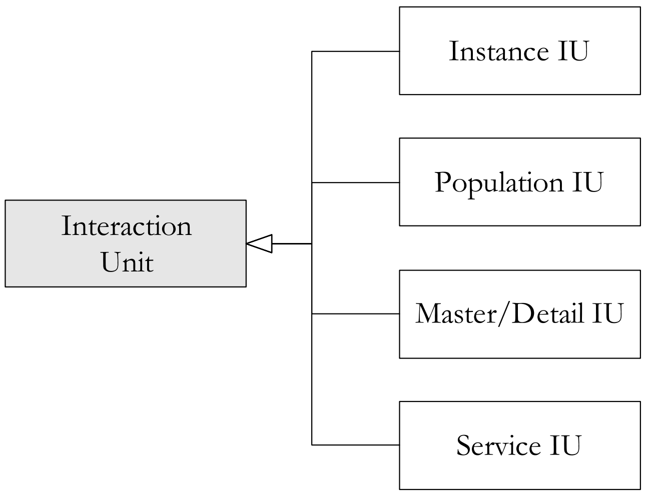


Figure 7. Interaction Unit sub-types.

The meta-model related with Instance and Population Interaction Units are described in Figure 8. In such a figure, Instance and Population IUs are *defined* for classes. Display Set, Action and Navigation are *applied* to Instance and Population IU. Filter and Order Criterium are also applied to Population IU.

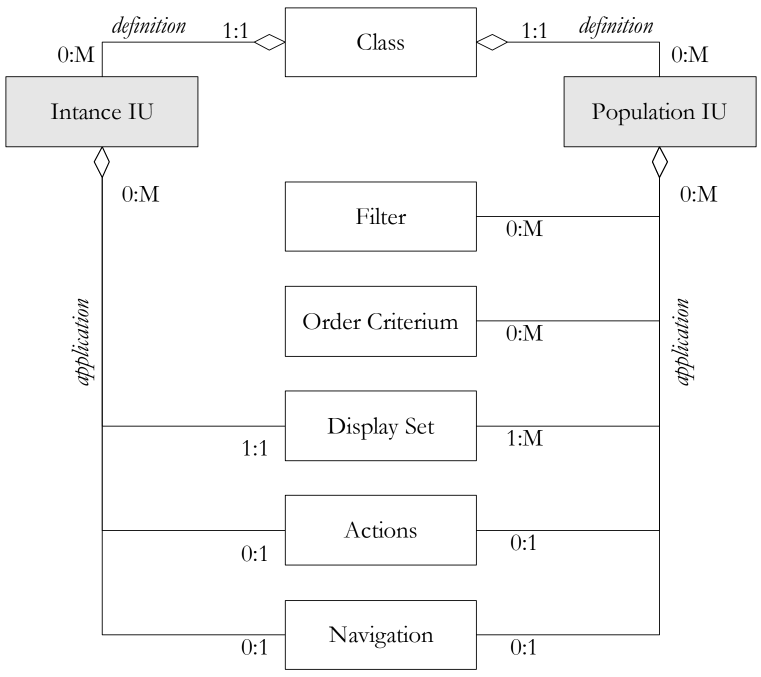


Figure 8. Meta-model for Instance and Population Interaction Units.

The meta-model for Master/Detail Interaction Unit is shown in Figure 9. Master/Detail IU is defined for a given Class. Instance and Population UIs can play the role of Master and Details. However, Master/Detail UI can only play a Detail role.

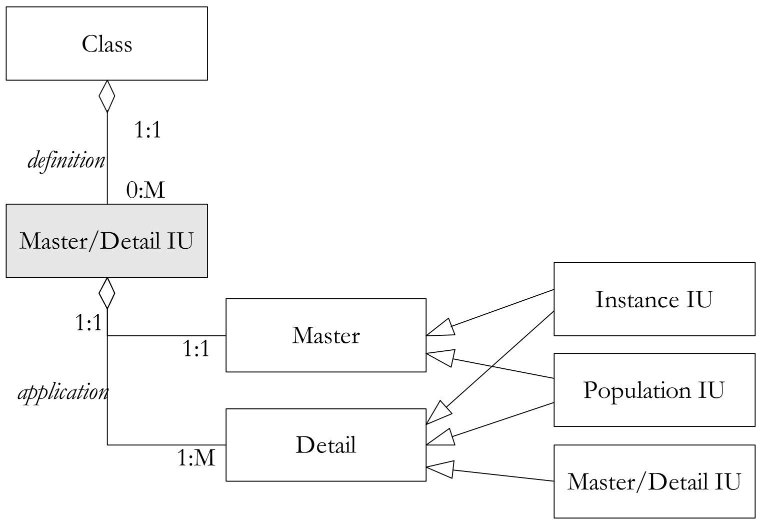


Figure 9. Meta-model for Master/Detail IU.

Figure 10 shows the meta-model for Service Interaction Unit. Service IUs are *defined* for Services. Auxiliary patters (like Introduction, Dependency, etc. not showed here for brevity) can also be applied to Service IU.

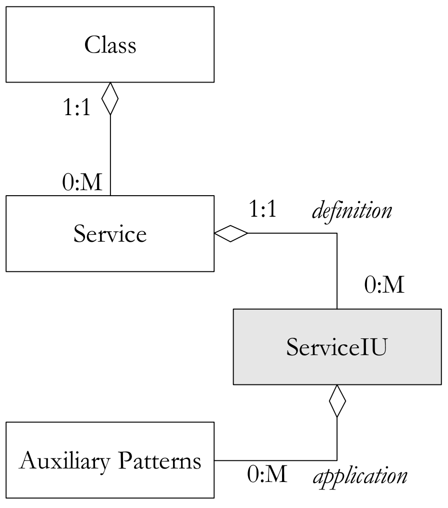


Figure 10. Meta-model for Service Interaction Unit.

In this way, we obtain a unified meta-model comprising the user interface, functional, and structural O-O specifications. The use of *definition* and *application* relationships allow for a high level of reusing of the patterns specified in the model. The reuse at the Conceptual level makes easier to maintain the homogeneity in the specification, and consequently, in the final implemented system.

3 A Case of Study: Golf Tournament Tracking System

To clarify the approach, and put the corresponding ideas into practice, a brief case of study based on a Golf Tournament Tracking System *"ProAm"* is going to be introduced. This system is successfully been used in one of the tournaments of the *Tour de España* every year. The example manages information about golf players, tournaments, rounds, scores, flights, controllers, etc. The goals are to track the enrolments, the score of each player and the final classification of several tournaments in a golf championship. The system is too big to be fully described. Therefore, we will describe only a subsystem to exemplify the concepts employed related to the user interface part.

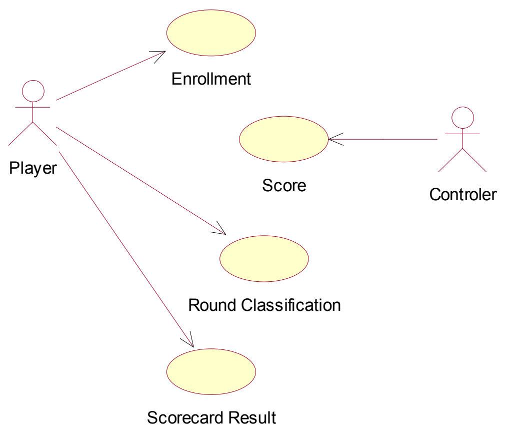


Figure 11. Use Case for Player and Controller agents.

3.1 Analysis and User Tasks

Analysis of the systems starts with a Use Cases definition. A subset of the functionality is going to be introduced as an example. Figure 11 shows a basic Uses Cases for Player and Controller actors. Actually, the system has defined more actors like Administrators or UserGolf. However, for this example, only players and controllers will be described. Going deeply, each Use Case can be described with more detail.

Use Case:	*Enrollment*
Actors:	Player
Type:	Primary
Descriptions:	Players must be enrolled for a Tournament. They must select the tournament from the championship and fill an enrollment form. After the inscription has been done the player is allowed to play in the tournament.
Preconditions:	Player must not be enrolled yet.
Frequency:	Low. A player is enrolled once for a tournament.

Table 1. Enrollment Use Case.

The first one (Table 1) corresponds to the enrollment task. Before each tournament, participants must be inscribed in the competition. Each player must fill an inscription form in the tracking system.

Use Case:	*Score*
Actors:	Controller
Type:	Primary
Descriptions:	Controllers input player's scores in the system using PDAs in real-time. A controller is in charge of three or four holes in a round.
Preconditions:	A score for a given player and hole is done only once.
Frequency:	High. A score is produced for every player in each hole in a round.

Table 2. Score Use Case.

The second Use Case (Table 2) describes the controlling task. A controller must track three or four holes in the course, checking player scores and write them down in the tracking system. Controllers use PDAs (*Portable Device Assistant*) to introduce the data in the system in real time. PDAs are connected with a central server using a wireless network.

Use Case:	*Round Classification*
Actors:	Player
Type:	Primary
Description:	At any moment, players can check the results and provisional classification of the current journey. The system must provide such classification in real-time.
Preconditions:	Round must been started.
Frequency:	Medium. Players usually look at this information three or four times per journey.

Table 3. Round Classification Use Case.

Next Use Case (Table 3) is used by players. When a player finished his flight (a play in a journey) he can follow the classification using the tracking system. A player needs to look at the overall classification for a given round.

Use Case:	*Scorecard Result*
Actors:	Player
Type:	Primary
Descriptions:	At any moment, players can look at their scorecard. The scorecard shows the score for each hole in the field.
Preconditions:	None.
Frequency:	Medium. A scorecard is produced for every player in each controlled hole in a round.

Table 4. Round Classification Use Case.

Finally, the last one Use Case (Table 4) refers to the scorecard observation task. A player can focus on a particular scorecard to check the shots in different holes.

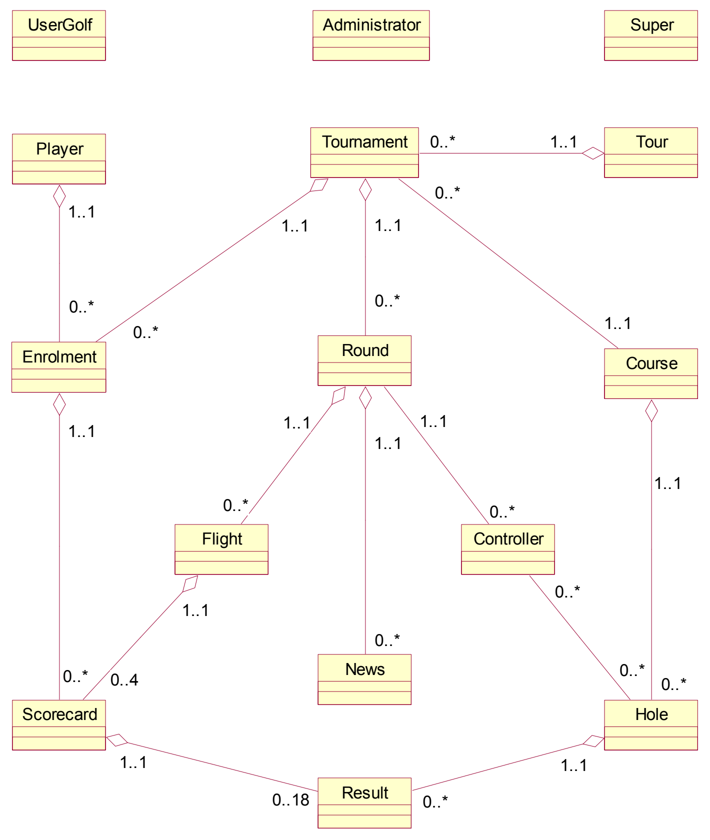


Figure 12. Class Diagram for the ProAm Schema.

Afterwards, analysts have built a Class Diagram to capture the analysis classes in the system (Figure 12). Attributes and methods are not shown for brevity. The main classes of the systems are *Player* (to maintain player information), *Tour, Tournament, Round, Flight* (different divisions in the competition), *Course*, *Hole* (representing the course and its holes), *Enrolment* (a player plays in a tournament), *Scorecard* (the aggregated score in the tournament) and *Result* (the score in a given hole).

At this stage, analysts can start building the Navigational Diagrams derived from the Use Cases. Different actors can have a different User Interface to achieve his tasks. In this example, analysts have decided to create to different interfaces: one for players, and a second one for controllers.

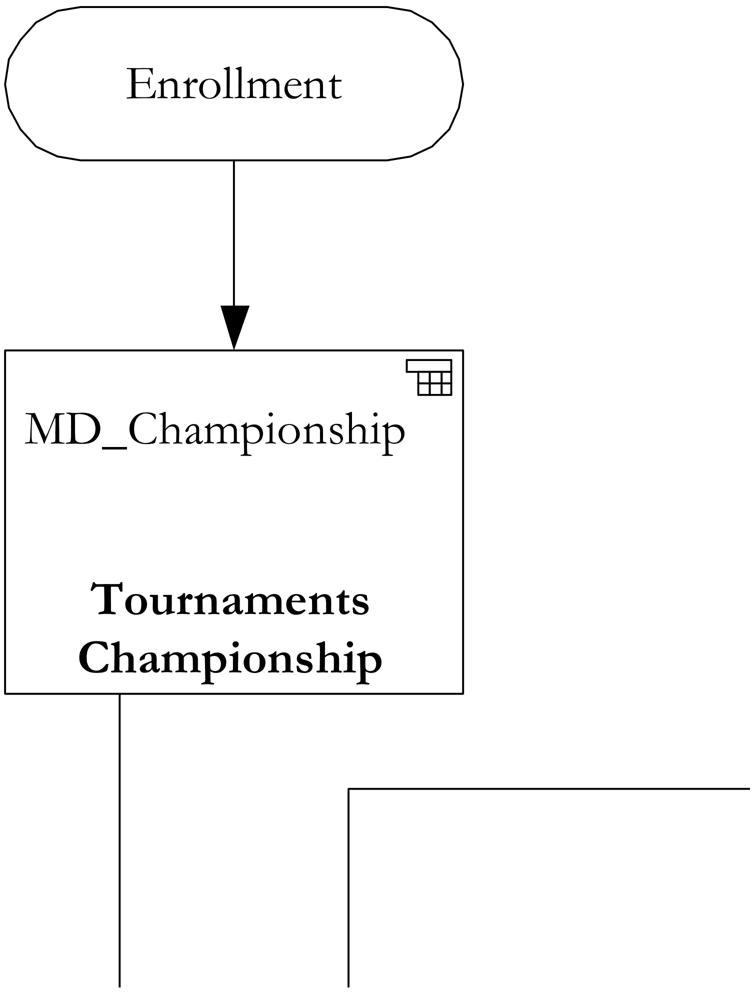


Figure 13. Navigational Diagram for Player actor.

Figure 13 shows the final Navigational Diagram for the Player User Interface specified. In this user interface specification, we have followed the ideas of Lausen (2001) applying reuse techniques to achieve a more compact and usable interface. Therefore, the specification has three entry-points (marked using round boxes): one entry for each task described in the Uses Cases. The diagram shows seven Interaction Units that allow to perform all the tasks requested by players.

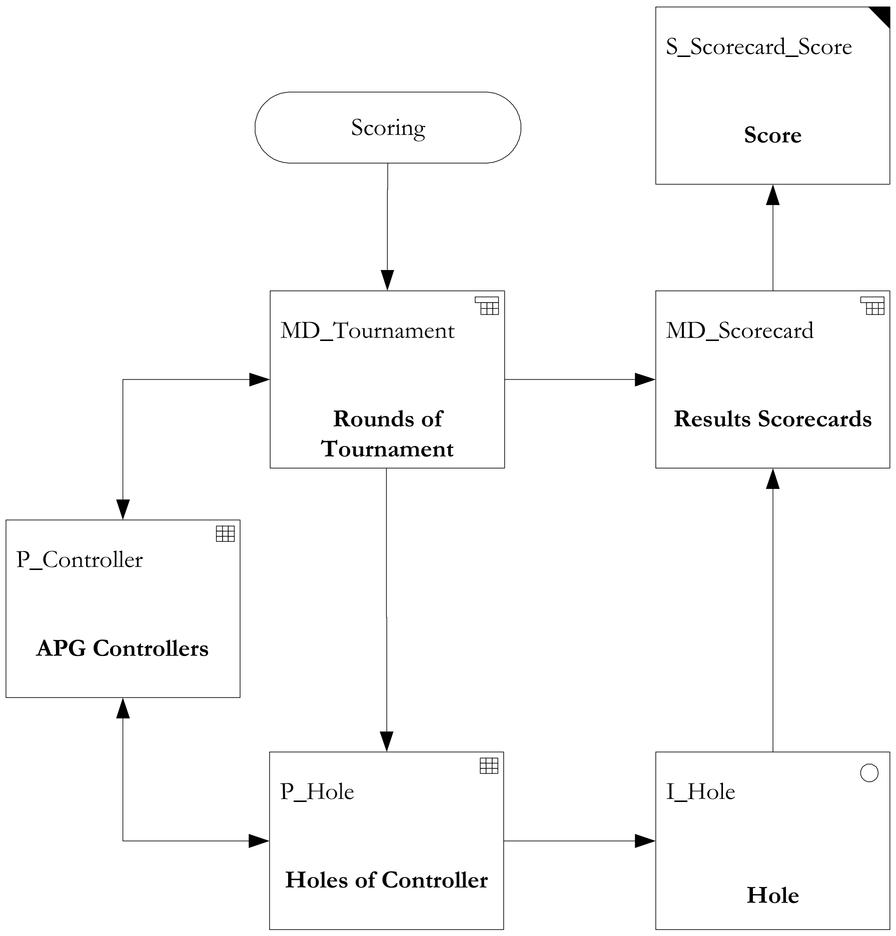


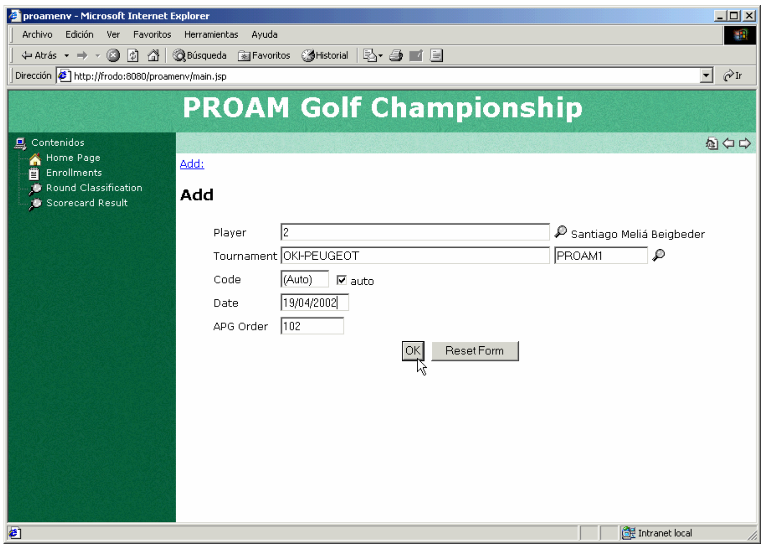
Figure 14. Navigational Diagram for Controller actor.

In the same way, Figure 14 shows the user interface specified for Controllers. Controllers need a small user interface adapted to their main task: Score (verify and introduce the players' scores) and device employed: PDA (*Personal Device Asistance*).

This kind of diagram can be explained to the customer as a paper sketch of the user interface. The main goal of doing in this way is to achieve an early validation of the user interface requirements.

3.2 Implementation

The implementation proposed has been obtained using code translators for obtaining a prototype of the user interface. A pair of implementations (Windows and Web) is presented as example of final user interfaces implemented in different platforms.


proamenv - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección http://frodo:8080/proamenv/main.jsp
PROAM Golf Championship
Contenidos
Home Page
Enrollments
Round Classification
Scorecard Result
Intranet local

Add:

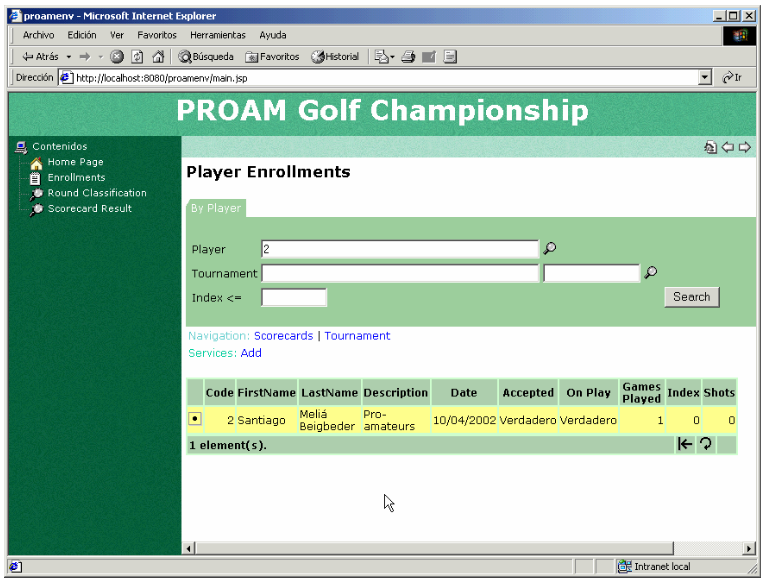
Add

Player	2	Santiago Meliá Beigbeder
Tournament	OKI-PEUGEOT	PROAM1
Code	(Auto)	✓ auto
Date	19/04/2002	
APG Order	102	

OK Reset Form

Figure 15. Enrolment Form.

In the use case *Enrolment* in which *Player* is enrolled for a *Tournament*, a Web User Interface has been produced. Players can be enrolled for a Tournament and fill an enrolment in a web form. Figure 15 shows the Enrolment Form to inscribe players in the competition. This form implements the Service Interaction Unit and launches the *Add Service* in the Enrolment class.


proamenv - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección http://localhost:8080/proamenv/main.jsp
PROAM Golf Championship
Contenidos
Home Page
Enrollments
Round Classification
Scorecard Result
Intranet local

Player Enrollments

By Player

Player	2	
Tournament		
Index <=		

Search

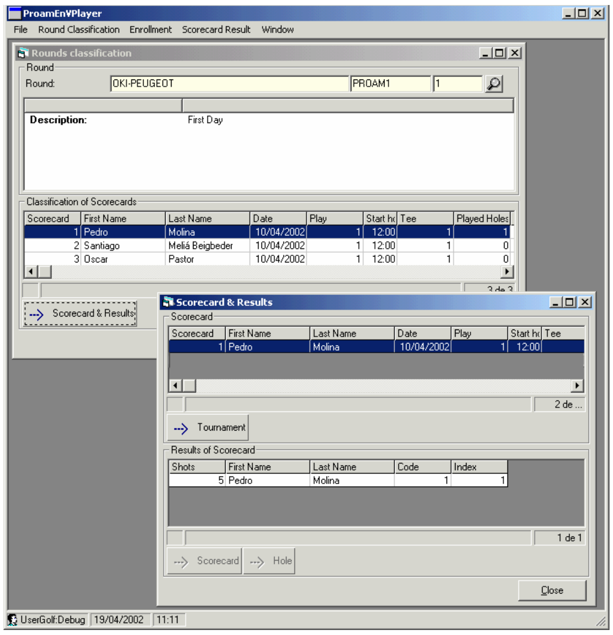
Navigation: Scorecards | Tournament
Services: Add

	Code	FirstName	LastName	Description	Date	Accepted	On Play	Games Played	Index	Shots
●	2	Santiago	Meliá Beigbeder	Pro-amateurs	10/04/2002	Verdadero	Verdadero	1	0	0

1 element(s).

Figure 16. Enrolments Population Interaction Unit implementation.

Once a player is enrolled, the scenario showed in Figure 16 implements a Population Interaction Unit for displaying Enrollments data. Players can use the filter to search by player, tournament and/or index. The filter is implemented simulating a tab labelled *"By Player"*. After searching, the enrolments satisfying the searching criteria will appear formatted using an HTML table.



Round: OKI-PEUGEOT PROAM1 1

Description: First Day

Scorecard	First Name	Last Name	Date	Play	Start h	Tee	Played Holes
1	Pedro	Molina	10/04/2002	1	12:00	1	1
2	Santiago	Meliá Beigbeder	10/04/2002	1	12:00	1	0
3	Oscar	Pastor	10/04/2002	1	12:00	1	0

3 de 3

Scorecard	First Name	Last Name	Date	Play	Start h	Tee
1	Pedro	Molina	10/04/2002	1	12:00	

2 de ...

Shots	First Name	Last Name	Code	Index
5	Pedro	Molina	1	1

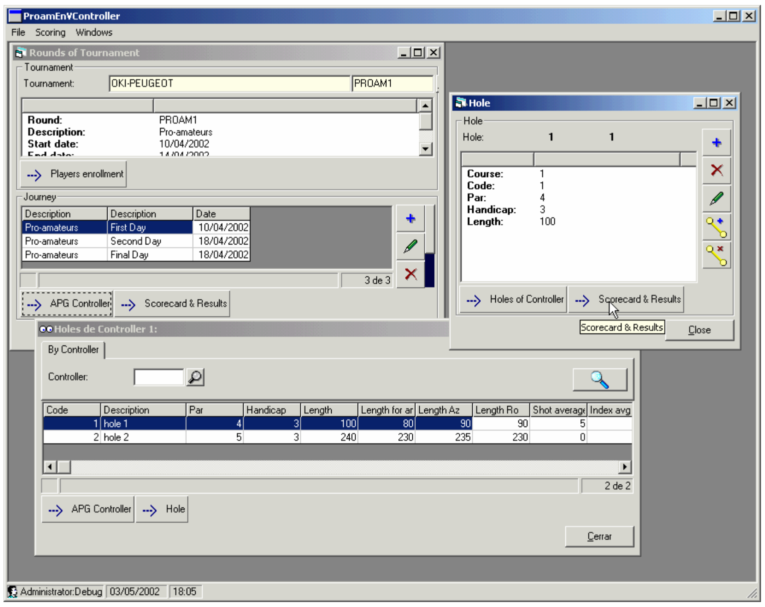
1 de 1

UserGolf:Debug 19/04/2002 11:11

Figure 17. Examples of Master/Detail Interaction Units implementation.

On the other hand, Figure 17 shows the Windows implementation for a pair of Master/Detail Interaction Units. The first child window (on the background) displays Scorecards for a given rounds. The second one child window (on the foreground) displays Scorecards and its related Results. In both implementations, objects can be selected in order to navigate to related data (implemented as an horizontal toolbar in Windows and links in the Web solution). Actions, were available, are also showed (implemented as a vertical toolbar in Windows and links in the Web solution).

The next example shows part of the user interface for controllers. Figure 18 describes how a controller in a Master-Detail selects a journey in the tournament (first background child window). For each journey, controllers can check the holes they have been assigned to (bottom child window). Furthermore, controllers can select a hole and ask for details using an instance interaction unit for showing Hole information (foreground right child window).


ProamEnVController

File Scoring Windows

Rounds of Tournament

Tournament

Tournament: OKI-PEUGEOT PROAM1

Round: PROAM1
Description: Pro-amateurs
Start date: 10/04/2002

Players enrollment

Journey

Description	Description	Date
Pro-amateurs	First Day	10/04/2002
Pro-amateurs	Second Day	18/04/2002
Pro-amateurs	Final Day	18/04/2002

3 de 3

APG Controller | Scorecard & Results

Hole

Hole

Hole: **1** **1**

Course: 1
Code: 1
Par: 4
Handicap: 3
Length: 100

Holes of Controller | Scorecard & Results

Scorecard & Results | Close

Holes de Controller 1:

By Controller

Controller:

Code	Description	Par	Handicap	Length	Length for ar	Length Az	Length Ro	Shot average	Index avg
1	hole 1	4	3	100	80	90	90	5	
2	hole 2	5	3	240	230	235	230	0	

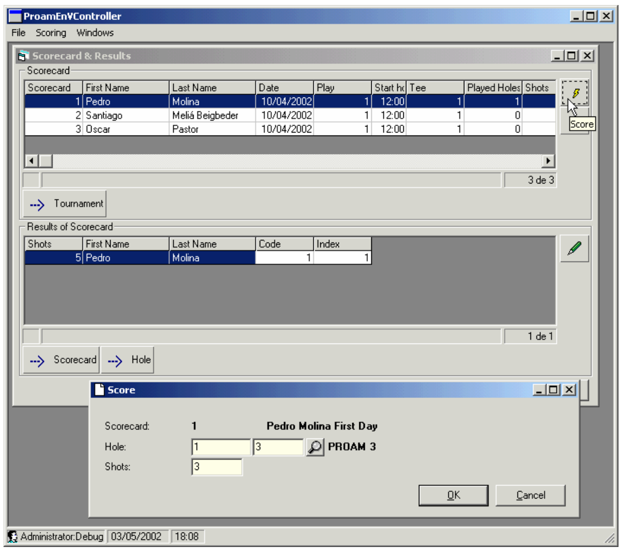
2 de 2

APG Controller | Hole

Cerrar

Administrator:Debug 03/05/2002 18:05

Figure 18. Part of the controllers user interface.


ProamEnVController

File Scoring Windows

Scorecard & Results

Scorecard

Scorecard	First Name	Last Name	Date	Play	Start ho	Tee	Played Holes	Shots
1	Pedro	Molina	10/04/2002	1	12:00	1	1	
2	Santiago	Meliá Beigbeder	10/04/2002	1	12:00	1	0	
3	Oscar	Pastor	10/04/2002	1	12:00	1	0	

Score

3 de 3

Tournament

Results of Scorecard

Shots	First Name	Last Name	Code	Index
5	Pedro	Molina	1	1

1 de 1

Scorecard | Hole

Score

Scorecard: **1** **Pedro Molina First Day**
Hole: 1 3 **PROAM 3**
Shots: 3

OK | Cancel

Administrator:Debug 03/05/2002 18:08

Figure 19. Scorecards result and score service.

Finally, Figure 19 deals with the main task for controllers: scoring the results of scorecards. The first child window is a Master/Detail that allows to select scorecards and to display its results. The second child window is used for the score service: for a given player and hole the controller fills in the input form the number of shots done.

4 Generating the User Interface

The specification collects the user interface requirements expressed in terms of the problem domain. It constitutes a good documentation of the system itself. This specification can be also used to generate code in order to implement the user interface for the Information System. Depending on the target platform, different translations must be done. In CARE Technologies S.A. we have developed translators to produce automatically such user interfaces. For the moment, we have developed translators for the following target languages (and platforms): Visual Basic (Windows), Java (Swing), JSP, ASP, and ColdFusion (Web pages). Depending on the target environment, a different style guide is used to obtain a homogeneous user interface with respect to other applications.

Each pattern in the Model can be expressed in terms of AIOs (Vanderdonckt, 1993) (*Abstract Interface Objects*) and behaviour attached to such AIOs. In the translation phase these AIOs are mapped to CIOs (*Concrete Interface Objects*, controls or widgets) following a table of transformations based on ergonomic rules (Vanderdonckt, 1993) & (Molina, 1998).

The implemented systems we produce follow the classical n-tier architecture. The User Interface layer acts as a client of a Business-Logic layer acting as a server in a client-server paradigm. Therefore, the user interface contains the code to make calls to the Business-Logic layer: the responsible for executing the services and maintaining the consistency of the data. Code for error trapping is also produced to isolate the user interface for any potential failure that could occur in communications or in other layers.

Figure 20 shows the three/five layers architecture for desktop applications and web environments, respectively. Persistence layer is currently mapped to RDBMS (*Relational Data Base Management Systems*). However, it could be replace by other technology like XML or object oriented databases. Business logic components can be automatically translated from the specification to Microsoft COM+ objects or, alternatively, Enterprise Java Beans for Java. On the other hand, the client part implements the Graphical User Interface.

The communication between client and server components is performed using an RPC (*Remote Procedure Call*) protocol based on XML (*eXtensible Markup Language*) messages over HTTP (*Hyper-Text Transfer Protocol*) as in XML Protocol (World Wide Web Consortium, 2000). Doing it by this way, it provides full inter-operation: clients with servers implemented with different languages or in different platforms.

Traditionally, desktop interfaces are more powerful than Web based ones. Desktop GUIs can employ the full set of widgets and guidelines available for such platform. Rich and mature IDEs (*Integrated Developments Environments*) are available to support the design and implementation phase.

On the other hand, Web based User Interfaces disperses its functionality across three different physical layers: the Server Application (ASJ, JSP, ColdFusion, PHP, etc.), the Web Server (like Apache or IIS) and the final User Agent (Netscape, Internet Explorer, or Opera Web browsers, WAP devices, PDA device, etc.). The diversity of technologies of the user device has to be dealt in order to adapt the user interface to the device capabilities accordingly.

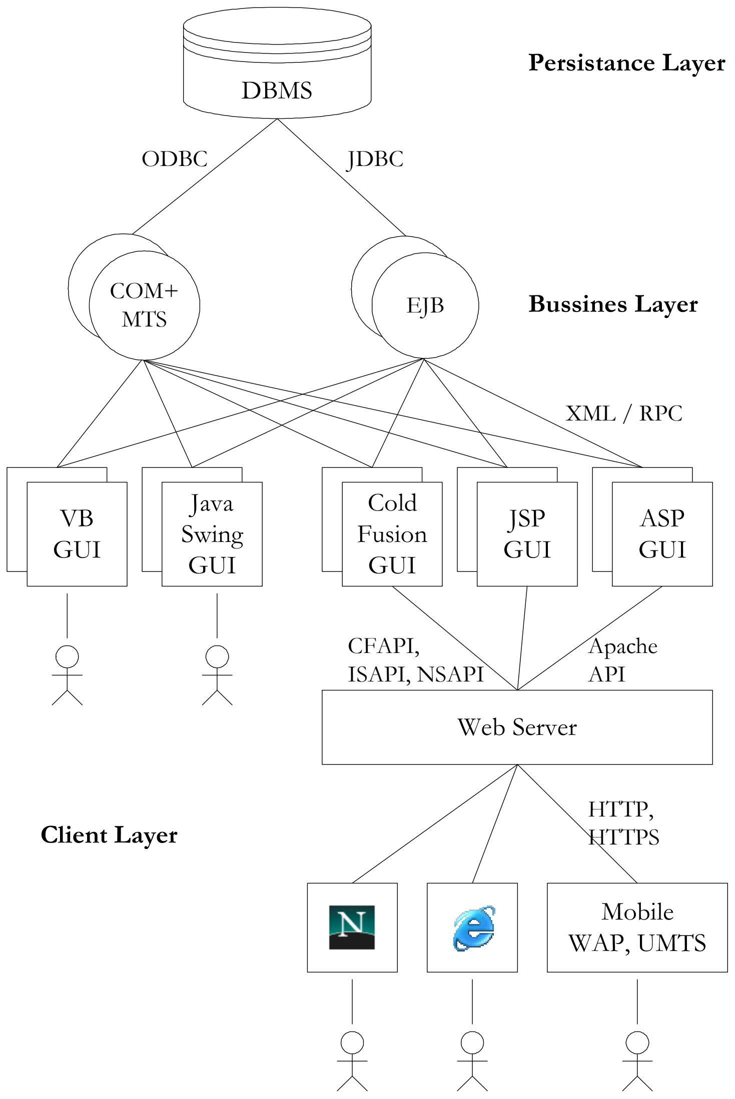


Figure 20. Translated Application Architecture.

In any case, the presented approach separates clearly conceptual interface specification from concrete, technology-dependent implementation. In consequence, it is possible to reificate a given user interface specification in any target software development platform, just by defining and implementing the mapping between conceptual user interface primitives, and their corresponding particular software representation.

5 Related Work

In the area of requirement elicitation, Ruble (1997) presented a simple notation useful for describing navigational maps of User Interfaces. Such a notation employs boxes as representation of windows and arrows as links between windows. Ruble also proposed working with the customer to build such navigational diagrams. On the other hand, Constantine & Lockwood's (1999) proposes the Essential Use Cases for gathering User Interface requirements. In a similar way as Ruble, Constantine & Lockwood recommend to work with the user in order to search the user interface requirements. In such sessions, they recommend to employ: paper, pens and Post-Its. Each Post-It represents a window for supporting user tasks. User and analysts work together for rearranging such post-its in the paper and drawing lines to discover the navigation paths needed.

Just-UI is inspired in these two works for providing an easy notation to work with: easy to understand, and intelligible by users (in the same way as Use Cases are). Nevertheless, we propose to extend such specification primitives with a more precise semantic in order to support code generation in the subsequent stages.

Lausen (2001) proposed a method for designing user interfaces mainly based on user tasks and scenarios. Once the scenarios are built, Lausen proposed to reuse and reduce the scenarios to simplify the user interface in order to increase the usability and learnability. Ideas from this work were also followed to design the example provided in this work.

Respect to the specification part, works like TRIDENT (Bodart, 1994), Janus (Balzert, 1996), and Teallach (Barclay, 1999), are well known examples of MB-IDEs providing user interface generation capabilities. However, no previous work has been focused on a pattern oriented approach to the specification of user interfaces.

Granlund (2001) have developed the PSA (*Pattern Supported Approach*) to cover the User Interface Design Process. In such approach, they advocate for using patterns at different levels as medium for collecting experience and document the system. We absolutely agree with them, but we also intent to use patterns for supporting code generation.

On the other hand, OVID (Robberts, 1998) and WISDOM (Nunnes, 2000) are methods to extend classical UML models (OMG, 1997-a) for user interface specification. The concepts of *View* in the former and *Interaction Spaces* in the later play a similar role to the *Interaction Unit* concept here introduced. However, both concepts have no additional conceptual behavioural semantic and must be added by the designer or implementer.

In the field of user interface generation, SEGUIA (Vanderdonck, 1999-a & 1999-b) is a semi-guided expert system that helps designers to select best designs and layouts. In SEGUIA, AIOs are mapped to CIOs based in a rule-based language. Otherwise, as a commercial product, Genova is a generator plugging for Rational Rose from Genera AS (2001). Genova helps to produce windows from UML class diagrams and allows to parameterise different design aspects. Such parameterisation can be stored, and therefore, be reused. However, Genova do not provides additional semantic for building IU specifications. It only uses the information in the UML model (as Janus (Balzert, 1996) does with OOA models).

6 Conclusions

A model for the specification of user interfaces based in Conceptual Interface Patterns and a graphical notation has been presented.

Just-UI provides a set of patterns that can be used as building blocks to create user interface specifications for Information Systems. The whole concepts used are located in the problem domain, thus allowing an easy matching of requirements to the user interface model. Analysts can create complete specification with very little effort. Besides, graphical designers can model without needing to be expert analysts. Users can also understand and review the navigational diagrams with analysts in order to validate the user interface specification.

The specification model is strategy neutral. Analyst can freely employ top-down, bottom-up approaches, or even, a mixed approach as well to build the abstract user interface. Note that, this article starts presenting the patterns following a bottom-up strategy and then, deliberately, subsection 2.4 summarizes the pattern language in a top-down way, just for the fun of view the same problem using a different point of view (*top-down*).

The specification obtained with Just-UI is also platform neutral. The specifications do not contain any design details. In this way, the specification is reused to provide similar user interfaces across several target platforms. At the same time, the concepts used have direct translation to the final implementation. Each presentation pattern is reified (mapped) to windows, forms, web pages or any other implementation of presentation unit that can be presented in the solution domain.

The need to obtain tailored user interfaces depending on the final platform (for example PDAs or UMTS devices) is beyond the scope of this paper, but this can be naturally achieved by providing design tools for specific environments capable of refining the initial Just-UI specification to add and maintain such design topics.

Using Just-UI to specify the user interface in conjunction with OO-Method to gather the functionality of the system opens the door to automatically obtaining complete prototypes of Information Systems, copying with Data Persistence, Business-Logic, User Interface and User Help.

If the specification is complete enough and non ambiguous, more and more code could be generated as final application code without additional manual changes. Therefore, the maintenance phases of software can be moved from code to specification as it was stated by Balzer (1983) in the *Automatic Programming Paradigm*.

Acknowledgments

We would like to thanks to the R&D Department of CARE Technologies and to the Department of Information Systems and Computation of the Valencia University of Technology for supporting this work.

References

Balzer, R., Cheatham T.E. and Green C. (1983). "Software Technology in the 1990s: Using a New Paradigm". *IEEE Computer*, pp 39-45, November 1983.

Balzert H. (1996). "From OOA to GUIs: The Janus system". *IEEE Software*, 8(9), pp 43-47, February 1996.

Barclay P.J., Griffiths T., McKirdy J., Paton N.W., Cooper R., and Kennedy J. (1999). "The Teallach Tool: Using Models for Flexible User Interface Design" In *Proceedings of CADUI'99.* pp 139-158, Kluwer Academic Publishers, 1999.

Bodart F., Hennebert A.M., Provot I., Leheureux J.M., and Vanderdonckt J. (1994) "A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype". *Proc. of the Eurographics Workshop on Design, Specification, Verification of Interactive Systems*, Carrara, Italy, Focus on Computer Graphics, Springer-Verlag, pp 77-94, Berlin, June 1994.

Constantine L. and Lockwood L. (1999). *Software for use. A practical Guide to the Models and Methods of Usage-Centered Design.* Addison Wesley, 1999.

Genera AS. (2001) *Genova 7.0.* http://www.genera.no/2052/tilkunde/09.04/default.asp [on-line]

Granlund Å., Lafrenière D., and Carr. D.A. "A pattern-supported approach to the user interface design". In *Proceedings of HCI International 2001, 9th International Conference on Human-Computer Interaction*, pp 282-286, New Orleans, LA, USA, August 2001. [on-line] http://www.sm.luth.se/csee/csn/publications/HCIInt2001Final.pdf.

IBM. (1992). *Object Oriented Interface Design: IBM Common User Access Guidelines.* Que, Carmel, Ind. 1992.

Lausen S. and Harning M.D. (2001) "Virtual Windows: Linking User Tasks, Data Models & Interface Design". *IEEE Software*, pp 67-75, July-August 2001.

Microsoft Corp. (2002). *VISIO* [on line] http://www.microsoft.com/visio, 2002.

Molina P.J. (1998). Master Science Thesis. *User Interface Specification in OO-Method (Especificación de Interfaz de Usuario en OO-Method)* [in spanish] DISC, Technical University of Valencia, Valencia, Spain, 1998.

Molina P.J., Pastor O., Martí S., and Insfran E. (2001-a). "Ingeniería de Requisitos aplicada al modelado conceptual de interfaz de usuario" [in spanish]. In *Proceedings of IDEAS'2001* (Ibero-american Workshop on Requirement Engineering and Software Environments), pp 181-192, Ed. CIT, Santo Domingo, Heredia, Costa Rica, April 2001.

Molina P.J., Pastor O., Martí S., Fons J., and Insfran E. (2001-b). "Specifying Conceptual Interface Patterns in an Object-Oriented Method with Code Generation". In *Proceedings of User Interfaces for Data Interactive Systems UIDIS 2001*, pp 72-79, Zurich, Switzerland, IEEE Computer Society, May 2001.

Molina P.J., Meliá S., and Pastor O. (2002). "Just-UI: A User Interface Specification Model". In *Proceedings of 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002* (Valenciennes, France, 15-17 May 2002.). Kolski Ch. & Vanderdonckt J. (eds.), pp 63-74, Kluwer Academic Press, Dordrecht, 2002.

Nunnes N.J. and Cunha J.F. (2000). "Wisdom: A Software Engineering Method for Small Software Development Companies" In *IEEE Software*, pp 113-119, September 2000.

OMG (1997-a). *UML Notation.* Version 1.1, OMG document ad/97-08-05, September 1997.

OMG (1997-b). *UML Semantics.* Version 1.1, OMG document ad/97-08-04, September 1997.

Pastor O., Hayes F., and Bear S. (1992). "OASIS: An OO Specification Language". In Proceedings of CAiSE-92, *Lecture Notes in Computer Science* 593, pp 348-363, Springer-Verlag, Manchester (UK), 1992.

Pastor O., Insfrán I., Pelechano V., Romero J., and Merseguer J. (1997). "OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods". In Proceedings of 9th International Conference CAiSE97, *Lecture Notes in Computer Science* 1250, pp 145-159, Springer-Verlag, Barcelona, Spain, June 1997.

Pastor O., Molina P.J., and Aparicio A. (2000). “Specifying Interface Properties in Object Oriented Conceptual Models”. In *Proceedings of Working Conference on Advanced Visual Interfaces, AVI 2000*, pp 302-304, Palermo, Italy. Ed. ACM, May 2000.

Paternò F. (20002). *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.

Pelechano V., Insfrán E., and Pastor O. (1998). “El Metamodelo OO-Method y su Repositorio Relacional” [in spanish]. *Technical Report*, DSIC. Universidad Politécnica de Valencia, 1998.

Pinheiro P., Paton N.W. (2000-a). “User Interface Modelling with UML”. *10th European-Japanese Conference on Information Modelling and Knowledge Representation*, Saariselka, Finland, May 2000.

Pinheiro P. (2000-b) “User Interface Declarative Models and Development Environments: A Survey”. *In Interactive Systems: Design, Specification and Verification* (7th International Workshop on Design, Specification and Verification of Interactive Systems), Limerick, Ireland. LNCS Vol.1946, pp 207-226, Springer-Verlag, June 2000.

Quarani T. (1998). *Visual Modeling with Rational Rose and UML*. Addison-Wesley, 1998.

Robbins J., Hilbert D., and Redmiles D. (1997). “ARGO: A Design Environment for Evolving Software Architectures”. In *Proceedings of ICSE'97*, pp 600-601, Boston, MA, ACM Press, May 1997.

Roberts D., Berry D., Isensee S., and Mullaly J. (1998). *Designing for the User with OVID: Bridging User Interface Design and Software Engineering*, MacMillan, 1998.

Ruble D.A. (1997). *Practical Analysis and Design for Client/Server and GUI Systems*. Yourdon Press Computing Series, 1997.

Szekely P. (1996). “Retrospective and Challenges for Model-Based Interface Development”. In *Computer-Aided Design of User Interfaces CADUI’96*, pp xxi-xliv, Namur, Belgium, Namur University Press, 1996.

Tidwell J. (1999). *Common Ground: A Pattern Language for Human-Computer Interface Design*. [on line] http://www.mit.edu/~jtidwell/common_ground.html, 1999.

TogetherSoft. (2002) [on line] http://www.togethersoft.com, 2002.

Van Welie M. (2000). *The Amsterdam Collection of Patterns in User Interface Design*. [on line] http://www.cs.vu.nl/~martijn/patterns/index.html, 2000.

Vanderdonckt J. and Bodart F. (1993). “Encapsulating knowledge for intelligent automatic interaction objects selection”. *Conference Proceedings on Human Factor in Computing Systems*, pp 424-429, Amsterdam, The Netherlands, ACM press, 1993.

Vanderdonckt J. (1999-a) “Advice-Giving Systems for Selecting Interaction Objects”. *Proceedings of UIDIS’99*, 1999.

Vanderdonckt J. (1999-b) “Assisting Designers in Developing Interactive Business Oriented Applications”. *HCI’99*, 1999.

World Wide Web Consortium (2000). *XML Protocol*. [on line] http://www.w3.org/2000/xp, The World Wide Web Consortium (W3C), 2000.

Zloof M. (1977). “Query-By-Example: A Data Base Language”. *IBM System Journal*. Vol 4, pp 324-343, December 1977.