

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

CONCILIATING USER INTERFACE AND BUSINESS DOMAIN ANALYSIS AND DESIGN

Isabelle MIRBEL, Violaine de RIVIERES

Projet MECOSI

Rapport de recherche
ISRN I3S/RR-2003-08-FR

Avril 2003

Conciliating User Interface and Business Domain Analysis and Design

Isabelle MIRBEL¹ and Violaine de RIVIERES²

¹ Laboratoire I3S
Route des Lucioles - BP 121
06903 Sophia Antipolis Cedex
France
Isabelle.Mirbel@unice.fr

² Amadeus sas
485 Route du Pin Montard, B.P. 69
06902 Sophia Antipolis Cedex
France
vrebuffel@amadeus.net

Abstract. User Interfaces (UI) are an essential part of most softwares, especially data intensive ones as database or Web based applications. Most of the approaches dealing with UI focus on design and implementation aspects and are driven by a specific technology. We believe UI specification has already to be taken into account through the analysis phase, because it is closely related to the application Business Domain (BD), which is apprehended in the earliest steps of the Analysis and Design Process (A&D-Pr). Moreover, during the analysis phase, the UI is described in a more abstract way and a larger variety of UI can be taken into consideration.

The UI and the BD aspects of the application have to be studied in a closely-related way. But most of the existing approaches propose a specific notation for the UI, different from the one used to define the BD. And when guidelines are associated with the notation, they focus on the UI and are poorly related to the BD. We propose an integrated approach (i) using a single notation in order to facilitate and support the integration of BD and UI aspects modeling and (ii) proposing an application model distinguishing the UI and BD aspects via separate views and clearly setting relationships between them. We choose the UML notation to illustrate our work in order to facilitate its use in any standard A&D-Pr. Our approach also includes a methodology to explain how to create the application model. Its aim is not only to model the UI from a pure technical point of view, but from a business one, providing guidelines for error management, or business rules support for instance. In this paper, we present our UML profile for UI and show how we could take advantage of it through the different phases of the A&D-Pr.

1 Introduction

Softwares evolve continuously increasing their heterogeneity and the need for integration. Replacing softwares with new ones is not acceptable, since it will

result in losses in time, money and productivity. Therefore, softwares have to be developed in a way allowing their evolution and integration with existing ones. And this constraint has already to be handled during the analysis and design phases of the development process.

Moreover, data-intensive software, as Web applications for instance, require a clear dissociation between the User Interface (UI) and the Business Domain (BD) in order to allow their accurate evolution: First of all because the BD evolves relatively independently from the way it is shown to the end-users of the software and reciprocally; Also because UI evolves faster than BD following the fast evolution of the technologies it is based on; And finally because several UIs may be developed on top of a given BD.

But on the other hand, even if it is obvious that UI and BD specifications have to be dissociated, UI relies on BD and can't be analyzed and designed fully independently from it. There are many approaches of analysis and design methodologies covering the whole development process from a software engineering point of view [6, 4] and also approaches focusing on a particular aspect of the development, as UI modeling, for instance. But they are few approaches integrating UI aspects into software engineering development processes: There are many examples of projects demonstrating the effectiveness of UML for software engineering development ; but most of them are silent in terms of UI [3].

The UI needs to be specified during the analysis and design phases of the development process and with the help of the same notation in order to reinforce the homogeneity, consistency and usability of the final software. The aim is to specify as much information as possible to avoid UI-related choices to be done when coding the software. And approaches dealing with UI focus on design and implementation aspects [8, 1] and are driven by specific technologies. They provide automatic mappings from specification to code. But automation does not reinforce usability aspects.

In [9], Ovid, an OO methodology to design UI is presented. Ovid can start from any task model built through use-cases or any other kind of task modeling technique. BD and UI aspects are not handled at the same moment. The authors assume the BD has already been defined when the UI specification starts on the contrary of our approach where we conciliate the two aspects which are studied in parallel.

We propose an integrated approach conciliating UI and BD modeling. We focus on the analysis and design phases of the development process. This approach is based on the UML notation [5], the industry standard language for object-oriented software development specification, on the contrary of approaches like [2] which are not fully UML compliant. As in [7], we define a profile to cope with UI and BD aspects. In [7], stereotypes from the RUP are adapted and enrich to cope with UI specification on the contrary of our work where we define stereotypes independently from any development process in order to allow our work to be useful in any standard Analysis and Design Process (A&D-Pr). Our

aim is to allow UI and BD specification at the same time through a dedicated profile to help in emphasizing the specificities of BD and UI while highlighting relationships and constraints between them. In our approach, we propose (i) an **Application Model** distinguishing UI and BD aspects via separate views and clearly setting relationships between them, (ii) a methodology to explain how to take advantage of this **Application Model**. Its aim is not only to provide a model of the UI from a pure technical point of view, but also from a business one: guidelines for error management, or business rules support for instance are also provided, because they have a direct impact on the usability of the software.

In this paper we focus on the behavioral aspects of UI modeling. We start by discussing how to conciliate BD and UI in a single model. We show what has to be emphasized from the BD and UI points of view and we proposed a dedicated model. Section 3 deals with our profile to extend the UML notation to support the conciliation of BD and UI in the **Application Model**. In Section 4 we show how to take advantage of the **Application Model** and the dedicated profile through in our methodology. Finally, we explain in section 5 how we validated our approach in various projects at Amadeus and then, we conclude.

2 The Application Model for User Interface and Business Domain modeling

When dealing with software development, it's especially important to well understand the end-user goals and how to achieve them. They represent what is often called the **business**. The **business** scope is larger than the scope of the software to be developed; it may even cover parts that cannot be automated. In consequence, the model that is going to be the base of the future software development is extracted from the **business** (which may be impacted by the new software): we call it the **Application Model**. To conceive efficient software that interfaces directly with the end-user, it is critical to identify the main factors impacting its understanding and its usage by an end-user. It is commonly accepted that the three main factors to achieve this goal are related to: (i) The software workflow, i.e. the series of functionality interacting with the end-user; (ii) The content and dynamic behaviors of the UI, i.e. the human computer interface; (iii) The graphical "skin" of the UI, i.e. the look and feel.

The software workflow and the content and dynamic behavior of the UI must be studied first to lay down stable foundation of the software to develop. When this base is reliable, then the look and feel and the technical aspects of the UI can be handled. In our approach, we focus on (i) the software workflow, specified with the help of a **BD view**, and (ii) the UI content and dynamic behavior, specified through a **UI view**. These two views and the relationships between them constitute the **Application Model**. Different persons contribute to the software development. Each of them plays a specific role in the development process, provide specific information to build the whole software specification and refer to it with different objectives. Therefore, it is important to have different

accurate views of the software (BD and UI) and to clearly keep the relationships between them.

2.1 The BD view

The **BD view** is an extraction of the **business**, focusing on formalizing the business covered by the software to develop. It describes the software workflow independently of any technical implementation. The model provided in the **BD view** aims at anticipating UI requirements. Therefore, we focus on diagrams useful for UI specification (we do not present all the interesting diagrams for BD specification) while studying the structural and behavioral aspects of the BD.

Structural aspect: To anticipate UI modeling through BD means to clearly set where a UI will be required by enforcing:

- The separation between BD and UI use-cases;
- The specification of relationships among use-cases: When the **business** associated with a use-case is complex, it may be difficult to understand it only through one use-case and it is usually required to split it into different use-cases to better identify its sub-functionalities. The use-case which drives others is called a **workflow** use-case. It is also important to distinguish it from standard use-cases when modeling the UI, because the general chaining among the different screens may be deduced from the BD.
- Where interactions with the end-user are needed: A UI will be required only where a human-being will be interacting with the software : **System** actors have to be distinguished from **human** actors. And among human-being interacting with the software, different UI may be used (graphical or vocal ones for instance). Therefore, associations between **human** actors and BD use-cases have to be carefully specified.

In our profile we provide use-case, actor and association stereotypes to answer this threefold need.

Behavioral aspect: To anticipate the dynamic aspect of the UI through BD modeling means at some point to describe more in detail the tasks associated with the different use-cases and especially the control flow that drives them. Of course, the goal is not to provide the different windows/screens associated with the software, nor to show the sequencing of the different windows, as we are still focusing on the BD modeling; But the kind of information used to drive the flow has to be specified: information may be provided (computed) by the system or given by the end-user. In the last case, it shows explicitly when a UI is required and anticipates the information which has to be provided through it to the system. Therefore, in our profile we provide dedicated decision stereotypes to highlight when the control flow is driven by information given by the end-user.

After anticipating the structural and dynamic aspects of the UI from a BD point of view with the help of our profile, we can now concentrate on the UI problematic through the **UI view**.

2.2 The UI view

The **UI view** handles the UI aspects required to satisfy the **BD** presented in the **BD view**: the graphical display (screen content, graphical controls, etc.), the behavior (condition for visibility and for enabling, etc.) and the relationship between the business elements and their graphical representation. Use-cases specified through the **BD view** have to be refined from the UI point of view. In addition, the different windows/screens associated with the use-cases and especially their sequencing have to be specified. Moreover, specific aspects of behavioral UI modeling also have to be studied: error management and business rule support.

Use-case refinement: First of all, UI use-cases are deduced from **BD** use-cases, and therefore justified by them. As well as use-cases may be deduced from the **BD** to the **UI view**, dependencies among them may also be deduced. They have to be distinguished from dependencies created among UI use-cases and not present in the **BD view**.

Moreover, as well as we introduced the notion of **workflow** use-case in the **BD view**, we may have to answer a similar need in the **UI view**. A requirement, while refined with regards to its UI, may lead to the specification of a **UI workflow** use-case.

In our profile we provide use-case and dependencies stereotypes to answer these needs. **BD extend** and **BD include** dependencies are deduced from the **BD view** and justified by it on the contrary of usual **extend** and **include** dependencies which may appear in the **UI view** and indicate dependencies belonging only to the **UI view**.

Window sequencing: The state-chart diagram is the most suitable UML diagram to show the sequencing of the different windows constituting the software. In order to conciliate **BD** and **UI** specification, screens have to be associated with the use-cases describing the **BD view**. Therefore, a state-chart diagram specifying the windows is associated with each use-case. It allows to keep the modularity of the specification introduced in the use-case diagram(s): if requirements have been decomposed into different use-cases, it means that some of them are for instance optional (i.e. related through extend dependencies) or may be used several times at different point of the specification. Therefore, screens have to be described within each use-case, to allow for instance not to present the windows associated with an optional use-case if it is not required.

But such a modularity in the presentation of the use-case leads to a difficult understanding of the whole sequencing of screens. Therefore, a top level diagram is also necessary to show how the different sets of windows associated with each use-case are related together.

We believe it is necessary to keep the 2 specifications: one at the level of each use-case and one among the different use-cases, for the reasons already given, and also because it allows a better traceability of specification: windows are associated with a given use-case, and if the use-case is changed or removed, it is easier to find the corresponding screens and to modify or to remove them. Each

window is justified by the use-case it is related to. Moreover, by providing these two levels of specification, the sequencing inside the use-case is clearly separated from the sequencing among the different use-cases and therefore constrained by the dependency among use-cases. Consistency is enforced because it is easier to check that the sequencing is made coherent with regards to dependency relationships. To allow screen chaining specification with the help of state-chart diagrams, we propose dedicated stereotypes in our profile. State stereotypes are provided to deal with the different kinds of screens (page, set of pages, page zone, etc.), event stereotypes are provided to clearly show when they are fired by the end-user or by the system. And to enforce the coherency between the UI and the BD **views** and this way the conciliation between UI and BD, realization stereotypes and action stereotypes are provided. The last ones aim at referring activities or use-cases of the BD **view** on the transition between the states.

Error Management Diagram: Errors related to the UI have to be modeled in a coherent and homogeneous way for the whole software. To help through this work, a dedicated class diagram may be provided to reassemble all the UI errors. Compensating actions/tasks may be associated to them.

Business Rule support Diagram: Business rules may be generic and associated to the whole software while called in several use-cases and windows. To be coherent through the whole UI specification, these business rules have to be isolated and referred to in the diagrams describing each concerned screen. A typical example of such generic business rules is the control among two dates (to be sure a first date is before the second one, for instance). It may be expressed through activity diagrams.

Because of space restriction, we do not detail the **Error Management** and **Business Rule Support** diagrams. They are based on class diagrams.

In the UI **view**, specifying the software UI with the help of our profile aims at getting a better integrated/justified UI with regards to the BD.

2.3 Relationships between UI and BD views

Dissociating explicitly BD from UI specification improves considerably the final product because the business thought is reinforced, in consequence the essential product objectives are clearly established. The thought process is driven by the business and not by the screen aspects (which are apparently the easiest aspects to cope with), in consequence the business end-user objectives are better targeted. The localization (translation) is made easier, because the screen layout is never used as business reference. The same BD can be implemented for different UIs. It facilitates the branding because the look and feel (graphical "skin") is handled independently of the UI behavior. But relationships have to be explicitly shown between UI and BD, to keep in mind they have been modeled in an integrated and coherent way. For this purpose, we introduce in our profile dependency stereotypes. Moreover, as the software architecture has to be layered with

a BD layer and a UI layer, in the diagrams specifying the software architecture BD components are distinguished from UI ones through dedicated component stereotypes.

2.4 Model organization

To organize the different diagrams associated to the UI and BD views, dedicated package stereotypes are introduced. The UI view is only made of **UI packages** while the BD view is only made of **BD packages**.

Figure 1 summarizes the **Application Model**. Its 2 views and the different models they embodies. For each of the model, the useful UML diagram are indicate in addition to the relationships among them. Relationships are supported by UML dependencies and UML actions on transition between states as it will be explained in detail in the profile description.

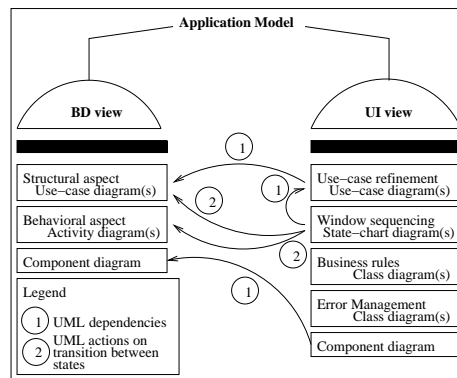


Fig. 1. The Application Model

3 A dedicated profile

3.1 Use-case stereotypes

About use-cases, the purpose of our profile is to distinguish BD use-cases from UI ones, as well as use-cases driving others.

Stereotype	Base Class	Parent	Tags	Constraints	Description
WF	Use-Case	N/A	N/A	Abstract	(d-1)
UI	Use-Case	N/A	For		(d-2)
BD	Use-Case	N/A	N/A	N/A	(d-3)
BD-WF	Use-Case	BD, WF	N/A	N/A	(d-4)
UI-WF	Use-Case	UI, WF-UC	N/A	N/A	(d-5)

- (d-1) Abstract stereotype to highlight a use-case driving others.
- (d-2) Use-case integrating a UI.
- (d-3) Use-case dedicated to BD (i.e. not dealing with UI specification).
- (d-4) A BD use-case which drives others.
- (d-5) A <<UI>> use-case which drives others. It may also be used to represent a UI use-case associated to a <<BD-WF>>.

Tag	Stereotype	Type	Multiplicity	Description
For	UI	Use-Case	0..1	If a <<UI>> use-case has been built from a <<BD>> use-case, the <<BD>> use-case it has been deducted from has to be indicated.

Figure 2 summarizes the different use-case stereotypes (using the currently proposed UML 2.0 notation).

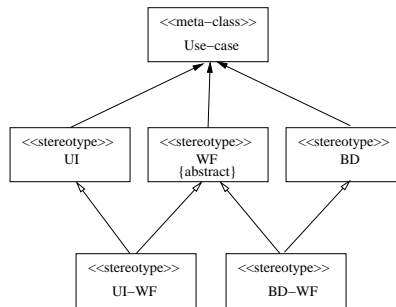


Fig. 2. Use-case stereotypes description

3.2 Actor stereotypes

UIs are required when human-being interacts with the software. Therefore, human-being actors have to be distinguished from system ones.

Stereotype	Base Class	Parent	Tags	Constraints	Description
Human	Actor	N/A	N/A	N/A	Human-being actor
System	Actor	N/A	N/A	N/A	

3.3 Association stereotypes

In the BD view, it has to be indicated where a UI is required, even without giving details about it, to clearly identify the concerned use-cases and actors and to

anticipate the UI specification. It is the purpose of the following stereotypes.

Stereotype	Base Class	Parent	Tags	Constraints	Description
UI	Association	N/A	N/A	(c-1)	

- (c-1) The associated use-cases must always be stereotyped <<BD>> or <<BD-WF>>; and the associated actor must be stereotyped <<Human>>.

Figure 3 presents a software through the use of <<BD>> and <<BD-WF>> use-case stereotypes. The necessity of an UI is enlightened by the <<UI>> association stereotype used between **customer** and **order**. The identifier of the inclusion dependency between **order** and **Manage customer info**, which should be `BDView::Order->ManageCustomerInfo`, has been named #1 to be shorter; In the same way, the inclusion dependency between **order** and **Manage item info** has been named #2.

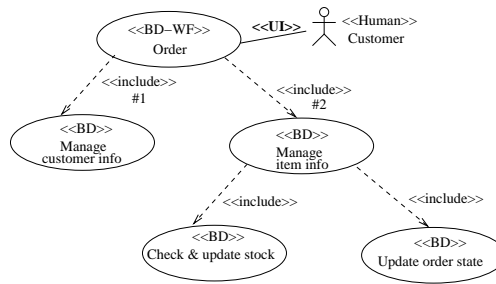
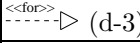


Fig. 3. Examples of <<UI>> association

3.4 Dependency relationship stereotypes

It is recommended to provide UI specification separately from BD specification. But UI specification is dependent to some extent from the BD: business rules have to be fulfilled through the UI. And when specifying UI, it is important to understand when dependencies among use-cases have been deduced from business rules. The following dependency relationships stereotypes allow to conciliate BD and UI specification: The **UI** realization allows to anticipate the UI in the **BD view**; **BD-extend** and **BD-include** allow to explicitly show dependencies among UI use-cases deduced from **BD** use-cases; And the **For** realization keep track of the dependencies among **BD** and **UI** specifications.

Stereotype	Base Class	Parent	Tags	Constraints	Description
BD-extend	Dependency	N/A	For	(c-1)	
BD-include	Dependency	N/A	For	(c-2)	
For	Realization	N/A	N/A		 (d-3)
UI	Realization	N/A			(d-4)

- (c-1) • The use-cases associated with such a dependency must be <<UI>> use-cases. Their **For** tag must be not null.
- A corresponding <<extend>> dependency must exist among the BD-UC the UI-UC under consideration has been deduced from.
- (c-2) • Use-cases associated with such a dependency must be <<UI>> use-cases. Their **For** tag must be not null.
- A corresponding <<extend>> dependency must exist among the BD-UC the UI-UC under consideration has been deduced from.
- (d-3) Relationships between UI elements and BD elements exist at the use-case level. They have already been introduced via the <<UI>> use-cases, <<BD-extend>> and <<BD-include>> stereotypes where a **for** tag indicates the BD dependency or the use-case from which the dependency or use-case of the UI **view** are derived from. But to explicitly show such relationships, a dedicated realization stereotype is provided. The <<For>> realization indicates when a <<UI>> or <<UI-WF>> use-case is derived from a <<BD>> or <<BD-WF>> use-case, or when a <<BD-include>> has been derived from an <<include>>, or when a <<BD-extend>> has been derived from an <<extend>>.
- (d-4) To explicitly show the relationship between use-cases and state-chart diagrams and for better understanding purposes, it is recommended to name the state with the name of the associated use-case. A dedicated realization dependency is also introduced.

Tag	Stereotype	Type	Multiplicity	Description
For	BD-extend, BD-include	dependency	1	The dependency from which the current dependency has been deduced.

In Figure 4, the UI and BD of the software presented in Figure 3 are shown. UI and BD are connected with the help of the <<For>> realization. One may also note that the decomposition of the WF use-cases may be different, because the BD does not have to be described through its UI, as well as the UI may be decomposed differently from the BD. In this example, the **Check & update stock** and **Update order state** use-cases do not need any UI; on the contrary, the BD use-case **Manage customer information** has been decomposed into <<UI>> **Manage payment info** and <<UI>> **Manage delivery info** because they need to be handled separately from a UI point of view.

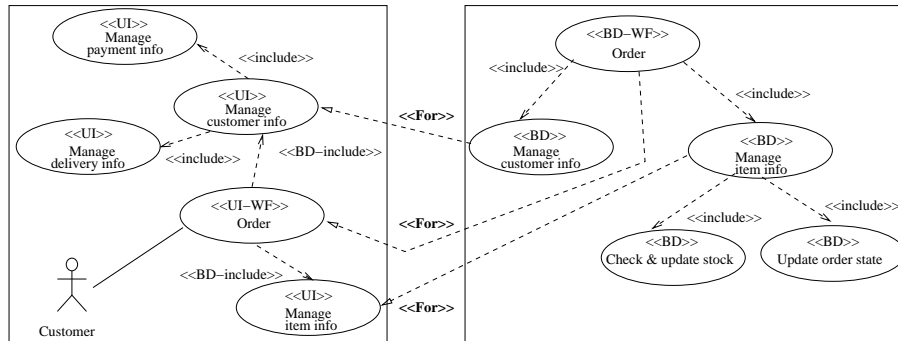


Fig. 4. Examples of deduced relationships

3.5 State stereotypes

In the **UI view**, it is required to specify the possible paths among the windows in accordance with the use-case specification. Sequencing of windows associated with a use-case to windows associated with another use-case, as well as window sequencing inside a use-case have to be documented. State-chart diagram is the most suitable diagram to document paths among windows. Especially compared to activity diagram because a window better correspond to a state than an activity: there is no completion with regards to window; The transitions among them are fired by event generated from end-user action; Specific actions may be associated to window especially when displayed or when quitted (a validation procedure for instance may be associated with the fact to leave a window); entry and exit actions provided with states are particularly suitable; And finally, state-chart diagrams also allow actions to be associated with transition (a *search* for instance is done after leaving a first window where *search criteria* have been entered and before displaying the *result* window(s)).

A state represents a window (or page), a window area or a collection of windows. It is stereotyped in consequence, with the main objective of distinguishing clearly when the window element is or not self-sufficient.

Stereotype	Base Class	Parent	Tags	Constraints	Description
UI	State	N/A	for displayMode	Abstract	
Page	State	UI	screenShot windowType detailedContent		(d-1)
Page-set	State	UI			(d-2)

Page-zone	State	UI	screenShot windowType detailedContent		(d-3)
Satellite-zone	composite state	Page-zone		(c-4)	(d-5)

- (d-1) Only one window is associated with the state
- (d-2) A set of distinct windows is associated with the state. The windows have to be presented in a determined order, given through a state-chart diagram.
- (d-3) It specifies an area in a larger window.
- (d-5) It is a specific kind of <<Page-zone>> which is not always displayed but appears on demand. A typical example of <<Satellite-zone>> is the dialog box.
- (c-4) When this state is reached, all the events not declared in the state are inhibited.

Tag	Stereotype	Type	Multiplicity	Description
for	UI	Use-case	1	The use-case the state is associated to. It must be a <<UI>> or <<UI-WF>> use-case.
screenShot	Page, Page-zone	String	1	The name of the drawing describing the windows associated with the state.
windowType	Page, Page-zone	String	1	To indicate the kind of window: full page, tab, dynamic form...
displayMode	UI-state	(String,String)	[0..*]	(d-1)
detailedContent	Page, Page-zone	class diagram	1	A class diagram specifying the information presented in the window.

- (d-1) Depending on the actor interacting with the software, on the task under consideration, on the information provided, a window may for instance be not-visible, active, non-active. The first term of the couple indicates the visibility. Conditions are associated with the visibilities: It is the purpose of the second term, which default is **no condition**. This composed tag indicates the visibilities associated with the current window and the conditions respectively associated with each visibility.

In the specification, we distinguish the permanency of the reservation of the window area where a given set of information may be displayed (**UI states**) from the visibility associated with the data to be displayed in the window (area). The visibility is documented through the `displayMode` tag. It is important to distinguish these 2 aspects while modeling the UI, because one is checked at the moment when the window (area) is displayed, while the other is checked while the window (area) is displayed, as an answer to a specific event.

Figure 5 summarizes the state stereotypes.

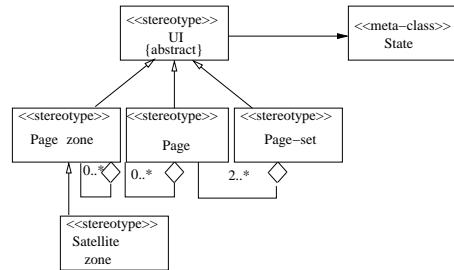


Fig. 5. State stereotypes

3.6 Transition stereotypes

Some events correspond to end-user interactions (button pressed for instance), others are generated by the system (when a task is ended for instance). In order to anticipate the UI specification, it is important to clearly distinguish these 2 kinds of events.

Stereotype	Base Class	Parent	Tags	Constraints	Description
User	Event	N/A			(d-1)
System	Event	N/A			(d-2)

- (d-1) An event represents an event generated by an interaction with the end-user (through the UI not fully described here).
- (d-2) An event generated by the software without any interaction with the end-user.

Actions may be associated to events on transitions. These actions may be taken from the UI or the BD specification. In this last case, they must correspond to a use-case or an activity.

Stereotype	Base Class	Parent	Tags	Constraints	Description
UI	Action	N/A			(d-1)
BD	Action	N/A	For		(d-2)

- (d-1) A task associated with the UI.
- (d-2) A use-case or activity associated with the BD. Relationships between actions on transitions and use-cases or activities have already been shown in Figure 1 (first kind of dependency).

Tag	Stereotype	Type	Multiplicity	Description
for	BD	Use-case or Activity	1	Name of the use-case or the activity the action is associated to.

3.7 Decision stereotypes

When decisions are required in the flow of control, it has to indicated if the condition depends on information known by the system or given by the end-user. In this last case, it implies that a UI is associated with the use-case to allow the end-user to give the information required to evaluate the condition.

Stereotype	Base Class	Parent	Tags	Constraints	Description
User-source	Decision	N/A	N/A	N/A	(d-1)

- (d-1) Decision made with regards to information given by an actor (most of the time, the end-user), even not at the point when the transition is fired.

3.8 Component stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
UI	Component	N/A	N/A	N/A	(d-1)
BD	Component	N/A	N/A	N/A	(d-2)

- (d-1) Component dedicated to UI specification.
(d-2) Component dedicated to BD specification.

3.9 Package stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
UI	Package	N/A	N/A	N/A	(d-1)
BD	Package	N/A	N/A	N/A	(d-2)

- (d-1) Packages dedicated to UI specification.
(d-2) Packages dedicated to BD specification.

3.10 Synthesis

The forthcoming table summarizes the different stereotypes provided in our profile. Stereotypes provided for the **BD** view aim at anticipating the **UI** through **BD** modeling. Stereotypes provided for the **UI view** aim at specifying the **UI** in conciliation with the **BD**. And relationships among **UI** and **BD** are also highlighted through dedicated stereotypes.

UML element	BD view	UI view	Relationships between views
Use-case	<<BD>> <<BD-WF>>	<<WF>> <<UI>> <<UI-WF>>	
Actor	<<Human>>, <<System>>		
Association	<<UI>>		
Dependency		<<BD-extend>> <<BD-include>>	
Realization	<<UI>>		<<For>>
State		<<UI>>, <<Page>> <<Pages-set>> <<Page-zone>> <<Satellite-zone>>	
Event		<<User>> <<System>>	
Action		<<UI>>, <<BD>>	
Decision	<<user-source>>		
Component	<<BD>>, <<UI>>		
Package	<<BD>>, <<UI>>		

4 Using the Dedicated Profile to Build the Application Model

In this section we present the required steps to build the **Application Model** through the analysis and design process (A&D-Pr). The process is decomposed into 5 phases. First, the **Requirement Analysis** deals with the requirement formalization. It covers explicit requirements, expressed by the end-user, as well as implicit ones, deducted by the analyst. Functional requirements, as well as technical ones, are carefully considered. Their pertinent organization facilitates the progress of the forthcoming phases of the A&D-Pr.

Then, the **Domain and Business Object Analysis** focuses on the specification of the **BD** covered by the software to be developed. This is a crucial point of the analysis activity, in order to get a software fulfilling the requirements previously captured and also to anticipate future enhancements not yet fully identified.

Requirement Analysis and *Domain and Business Object Analysis* may be processed in parallel. They provide two complementary and essential view points on the software to be developed.

The third phase concentrates on the **Software and System Architecture**. Softwares are more and more complex from the functional and technical points of view. Most of the time, they involve heterogeneous environments, which increase considerably the complexity of the architecture to be deployed. Therefore, part of the A&D-Ac has to be dedicated to the specification of the software and the system architecture.

Then, the fourth phase, **Component Specification**, copes with the integration of the different components mainly through the identification of the interfaces among them. Potential reuse aspects are handled through this phase of the A&D-Pr.

Software and System Architecture and *Component Specification* phases may be processed in parallel. They provide two complementary and essential view points on the software to be developed.

Finally, through the *Internal Design* phase, the specification of each element (mainly component) previously defined is refined to allow the implementation.

In the following, we will show how to conciliate UI and BD modeling via our UML profile through the different phases constituting the A&D-Pr.

4.1 Requirement Analysis

Building the BD View:

- Model the BD with the help of the <<BD>> use-case stereotype.
- Stereotype actors interacting with the application as <<Human>> actor or <<System>> actor to clearly emphasize where a UI is required (it is never the case with <<System>> actor).
- Stereotype associations between actors and use-cases as <<UI>> associations where a UI is required to access the use-case. Such an association is allowed only between <<Human>> actor and BD use-case.
- Group the models related to the **BD View** in <<BD>> package(s).

Specifying the BD workflow (BD view):

- Enlighten use-cases which drive other use-cases (through **extend** and **include** relationships) by stereotyping them as <<BD-WF>> use-case.
- Activity diagrams may be provided to complete the description of <<WF>> use-cases to explicit their control flow.
- Use decision point stereotypes if required in the activity diagram to indicate if choices associated to the splits are based on information given by the actor interacting with the application (<<Actor-based>>) or known by the system (<<System-based>>).

Building the UI View:

- Deduct UI use-cases from <<BD>> and <<BD-WF>> use-cases. UI use-cases are derived from BD use-cases, at least as a starting point. A UI use-case is created for each BD use-case related to an actor through a <<UI>> association. Use-cases which are included or extend BD use-cases related to actor(s) through <<UI>> association may also lead to the creation of <<UI>> use-cases, except if the contrary is specified.
- If dependency relationships exist among BD use-cases used to deduct <<UI>> use-cases, they also have to be taken into account through the **UI view**. Use <<BD-extend>> or <<BD-include>> dependency stereotypes to show the dependencies deducted from BD dependencies.
- If a BD use-case has been stereotyped <<BD-WF>>, then the deducted UI use-case is stereotyped <<UI-WF>>.
- Group the models related to the **UI View** in <<UI>> package(s).

Highlighting relationships between UI and BD views:

- When UI use-cases have been deducted from BD use-cases, show it explicitly via <<For>> dependencies.

Specifying layout policies (UI View):

- It is a first attempt to define the look and feel of the application, but without looking at a specific window. First of all, guidelines have to be decided to present information in the same way everywhere in the application. For instance, one will decide if contextual menus are provided, and for which kind of situation. It reinforces the homogeneity of the software to be developed.

4.2 Domain & Business Object Analysis

Refining UI use-cases (UI View):

- If there is no <<BD-WF>> in the **BD view**, the **UI view** deducted from the **BD view** is only made of distinct use-cases. Choose or create one <<UI>> use-case to drive the UI tasks. Stereotype it as a <<UI-WF>> use-case.
- If the UI is complex, it is also required to decompose it into different use-cases, introducing a <<UI-WF>> use-case if necessary.
- Refine use-cases to highlight constraints and requirements related to the UI.

Associating windows to use-cases (UI View): Here are the different steps to be followed to provide this two-level state-chart diagram specification:

- Create a top level state-chart diagram to show how to chain windows associated with use-cases:
 - Use <<UI>> state stereotypes.
 - Transition among states are allowed only if dependency exists among the corresponding use-cases.

- Stereotype events as <<User>> event if it represents a user interaction, <<System>> event otherwise.
 - Actions may be associated with events on transitions. They have to correspond to UI actions or to actions associated with BD use-case or activities explicitly described in the activity diagram associated with the BD use-case. Use dedicated action stereotypes: <<UI>> or <<BD>>.
- If required (i.e. <<Pages-set>> stereotype used), refine each state from the top level state-chart diagram to specify the different windows associated with a use-case and how to navigate through them.
- Use <<UI>> state stereotypes.
 - Stereotype events on transitions as <<User>> or <<System>>
 - Actions may be associated with events on transitions. They have to correspond to UI actions or to actions associated with BD use-case or activities explicitly described in the activity diagram associated with the BD use-case. Use dedicated action stereotypes: <<UI>> or <<BD>>.

In Figure 6, the UI specification through use-case already introduced in Figure 4 is completed by state-chart diagrams. A top-level state-chart diagram is provided to show the different <<UI>> states associated with the <<UI-WF>> use-case. **Manage customer info** and **Manage item info** are presented as <<Pages-set>> states, which means that they should be detailed in another diagram. The detailed state-chart diagram associated with **Manage item info** is presented as composed of 3 distinct <<UI>> states (**Item list**, **Item detail** and **Search criteria**).

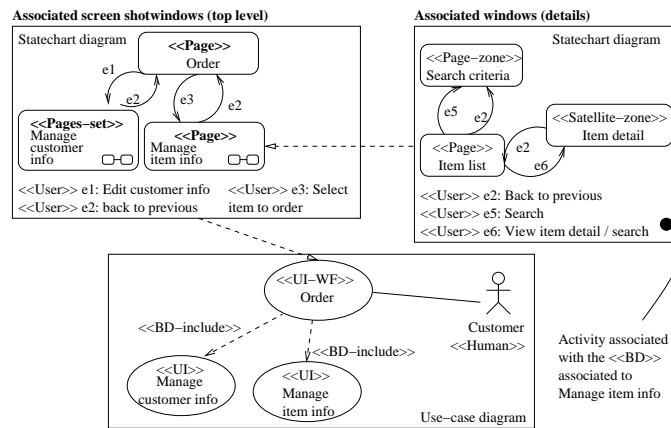


Fig. 6. Example of window specification via the UI profile

Specifying window content (UI View):

- Detail each window (i.e. each <<UI>> state except <<Pages-set>> state) with regards to the information displayed in the window.

Uniforming UI business rules behavior (UI View): Some business rules may be generic and associated to the whole software while called in several use-cases and windows. To be coherent through the whole UI specification, such business rules have to be isolated in a specific class diagram and explicitly used in the diagrams describing each concerned windows. A typical example of such generic business rules is the control among two dates (to be sure a first date is before the second one, for instance).

- Give a class diagram summarizing the general business rules.

Error management (UI View):

- Group all errors related to a bad use of the UI by the user in a (set of) class diagram for coherency purpose.
- Specify each error carefully: Associate a compensating task/action to each error.

4.3 System & Software Architecture

Separation of concerns (UI View):

- Separate the UI layer from the BD layer in the component diagram by using dedicated packages: <<UI>> and <<BD>> packages to show the layers.
- Highlight components dedicated to the UI from components dedicated to the BD by using the corresponding <<UI>> and <<BD>> component stereotypes.

4.4 Component Specification

Specifying layout policies (UI View):

- Isolate generic control provided all over the software to be developed in dedicated components to better ensure that a similar policy with regards to the way information will be displayed, error message will be given, etc. is followed every where.

Locating display rules (UI View):

- Place business rules in the <<BD>> and/or <<UI>> components: business rules may be managed directly by <<UI>> components and/or by <<BD>> components.
 - If the tool/technology used to implement the UI supports the business rule one is interested in (a display mode for instance), place it in the <<UI>> component.
 - If the business rule is useful in several windows over the UI, isolate it in a dedicated component to be then able to change it if required and to have an homogeneous behavior of the application all over the UI.

4.5 Internal Design

UI design (UI View):

- Select the <<UI>> components which specification need to be refined.
 - A component may is not selected if its specification already allows its implementation, especially with the help of a dedicated tool.
 - A component is selected if it needs more specification to allow its right implementation.

Figure 7 summarizes the different steps required to built the **Application Model**.

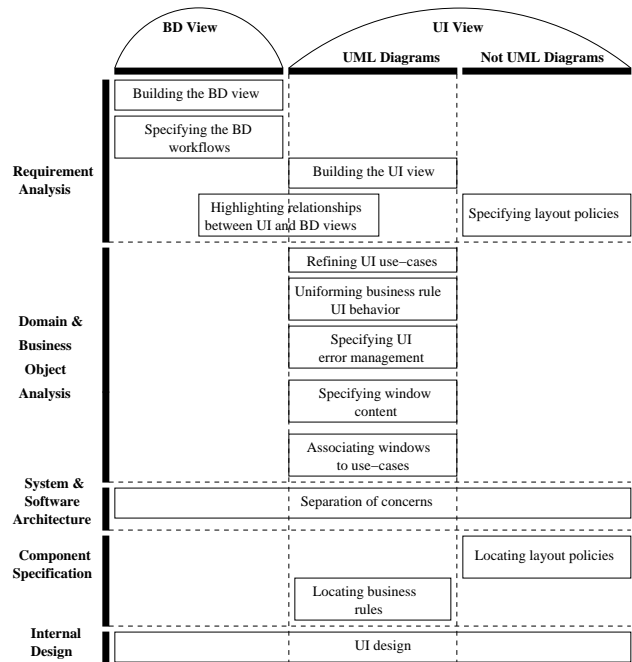


Fig. 7. A&D-Pr steps

5 Validation of the approach

This proposed profile is part of the UML usage recommendations and guidelines of the Amadeus Software Development process. Amadeus is a worldwide leader in the electronic distribution of travel services. With a workforce of more than 5000 people worldwide, and development sites spread around the globe (1500

people in Sophia-Antipolis in France, 700 people near Frankfurt in Germany, and other sites in Boston, Miami, London, Sydney, etc.), software development process must cope with geographic distribution. Most of Amadeus projects are multi-sites. UML is used as Amadeus linguafranca since more than seven years for analysis and design of all products. Since October 2002, the profile presented in this paper is available on the Amadeus intranet, providing detailed usage guidelines. It has been used directly by more than 120 people from the various development sites: these people are not necessary in direct contact with the Software Engineering team, responsible of the corporate software development process (including the UML usages). One of the main concrete benefits of this profile is to reinforce the business thought coming with the ability to provide several user interfaces for the same business: cryptic interfaces for 3270 screens, graphical user interface for the Web applications or for Windows platforms (plasticity of the UI). An important effort is made today on the product usability; this profile fully integrates the usability needs in term of formalization; and it is used by the Amadeus ergonomists: the communication between these two populations, ergonomists and developers, has been considerably improved with the usage of a common language coming with standard usages. Last but not least, the relationship between the different involved development teams has been improved because the split between the business and the user interface part is clearer: core business specification is no more included in the UI specification. In addition, the review process has been simplified due to this better repartition. In conclusion, the usage of this proposed profile has considerably improved the analysis and design of the software development; and a direct impact on the developed product has already been noticed. It does not concern only the graphical user interface application, but also the cryptic entrees ones and the servers that cover the business.

6 Conclusion

Most of the approaches dealing with User Interface (UI) focus on design and implementation aspects and are driven by a specific technology. We believe UI specification has already to be taken into account through the analysis phase, because it is closely related to the software Business Domain (BD), which is apprehended in the earliest steps of the Analysis and Design Process (A&D-Pr). Moreover, during the analysis phase, the UI is described in a more abstract way and a larger variety of UI can be taken into consideration.

The UI and the BD aspects of the software have to be studied in a closely-related way. But most of the existing approaches propose a specific notation for the UI, different from the one used to define the BD. And when guidelines are associated with the notation, they focus on the UI and are poorly related to the BD.

We proposed an integrated approach for UI modeling. In terms of model, we proposed an **Application Model** distinguishing the UI and BD aspects via separate views and clearly setting relationships between them. We choose the

UML notation to illustrate our work in order to facilitate (i) the integration of BD and UI aspects modeling and (ii) its use in any standard A&D-Pr. We presented a UML profile to support the **Application Model**. We focus on the dynamic aspect of UI. Our approach also includes a methodology to show how we could take advantage of the **Application Model** through an A&D-Pr. Its aim is not only to model the UI from a pure technical point of view, but also from a business one.

In the future we would like to complete our work by providing stereotypes to suit the specific features of specific kinds of UI. And we would also like to enrich our approach to take into account the preferences associated with specific users or groups of users.

References

1. T. Browne, D. Davila, S. Rugaber, and K. Stirewalt. *Formal methods in Human-Computer Interaction*, chapter Using declarative descriptions to model user interfaces with MASTERMIND. Springer-Verlag, 1997.
2. L.L. Constantine and L.A.D. Lockwood. *Software for use: a practical guide to the models and methods of usage-centered design*. Addison Wesley, 1999.
3. P. Pinheiro da Silva and N.W. Paton. UMLi: the unified modeling language for interactive applications. In *3rd International Conference on the Unified Modeling Language*, pages 117–132, October 2000.
4. D. D'Souza and A. Wills. *Objects, Components and Frameworks With UML: The Catalysis Approach*. Addison-Wesley, 1998.
5. Object Management Group. The UML notation. <http://www.omg.org/>.
6. P. Krutchen. *The Rational Unified Process*. Object Technology Series. Addison-Wesley, 2000.
7. D.N.J. Nunes. *Object Modeling for User-Centered Development and User Interface Design: the Wisdom Approach*. PhD thesis, Universidade da Madeira, Portugal, 2001.
8. A. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. In *International Conference on Intelligent User Interfaces*, pages 171–178, January 1999.
9. D. Roberts, D. Berry, S. Isensee, and J. Mullally. *Designing for the user with OVID: bridging user interface design and software engineering*. Macmullan Tech, 1998.