User experience storyboards: Building better UIs with RUP, UML, and use cases

by Jim Heumann
Requirements Evangelist
IBM Rational
IBM Software Group

Use cases describe the way a system will be used, but they don't specify the system user interface (UI). User experience storyboards, based on the use cases, add the process and techniques necessary to design and build a user interface that will meet requirements and allow users to exercise all the system behavior described in the use cases. These storyboards provide a firm foundation for the detailed UI design and also facilitate communication among development team members -- especially between the creative team that designs the UI and the architects and developers who design and implement the system.

IBM Rational Unified Process

IBM Rational Unified Process,® or RUP,® recommends user experience (UX) storyboards along with a user experience model. RUP is a software process framework that gives teams a head start in creating a repeatable way to develop software within their environment. RUP organizes the activities involved in developing software into nine disciplines, as shown in Figure 1.

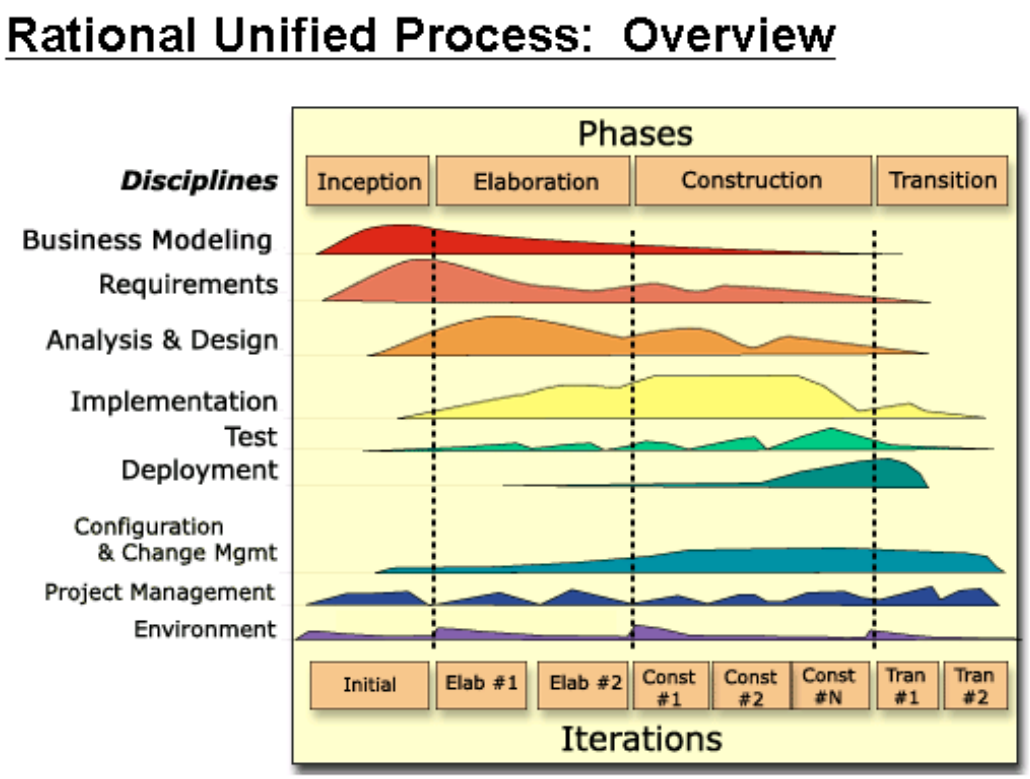

Figure 1: Phases, disciplines, and iterations in IBM Rational Unified Process

The two most important disciplines relative to UX storyboards are Requirements and Analysis & Design. It is in Requirements that use cases are developed; in Analysis & Design they are used to design the system, including the UI. RUP uses models to represent the various parts of a software system. In Requirements, the use-case model is the most important one to build. In Analysis & Design, as one might expect, the analysis model and the design model are developed. For systems with a significant amount of user interaction, the development team should also create a UX model and storyboards within that discipline. This integrates usability concerns into the RUP development approach.

Unified Modeling Language

The Unified Modeling Language, or UML, is the *de facto* standard for software modeling. It is used for creating models in RUP, including the use-case model, the analysis and design models, and the UX model.

Using multiple models to represent a system is important because it allows a team to break up the huge problem of developing a system into smaller, digestible pieces that can be solved one at a time. However, when you represent a system specification/design in disparate models, it becomes very important to keep the models synchronized with each other. If just one model is out of date, there is a risk that the team will develop the wrong system. In RUP, the use-case model is the foundation for the other models -- the one that keeps all models in sync. Analysis & Design activities are based directly on use cases, as is the UX model.

Use cases

Use cases describe functional requirements. RUP and UML define a use case as follows:

> A use case describes a sequence of actions a system performs that yields an observable **result of value** to a particular actor.

In order to provide value, a use case must describe a significant chunk of functionality. The narrative text of use cases tells the story of how a user and the system interact -- in a simple, structured, and complete way.

Interestingly, although the UML defines a use case, it does not prescribe any format or style for writing one. Because of this, different authors promote many different styles. However, RUP specifies a use case with the basic structure shown in Figure 2.

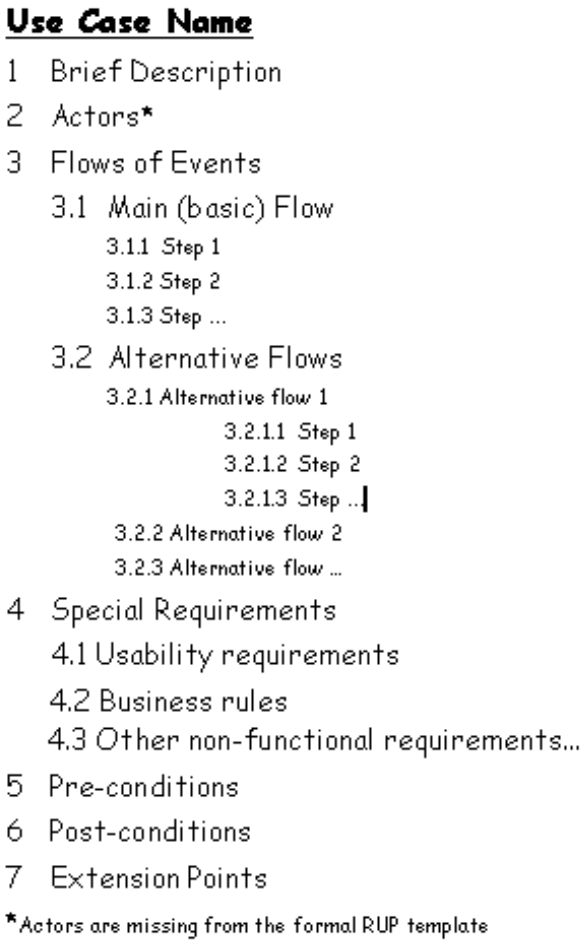

Figure 2: Structure for a RUP use case

The flows of events are the most significant part of the use case; this is the *narrative text* we mentioned above. Flows of events, which describe the behavior of the system, are the part of the use case applied most directly in designing and testing the system, including the user interface. A use case has one basic (or main) flow and multiple alternative flows. The basic flow is the "successful" one: The user achieves the main goal and

receives the value promised by the use case. The alternative flows specify either regular variants of the main flow or exceptional and error flows. The collection of all flows constitutes the basis for the design and testing activities that will come next.

Figure 3 shows an example of the basic flow of events for a use case describing an actor bidding on an item in an auction system (Bid on Item use case).

3. Flow of Events

3.1 Basic Flow

3.1.1 BID
This use case starts when the Buyer bids on a displayed item currently available for auction.

3.1.2 ENTER AMOUNT
The Buyer enters the bid amount. The system validates the bid amount. The entered bid must be greater than the current (i.e., greatest) bid by an amount greater than the minimum bid increment specified for the auction.

3.1.3 BUYER CONFIRMS BID
The system provides information that tells the legal obligations of placing a bid. The Buyer confirms that the bid should be placed.

3.1.4 POST BID
The system posts the bid for the auction. The entered bid becomes the current (i.e., greatest) bid.

3.1.5 SEND EMAIL
The system sends an email confirmation to the Buyer, including the bid amount for the auction item, as well as when the auction will close.

3.1.6 SYSTEM CONFIRM BID
The system notifies the Buyer that the bid has been accepted and displays the Buyer's name and email address, as well as the item name and the bid amount. The use case ends.

Figure 3: Basic flow of events for Bid on Item use case

And Figure 4 shows alternative flows for the same use case.

3.2 Alternative Flows

3.2.1 AUCTION IS CLOSED
At BF BID, the auction for the item has been closed, a message is displayed to the Buyer stating that the auction has been closed, and the bid is not accepted. The use case ends.

3.2.2 BUYER HAS PENDING PAYMENTS
At BF BID, the Buyer has pending payments, the system displays a message to the Buyer, reminding him/her that he/she has pending payments that are due and that until those payments are made, the Buyer cannot participate in an auction, either as a Buyer or a Seller. The use case ends.

3.2.3 ENTERED BID IS INVALID
At BF ENTER AMOUNT, the User enters an invalid bid, the system indicates to the User the reason the bid is invalid (e.g., the entered bid was not greater than the greatest bid by an amount greater than the minimum bid increment specified for the auction). Resume at BF ENTER AMOUNT.

3.2.4 BID NOT CONFIRMED
At BF CONFIRM BID, the Buyer does not confirm that the bid should be placed. The use case resumes at BF ENTER AMOUNT.

Figure 4: Alternative flows for Bid on Item use case

UX storyboards use the special requirements section of a use case, which stores all non-functional requirements specifically related to the use case (non-functional requirements about the system as a whole are stored in a different document -- the supplementary specification -- which we will not address here). Note that there are no user interface details in this use case -- no mention of buttons, text boxes, screens, or pages. This is intentional. A use case should describe what a system should *do*, and not how it should be done.

Modeling the user experience with storyboards

Since use cases do not specify user interface details, we need something else to do that. This is where the UX model and storyboards are very effective.

The UX model is conceptual; it specifies user experience elements, screens, and screen content in an abstract representation. It helps architects and designers determine what will go into the UI before worrying about the details of specific UI elements (buttons, pull-down

menus, etc.) The UX model encourages a good interface architecture. In fact, it serves as a sort of contract between the UI team and the development team, ensuring that developers don't just "make it up" as they go.[1]

UX modeling is not required on every project. Creating a model takes effort that should not be expended without weighing the potential benefit. Typically, the benefit of a UX model outweighs the cost and effort required if the user experience is critical to the system's success, if the system is very new to the users, or if UI errors might have costly consequences. The latter category might include systems for which user errors can create safety risks -- such as air traffic control or nuclear power plant systems. UI errors can also be costly in systems with a high potential for loss, such as a financial system, a network control center, or any system for which usability is very important to success.

A UX model and its storyboards describe actors (user characteristics) and screens, as well as input forms, screen flows, navigation between screens, and usability requirements. The actor characteristics and usability requirements are added to the use-case descriptions. The other elements are described in UML and remain part of the UX model; they may also be used in other models.

Creating UX storyboards

There are five steps for creating UX storyboards.

1. Add actor characteristics to the use case.
2. Add usability guidance and usability requirements to the use case.
3. Identify UX elements.
4. Model the use-case flows with the UX elements.
5. Model screen navigation for the use case.

Step one: Add actor characteristics to the use case.

Knowing a significant amount of detail about the system users is critical to creating a usable system. The first step in creating a UX storyboard is to add that detail to the use case. This may include the users' average level of domain knowledge, general level of computer experience, and physical environment. It may also include how frequently they will use the system, and the approximate number of users represented by an actor.

To keep the main part of the use case simple and easy to read, you should put the actor characteristics in the special requirements section of the use case. Figures 5 shows some actor characteristics for the Bid on Item use case.

4.3 Actor Characteristics

4.3.1 BUYER

4.3.1.1 FREQUENCY OF USE

4.3.1.1.1 The typical Buyer will bid on an item three times per week.

4.3.1.1.2 Near the end of an auction, bidding activity may be very intense.

4.3.1.2 GENERAL LEVEL OF COMPUTER EXPERIENCE

4.3.1.2.1 The typical Buyer only uses his/her computer on a casual basis.

4.3.1.3 ENVIRONMENT

4.3.1.3.1 The typical Buyer uses the system from his/her home.

4.3.1.4 NUMBER OF USERS

4.3.1.4.1 The targeted number of users is 50,000.

Figure 5: Actor characteristics for the Bid on Item use case

This information will help the UI designers make decisions about how the UI should work.

Step two: Add usability guidance and usability requirements to the use case.

The second step in creating a UX storyboard is to add usability guidance and usability requirements to the use case. Remember, the original use case detailed system behavior -- in other words, the functional requirements. Guidance and usability requirements are considered non-functional, so they go in the special requirements section of the use case.

Usability guidance includes things such as advice for making the UI easier to use (usage guidance), average attribute values and volumes of objects, and average action usage.

Figure 6 shows some examples of usability guidance for the Bid on Item use case; items 4.1.1 and 4.1.2 show average action usage; 4.1.3 shows an average attribute value; and 4.1.4 shows usage guidance. All of these items provide extra information the UI designers can use to optimize the UI.

4. Special Requirements

4.1 User Experience Guidance

4.1.1 At AF PENDING PAYMENTS, pending payments normally occur in only 10% of the cases.

4.1.2 At AF INVALID BID ENTERED, Invalid bids are normally entered 15% of the time.

4.1.3 At BF BUYER CONFIRMS BID, the legal statement will be approximately 150 characters in length.

4.1.4 At BF ENTER AMOUNT the system should automatically provide choices at the next three bid increments.

Figure 6: Usability guidance for the Bid on Item use case

Usability requirements are different from usability guidance. Because they are requirements rather than helpful hints, they should be officially tested by the QA group. You can think of usability requirements as boundaries the system should not cross, and of usability guidance as hints on how users will use the system. Usability requirements might specify how fast a user must be able to do something, how fast the system must respond, maximum error rates, maximum number of mouse clicks, and learning times. The UI designers and developers must know these requirements before they get deeply into the UI development process. Again, to keep the original use case clean and easy to read, put usability requirements in the special requirements section of the use case. See Figure 7 for an example.

4.2 Usability Requirements

4.2.1 The Buyer must be able to confirm his/her bid with a single mouse click.

4.2.2 The system must update the current bid within 5 seconds of the Buyer confirming his/her bid.

4.2.3 The system must return confirmation of an accepted bid within 2 seconds.

Figure 7: Usability requirements for the Bid on Item use case

Step three: Identify UX elements.

The next step in creating UX storyboards is to identify the UX elements, including screens and input forms. Each of these can include dynamic content, user actions, environmental actions, and messages. As mentioned earlier, you can represent these elements in abstract form, using UML.

UML, originally developed for modeling objects and classes, was designed to be extensible. The mechanism that makes it extensible is the *stereotype*. Stereotypes allow you to repurpose an existing UML element, giving it a new meaning and a new visual representation. Figure 8 maps the elements UML defines for Object-Oriented Application Design (OOAD) modeling to elements used for UX modeling.

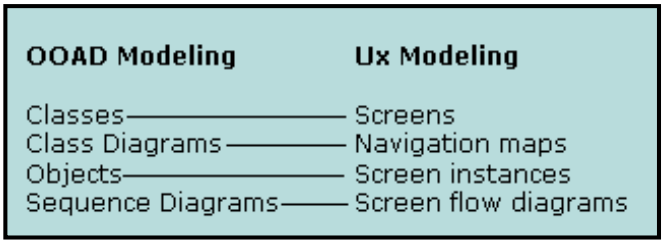

Figure 8: Mapping between OOAD and UX modeling elements

Figure 9 compares the characteristics of a *class* with those of a *screen*.

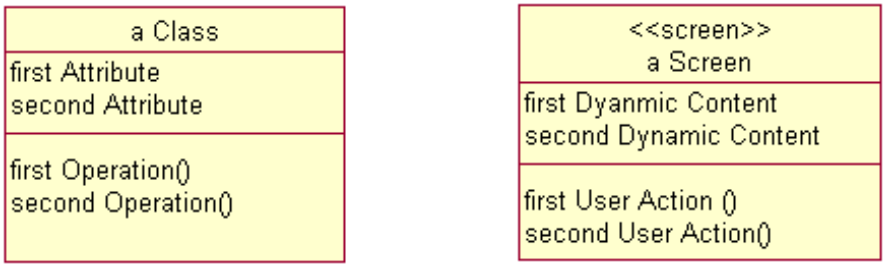

Figure 9: Characteristics of OOAD *class* modeling element and UX *screen* modeling element

The left side of Figure 9 shows the way UML depicts a class: name in the top compartment, attributes in the middle compartment, and operations in the lower one. The right side depicts a screen in a similar way, adding the stereotype <<screen>> to distinguish this element from a typical class. Here, the middle compartment shows the screen dynamic content (instead of attributes) and the lower compartment contains user and environmental actions (instead of operations). It should be clear why we say that UX elements such as screens are abstract; they don't look at all like actual GUI screens, but they *represent* GUI screens -- and what users can do with them.

Figure 10 shows the components of the "item detail" screen for the Bid on Item use case.

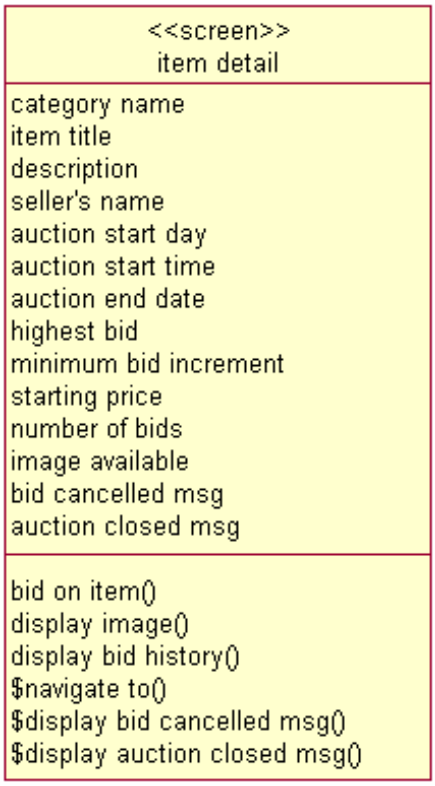

Figure 10: Components of the "item detail" screen for the Bid on Item use case

Defining screens

Here are some rules of thumb for defining screens for a use case:

- For the application, identify one primary screen for each actor -- in other words, the main screen with which the actor would usually work.
- For each use case, add a candidate screen; a use case represents a significant, independent chunk of functionality, so often (but not always) each use case has enough behavior to have a main screen.
- For important sets of information that must be managed by a use case, consider adding a candidate screen.

A very important caveat: Don't forget the creative aspects of designing a UI. The above rules of thumb will provide a good start, but you will need more to produce a fully optimized, usable UI.

Dynamic content of screens

Once you define what screens you will require, it is time to identify the dynamic content of each one. Dynamic content is information provided by the business logic, not static information such as headers and footers on a Web page. You can discover dynamic content in the flows of events for the use case under consideration. Figure 11, for example, shows the text for a step in the basic flow for the Bid on Item use case. It says that the system "...displays the Buyer's name and email address, as well as the item name

and the bid amount."

> 3.1.6 SYSTEM CONFIRM BID
>
> The system notifies the Buyer that the bid has been accepted and displays the Buyer's name and email address, as well as the item name and the bid amount. The use case ends.

Figure 11: Text for a step in the basic flow for the Bid on Item use case

This dynamic content is shown in the middle compartment of the UML representation of the screen (similar to a class representation). See Figure 12.

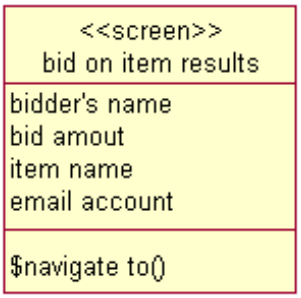

Figure 12: UML representation of a screen for Bid on Item use case (basic flow)

User actions, environmental actions, and messages on screens

You must also identify user actions for each screen. These are the things the user performs with the UI to get value from the use case. Again, you can discover user actions in the text of the use case. The step in the use case basic flow shown in Figure 13 states that "The Buyer confirms that the bid should be placed." This user action is shown in the lower compartment of the UML representation in Figure 14.

> *3.1.3 BUYER CONFIRMS BID*
>
> The system provides information that tells the legal obligations of placing a bid. The Buyer confirms that the bid should be placed.

Figure 13: Text for another step in the basic flow for the Bid on Item use case

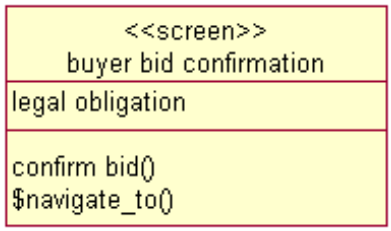

Figure 14: UML representation of another screen for the Bid on Item use case (basic flow)

Next, you should identify environmental actions. These are actions the system performs, which result in either a new screen or the appearance of new information on an existing screen. The most common environmental action is to navigate from one screen to another, and each screen has a $navigate_to action. (Note that environmental actions are distinguished from user actions by a dollar sign in front of their name.)

Another system action is to display messages. These can be status, error, warning, or informational messages. Once again, the place to find these messages is in the use-case text. Figure 15 shows an alternative flow that says "...the system indicates to the user the reason the bid is invalid." This then becomes a message on a screen, as shown in Figure 16.

3.2.3 ENTERED BID IS INVALID

At BF ENTER AMOUNT, the User enters an invalid bid, the system indicates to the User the reason the bid is invalid (e.g., the entered bid was not greater than the greatest bid by an amount greater than the minimum bid increment specified for the auction). Resume at BF ENTER AMOUNT.

Figure 15: Text for alternative flow for the Bid on Item use case

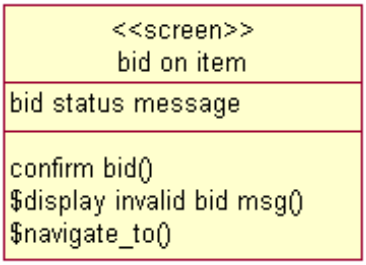

Figure 16: UML representation of a screen for an alternative flow for the Bid on Item use case

The last thing to model on a screen is another user interface element: an input form. Think of an input form as a screen within a screen. It provides a place for users to enter data and direct the system to act upon it. UML represents input forms as separate "boxes" and indicates them with an <<input form>> stereotype. Input forms never "live on their own"; they are always contained within a screen. This containment relationship is

shown as an aggregation in UML.

Figure 17 shows the use-case text in which we discover an input form ("The Buyer enters the bid amount"), and Figure 18 show the resulting screen and form in a UML representation.

> *3.1.2 ENTER AMOUNT*
>
> The Buyer enters the bid amount. The system validates the bid amount. The entered bid must be greater than the current (i.e., greatest) bid by an amount greater than the minimum bid increment specified for the auction.

Figure 17: Text with an input form requirement -- a step in the basic flow of the Bid on Item use case

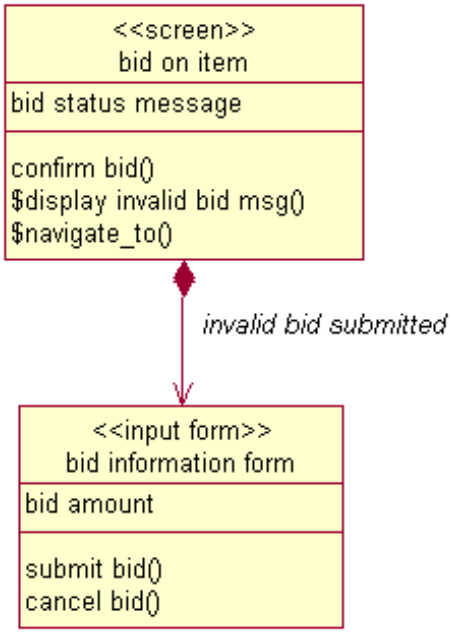

Figure 18: UML representation of a screen with an input form

Step four: Model the use-case flows with the UX elements.

The fourth step in modeling the user experience with UX storyboards is to show how the use-case flows of events are implemented among the screens of the conceptual UI. You can do this with UML sequence or collaboration diagrams, both of which are types of interaction diagrams. These diagrams are dynamic, meaning that they show something happening -- in this case the flow of a use case. Modeling the flows helps not only to allocate behavior to screens, but also to show how a user would navigate between screens to accomplish a task. It also helps to ensure that there is either a screen or a user action for everything the user has to accomplish. In addition, modeling verifies that the screens have the content necessary to accomplish the use case.

Figure 19 shows a sequence diagram for the basic flow of events for the Bid on Item use case.

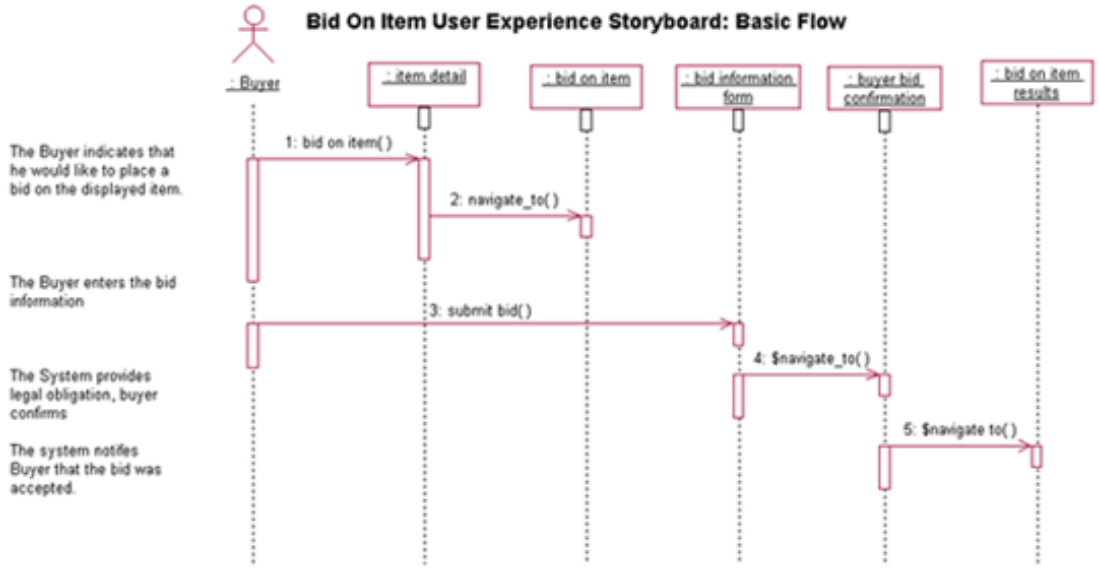

Figure 19: Sequence diagram showing the basic flow of events for the Bid on Item use case
Click to enlarge

This view gives you a clear picture of how many screens (represented by boxes at the top of the diagram) are involved in the flow of events and provides an opportunity to analyze whether the initial design is effective. Perhaps there are too many screens or too few; it is easy to experiment with different options. Remember: These diagrams depict presentation elements only, and they should show only the interaction between the user and the screen or between the screens themselves.

Step five: Model screen navigation for the use case.

The fifth step in creating a UX storyboard is to draw a navigation diagram for the use case. The diagram should not show any one flow of events, but rather the relationships among all the screens, or how the user can navigate between them. This diagram is valuable because it provides the big picture: the number and complexity of the screens needed to implement the use case. Again, this diagram will help you determine whether you have a good initial design. If not, then it is easy to experiment with alternative designs.

Navigation diagrams are a form of UML class diagram; elements are screens and input forms instead of classes. Figure 20 shows the navigation diagram for the Bid on Item use case. As you look at it, remember that an input form is just part of a screen; you can navigate only to a screen and not directly to an input form. Notice, too, that each navigation has a label indicating what triggers it.

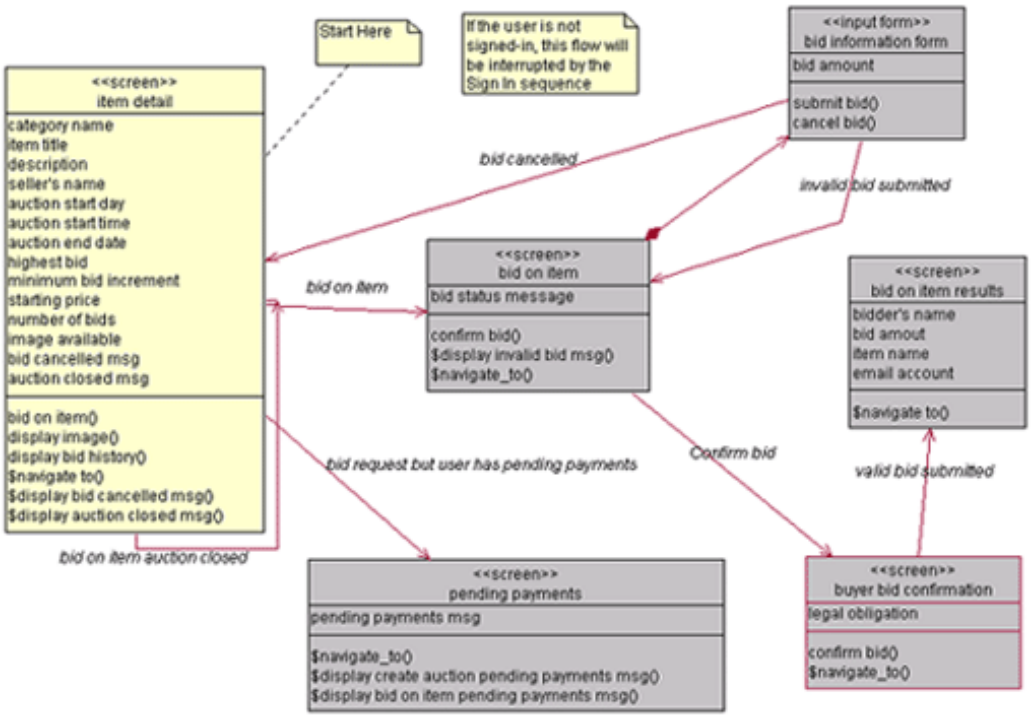

Figure 20: Navigation diagram for the Bid on Item use case
Click to enlarge

A UX model contains one navigation diagram for each use case and also a navigation diagram for the entire application, showing how all of the screens fit together.

Once you complete the five steps required to create a UX storyboard, it is time to review what you have done so far.

Review and reconcile

When you use the iterative software development approach in RUP, you assign a group of use cases, or use-case scenarios, to each iteration. The end of each iteration is a good time to review all of the UX storyboards, identify and resolve any inconsistencies, and remove any duplication. This is particularly important if different teams are working on different use cases. Architects should be key players in this review process, as they have an overall understanding of the application and can more easily see where changes might need to be made.

As you plan your review, you should decide whether to involve users. Experience has shown that the usefulness of their involvement is highly dependent on their technical sophistication. Some users might be confused by the abstract nature of the UX model and its UML representations, but others who are familiar and comfortable with UML models would be able to provide reasonable feedback.

Ultimately, you should ensure that you have followed the five steps we discussed in the previous section, and that your UX storyboards result in:

- An augmented use case with actor characteristics and usability guidelines and requirements.

- A set of screens and input forms with dynamic content, and user and system actions.
- Flow diagrams showing how the use-case flows of events are implemented via the screens.
- A navigation diagram showing all of the screens and their relationships for implementing the use-case user interface.

Once you have done your review, you can use the UX storyboards to support the ongoing work of both the UI team doing the detailed design and implementation of the UI, and the designers and developers designing and implementing the business logic.

Tracing to other models

Remember the importance of keeping in sync the various models that specify the system. The UX model clearly has a direct relationship with the use-case model. The two models should trace to each other, so that any changes in the use-case model are propagated to the UX model. You can do this automatically by using the same name for both the use case and its corresponding UX storyboard, or you can model the changes explicitly, using UML diagrams.

The UX model also has a relationship with the analysis and design models. The analysis model, which shows the initial design of the system at a very high level, contains <<boundary>> classes that model interfaces to users and other actors. The general rule in an analysis model is that you need one boundary class for each actor/use-case pair. The boundary classes are placeholders for the screens developed in the UX model, so you should establish traceability between the boundary classes and the screens.

The design model takes the analysis model to the next level of detail. It contains more specific interface implementation elements such as server pages, EJBs, and servlets. You can trace these elements to screens in the UX model.

The benefit of tracking these relationships is that it enables you to manage change. If a use case changes, it is easy to see what UI elements will be impacted and may have to change as well. Tracking also ensures that every screen will have detailed design elements and ultimately code that implements them, so nothing gets left out by mistake.

A firm foundation that adds value

Creating UX storyboards for a highly interactive system adds value by building a firm foundation for both the overall product concept and a strong interface architecture. UX storyboards are faster to develop and easier to experiment with than the actual UI or a prototype. The UX model facilitates communication between the UI team and the development team, in effect serving as a contract between these teams. It ensures that the UI fulfills specified requirements because it is based on the system use

cases. The UX model helps the UI team determine what should go in the UI before they decide how it should look or begin building it.

Not all systems need a full UX modeling effort, but if mistakes in your UI might have serious consequences for the system and its users, then this effort is critical. The project will benefit from using a UX model and storyboards to ensure the creation of an effective UI, and from tracing the UX model to other UML models in the Rational Unified Process to ensure compatibility and consistency among all system elements.

References

Philippe Kruchten and Stefan Ahlqvist, "User Interface Design in the Rational Unified Process." In M. van Harmelan, Ed., *Object Modeling and User Interface Design*. Addison Wesley, 2001.

IBM Rational Unified Process. IBM Rational Software, 2003. (See http://www-3ibm.com/software/awdtools/rup

L. L.Constantine and L.A.D. Lockwoood, *Software For Use: A Practical Guide to the Models and Method of Usage-Centered Design*. Addison-Wesley, 1999.

Notes

1 User experience storyboards are also called use-case storyboards.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!