USER INTERFACE DRIVEN SYSTEM DESIGN

Stevan Mrdalj, Eastern Michigan University, stevan.mrdalj@emich.edu
Vladan Jovanovic, Georgia Southern University, vladan@gasou.edu

ABSTRACT

Most of the object-oriented design approaches emphasize Use Cases as a substitute for functional decomposition and concentrate on conceptual system decomposition into objects in the early analysis phase resulting in a list of candidate classes. Practice has shown that user interface design before domain modeling can be used as a systematic basis to identify classes. This way the initial domain model will contain all the necessary classes and would eliminate considerable refinement caused by candidate classes. This paper focuses on the user interface driven system design and presents techniques that assure better consistency than in a typical design process including a diagram developing procedure as well as diagram balancing criteria.

Keywords: Object-oriented system design, user interface driven, diagram completeness, UML.

INTRODUCTION

Most of the object-oriented design approaches/methodologies (3,5,7) emphasize system decomposition into objects in the early analysis phase. The following three techniques are commonly used in the process of domain modeling: a - Noun phrase (1): b - Conceptual class category list (6), c - Analysis patterns (2). All three techniques are used to make a list of candidate classes. Larman states (6), "It is better to over specify a domain model … than to under specify it" as the usual guidelines in identifying conceptual classes. Approaches like this require additional effort to identify all the possible classes during the early elaboration phase of the project and considerable effort to verify if they are indeed needed or if all needed classes are identified. Consequently, all diagrams that are developed using the candidate classes that need to be modified or eliminated need to be refined. This verification concludes at the design phase using detailed User Interface (UI) design.

In order to eliminate considerable refinement caused by candidate classes, our approach suggests performing detailed UI design before domain modeling and to use it as a basis to identify classes. This way the initial domain model will contain only the necessary classes. Such a domain model can be further refined to reflect the various nonfunctional requirements like performance, expandability, maintainability, etc. In this paper we will use Unified Modeling Language (UML) diagrams (1) that are the de facto industry-standard modeling notation for object-oriented development.

USER INTERFACE BASED DESIGN PROCESS

The goal of our approach is to minimize the possibility to overlook UI requirements by developing them early on in the design process and by using them as a basis for developing all other components of the system model. The design process described in this paper recommends the following steps and criteria to be used:

1. Develop Use Case Model.
2. Design User Interface for each use case.
3. Develop Class diagram (conceptual user model) based on UI.
4. Develop Sequence diagram based on Use Case diagram, UI and the Class Diagram.
5. Develop Collaboration Diagram based on Sequence and Class Diagrams.
6. Develop Statechart diagram based on UI and Class diagram.

The entire system description is a network of interrelated parts/diagrams, as indicated on Figure 1. Therefore, it is necessary to verify that all the dependencies are modeled correctly. This paper describes some techniques that ensure that modeling diagrams are more completely examined for consistency than in a typical design process.

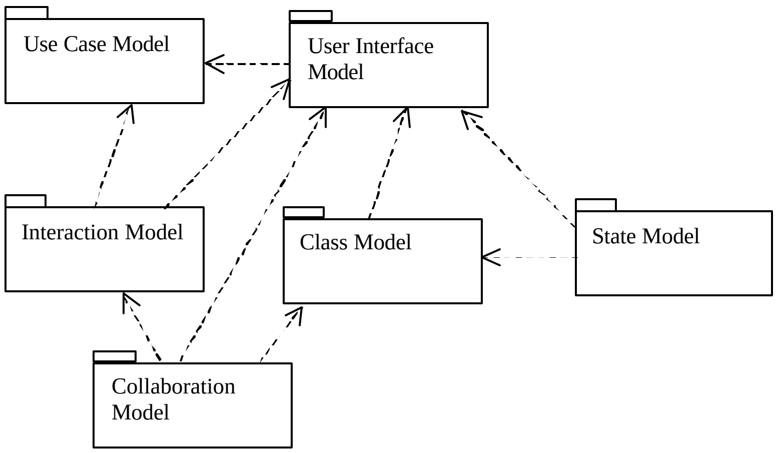


Figure 1: System Model Dependencies

There are two basic criteria that we will use for balancing diagrams, and each can be the focus of a specific balancing task: a - Completeness: A diagram is complete if no required elements are missing according to the UI; b - Consistency: A diagram is consistent if there are no contradictions among its elements and the corresponding ones in the dependent diagrams.

USE CASES AND USER INTERFACE PROTOTYPES

A use case description is a formal way to represent how an information systems interacts with its environment and who are the actors involved in such interactions. There are many ways in which use case diagrams and their descriptions may be constructed. In conjunction with use cases we will consider the early availability of the interaction prototype consisting of the User Interface (screens, dialogs, reports, etc.) and their flows (3); this alternative is consistent with many of the user-centered approaches (9).

Each use case is associated with its UI prototype. Each UI prototype consists of a central form and possible sequence of subsequent forms. Each form corresponds to a window/dialog/Web page/report allowing the access to the functionalities associated with that use case. For the purpose of this paper, we will consider that the complete UI is available to us. The principles and process of UI design can be found for example in (9,10). The UI driven approach advocates developing UI as early as possible and using it as a basis for developing of all other diagrams.

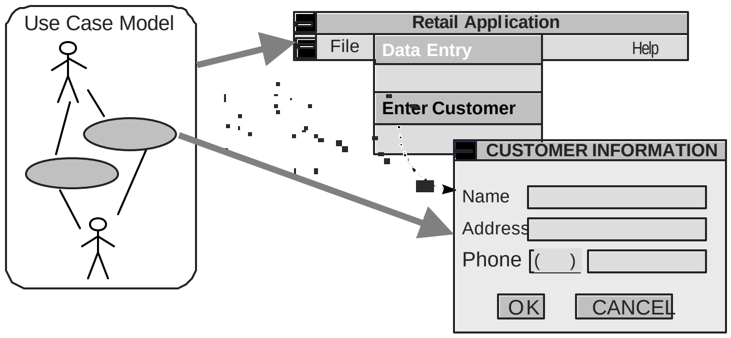


Figure 2: Use Case vs. User Interface Diagrams

An obvious completeness fault may be the omission of a valid use of the system. This means that each actor interacting with the use case needs to have an appropriate manner of interaction represented in the UI prototype, while UI elements not covered by use case descriptions should also arouse suspicion. Either can cause omission of a large number of required classes and interactions.

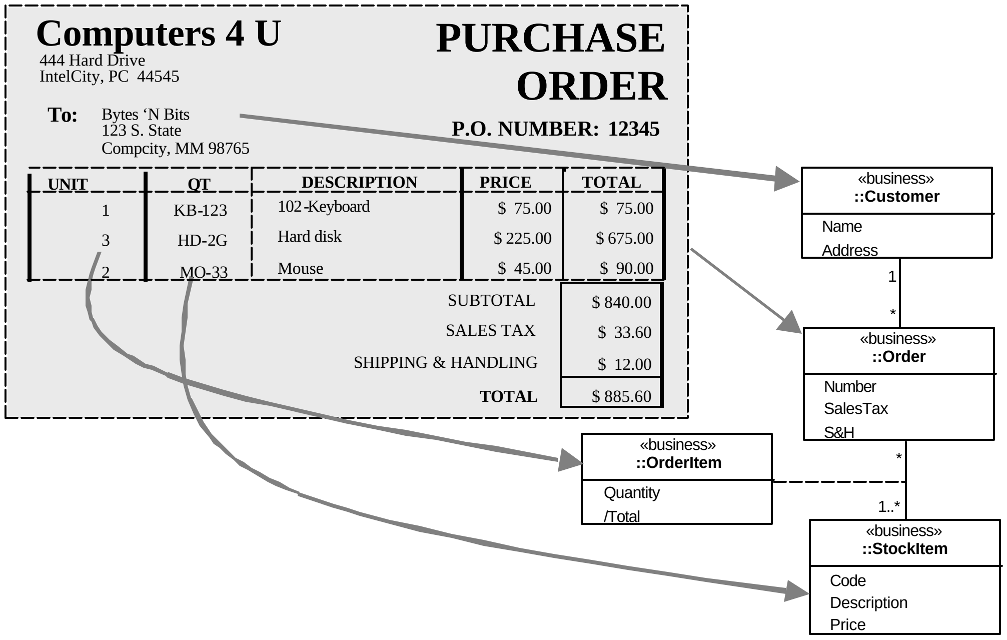
FROM USER INTERFACE TO CLASS DIAGRAMS

The UI forms are essential in capturing data requirements. Therefore, we can use the UI prototype as a source for developing class diagrams. The existence of a field on a dialog/Web page /report means that the data must either be an attribute of some object, the result of some operation on an object or series of objects, or be calculated from some object(s) attributes. Existence of the data about different objects on the same user interface means that dose objects are related to each other and results into an association between classes in the class diagram. Initial multiplicities for such associations might be detected by the occurrences of the related objects. Figure 3 shows how the key objects, attributes and relationships can emerge from the user interface prototype.

The union of all classes appearing in all UI forms represents a set of classes necessary to support that UI. Consequently, the union of all attributes for a given class that appear on all UI forms represents a set of necessary attributes for that class. The union of all associations among classes represents a set of required associations to support the given UI. Such an essential class diagram might be further improved to satisfy other requirements like accessibility, reusability, scalability, etc. The use of UI forms and reports to capture requirements for database design was also demonstrated in (4).

FROM USER INTERFACE TO SEQUENCE DIAGRAMS

Developing sequence diagrams is heavily influenced by the UI prototype, especially in understanding the different steps in a scenario or the sequence of scenarios. In our approach, the sequence diagrams are used to formally describe UI navigation. Note also that the ICONIX methodology (7) recommends the use of stereotyped class diagrams as an additional intermediate step between use cases and sequence diagrams. Our approach is fully consistent with such a possibility.



Computers 4 U
444 Hard Drive
IntelCity, PC 44545

PURCHASE ORDER

To: Bytes 'N Bits
123 S. State
Compcity, MM 98765

P.O. NUMBER: 12345

UNIT	QT	DESCRIPTION	PRICE	TOTAL
1	KB-123	102-Keyboard	$ 75.00	$ 75.00
3	HD-2G	Hard disk	$ 225.00	$ 675.00
2	MO-33	Mouse	$ 45.00	$ 90.00

SUBTOTAL	$ 840.00
SALES TAX	$ 33.60
SHIPPING & HANDLING	$ 12.00
TOTAL	$ 885.60

Figure 3: User Interface to Class Diagram

Each form of interaction with the use case needs to have the appropriate representation in the sequence diagram. In other words, each interface prototype is associated with a central view class (Main Menu object in Figure 4) and each window/dialog corresponds to a view class like CustomerInfo in Figure 4. At the same time, the existence of each form means that there needs

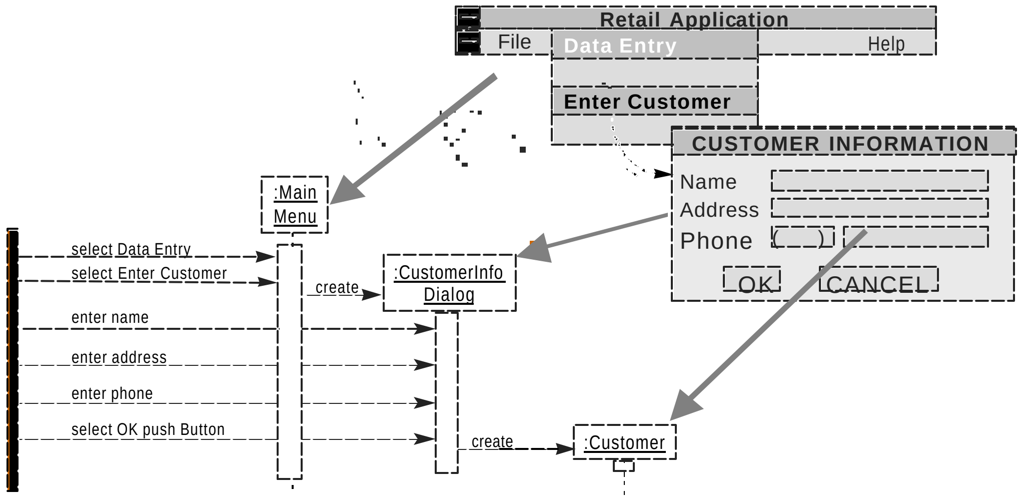


Figure 4. User Interface prototype to Sequence Diagram

to be a view class in the view class model. We use the view class/model name to avoid confusion with interface classes from UML. Menu options or buttons on screens typically trigger events sent either to the subsequent screen/dialog or to the application. Omission of any of these classes can cause omission of a large number of required operations for these classes.

The screen/dialog flows can lead to the definition of the interactions in the sequence diagram. For each screen/dialog all events generated through its buttons and menus need to be captured by the appropriate messages/events in a Sequence diagram.

If a sequence diagram covers the interaction among business classes, then each business class in a sequence diagram has to exist in the class diagram, see Figure 5. The fact that the object of one class sends a message to the object of another class means that there needs to be an association between these classes. The exception to this rule is the procedural relationship described in the next part. Simultaneously, each message received by some object in a sequence diagram has to be an operation for the corresponding class in the class diagram.

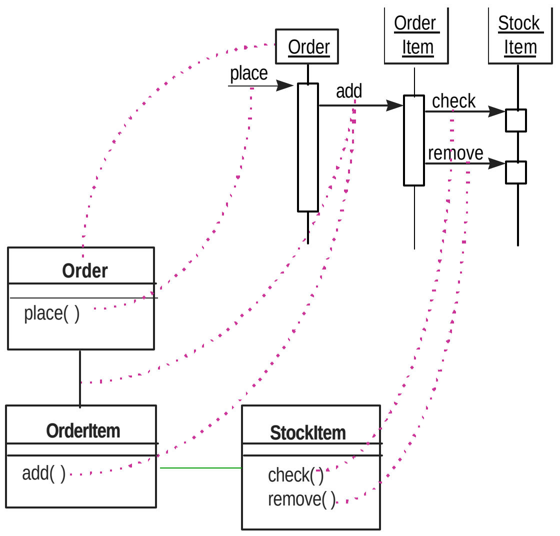


Figure 5. Sequence to Class Diagram

FROM USER INTERFACE TO COLLABORATION DIAGRAMS

Because sequence and collaboration diagrams arrive from the same information in UML's meta-model (1), these diagrams are semantically equivalent. As a result, a collaboration diagram can be directly derived from a given sequence diagram(s), see Figure 6.

Since a collaboration diagram emphasizes the organization of the objects that participate in an interaction, as Figure 6 shows, each <<business>> object from a collaboration diagram has a corresponding class in a class diagram. Next, each link that connects these objects and is not otherwise stereotyped has to map to an existent association in a class diagram. The role names for those links and associations need to be the same. There might be exceptions to this rule in a case of "procedural relationships" (8) in which case there will be no explicit association between these classes.

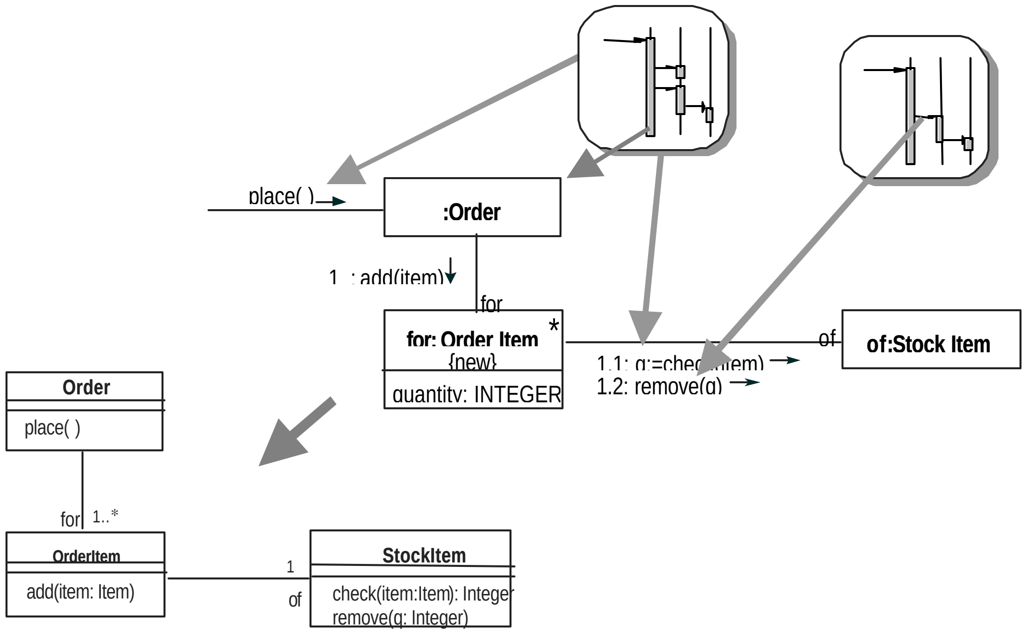


Figure 6. Sequence to Collaboration and Collaboration to Class Diagram

FROM USER INTERFACE TO STATECHART DIAGRAMS

Statechart diagrams are one of the UML diagrams for modeling the dynamic aspects of systems. They can be attached to a class, a use case or even an entire system. For the most part, they are used for modeling the behavior of reactive objects. A reactive object is one whose behavior is best characterized by its response to events dispatched from outside its context. Therefore, any such business object and almost all view objects may have associated statechart diagrams. In this paper, we will use statecharts to formally represent user interaction with each form in the given UI.

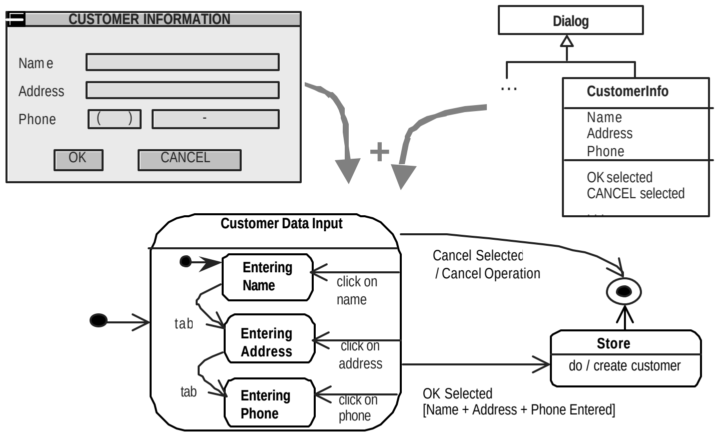


Figure 7. User Interface Prototype to Statechart Diagram

In the case of a statechart diagram for a view class, each independent event like data entry or button click need to be appropriately modeled as transitions in a statechart. This can be modeled using composite states as shown in Figure 7. All operations that are listed as part of a transition specification must also exist in the class diagram for the appropriate class. The same principle applies for all operations listed as actions or activities for the states. At the same time, role names used in a state chart must correspond to the appropriate role names in a class diagram.

CONCLUSIONS

The intention of this paper is to help software developers to successfully design information systems using UML diagrams. Emphasis is given to the use of the user interface as a basis for developing almost all diagrams. This paper presents the techniques on how to derive the Class, Sequence, Collaboration and State diagrams from the UI. The experience in using these techniques is that the larger the project, the more beneficial diagram derivation is.

Despite a growing number of CASE tools that support UML, very few of them emphasize diagram balancing. Most of the time they make sure that all elements belong to certain concepts without consideration if those concepts are really related to that diagram or not. This paper describes some techniques that ensure that modeling diagrams are more completely examined for consistency than in a typical design process.

The ultimate goal is to incorporate diagram balancing into the design process and that this evaluation is continuous throughout the design process. For example, as the sequence diagram is created, the class diagram can be checked to be certain that all messages in the sequence diagram correspond to associations in the class diagram.

REFERENCES

1. Booch, G. Rumbaugh, J. and Jacobson, I. (1998). The Unified Modeling Language User Guide, Addison-Wesley.
2. Fowler, M. (1997). Analysis Patterns: Reusable Object models, Addison-Wesley.
3. Gossain, S. (1995) Tracking Requirements in Object Development: Part II, Report on Object Analysis and Design, (2.3), 18-19,55.
4. Jovanovic, V. and Mrdalj, S. (1990). Three-Layered Approach to the Analysis of Forms and Transactions, Dallas, TX: Proceedings of the IAMM Conference, 141-148.
5. Kruchten, P. (2000). Rational Unified Process (2nd ed.), Addison-Wesley,
6. Larman, C. (1988). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice Hall.
7. Rosenberg D. (1999). Use Case Driven Object Modeling with UML, Addison-Wesley.
8. Rumbaugh, J., (1998). Depending on Collaborations: Dependencies as Contextual Associations, The Journal of Object-Oriented Programming, (11.4), 5-9.
9. Van Hartman M., editor (2001). Object Modeling and User Interface Design, Addison-Wesley.
10. Weinschenk, S., Jamar, P. and Yeo. S. (1997). GUI Design Essentials, John Willey & Sons.