Application System Requirements Definition Using the Unified Modeling Language (UML)

Robert Jackson
Brigham Young University

Follow this and additional works at: http://aisel.aisnet.org/amcis1998

Recommended Citation
http://aisel.aisnet.org/amcis1998/231
Application System Requirements Definition Using the Unified Modeling Language (UML)

Robert B. Jackson
Brigham Young University

Abstract

In the development of a new information system the first activity is usually a feasibility study. During the feasibility study it is frequently useful to elicit user requirements and develop a Requirements Definition Document. One method to document these requirements is by building high-level, abstract models. In this paper, we consider the utility of using diagrams from the Unified Modeling Language (UML) as the means to document these user requirements.

Introduction

In the Systems Development Life Cycle, normally the first activity is a feasibility study. Primary objectives of the feasibility study are to understand the users’ requirements, identify possible solution alternatives, analyze risks, and determine costs and benefits. At the conclusion of the feasibility study a management review takes place, agreement is reached on the scope and functionality of the proposed solution, and funds are allocated for the next phases of the project. An integral part of the results of the study and the documentation provided for the management review is the Application System Requirements Definition.

Definition of Terms

Application System Requirements, sometimes called Business Requirements, are the needs, expressed and unexpressed, of the information system user. Specifically, they are the information needs – information needs within the context of the users’ business or application. It is not uncommon for the user to recognize only a general need. He may be unable to cogently describe his detail requirements. Thus, one of the tasks of the feasibility study is to uncover these requirements, albeit at a fairly general level.

Specifications, on the other hand, are written descriptions that precisely describe requirements. There are all types of specifications: design specifications, programming specifications, hardware specifications, performance specifications, and so forth. During the analysis phase of a development project, Requirements Specifications are developed and documented in a set of models. At times formal specification languages are used. During the feasibility study, however, the level of detail and precision is necessarily limited because of time and budget considerations. However, the work done does provide the foundation for the analysis phase and the initial iteration of the specifications. This early high-level description is what is known as Application System Requirements Definition.

Components of a Requirements Definition

A Requirements Definition must satisfy several needs. First and foremost, as its name implies, it must capture and describe user information requirements. It must document these requirements in a form that is precise and easily understood by users. In large system development projects it must provide a mechanism to organize and structure these requirements. Its orientation must be from the user perspective.

A second objective of the requirements definition is to establish a scope for the new system. One ubiquitous problem in system development is “scope creep”. By defining the scope of the system (the services and capabilities to be provided), the project can be controlled better. Later scope changes can be identified easily, and their impacts on the project can be determined more precisely.

A third purpose of the requirements definition is to provide a blueprint for the planning and scheduling of the analysis phase. There are two aspects to this planning. First, it should provide structure for identifying and partitioning analysis tasks for the project schedule. Second, it is the preliminary version of the model instance diagrams that will be developed later during the analysis phase.

Unified Modeling Language and Object-oriented Development

The Unified Modeling Language (UML) is the unification and combined work of Grady Booch, James Rumbaugh, and Ivar Jacobson. UML has been proposed as an OO standard model to the Object Management Group (OMG). Version 1.1 of UML is available on the Internet for review and comments (http://www.rational.com/uml). Documents available include a summary, notation guide, semantics manual, meta-model description, constraint language description, and so forth. The authors are currently developing system development method guidelines. Defined within the language of UML is an extension mechanism that allows one to extend the modeling constructs as required by special situations.

Version 1.1 of UML has ten different types of diagrams defined. There is considerable overlap of information provided by these ten different diagrams. It may be inappropriate to call each one of these ten a separate model since the information content does overlap. For example, the collaboration diagram and the sequence diagram essentially capture the same
information. They can be considered the same model instance with slightly different diagram presentations. Some are strictly object-oriented in that they conserve object boundaries, and it is possible to identify objects and object classes within the diagram. Other diagrams are not strictly object-oriented because they do not identify, nor do they necessarily conserve, object boundaries.

For the feasibility study there are two UML diagrams that are particularly useful: the Use Case Diagram and the Activity Diagram. These are two of the diagrams that do not preserve object information. In fact, that is precisely the reason why they are beneficial for requirements definition. The development of these two diagrams is based on users’ answers to such questions as: “What do you need to do with the system?” and “How are you going to use the system?” The development of these diagrams does not require the systems analyst to ask the user to identify classes or objects in the system.

One other diagram in UML which may be useful for requirements definition is the Package Diagram. This diagram does preserve class integrity, not by requiring the definition of classes, but because it operates under an encapsulation rule: i.e., any given object can belong to only one package.

Package Diagram

The information provided by a Package Diagram for requirements is system boundary information as it relates to other systems or subsystems in a larger information system environment. In Figure 1 we see a Library Resource Management System, which we shall use for our example, a Student Registration System, and an Employee Database System. Each system or subsystem is drawn as a box with a tab on the top left corner. From this diagram, we observe that there are three separate systems and that the Library Resource Management System has dependencies on the other two systems. In other words, if either of the other two systems change, then changes will probably be necessary in the Library Resource Management System.

Use Case Diagram

A Use Case Diagram identifies all the ways that the system can be used. It is a diagramatic listing of all of the “uses” or functions of the system. There are four major constructs in a use case diagram: (1) System Boundary, (2) Actors, (3) Use Cases, and (4) Relationships. Figure 2 shows an example (partial) Use Case Diagram for the Library Resource Management System. The two Actors are the Library Employee and the Patron. It should be noted that actors represent roles and not necessarily physical persons or things. In this example, there are five “uses” identified. The Library Employee Actor will use the system to do such things as “Check out Item”, “Return Item”, and “Add New Item”. The Patron Actor, on the other hand, will use the system to “Search Catalogue”. The relationships between the actors and the use cases indicate which actors will use which use cases.

A Use Case diagram is an effective method of defining user requirements. When the use case diagram is complete, it identifies all the possible users of the system; i.e., all of the actor roles. It also identifies all of the anticipated uses of the system. In other words, all of the user functions that must be supported are identified. The system boundary is also identified, indicating which entities are external (actors) and which are internal (use cases). The relationship between the actors and the use cases is identified by the arrows. Although not shown in this simple example, a Use Case diagram also indicates relationships between use cases themselves.

Activity Diagram

Because the name of each use case is not descriptive enough to ensure that there is no miscommunication concerning the actual function to be supported, it becomes necessary to elaborate a use case with an activity diagram. This additional detail does several things. First, it ensures that both the developer and the user understand what is meant by each use case. This diagram enhances communication and understanding. The second major function of an activity diagram is to provide assistance in the planning of the analysis phase. Activity descriptions aid in estimating the scope of the required solution system. They also assist in the planning of tasks necessary to conduct a thorough analysis of the problem domain.

Each Use Case will have its own activity diagram. Figure 3 is an example of an Activity Diagram for the Check Out Item Use Case. The ovals represent activities, and the arrows represent the flow between activities. The cross-bar is a synchronization bar and is used to synchronize the beginning or ending of parallel activities. We also can decorate the diagram with notes (in text) and constraints or guards (in brackets). The “*” indicates repetition, with the note For each Item indicating the repetition condition.

Activity diagrams are good for telling what happens, but do not indicate which object does which activity. As indicated earlier, activity diagrams do not preserve object class boundaries. This problem can be alleviated somewhat by the addition of “swimlanes”. In Figure 3, swimlanes are added to indicate which activities are performed by the Library Employee actor, and which are performed by the system.

Activity Diagrams can be made more complex, but this simple example does illustrate the utility of using them to describe user requirements.
Information Flows

The one other type of information that is needed in a requirements definition is the information flow across the system boundary. In structured methodology this information is captured by the DFD Context Diagram. Unfortunately, there is not a similar, simple construct in UML to show the same information flow. The dilemma can be solved in one of two ways. One option is to add a Context Diagram to the set of requirements diagrams. Another alternative is to modify one of the existing UML diagrams. The best candidates for modification are the Use Case Diagram and the Collaboration Diagram. In the Use Case Diagram, each relationship can be decorated to show inflow and outflow information. Or the Collaboration Diagram can document interactions between objects. In this case, the objects consist of the actors and the system, and the interactions are identified, as well as the information being passed. At present, neither of these UML diagrams satisfy the need for information flow definition as well as a Context Diagram.

Conclusion

UML is a positive step forward in the development of some type of standard object-oriented methodology. The models have been defined, but there is not yet adequate methodology. In the case of Requirements Definition, there are some UML diagrams that are extremely useful. However, as seen in the case of information flows, one additional construct is needed. Obviously, the method of using these models needs to be more fully developed.

References

References will be furnished upon request from author (robert_jackson@byu.edu).