

Understanding Use Case Modeling

By Sinan Si Alhir

Sinan Si Alhir (salhir@earthlink.net, <http://home.earthlink.net/~salhir>) is a consultant and author of "UML in a Nutshell" (O'Reilly and Associates, Inc., 1998).

Published in "Methods & Tools" (April 2000) – An international software engineering digital newsletter published by Martinig & Associates.

Introduction

Following the "method wars" of the 1970s and 1980s, the Unified Modeling Language (UML) emerged from the unification that occurred in the 1990s within the information systems and technology industry. Unification was lead by Rational Software Corporation and Three Amigos, Grady Booch, James Rumbaugh, and Ivar Jacobson. The UML gained significant industry support from various organizations via the UML Partners Consortium and was submitted to and adopted by the Object Management Group (OMG) as a standard (November 17, 1997).

The UML is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. The language is broadly applicable to different types of systems (software and non-software), domains (business versus software), and methods and processes. The UML enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative, and incremental process that is object oriented and component based. The UML enables the capturing, communicating, and leveraging of knowledge: models capture knowledge (semantics), architectural views organize knowledge in accordance with guidelines expressing idioms of usage, and diagrams depict knowledge (syntax) for communication.

System development may be characterized as problem solving, including understanding or conceptualize and representing a problem, solving the problem by manipulating the representation of the problem to derive a representation of the desired solution, and implementing or realizing and constructing the solution. This process is very natural and often occurs subtly and sometimes unconsciously in problem solving.

Models are complete abstractions of systems. Models are used to capture knowledge (semantics) about problems and solutions. Architectural views are abstractions of models. Architectural views are used to organize knowledge in accordance with guidelines expressing idioms of usage. Diagrams are graphical projections of sets of model elements. Diagrams are used to depict knowledge (syntax) about problems and solutions. Within the fundamental UML notation, concepts are depicted as symbols and relationships among concepts are depicted as paths (lines) connecting symbols.

Use case modeling from the user model view (also known as the use case or scenario view), which encompasses a problem and solution as understood by those individuals whose problem the solution addresses, involves use case diagrams to depict the functionality of a system.

Use Case Diagrams

To successfully apply use case diagrams, we must first understand the types of elements used in use case diagrams.

Actors

Actor classes are used to model and represent roles for "users" of a system, including human users and other systems. Actors are denoted as stick person icons. They have the following characteristics:

- Actors are external to a system.
- Actors interact with the system. Actors may use the functionality provided by the system, including application functionality and maintenance functionality. Actors may provide functionality to the system. Actors may receive information provided by the system. Actors may provide information to the system.
- Actor classes have actors instances or objects that represent specific actors.

Figure 1 shows a project management system with a project manager actor and a project database actor. The project manager is a user who is responsible for ensuring the success of project and uses the system to manage projects. The project database is a system that is responsible for housing project management data.

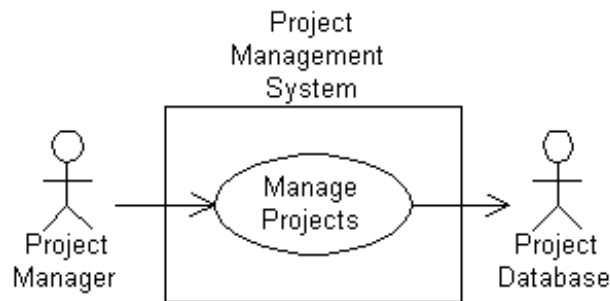


Figure 1

Use Cases

Use case classes are used to model and represent units of functionality or services provided by a system (or parts of a system: subsystems or classes) to users. Use cases are denoted as ellipses or ovals. They may be enclosed by a system boundary or rectangle labeled with the name of the containing system. They have the following characteristics:

- Use cases are interactions or dialogs between a system and actors, including the messages exchanged and the actions performed by the system. Use cases may include variants of these sequences, including alternative and exception sequences.
- Use cases are initiated by actors and may involve the participation of numerous other actors. Use cases should provide value to at least one of the participating actors.
- Use cases may have extension points that define specific points within an interaction at which other use cases may be inserted.

- Use case classes have use case instances or objects called scenarios that represent specific interactions. Scenarios represent a single sequence of messages and actions.

Figure 1 shows a project management system which provides the functionality to manage projects in which the project manager and project database participate.

Relationships

Association relationships between actor classes and use case classes are used to indicate that the actor classes participate and communicate with the system containing the use case classes. Association relationships are denoted as solid lines or paths. Arrowheads may be used to indicate who initiates communication in the interaction. If an arrowhead points to a use case, the actor at the other end of the association initiates the interaction with the system. If the arrowhead points to an actor, the system initiates the interaction with the actor at the other end of the association.

Figure 1 shows a project management system that provides functionality to manage projects. A project manager initiates this functionality and the system initiates the communication with the project database in providing this functionality.

Includes relationships from base use case classes to inclusion use case classes are used to indicate that the base use case classes will contain the inclusion use case classes; that is, the base use case will contain the inclusion use case. A base use case defines the location at which the inclusion use case is included. Includes relationships are denoted as dashed lines or paths with an open arrow-head pointing at the inclusion use case and are labeled with the `<<include>>` keyword (stereotype). The insertion of the inclusion use case involves the execution of the base use case up to the inclusion point, inserting and executing the inclusion use case, and then continuing with the execution of the base use case.

Figure 2 shows that a project manager may add projects and remove projects using the project management system. When removing projects, the functionality of finding a project is included into removing a project.

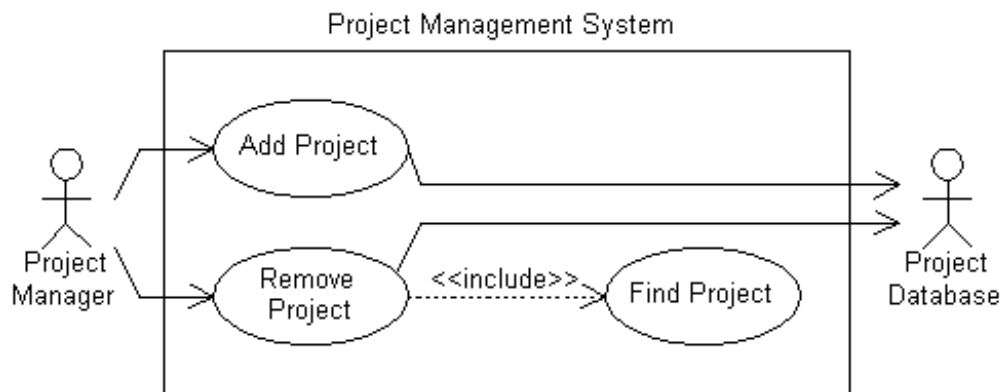


Figure 2

Extends relationships from extension use case classes to base use case classes are used to indicate that the base use case classes may be augmented by the extension use case classes; that is, the inclusion use case will augment the base use case if an extension condition is satisfied. A base use case defines the extension point. An extension use case defines the extension condition that must be satisfied in order to insert the extension use case into the base use case. The insertion of the extension use case involves the execution of the base use case up to the extension point, testing

the extension condition and inserting and executing the extension use case if the condition is satisfied, and then continuing with the execution of the base use case. Extends relationships are denoted as dashed lines or paths with an open arrow-head pointing at the extension use case and are labeled with the extension condition in square brackets, the <<extend>> keyword (stereotype), and the extension point name in parentheses. Extension points are identified in a compartment labeled "Extension Points" in the base use case.

Figure 3 shows that a project manager may update projects using the project management system. When updating projects, a project manager may manage tasks if the project manager selects the task option, and a project manager may manage resource if the project manager selects the resource option.

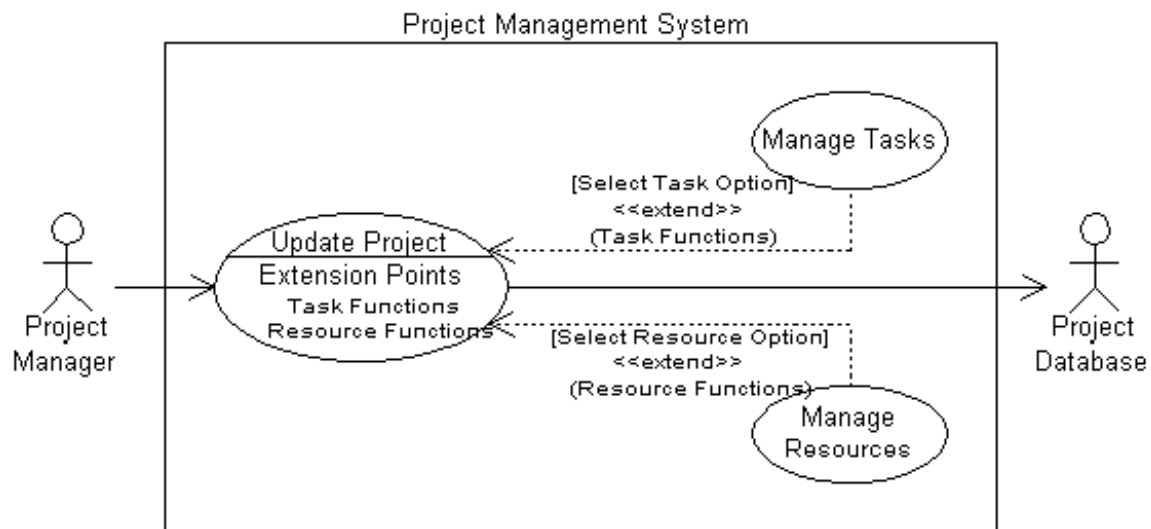


Figure 3

Generalization relationships from specialization use case classes to generalized use case classes are used to indicate the specialization use case classes are consistent with the generalized use case classes and may add additional information. A specialization use case may be used in place of a generalized use case and may use any portions of the interaction of the generalized use case. Generalization relationships are denoted as solid lines or paths with a hollow arrow-head pointing at the generalized use case.

Figure 4 shows that a project manager may publish a project schedule by sending e-mail to project team members using an e-mail system or by generating a web-site on a web-site host. In either case, there will be common functionality used from the generalized use case, for example: inputting project name, extracting the relevant project information from the project database, etc.

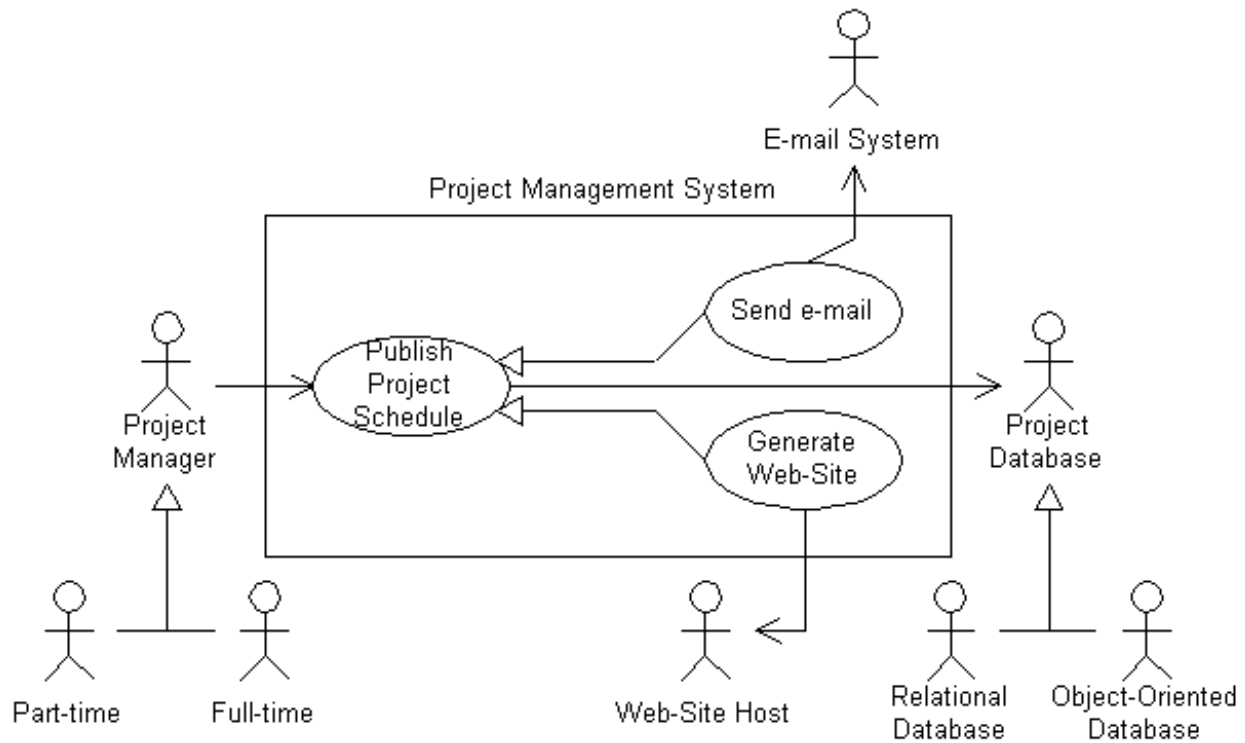


Figure 4

Generalization relationships from specialization actor classes to generalized actor classes are used to indicate the specialization actor classes are consistent with the generalized actor classes and may add additional information. A specialization actor may be used in place of a generalized actor and receives the characteristics of the generalized actor. Generalization relationships between actors are denoted similarly to generalization relationships between use cases.

Figure 4 shows that there are two types project managers, full-time and part-time, and that there are two types of project database, relational database management systems (R-DBMS) and object oriented database management systems (OO-DBMS). Any type of project manager may publish a project schedule using any type of project database.

Use Case Modeling

To successfully apply use case diagrams in use case modeling, we ought to be aware of various guidelines (lessons learned) from applying this technique.

Actors

When modeling actors, we ought to be aware of the following guidelines:

- Actors should be named using noun phrases.
- Actors should be described, indicating what interests an actor has in interacting with the system. For example, the project manager is responsible for ensuring the success of projects, and the project database is responsible for housing project management data.

- Actors define the scope of a system and identify those elements that reside at the periphery of the system and those elements on which the system depends. For example, these use case diagrams indicate that the project management system depends on a project database to provide functionality to a project manager, both residing on the periphery of the system.

Furthermore, other guidelines may be applied in addition to those above.

Use Cases

When modeling use cases, we ought to be aware of the following guidelines:

- Use cases should be named using verb–noun phrases.
- Use cases should be described, indicating how they are started and end, any conditions that must be satisfied before the use case starts (pre–conditions), any conditions that must be satisfied when the use case ends (post–conditions), the sequence of exchanged messages and performed actions, the data exchanged, and any non–functional characteristics (reliability, performance, supportability, etc. constraints). This descriptions may be captured using text and other UML diagrams.
- Use cases define the scope of a system and define the functionality provided by the system and those elements on which the system depends in order to provide the functionality. For example, these use case diagrams indicate that the project management system provides functionality to manage projects to a project manager, and this functionality is implemented using the project database.
- Use cases should facilitate actors in reaching their goals. Use cases are system functionality or responsibilities (requirements) that actors use in order to reach or satisfy their goals. Use cases are not simply actor goals. For example, a project manager is responsible for ensuring the success of projects, and a project database is responsible for housing project management data. The project management system provides functionality to manage projects to a project manager such that the project manager can ensure the success of projects.
- Use cases should facilitate the architecture of a system. Use cases may be organized and partitioned using includes, extends, and generalization relationships to identify, extract, and manage common, optional, and similar functionality. The organization of a set of use cases is not simply the architecture of the system. However, the architecture of a system is based upon the various technology, infrastructure, etc. considerations relevant to satisfying the use cases. For example, the project management system must interface with an e–mail system and a web–site host, thus appropriate subsystem elements must exist within our architecture to facilitate these interfaces.
- Use cases provide flexibility and power throughout the life–cycle process. They provide the freedom to work with a use case as a whole or any subset of a use case via scenarios. The use of includes, extends, and generalization relationships to identify, extract, and manage common, optional, and similar functionality provides further flexibility in working with use cases. Furthermore, use cases may be used to model interactions between actors and systems, subsystems, and classes at various levels of abstraction. This flexibility and power is propagated to every application of use cases. For example, if time, resources, or funding are not sufficient to implement a whole use case, various scenarios may be selected for implementation based upon these factors.
- Use cases may be used as the basis for planning. Time and resource estimates may be associated with use cases. If estimates for a use case cannot be derived, estimates for each scenario of a use case may be derived and used to potentially estimate the overall time and resource estimates for the use case as a whole. This helps ensure that planning is done with the objective of satisfying the requirements.
- Use cases may be used as the basis for analysis, design, and implementation. The sequence of exchanged messages and performed actions within the description of a use case are analyzed and the system is design

and implemented to specifically realize use case interactions. This helps ensure that every element of a system is created and used because it contributes to satisfying the requirements.

- Use cases may be used as the basis for testing. The sequence of exchanged messages and performed actions within the description of a use case may be used as test scripts for validating the functionality of a system. This helps ensure that the system is tested and validated against the requirements.
- Use cases may be used as the basis for documentation since use cases capture how users will use the system.

Furthermore, other guidelines may be applied in addition to those above.

Conclusion

As the Unified Modeling Language (UML) is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process, by understanding the types of elements used in use case diagrams and being aware of various guidelines (lessons learned) from applying this technique, we have a sound foundation for successfully applying the technique. Furthermore, it is experience, experimentation, and application of the standard and its various techniques that will enable us to realize its benefits.
