Use cases

Paul Jackson

School of Informatics
University of Edinburgh

The Unified Modeling Language

UML is a graphical language for recording aspects of the requirements and design of software systems.

It provides many diagram types; all the diagrams of a system together form a UML model, which must be consistent (in a weak sense...).

Mostly tailored to an OO world-view

Often used just for documentation, but in model-driven development, a UML model may be used e.g. to generate and update code and database schemas automatically.

Many tools available to support UML

UML: Use cases

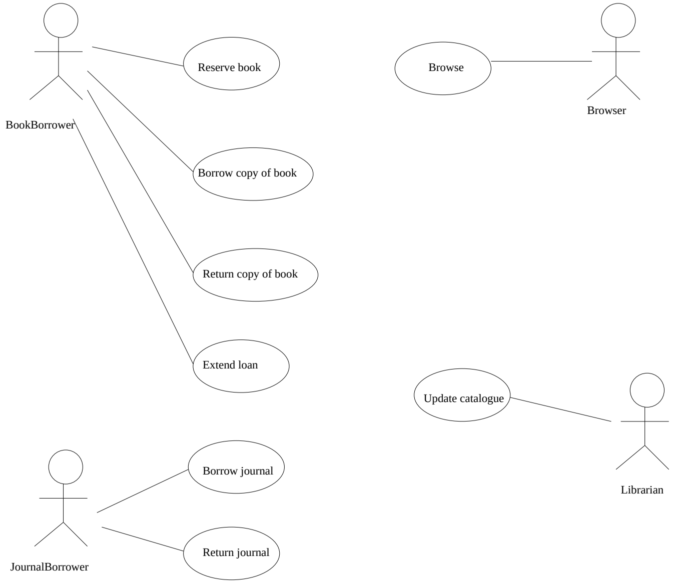
Document the functional requirements of the system *from the users' points of view.*

They help with three of the most difficult aspects of development:

- capturing requirements
- planning iterations of development which are good for users
- meaningful system testing

A set of use cases is *summarised* in a UML use case diagram.

A simple use case diagram



UML use cases: Actors

An **actor** – shown as a stick figure – can be:

- a human user of the system *in a particular rôle*
- an external system, which *in some rôle* interacts with the system.

Not a user: a particular *kind* of user. E.g., Bank Customer.

The same human user or external system may interact with the system in more than one rôle: he/she/it will be (partly) represented by more than one actor. (e.g., a bank teller may happen also to be a customer of the bank).

What is a use case?

A task involving the system which has value for an actor, e.g. `Borrow copy of book`.

Shown on diagram as named oval.

Each use case

- has a discrete goal the actor wishes to achieve
- includes a description of the a sequence of messages exchanged between the system and actors, and actions performed by the system, in order to achieve the goal.

Use cases primarily capture functional requirements, but sometimes non-functional requirements are attached to a use case.

Other times, non-functional requirements apply to subsets or all of use-cases.

Paths in a use case

Usually a use case describes a core of sequence of steps necessary to achieve its goal.

However might be alternatives, including some handling when all does not go to plan and the goal is not achieved.

Each path through use case called a *use-case instance* or *scenario*

- One talks about the main success scenario and alternate success or failure scenarios.

All scenarios in a use-case tied together by common user goal.

Warning: Sometimes *scenario* and *use-case* are synonyms

Example of use-case paths

Goal: Buy a Product

Main Success Scenario

1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills shipping info.
4. Systems presents full pricing information.
5. Customer fills in credit card info.
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirmation email to customer

Example of use-case paths (cont)

Extensions - variations on main success scenario

- 3a . If customer is regular customer
  - .1 . System displays current shipping and billing information
  - .2 . Customer may accept or override these defaults, returns to MSS at step 4, but skips step 5.
- 6a . System fails to authorize credit card purchase
  - .1. Customer may reenter credit card information or may cancel

Example fields in use case template

- Goal What the primary actor wishes to achieve
- Summary A one or two sentence description of the use case.
- Primary actor
- Secondary actors
- Trigger The event that leads to this use case being performed.
- Pre-conditions/Assumptions What can be assumed to be true when the use case starts
- Guarantees What the use case ensures when it succeeds
- Main Success Scenario
- Alternative scenarios

Use cases: connections and scope

A use case:

- can have different levels of detail, e.g. depending on where it is used in development process
- may refer to other use cases to provide further information on particular steps
- may be associated with other UML models (e.g. sequence and state diagrams) which show how it is realised;
- May describe different scopes: e.g. a system of systems, a single system or a single component of a system

Requirements capture organised by use cases

Use cases can help with requirements capture by providing a structured way to go about it:

1. identify the actors
2. for each actor, find out
   - what they need from the system
   - any other interactions they expect to have with the system
   - which use cases have what priority for them

Good for both requirements specification and iterated requirements elicitation.

There may be aspects of system behaviour that don't easily show up as use cases for actors.

Analysis vs design

Some actors are part of the requirements: usually the ones who derive benefit from a use case.

Others are part of the (business process) design: the ones who interact with the computer system to provide the benefit.

For example, consider a `FindBook` use case of a library, in which the user enters details of a book and wants to end up with a copy of it. Maybe the system will give the user directions to where the book is on the shelf. Maybe it will alert a librarian to go and fetch it. In the latter case, should the librarian be shown as actor? In some sense, the choice is a design decision.

Using use cases in development

Use cases are a good source of system tests: requirements documented as desired interactions, which translate easily into tests.

Earlier, they can help to validate a design. You can walk through how a design realises a use case, checking that the set of classes provides the needed functionality and that the interactions are as expected.

Politics

Through emphasising the value of tasks to actors, use cases help us understand *what is important to whom*.

To avoid a project being cancelled, should make sure system delivers added value:

- soon
- to all the people who might scupper it
- in every iteration

Of course, use cases might identify a project is not worth even starting because benefits to key actors are minimal

Possible problems with use cases

- Interactions spelled out may be too detailed, may needlessly constrain design
- May specify secondary actors that are not essential for fulfilling goal of primary actor
  - Does borrowing a book have to involve a librarian?
- Focus on operational nature of system may result in less attention to software architecture and static object structure
  - Refactoring during design may help
- May miss requirements not naturally associated with actors

Reading

Required: *Tokeneer ID Station System Requirements Specification*. See Section 5 for use-case-like scenarios. More generally, browse the document to get a feel of what a real Requirements Specification looks like.

Suggested: *Sommerville*. Use Cases discussed are both in Requirements and System Modeling chapters. Look up Use Cases in index to find the relevant sections.

Suggested: *Stevens*, Chapter 7.