

Requirements Engineering

Chapter 3.2.6 Using Use Cases to Elicit Requirements

Learning Objective

... To introduce the process of describing the system as a number of *use cases* that are performed by a number of *actors*. Actors constitute the entities in the environment of the system. Use cases describe what takes places within the system. A use case is a specific way of using the system by performing some part of the system functionality.

Frederick T Sheldon

Assistant Professor of Computer Science
University of Colorado at Colorado Springs

OOSE analysis models

- ⊗ Object-oriented software engineering (OOSE) proposes two analysis models for understanding the problem domain
 - Requirements Model
 - Analysis Model
- ⊗ The requirements model serves two main purposes
 - To delimit the system
 - To define the system functionality

Constructing the requirements model

- ⊗ Conceptual model of the system is developed using:
 - Problem domain objects
 - Specific interface descriptions of the system (if meaningful to the system being developed)
- ⊗ The system is described as a number of *use cases* that are performed by a number of *actors*
 - Actors constitute the entities in the environment of the system
 - Use cases describe what takes places within the system
 - A use case is a specific way of using the system by performing some part of the system functionality

The analysis model

- ⊗ The analysis model gives a conceptual configuration of the system. It consists of:
 - The entity objects
 - Control objects
 - Interface objects
- ⊗ The analysis model forms the initial transition to object-oriented design

Use case example (recycling machine)

- ⊗ A software system is to be developed to control a recycling machine for returnable bottles, cans and crates (used to hold bottles).
 - The machine can be used by several customers at the same time.
 - Each customer can return all three types of item on the same occasion.
 - The system should be able to distinguish between different types and sizes of bottle and can.
 - The system is required to register the number and type of items returned by each customer
 - The system should be able to print out a receipt on request showing: the items deposited, the value of the returned items and the amount to be paid to the customer.

Recycling machine example continued

- The system is also to be used by an operator. The operator requires a daily printout of the items deposited during the day. The printout should include the number of each item type
- The operator also requires a facility for modifying the item information stored in the system. For example, the deposit values of the items.
- In the event of an item getting stuck in the system, the system should alert the operator by setting off an alarm.

Requirements model

- ⊗ The requirements model for the recycling system will comprise three parts:
 - The use case model
 - The problem domain model
 - User interface descriptions

Actors

- ⊗ In order to identify use cases to be performed in the system, we need to first identify system users
- ⊗ The system users are referred to as *actors*.
- ⊗ Actors model the prospective ‘users’ of the system.
 - An actor is a user type or category. When an actor does something, the actor acts as an occurrence of that type.
 - An actor may represent a person or another system interacting with the intended system

Actors and role playing

- ⊗ One person can instantiate (play the roles of) several different actors
- ⊗ Actors define the roles that users can play
- ⊗ Actors model anything that needs to exchange information with the system.
- ⊗ Actors can model human users but they can also model other systems communicating with the intended system

Identifying actors

- ⊗ Actors constitute anything external to the system
- ⊗ Identifying all the relevant actors for a system may require several iterations
- ⊗ General guidelines include the following:
 - Ask yourself why the system is been developed
 - Who are people the system is intended to help?
 - What other systems are likely to interface with new system?
- ⊗ Actors who use the system directly (or in their daily work) are known as *Primary* actors
- ⊗ Primary actors are associated with one or more of the main tasks of the system

Identifying actors continued

- ⊗ Actors who are concerned with supervising and maintaining the system are called *secondary* actors
- ⊗ We can identify two actors in recycling system:
 - The customer
 - The operator
- ⊗ These are the entities in the environment of the system, that interact with it.

Interaction of actors

- ⊗ The customer interacts with the system by:
 - depositing items into the machine
 - receiving a receipt from the machine
- ⊗ The operator interacts with the system by:
 - Getting daily deposit reports
 - Maintaining item database
- ⊗ However the operator can occasionally deposit his/her own bottles in the machine. In which case he acts like an *instance* of the customer.

Primary and secondary actors

- ⊗ The distinction between the *primary* and *secondary* actors has a bearing on the system structuring
- ⊗ The system requirements are structured on the basis of its main functionality
- ⊗ Primary actors govern the system structure. Thus when identifying use cases, we first start with the primary actors

Use cases for the recycling system

- ⊗ After the actors have been identified the next step is to define the functionality of the system. This is done by specifying *use cases*.
- ⊗ Actors are a major tool in finding use cases. Each actor will perform a number of use cases in the system.
- ⊗ Each use case constitutes a complete course of events initiated by an actor and specifies the interaction that takes place between the actor and the system

Use cases continued

- ⊗ A use case is a special sequence of related transactions performed by an actor and the system in dialogue.
- ⊗ The collective use cases should specify all the existing ways of using the system

Instantiating use cases

- ⊗ Like actors use cases can be instantiated. This is done every time a user performs a use case in the system.
- ⊗ It is not always possible to identify instances of use cases early in the project

Telephone example

- ⊗ For example, consider a telephone exchange:
 - One actor is a subscriber and a typical use case to make a local telephone call. This use starts when the subscriber lifts his telephone receiver.
 - Another use case is to order a wake-up call.
 - Both use cases start when the subscriber lifts the telephone.
 - However, when the subscriber lifts his telephone, its not obvious which use case he would like to perform.
 - Thus uses cases may begin in a similar manner but we may not know which use case is to be carried out until its over.
 - The actor should be viewed as someone who initiates a course of events that eventually results in a complete use case. Rather than someone who demands that a use case be performed.

Identifying use cases

- ⊗ The first step in identifying the use cases is to examine the requirements from the user's perspective.
- ⊗ The following questions may be helpful in identifying use cases
 - What are the main tasks of each actor?
 - Will the actor have to read/write/change any of the system information?
 - Will the actor have to inform the system about outside changes?
 - Does the actor wish to be informed about the unexpected changes?

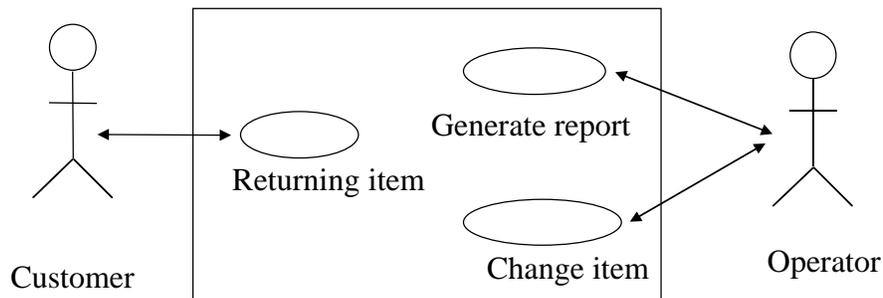
Use cases for recycling system

- ⊗ We can identify a number of use cases for the recycling system
- ⊗ Customer
 - Should be able to return deposit items (i.e. cans, bottles). The use case is *returning item*.
 - This use case should include all the events until a receipt is given
- ⊗ Operator
 - Should be able to get a daily report of the items deposit. This will be one use case, *Generate daily report*.
 - Should also be able to modify information in the system, for example, each item's deposit value. We will call this use case, *Change item*

Use case descriptions

- ⊗ Return item use case
 - Will be initiated by the *customer* when he/she returns deposit items. The system will keep a running count of the item types and their daily total.
 - When the customer has deposited all his/her items, he/she will press the receipt button to get a receipt. The printed receipt will list the items deposit, their totals and the amount to be paid to the customer.
- ⊗ Generate daily report use case
 - Will be initiated by *operator* when he wants to get a printout of the items returned on the day. The system will print out the type and number of each item, and the overall total.
- ⊗ Change item use case
 - Will initiated by the *operator* to modify the item information in the system. He will be able to change the return value and the size of each returnable item

First level use case for the RSM



Expressing use case variants

- ⊗ It is not always obvious which functionality should be placed in separate use cases, and what is only a variant of the same use case
 - If the differences are small they can be described as variants of the same use case
 - If the differences are large they should be described as separate use cases
- ⊗ Basic course of events
 - Describe the normal sequence of events in a use case
- ⊗ Alternative courses of an event
 - Describe variants of the basic course of events (e.g. errors)

Adding detail to the use case

⊗ Returning item

When the customer returns a deposit item, it is measured by the system. The measurements are used to determine what kind of can, bottle or crate has been deposited. If accepted, the customer total is incremented, as is the daily total for that specific item type. If is not accepted, a 'NOT VALID' message is displayed.

When the customer presses the receipt button, the printer prints the date. The customer total is calculated and the following information printed on the receipt for each item type:

item name

number returned

deposit value

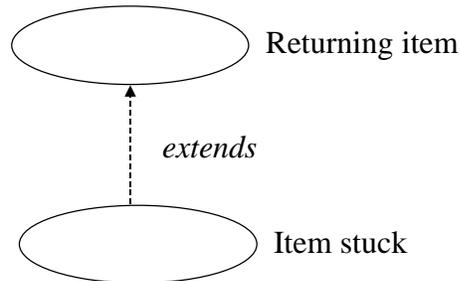
total for this type

Finally the sum to be paid to the customer is printed on the receipt

The extension relationship

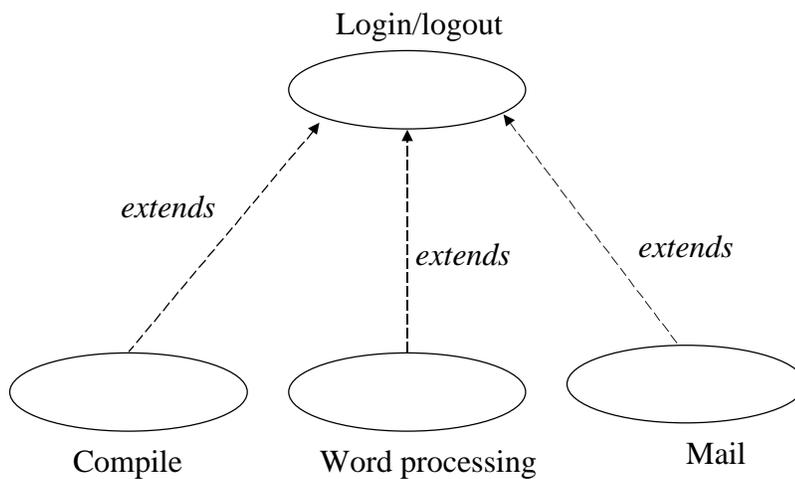
- ⊗ The *extension* specifies how one use case description may be used to extend another use case description.
- ⊗ The use case which is to be extended should be a complete course in itself.
- ⊗ An example of an extended use case is the following requirement:
when an item is stuck in the system issue an alarm
- ⊗ This can be described as a use case description that extends the *Returning item* use case

Extension example 1



When an item gets stuck the alarm is activated to call the operator
When the operator has removed the stuck item he resets the alarm
and the customer can continue to return items.

Extension example 2



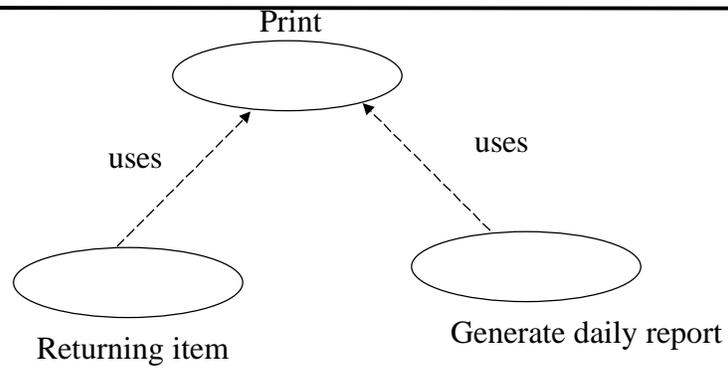
Extension (continued)

- ⊗ An extension is thus used to model extensions of other complete use cases. Examples include:
 - To model optional parts of use cases
 - To model complex and alternative courses which seldom occur, for instance *Item stuck*
 - To model separate sub-courses which are executed only in certain cases
 - To model the situation where several different use cases can be inserted into a special use case, such as login/logout

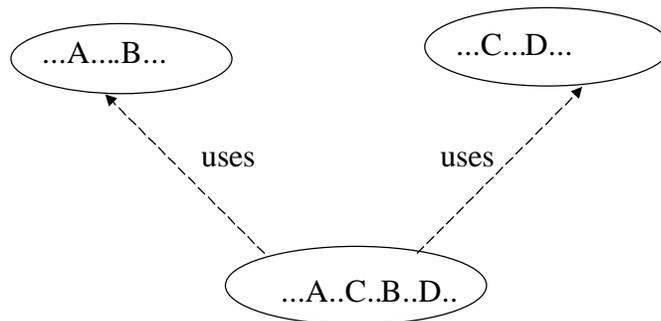
The uses relationship

- ⊗ It is possible to abstract similar behavior in use cases. Normally a similar behavior between use cases is identified *after* the use cases have been identified.
- ⊗ The two uses cases; *Returning item* and *Generate daily report* will both need to print out a receipt. We can identify an abstract use case *Print* that performs this printing.

Uses Example 1



Uses example 2



Interface descriptions

- ⊗ When describing use cases and communicating them to their potential users; it is often appropriate to describe the interface in more detail
- ⊗ Sketches of what the user is likely to see on the screen/display can often help make the use more understandable
- ⊗ Simulate the way the use case will appear to the user

Problem domain objects

- ⊗ Define the logical view of the system
- ⊗ Forms the initial transition to o-o design and eventually implementation