Themes and Use Cases: Comparison and Transformation

Erasmus Mobility at Lancaster University

Lecture 2

Valentino Vranić

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
vranic@fiit.stuba.sk
http://fiit.stuba.sk/~vranic/

November 23–26, 2009

Overview

Introduction

Aspect-Oriented Analysis

- As object-oriented development, aspect-oriented development also starts with analysis
- Use case modeling is a common approach to object-oriented analysis
- No common approach to aspect-oriented analysis yet
- Ivar Jacobson showed that use cases not only can be used in aspect-oriented development, but that they are intrinsically aspect-oriented
- Their full-fledged application in aspect-oriented development requires some extensions in the use case diagram notation

Theme

- Theme is an important approach aspect-oriented analysis and design
- Its analytical part, Theme/Doc, is a kind of conceptual modeling
- Theme/Doc is based on the notion of theme
- A theme bears the semantics of a use case
- It may be desirable or necessary to switch from one way of analysis to the other
- For example, an initial—possibly automated—theme identification may be to transformed to the more widely accepted use case modeling
- Also, a switch from use case modeling into Theme/Doc may be needed in order to employ Theme/UML for design

Aspect-Oriented Analysis

- A way of transforming a Theme/Doc model into a use case model and vice versa will be presented here
- The transformation uncovers differences and commonalities between use cases and themes in detail
- Despite Theme/Doc lacks the elements that would correspond to actors, flows, or extension points, the most of the themes are transformed directly into use cases and vice versa with a quite straightforward derivation of the relationships among them

Theme/Doc: An Example Scenario

The Theme/Doc Approach (1)

- Theme/Doc is the analytical part of the Theme approach[1]
- It consists of the following four main activities performed in iterations:
  1. Choose a starting set of potential themes
  2. Refine the set of themes
  3. Identify which of the themes are aspects
  4. Prepare for design

[1] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley, 2005.

The Theme/Doc Approach (2)

- A theme is a modularization construct that encapsulates a concern[2]
- Themes are graphically represented in a simple notation
- Three different views are used:
  - theme–relationship
  - crosscutting
  - individual view

[2] P. Sánchez, L. Fuentes, A. Jackson, and S. Clarker. Aspects at the right time. Transactions on Aspect-Oriented Software Development, IV:54–113, 2007.

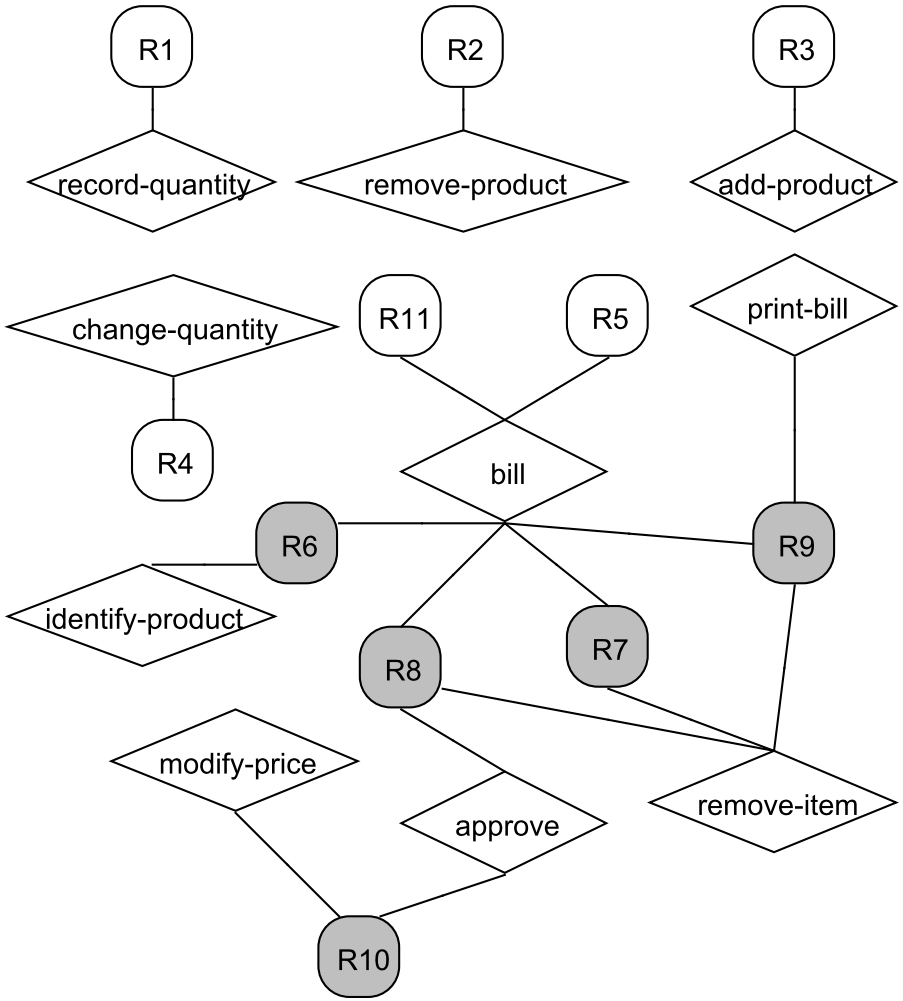
An Example Scenario (1)

- An example scenario of a simplified retail support application development that includes store management, price management, and cash desk functionality provided simultaneously on several PCs in the store
- Requirements:

An Example Scenario (2)

1. The application will record and maintain the product quantity in the stock in the central database.
2. The storekeeper can remove products from the database.
3. The storekeeper can add products into the database.
4. The storekeeper can change the product quantity in the database.
5. The cashier can bill the item by manually entering the bar code or with a bar code reader.
6. Only the products recorded in the database can be billed.
7. The billed items can be removed from the bill until it has been closed.
8. The billed item removal must be approved by a store manager by entering his authentication data.
9. The billed items will be printed on the cash desk bill as they are entered. The bill will embrace the store name, billed items, information on removed billed items, the total amount of money to be paid, and date and time.
10. The product price can be entered or modified only by a properly authenticated store manager.

The Themes in the Retail Support Application



Aspect Separation (1)

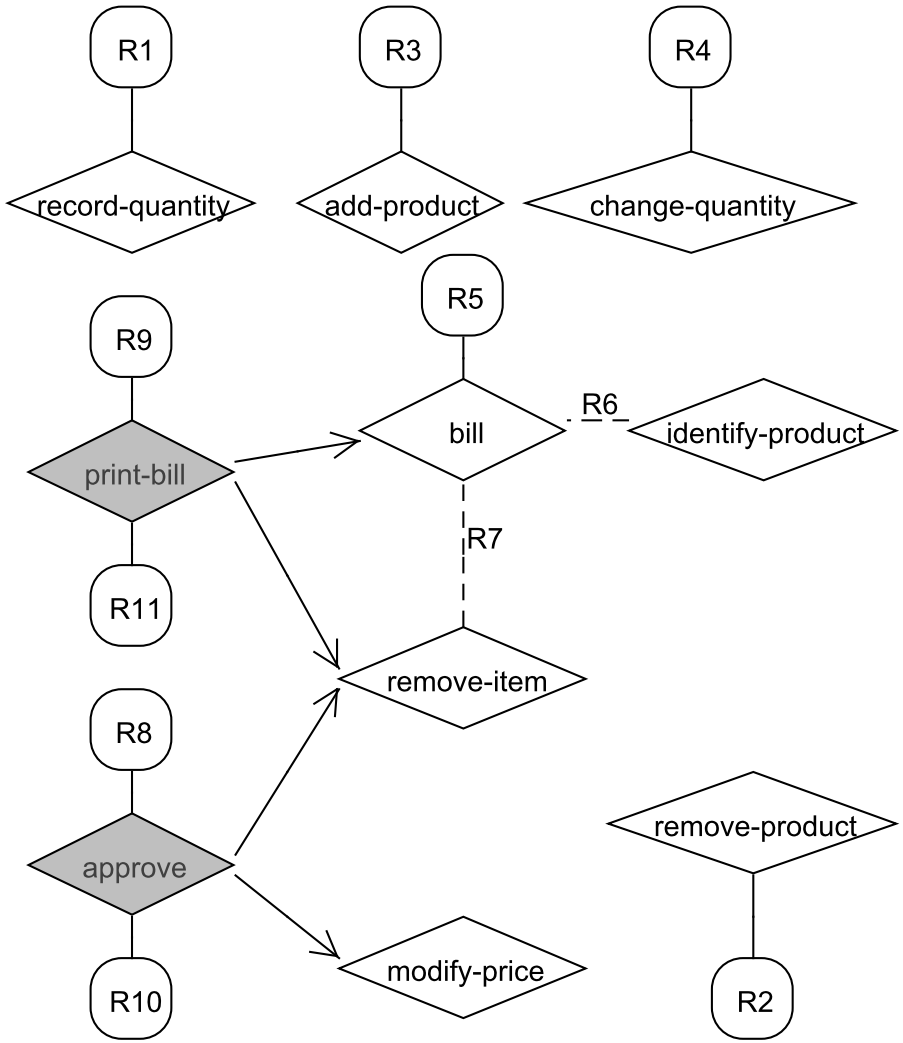
- Five shared requirements: R6, R7, R8, R9, and R10
- R8 is about removing the billed item, while the theme *bill* is about billing in the sense of adding items to the bill which is covered by the R7 requirement, so the connection between R8 and the *bill* theme can be omitted.
- Removed items printing can be separated from the R9 requirement into a new requirement R11, so the connection between the R9 requirement and the *remove-item* theme can be omitted:

Aspect Separation (2)

9. The billed items will be printed on the cash desk bill as they are entered. The bill will embrace the store name, billed items, the total amount of money to be paid, and date and time.
11. The removed items will be printed on the cash desk bill as they are entered.

- If no further requirement rewriting can help isolating requirements, the remaining requirement sharing is due to the crosscutting nature of some themes

The Crosscutting View



No Dominant Theme

- For some requirements sharing it may be impossible to determine the dominant theme—indicated by dotted edges between themes marked by respective requirement number
- Postponed to design

Transforming Themes into Use Cases

Basic Comparison

- Both themes and use cases represent functionality and have active names
- Use cases are not just any functionality, and it may seem themes are, but—if at all—only in initial phases of automated theme identification
- Use case identification is similar to theme identification (use cases are even acknowledged as one of the sources of themes)
- Similarity in relationships:
  - Crosscutting relationship between themes and extend relationship between use cases
  - Grouped themes resemble include relationship between use cases
  - Unified themes seem as a rudimentary form of generalization
- A significant difference: themes lack a direct description whereas use cases are primarily textual
- No actors in the theme model

Transforming Themes into Use Cases (1)

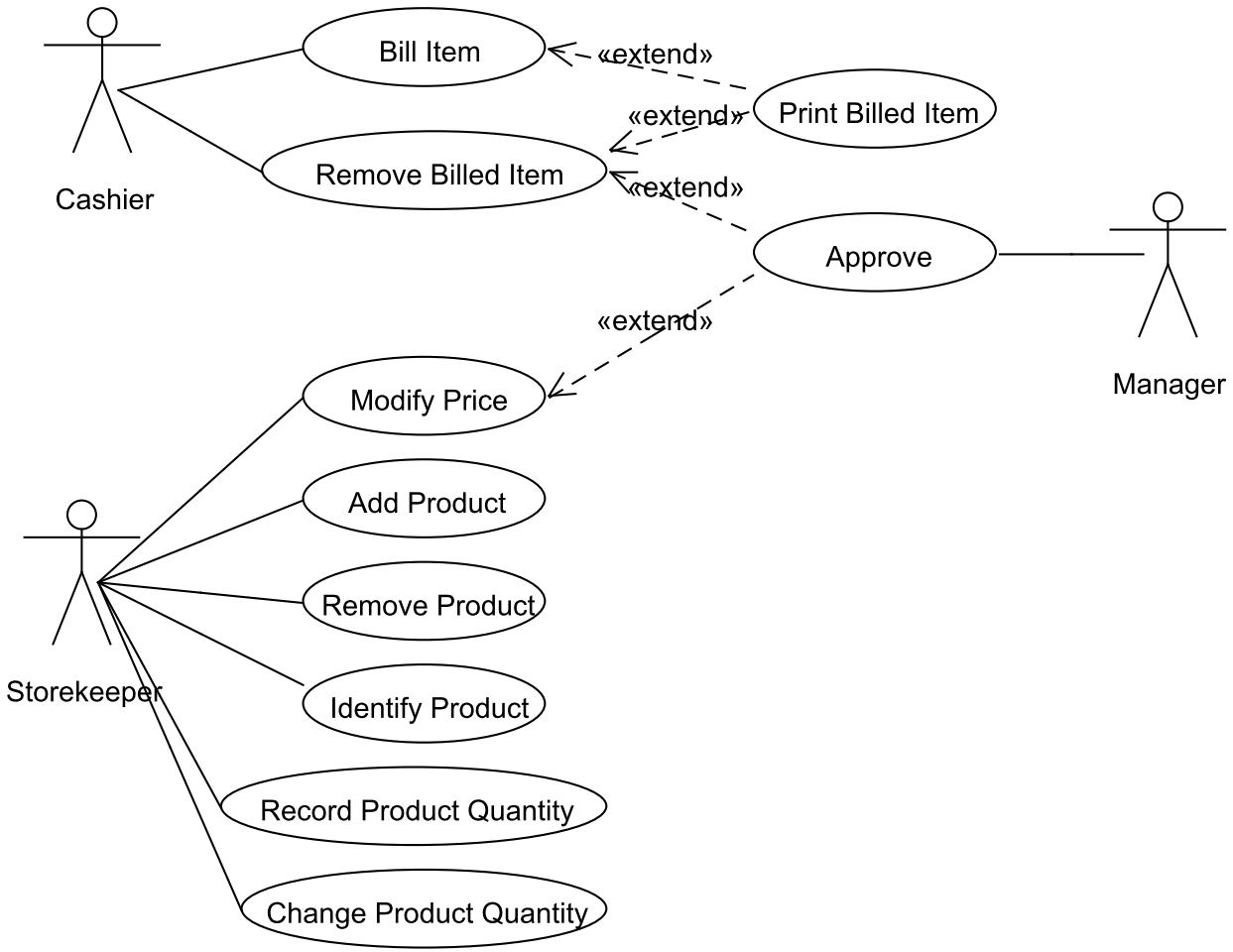
1. Create a use case for each theme. Identify actors in requirements.
2. Create an extend relationship for each crosscut relationship found in the crosscutting view preserving its direction.
3. Consider splitting themes. Identify grouped themes in individual theme views (both the existing ones and those obtained in step 1). Consider transforming each theme–subtheme relationship into an include relationship or into a generalization relationship if the theme and subtheme conceptually represent the same theme. Deciding not to transform the subtheme means deciding its functionality will be an integral part of the existing use case possibly as a separate flow.

Transforming Themes into Use Cases (2)

4. Consider unifying themes. Identify unified themes in the history of the operations performed upon the theme model if it is available. Consider transforming unified themes into generalizations.
5. Consider the granularity of the obtained use cases and restructure them as necessary by including too low level use cases as flows of regular ones.
6. Resolve the postponed relationships.

The Initial Use Case Model

- Identified use cases and extend relationships



Renaming Use Cases

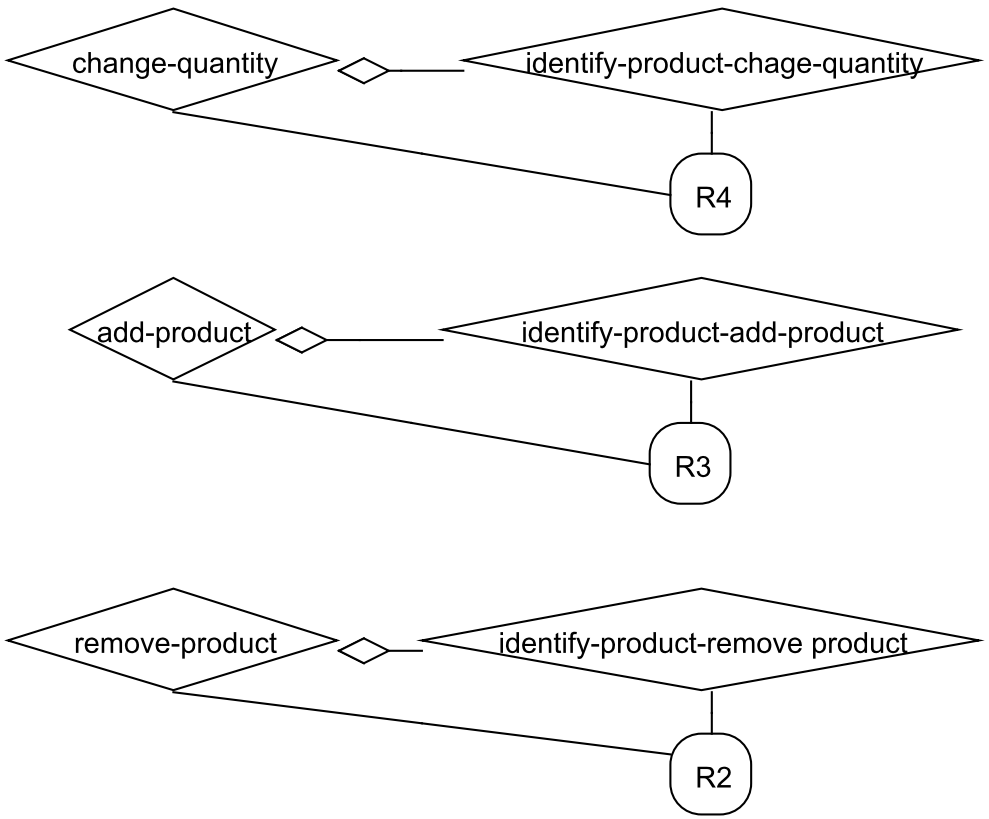
Theme Name	Use Case Name
record-quantity	Record Product Quantity
add-product	Add Product
change-quantity	Change Product Quantity
print-bill	Print Billed Item
aprove	Approve
bill	Bill Item
remove-item	Remove Billed Item
modify-price	Modify Product Price
identify-product	Identify Product
remove-product	Remove Product

Extension Points

- Extension points omitted since we don't deal with the actual flows in use cases
- Nevertheless, a Cockburn style descriptive reference to extension points in extending use cases can be provided
- Example: *Print Billed Item* is activated each time an item is added to a bill or removed from it
- Base themes (that do not appear as subthemes) in the initial process correspond to peer use cases—so no special effort needed

Subthemes (1)

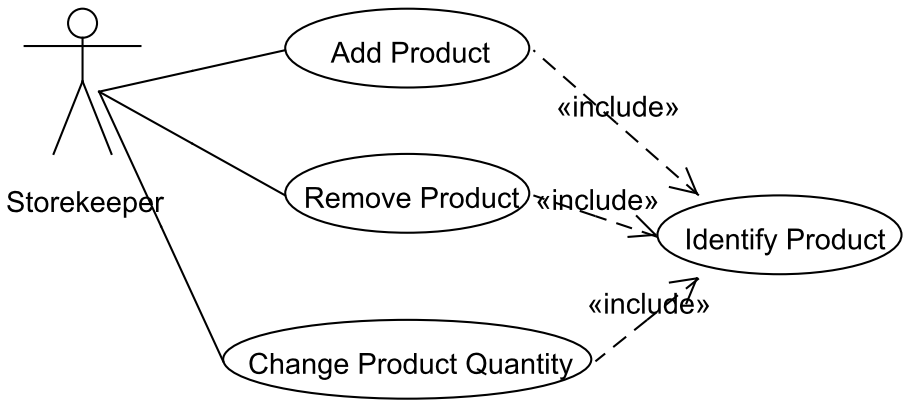
- Theme splitting may be needed



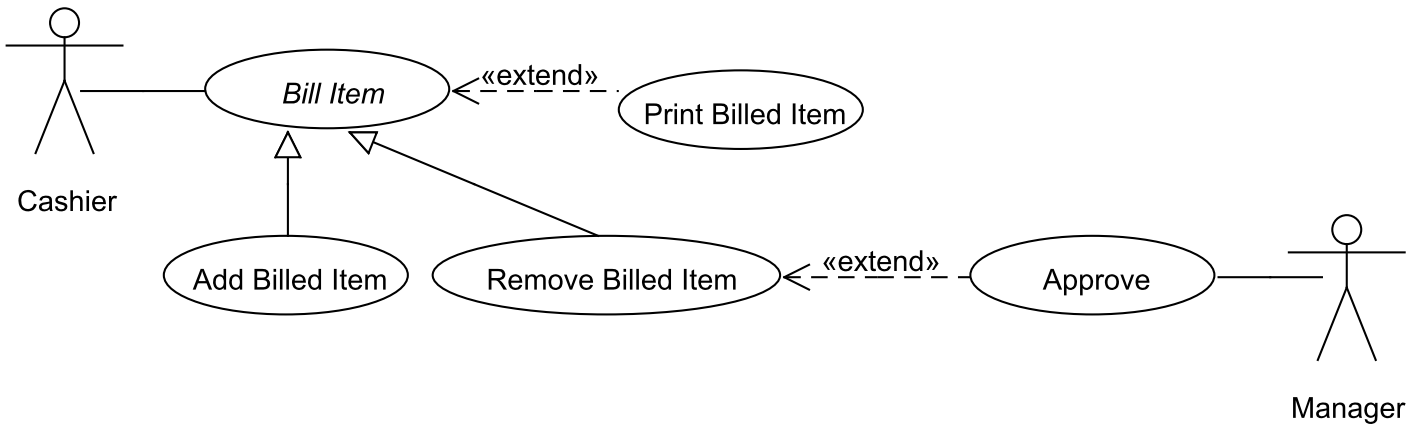
Subthemes (2)

- We could create a separate use case for each subtheme, but closer look at them reveals they represent the same functionality named differently merely due to Theme/Doc doesn't allow themes with equal names

Theme Unification into One Use Case

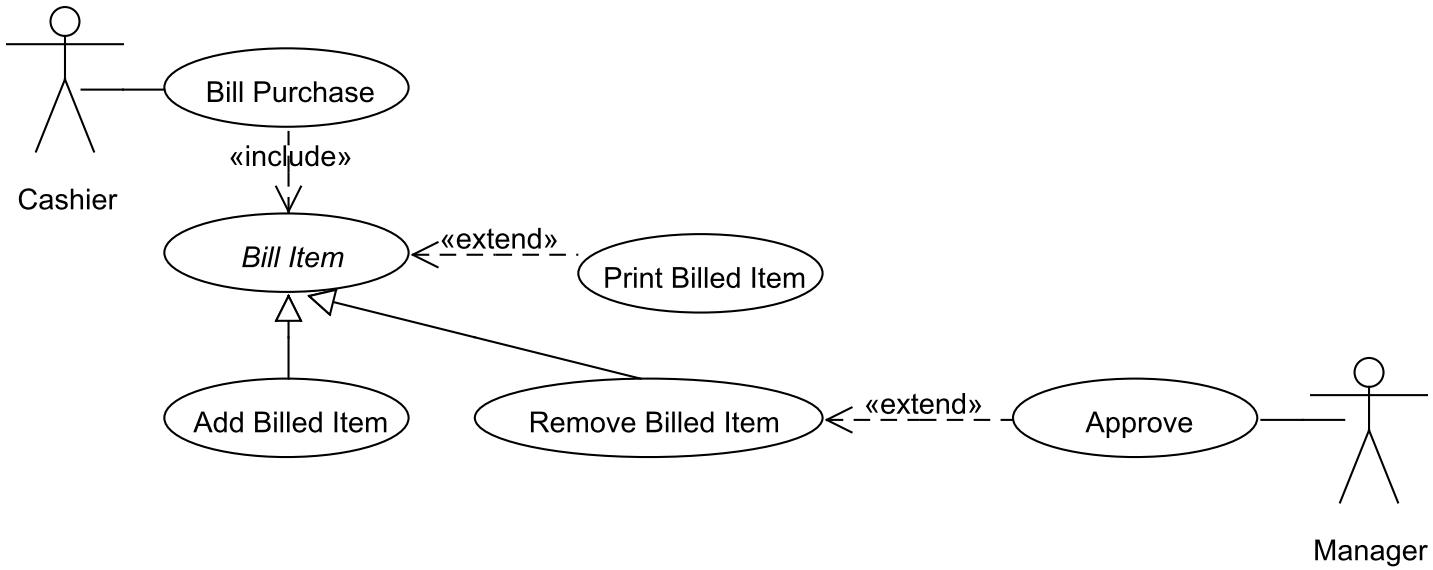


Theme Unification by Generalization



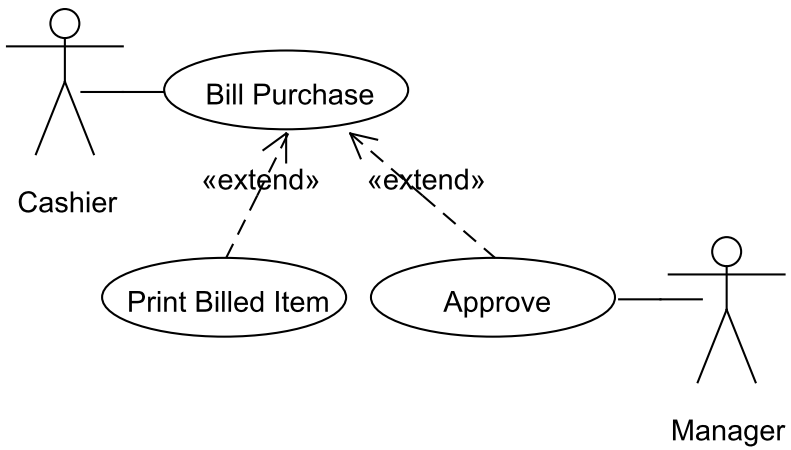
Granularity (1)

- Themes tend to be lower level than use cases—check the level
- Introducing an integral use case



Granularity (2)

- Keeping former use cases as inclusion-only use cases



Postponed Relationships

- Resolved as any kind of legal use case to use case relationship
- Can even be dismissed

Transforming Use Cases into Themes

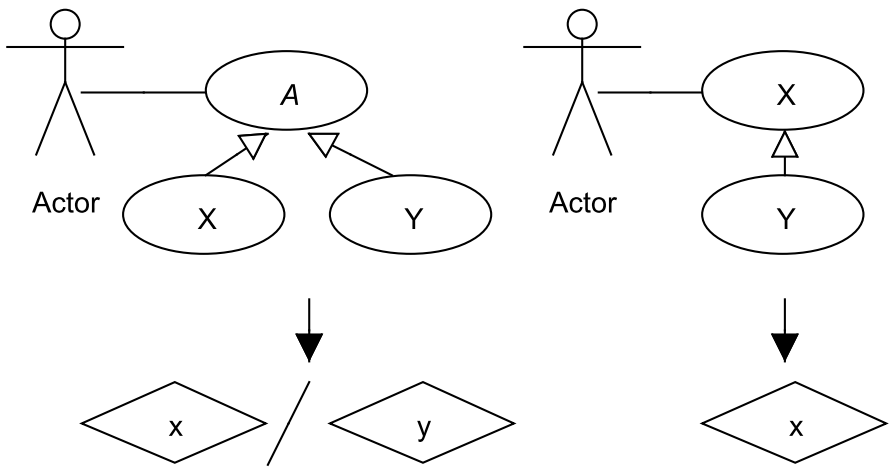
Transforming Use Cases into Themes (1)

1. Identify themes by transforming each use case not embraced in generalization into a theme and transforming each generalization among use cases into a unified theme. Optionally rename themes by shortening the corresponding use case names. Drop actors.
2. Create the crosscutting view by transforming each extend relationship between use cases into a crosscutting relationship between the corresponding themes preserving its direction.
3. Create the individual view by transforming each include relationship between use cases into a theme–subtheme relationship preserving its direction. Derive the data entities the theme operates on from the use case flows and attach them to the corresponding themes.

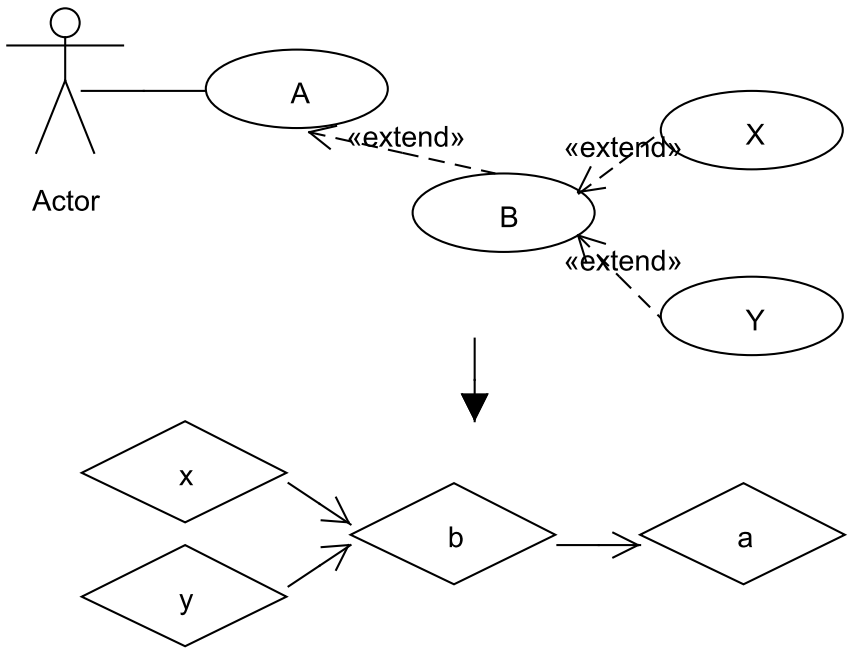
Transforming Use Cases into Themes (2)

4. Transform all requirements use cases refer to into requirements in the theme model. Transform each use case to requirement relationship into a relationship between the corresponding theme and requirement.
5. Derive the theme–relationship view by including all the themes in the crosscutting view and identifying shared requirements. Transform each unspecified dependency between use cases into a postponed relationship between the corresponding themes preserving its direction.

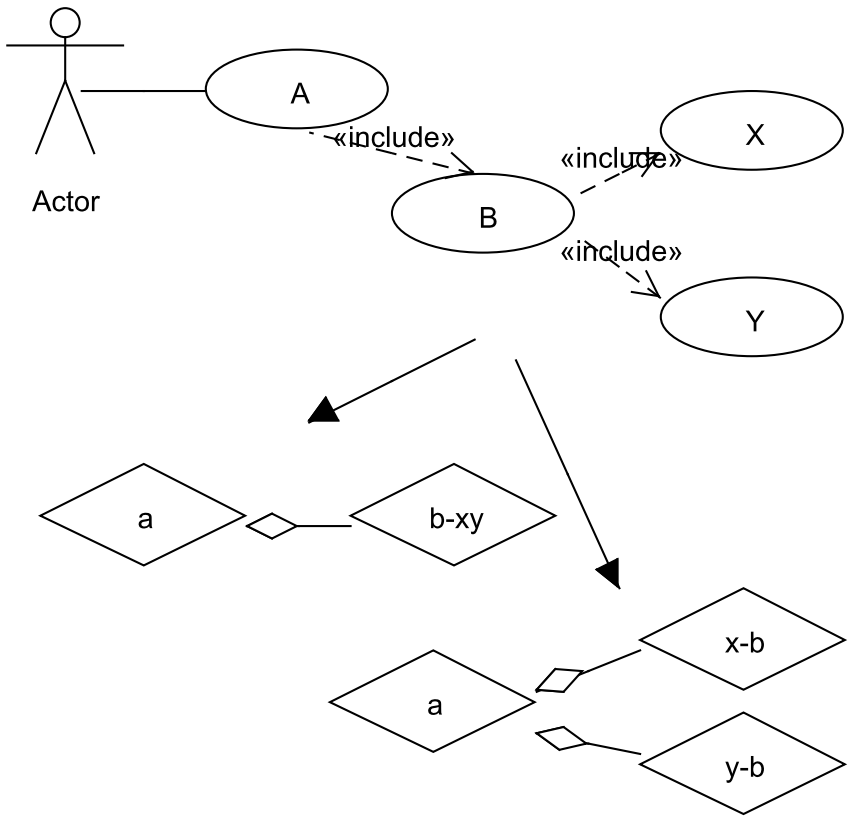
Generalizations



Transforming Extend Chaining



Transforming Include Chaining



Requirements

- Themes derived from use cases have no description How to solve this:
  1. Derive Theme/Doc style requirements from use cases and requirements attached to them
  2. Describe each theme according to the understanding of its meaning obtained during the transformation and attach this description as requirement attached to the corresponding theme
  3. Attach use cases as a kind of requirements derivative instead of requirements to the corresponding themes

Summary and Discussion

What the Transformation Revealed

- The way themes and use case express aspect-oriented decomposition
- Their relationship to functional decomposition and generalization
- Equivalence of mechanisms for aspect-oriented decomposition

Theme/Doc	Use Case Modeling
base themes	peer use cases
crosscutting relationship	extend relationship

- One of the main differences: themes are described indirectly by restructured requirements, while use cases are described by flows of events

When to Apply the Transformation

- Switching from themes to use cases as two ways of analysis
- But also to improve the process of aspect-oriented analysis and design based on the Theme:
  1. Create the Theme/Doc model from requirements (a part of it could be automated)
  2. Transform it into use cases
  3. Develop flows in use cases (to make easier creating the Theme/UML model)
  4. Develop the Theme/UML model
- The limitations of theme decomposition by which a subtheme can't have its own subthemes can be used as a kind of a test for functional decomposition of use cases by include relationship

Further Work

- Extend the transformation to embrace the Theme/UML model
- The excess information in use cases could provide a part of the information needed for modeling Theme/UML themes, and vice versa, the Theme/UML themes can provide flows to use cases
- Explore themes with respect to features in feature modeling