

Refactoring Use Case Models

A Case Study

Shengbing Ren, Kexing Rui, Greg Butler

Department of Computer Science, Concordia University
Montreal, Canada

Outline

- The Problem (*)
- Background on Refactoring (*)
- Requirements and Use Cases (*)
- The MetaModel
- The Case Study
- The Tool (In Progress)
- Conclusions and Future Work (*)

The Problem

Evolution of an existing software system

The Assumptions

System model of requirements is a *use case model*.

Part of the Solution

Refactor the use case model

The metamodel *defines* the important concepts and relationships of a use case model

Case studies validate the usefulness and coverage of refactorings.

Rest of Solution

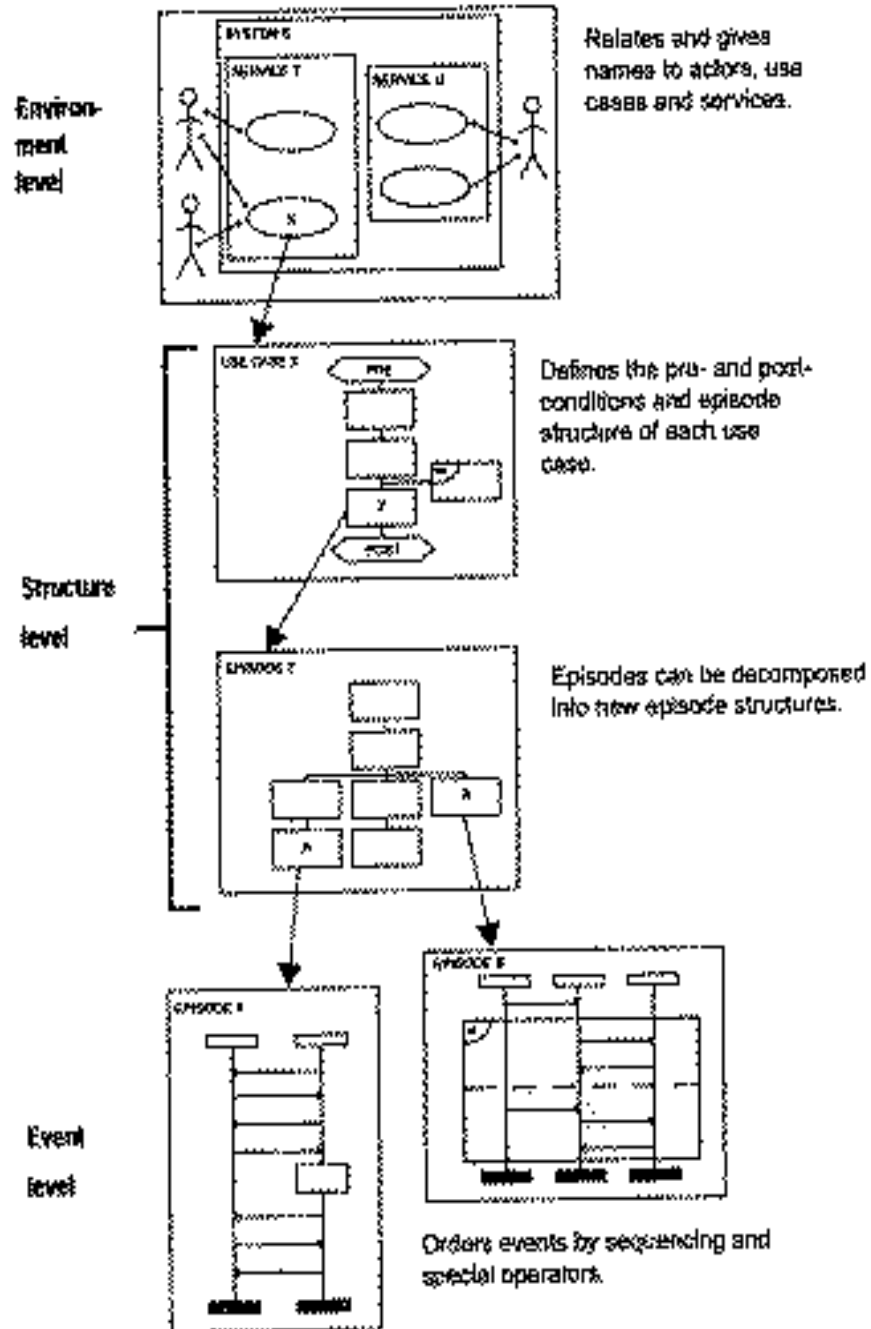
Evolution = Refactoring followed by Extension

Refactor each model in turn

Extend each model in turn

Tool support

Example Use Case Model



Background

Refactoring is a behavior-preserving program transformation that automatically updates an application's design and underlying source code.

Primitive refactorings perform simple edits such as adding new classes, creating instance variables, and moving instance variables up the class hierarchy.

Compositions of refactorings can create abstract classes, capture aggregation and components, and even install design patterns.

Evolution is *refactoring* followed by *extension*.

The new structure of the system and the refactorings to effect the re-structuring are chosen with the required extension in mind.

More on Refactoring

1992: Opdyke introduced the term and defined a set of behavior preserving transformations for object oriented applications.

1993–1997: Roberts, Johnson, Brant: Smalltalk Refactory Browser

1995: Tokuda and Batory refactoring to support design patterns; C++ tool

1997: Miller, McGregor, Major: *use case assortment* as re-structuring of use case model

1999: Fowler's book and XP increase visibility

1997–: several tools for refactoring (Java) source code

Requirements and Use Cases

There are many approaches to modeling requirements:

- task-oriented
- goal-oriented
- discourse modeling
- scenarios
- ... and use cases

Use case is a description of a cohesive set of dialogues that the primary actor initiates with a system.

use case encompasses a collection of *scenarios*

use case encompasses a collection of *episodes*

Various people have “enhanced” use cases

Cockburn (1997) — goals

Regnell (1999) — hierarchical model with 3 levels

...

The MetaModel Basic Questions for Refactoring

What notion of behaviour is appropriate for a model?

How does one record the intent of a refactoring?

How does one reason that the chosen set of refactorings achieves the desired intent?

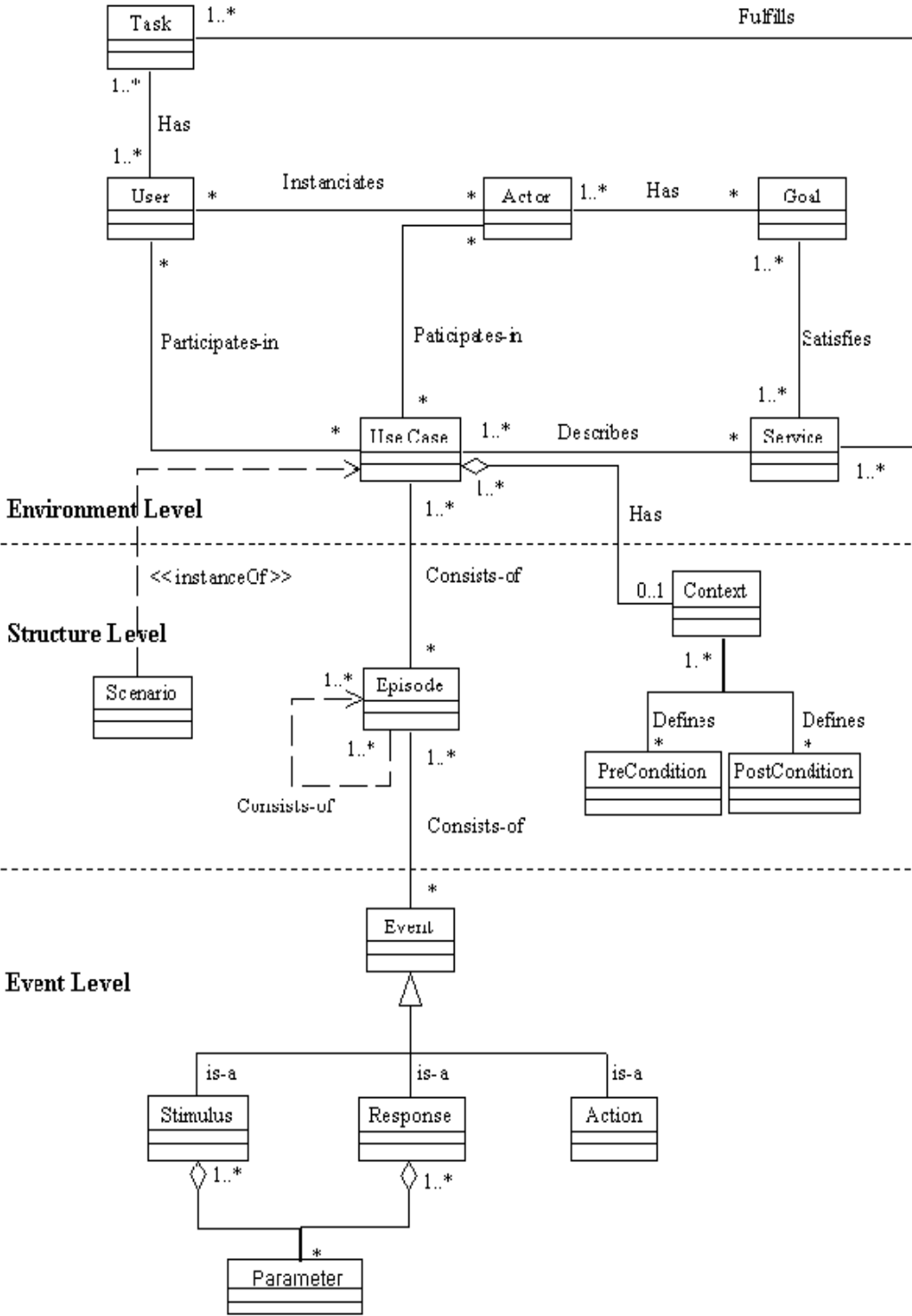
Refactoring needs a notion of “behaviour” for use cases

The functionality of the system is defined as the set of dialogues.

A refactoring of a use case model preserves the set of dialogues of the target system.

Metamodel defines “semantics” of use case model

The MetaModel



Cascaded Refactoring

Extend the notion of *refactoring*

- refactoring of feature models;
- refactoring of use case models;
- refactoring of architectures (preliminary); and
- clarify “behaviour” preserved by these refactorings.

Initial methodology for refactoring the set of models

- to view refactoring as an issue-driven activity;
- to document the rationale of an application of a refactoring as a triple:
 - intent of restructuring,
 - choice of refactoring(s), and
 - impact of the restructuring;
- the notion of cascaded refactoring, where the restructuring of one model determines constraints on the restructuring of other models (via the traceability and alignment maps).

Methodology — Refactoring Rationale

Each decision record has

- the *intent* of this step of restructuring, perhaps in the context of the overall intent;
- the *choice* of refactoring, or refactorings for a high-level refactoring that is composed of a series of lower-level refactorings; and
- the *arguments* for this choice, with a possible discussion of trade-off analysis for other candidate refactorings; and
- the *impact* or consequences of this step of restructuring.

Sample Refactorings of Use Case Model

create_abstract_actor identifies two actors a_1 and a_2 with a common super-actor a as their parent.

create_abstract_usecase identifies two use cases u_1 and u_2 as specializations of a common super-use case u .

merge_actors identifies two actors a_1 and a_2 as a common actor a .

merge_behaviours identifies two use cases u_1 and u_2 as a common use case u .

split_behaviour distributes a set of scenarios for a use case u across two new use cases u_1 and u_2 .

split_actor identifies the special cases a_1 and a_2 of an actor a .

make_episode_usecase takes a usecase u with an episode e and creates a new usecase u_1 with behaviour precisely that of the episode e . A relationship link u includes u_1 is added.

make_scenario_usecase takes a usecase u with a scenario s and creates a new usecase u_1 with behaviour precisely that of the scenario s . A relationship link u includes u_1 is added.

Conclusions and Future Work

Contributions of this Work:

Developed and justified a metamodel for use cases to act as semantics for reasoning about refactorings

Simple ATM case study to illustrate use case refactorings.

Contributions of Related Work:

Extend the notion of *refactoring* to use case model.

View refactoring as *issue-driven* activity

How to document a refactoring decision

Cascading of the refactoring decisions across models

Future Work

More extensive catalogue of use case refactorings

Document intent and correctness of refactorings using the metamodel.

Small and large case studies.

Tool support.