Bridging the Gap Between Requirements and Design with Use Case Maps

Daniel Amyot, Ph.D.
Daniel.Amyot@mitel.com
http://www.UseCaseMaps.org

Carleton University
November 15, 2001

Requirements Engineering Issues

- Early focus on low-level abstractions
- Requirements and high-level decisions buried in the details
- Evolution of functionalities difficult to handle (feature interactions, V&V, adaptability to legacy architectures...)
- Delay introduction of new services
Software Engineering Issues

- Requirements/analysis models need to support new types of dynamic systems
  - Run-time modification of system structure
  - Run-time modification of behaviour
- Need to go from a requirements/analysis model to design models in a seamless way
- We propose Use Case Maps (UCMs)!

Use Case Maps (UCMs)

- The Use Case Maps notation allows illustrating a scenario path relative to optional components involved in the scenario (gray box view of system)
- UCMs are a scenario-based software engineering technique for describing causal relationships between responsibilities of one or more use cases
- UCMs show related use cases in a map-like diagram
**UCM Notation - Basic**

**UCM Example: Commuting**

- **Basic Path** (from circle to bar)
- **Responsibility Point**
- **Component** (generic)

- **home**
  - secure home
- **transport**
  - commute
- **elevator**
  - take elevator

**Why Use Case Maps?**

- **Bridge** the **modeling gap** between requirements (use cases) and design
  - Link behaviour and structure in an explicit and visual way
  - Provide a behavioural framework for making (evaluating) architectural decisions at a high level of design
  - Characterize the behaviour at the architecture level once the architecture is decided
- Convey a lot of information in a compact form
- Use case maps **integrate many scenarios** - enables reasoning about potential undesirable interactions of scenarios
Why Use Case Maps?

- Provide ability to **model dynamic systems** where scenarios and structures may change at run-time
  - E-commerce applications
  - Telecommunication systems based on agents
- Simple, intuitive, low learning curve
- Document while you design
- Effective learning tool for people unfamiliar with the domain
- May be transformed (e.g. into MSC/sequence diagrams, performance models, test cases)

---

The Development Pyramid

- **Requirements**
  - NFR
  - Use cases
  - Problem modeling

- **Analysis/High-level Design**
  - Use Case Maps
  - Sequence/collaboration diagrams, statechart diagrams, class/object diagrams, component/deployment diagrams (UML);
    message sequence charts, SDL (ITU-T)

- **Detailed design**

- **Implementation**
  - Code
**UCM Notation - Hierarchy**

**UCM Example: Commuting**

- **Dynamic Stub** (selection policy)
- **Static Stub**

- **Transport**
  - Drive car

**UCM Notation - Simple Plug-in**

**UCM Example: Commute - Car (Plug-in)**

- **Transport**
  - Drive car

© 2001 Bridging the Gap Between Requirements and Design with Use Case Maps
UCM Notation - AND/OR

UCM Example: Commute - Bus (Plug-in)

- Person
- Transport
- Take 95
- Take 97
- Take 182

AND Fork OR Fork OR Join AND Join

UCM Notation - Dynamic Structures

Generic UCM Example

- Start
- Create
- Move in
- Move out
- Copy
- Destroy
- Slot (component)
- Pool (component)

Dynamic Responsibilities and Dynamic Components
The elevator control system case study is adapted from Hassan Gomaa's Designing Concurrent, Distributed, and Real-Time Applications with UML (p459-462), copyright Hassan Gomaa 2001, published by Addison Wesley. Used with permission.
Generic Problem with Scenarios

- Given a set of scenarios capturing informal (functional) requirements
- Specify (formally) the integration of scenarios
  - Undesirable emergent behaviour may result...
- Validate, i.e. look for logical errors and check against informal requirements
- Numerous tools and techniques can be applied (e.g. functional testing)
**UCM Validation by Inspection**

- Several problems detectable by inspection
  - Non-determinism in selection policies and OR-forks
  - Erroneous UCMs
  - Ambiguous UCMs, lack of comments
- Many **feature interactions** (FI) solved while integrating feature scenarios together
- Remaining undesirable FI need to be detected!
  - Many are located in stubs and selection policies
  - Need more formal analysis

**Feature Interaction**

- Conflict between candidate plug-ins for the same stub (preconditions of plug-ins are the same)
  - Call waiting (CW) vs. automatic re-call (ARC)
Feature Interaction

- Unexpected behaviour among different selected plug-ins for different stubs (postconditions of plug-ins are not the same)
  - Originating call screening (OCS) denies call whereas call forward (CF) redirects call to screened number

![Feature Interaction Diagram]

Analysis Model Construction

- Source scenario model ⇒ Target analysis model
- Q1. What should the target language be?
  - Use Case Maps Specification ⇒ ?
- Q2. What should the construction strategy be?
  - Analytic approach
    * build-and-test construction
  - Synthetic approach
    * scenarios "compiled" into new target model
    * interactive or automated
Specification-Validation Approach with LOTOS and UCMs

Complementary Yet Compatible!

<table>
<thead>
<tr>
<th>Use Case Maps</th>
<th>LOTOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario notation, readable, abstract, scalable, loose, relatively effortless to learn</td>
<td>Mature formal language, good theories and tools for V&amp;V and completeness &amp; consistency checking.</td>
</tr>
</tbody>
</table>

Both

- Focus on ordering of actions
- Have similar constructs → simpler mapping
- Handle specifications with or without components
- Have been used to describe dynamic systems in the past
- Have been used to detect feature interactions in the past
UCM-LOTOS Construction Guidelines

UCM-LOTOS Testing Framework
**Scenario Definitions**

- Enhances the behavioural modeling capability of UCM paths and path elements
- Requires a **path data model** (for conditions at various points along the path)
  - Currently, global and modifiable Boolean variables
    - Values may be assigned to variables along a path
  - In future, …
    - Variables may possibly have different types
    - Variables may be scoped to paths or components
    - Scenarios may be structured into sub-scenarios
Scenario Definitions

- Requires a more formal definition of some notational elements
  - Currently, logical expressions with global variables
  - Currently, OR forks, selection policies, start points, waiting places, & timers covered (in future: loops)
- Scenario definitions consist of …
  - Name of scenario (scenarios may be grouped for convenience)
  - Set of concurrent start points
  - Set of initial values assigned to global variables
UCM Path Traversal

- Starts at one or more parallel start points as defined by user
- Starts with initial values (true, false, or undetermined) for each path data variable as defined by the user
- Moves from path element A to path element B if continuation criteria are met for element A
  - Each UCM path element has specific criteria
- Issues a warning if path traversal is stuck

UCM Path Traversal - Example I

```
A,1 2,4
A,1 2,4,5
A,1 2,P
A,1,P 7,R
R
```

© 2001 Bridging the Gap Between Requirements and Design with Use Case Maps
**UCM Path Traversal - Example II**

![Diagram](image_url)

**Applications of UCM Path Traversal**

- **Highlighting**
- **Animation**
  - Requires sequence numbers
- **MSC generation**
  - Requires component information
  - Well-nestedness transformation and warning mechanism
- **LQN generation**
  - Requires arrival and device characteristics, device demands, data access modes, response-time requirements
- **Test case generation**
  - Requires controllable and observable activities
Example

**User**

- *down*
  - at floor
  - select elevator
  - already on list
  - add to list

**Elevator Control System**

- *moving*
  - motor up
  - motor down
  - motor stop
  - door open
  - door close
  - remove from list

**Service Personnel**

- switch on
- stationary
- memory
- in elevator

**Arrival Sensor**

- switch on
- approaching floor

**Example**

**User**

- *moving*
  - door close
  - above
  - decision
  - [on list]

**Elevator Control System**

- *moving*
  - motor up
  - motor down
  - door open
  - door close

**Service Personnel**

- switch on
- stationary
- memory
- in elevator

**Arrival Sensor**

- switch on
- approaching floor
Selected Contributions


- Cameron, D. et al. (2001) Draft Specification of the User Requirements Notation. Cardiff, UK
