Bridging the Requirements / Design Gap in Dynamic Systems with Use Case Maps

Daniel Amyot, Gunter Mussbacher
Strategic Technology, Mitel Networks

Objectives

- Introduce Use Case Maps (UCMs) Concepts and the UCM Notation
- Understand the relationship of UCMs with use cases, test cases, high level architecture, more detailed behavioral diagrams, and more detailed structural diagrams
- Give an overview on validation of UCMs (feature interaction), performance analysis based on UCMs, and UCM styles
- Compare and contrast Use Case Maps with other modeling techniques which focus on the same general problem area
- Understand the benefits of UCMs and how UCMs fit into use-case driven software development processes

Table of Contents

- **Introduction to Use Case Maps (UCMs) and the UCM Notation**
- Dynamic Aspects of the UCM Notation
- Enhancing UCMs with Formal Scenario Definitions
- Derivation of Structural Specifications from UCMs
- Validation of Use Case Map Specifications
- Generation of Message Sequence Charts From UCMs
- Generation of Performance Models from UCMs
- Tool Demonstration: UCMNav - The UCM Navigator
- Use Case Maps Exercise
- Use Case Map Styles for Small and Large Systems
- UCM Puzzles

Use Case Maps (UCMs)

- UCMs stands for **Use Case Maps** – a graphical notation that allows illustrating a scenario path relative to the 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
- A map shows the progression of scenarios along use cases

Use Case Maps (UCMs)

- The intent of UCMs is to facilitate reusability of scenarios across a wide range of architectures and to guide the design of high level architecture
- UCMs have a history of application to the description of object-oriented systems and reactive systems in various domains
  - Telecommunication, wireless, airline reservation, elevators, railway, agents, network management applications, web applications, graphical user interfaces, drawing packages, multimedia applications, banking applications, object-oriented frameworks, "work patterns" of software engineers, etc.

History of Use Case Maps

- End of 1980's at Carleton University, Canada
  - Buhr (design) and Woodside (performance)
- Slices: Vigder and Buhr, 1992
- Timethreads: Buhr and Casselman, 1993
- Use Case Maps: Buhr and Casselman, 1995
- Still evolving…
  - Scenario definition
  - Model generation (e.g. UCM to MSC)
  - Standardization
  - UCM patterns and styles
  - …

Use Case Maps Web Page

- **http://www.UseCaseMaps.org/**
- Prime source of information about UCMs
- Supports the UCM User Group
  - Nearly 200 members. Mailing list and newsletter.
- UCM Virtual Library
  - Over 50 publications and 25 presentations
- Tools (**UCM Navigator** and others)
- XML Document Type Definition (DTD)
- UCMs and UML

Why Use Case Maps?

- **Bridge** the **modeling gap** between requirements (use cases) and design
  - Link behavior and structure in an explicit and visual way
  - Provide a behavioral framework for making (evaluating) architectural decisions at a high level of design
  - Characterize the behavior 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

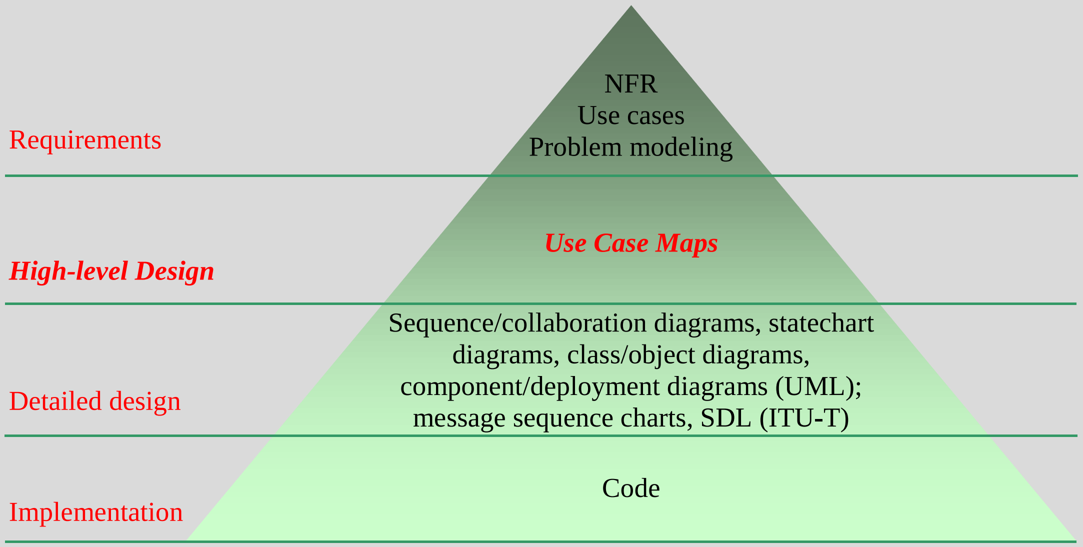
Why Use Case Maps?

- 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, very low learning curve
- Excellent documentation tool – document while you design
- Effective learning tool for people unfamiliar with domain

Information Needed to Construct Use Case Maps

- Informal requirements or use cases or extensive domain knowledge
- Responsibilities either stated or inferred from requirements/use cases/domain knowledge
- Clearly defined interface between the environment and the system (leads to start and end points of paths)
- Architectural components (optional)
  - High level description of these components (their nature, the relationships between them)
  - Vision of architectural structure

The Design Pyramid



Requirements

NFR
Use cases
Problem modeling

High-level Design

Use Case Maps

Detailed design

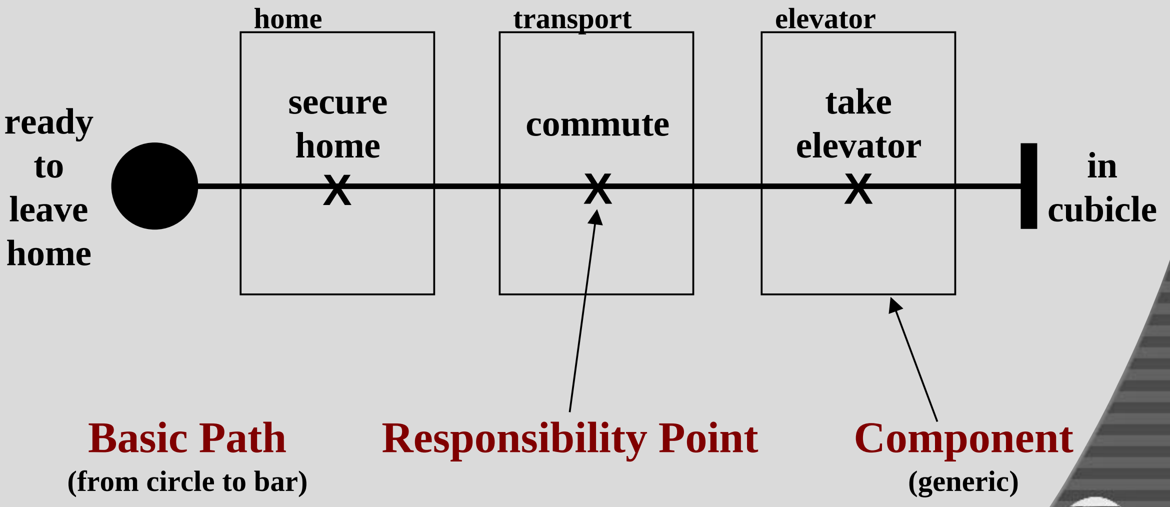
Sequence/collaboration diagrams, statechart diagrams, class/object diagrams, component/deployment diagrams (UML); message sequence charts, SDL (ITU-T)

Implementation

Code

UCM Notation - Basic

UCM Example: Commuting



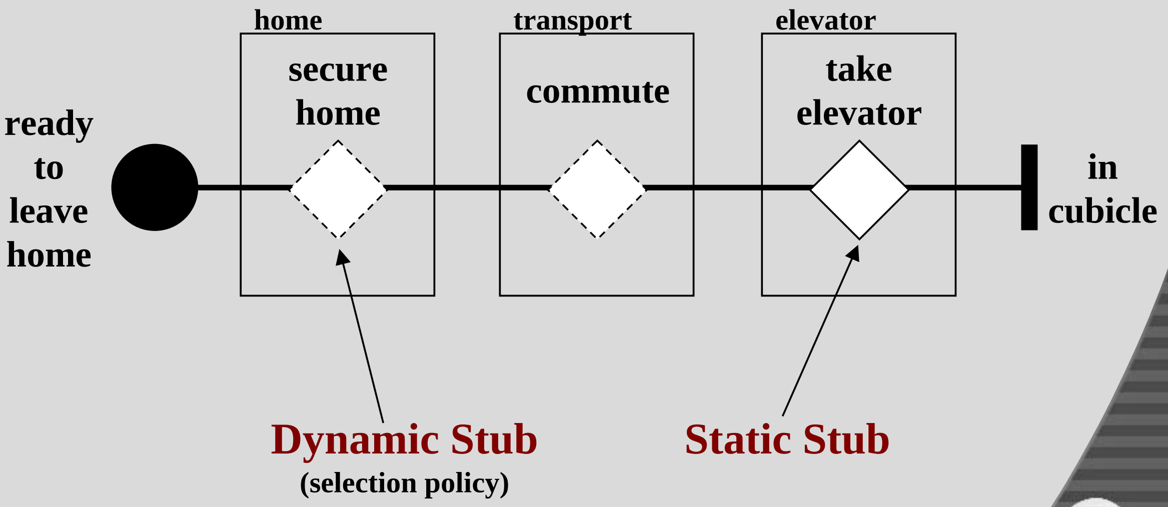
Basic Path
(from circle to bar)

Responsibility Point

Component
(generic)

UCM Notation - Hierarchy

UCM Example: Commuting



ready to leave home

home

secure home

transport

commute

elevator

take elevator

in cubicle

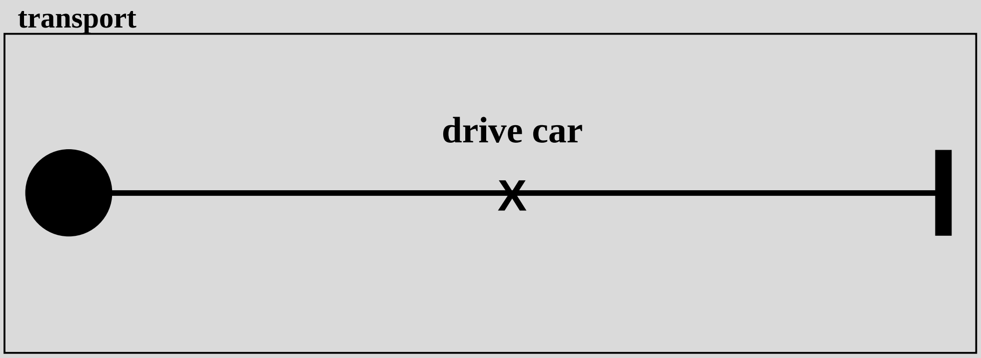
Dynamic Stub

(selection policy)

Static Stub

UCM Notation - Simple Plug-in

UCM Example: Commute - Car (Plug-in)



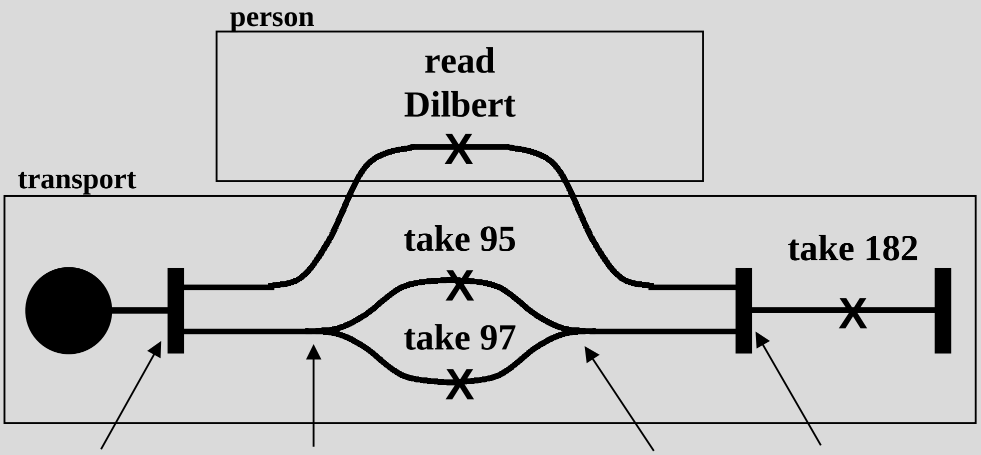
transport

drive car

X

UCM Notation - AND/OR

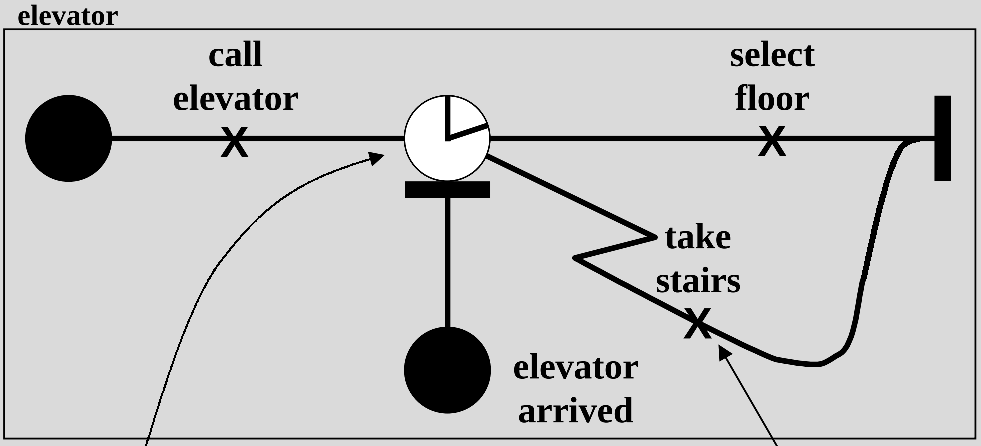
UCM Example: Commute - Bus (Plug-in)



AND Fork **OR Fork** **OR Join** **AND Join**

UCM Notation - Waiting Place / Timer

UCM Example: Take Elevator - Default (Plug-in)



Timer

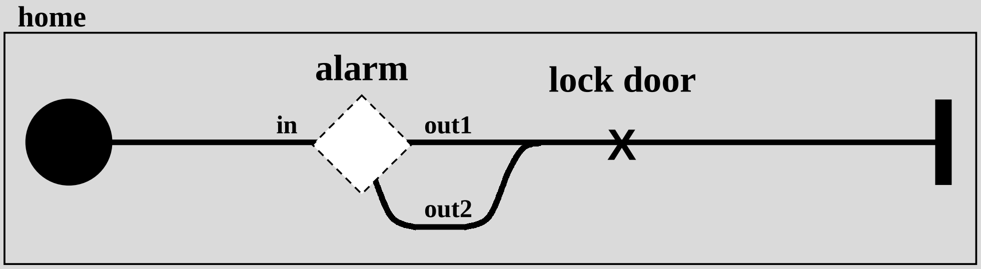
(special waiting place)

Timeout Path

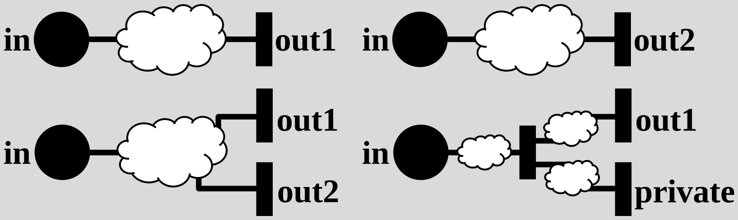
Note: Waiting places may be regular, memory, signal, or delay.

UCM Notation - Simple Plug-in with Stub

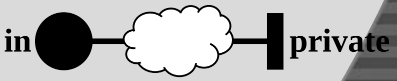
UCM Example: Secure Home - Default (Plug-in)



Possible plug-ins for a stub with one in path and two out paths:

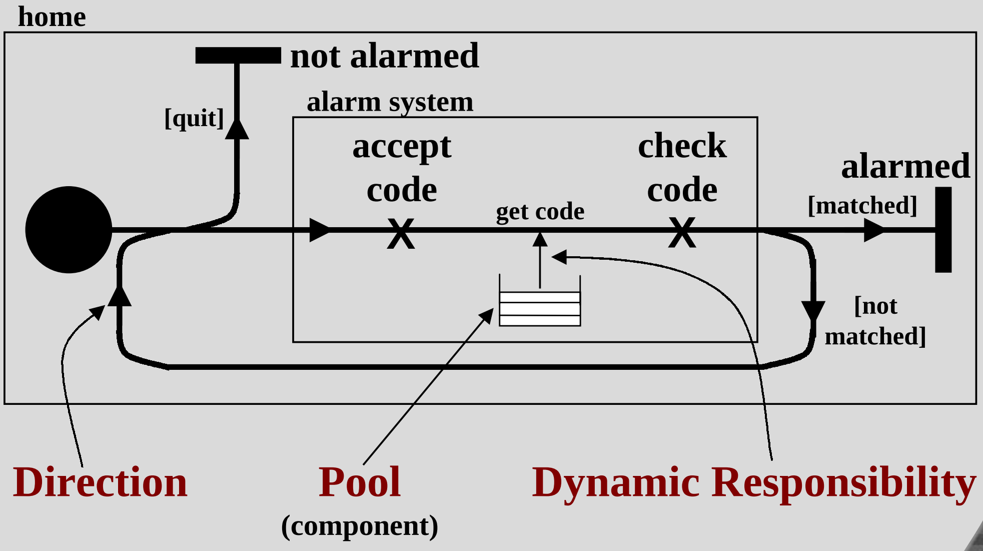


Possible but counterintuitive (avoid use):



UCM Notation - Component - Pool

UCM Example: Alarm - Installed (Plug-in)



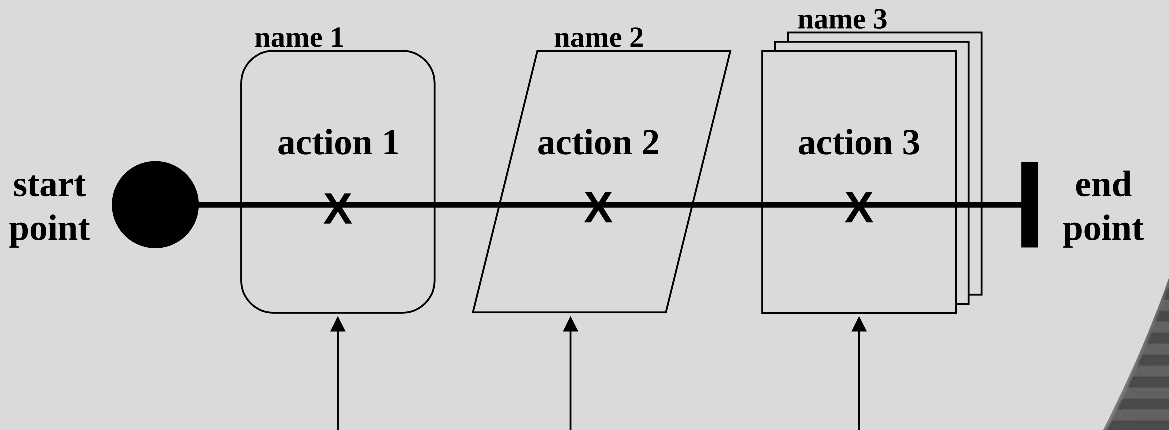
Direction

Pool
(component)

Dynamic Responsibility

UCM Notation - Component - Object, Process & Stack

Generic UCM Example



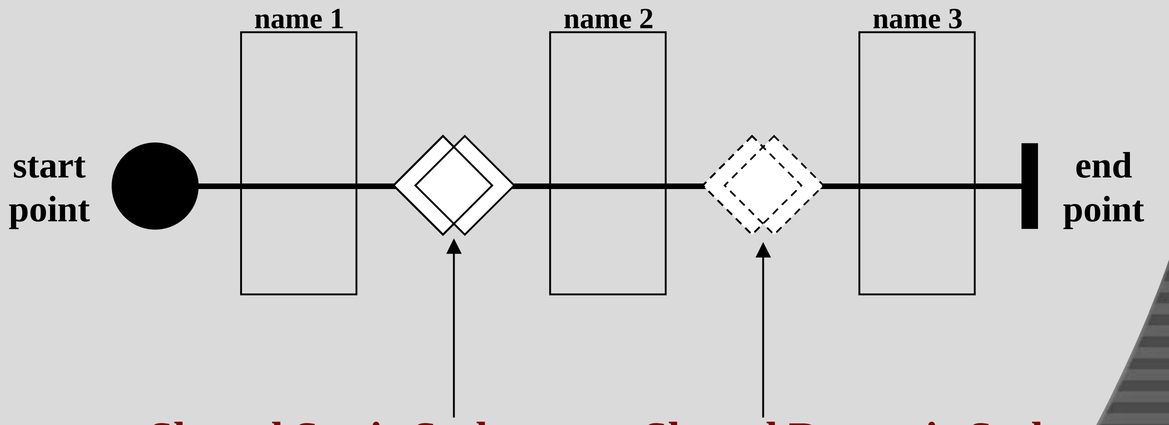
Object
(passive)

Process
(active)

Component Stack
(for any kind of component)

UCM Notation - Shared Stub

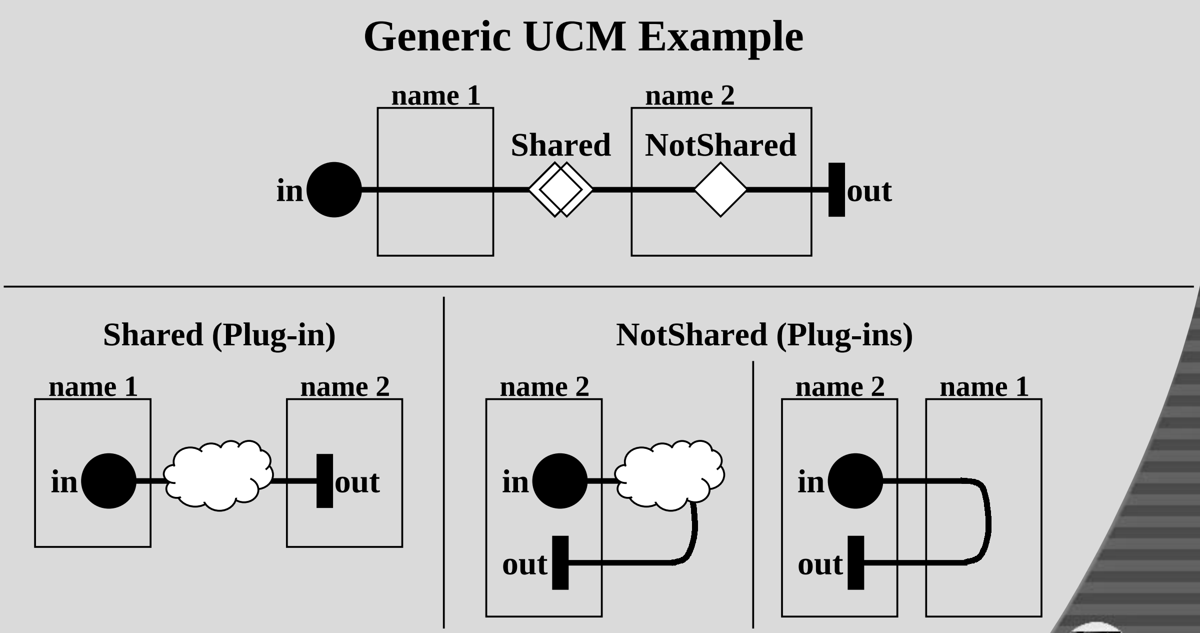
Generic UCM Example



Shared Static Stub

Shared Dynamic Stub

UCM Notation - Shared Stub



Key Points - UCM Introduction & Notation

- Modeling behavioral aspects of a system
  - UCM path including all path elements
  - More abstract than message exchanges (causal)
- Modeling structural aspects of a system
  - UCM components
    - Quite similar to the concept of roles in UML
    - Represent a slice of an architectural entity as required in a specific scenario
- Allocating behavior to structure
  - Place responsibility in component

Key Points - UCM Introduction & Notation

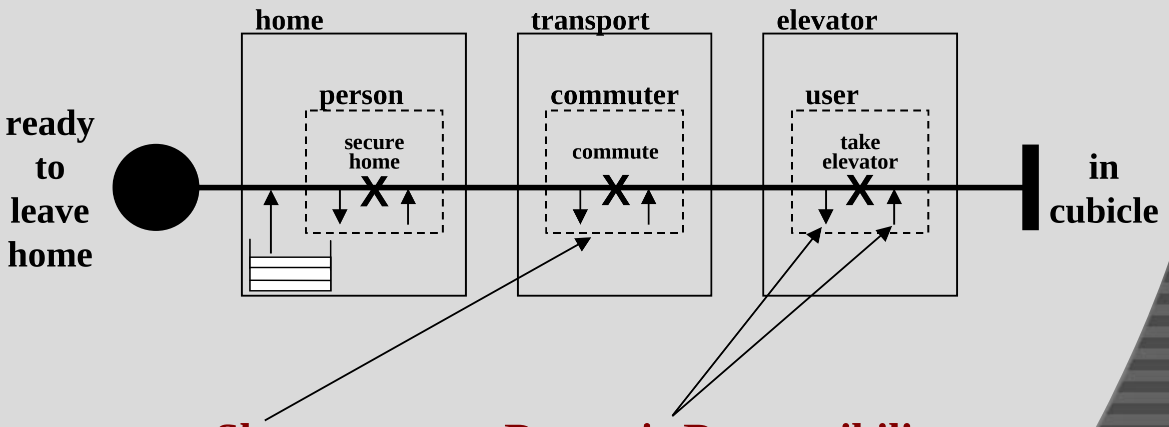
- UCMs (Use Case Maps) provide an integrated view of behavior and structure and bridge the conceptual gap between requirements and design
- UCMs are intuitive and easy to learn
- UCMs provide a gray-box view of the system
- The basic elements of the UCM notation are paths, start points, end points, responsibilities, static and dynamic stubs, AND/OR forks/joins, waiting places & timers, and components with dynamic responsibilities
- UCMs may provide reusable behavioral patterns even when the architecture evolves

---

Table of Contents

UCM Notation - Dynamic Aspects - Slot

UCM Example: Commuting

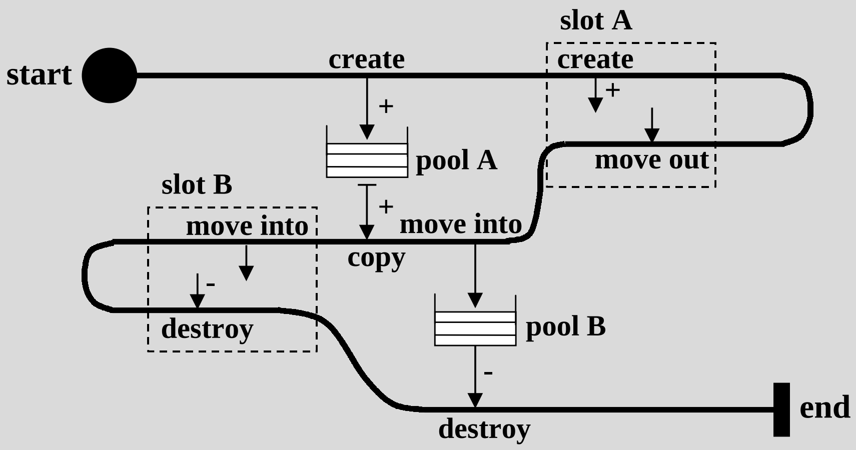


Slot
(component)

Dynamic Responsibility

UCM Notation - Dynamic Aspects - Dynamic Responsibilities

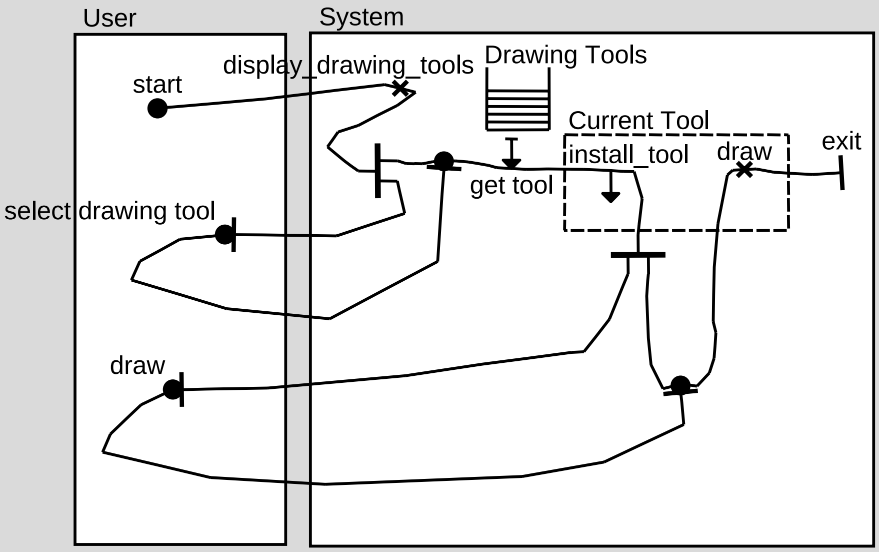
Generic UCM Example



Dynamic Responsibilities

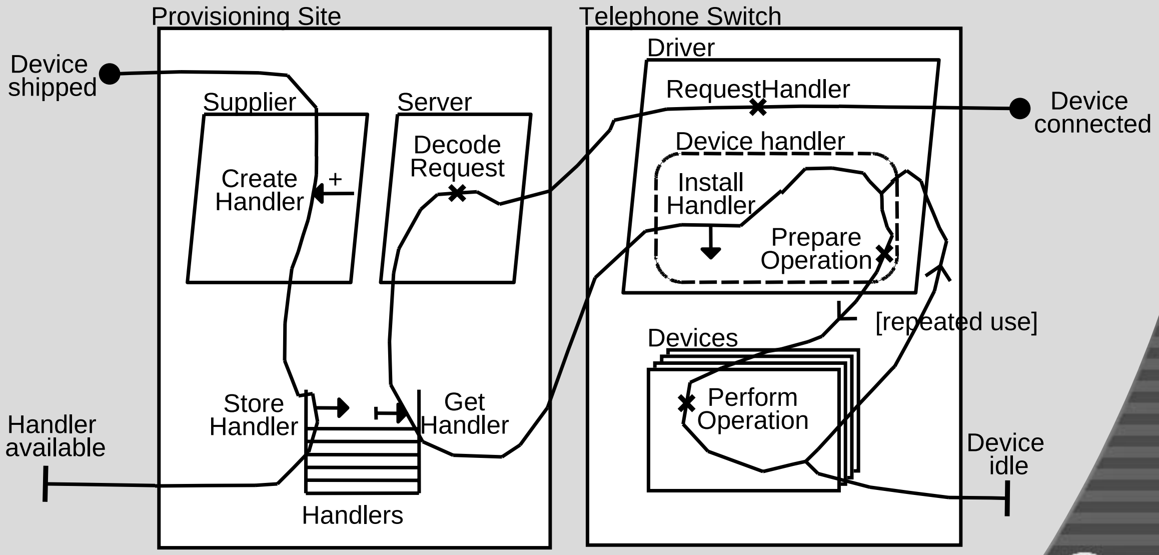
UCM Notation - Dynamic Aspects - Example 1

UCM Example: Drawing Tool



UCM Notation - Dynamic Aspects - Example 2

UCM Example: Plug'n'Play



Key Points - UCM Notation - Dynamic Aspects

- Modeling dynamic aspects of a system
  - Timers represent a basic dynamic element
  - Dynamic stubs decide on dynamic variations in behavior & structure with selection policies
  - Dynamic responsibilities create and delete dynamic components and manage their movement along paths
  - Slots are placeholders for dynamic components in execution
  - Pools are containers for dynamic components

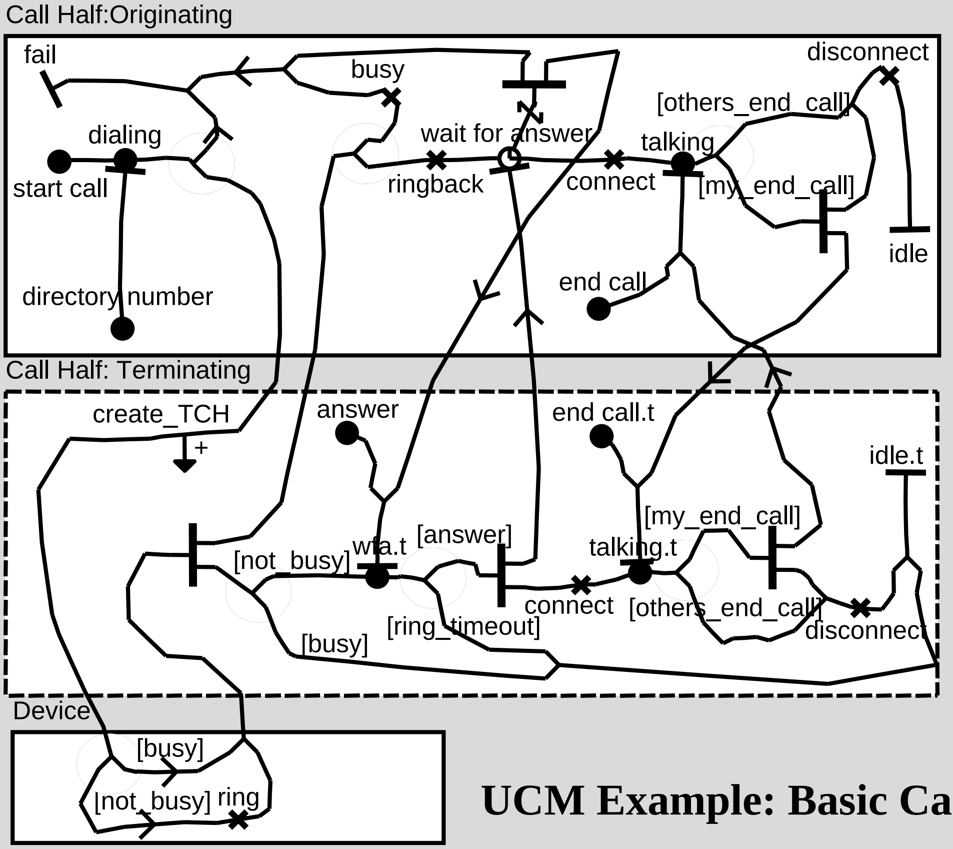
Table of Contents

Scenario Definitions

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

Scenario Definitions

- Requires a more formal definition of some notational elements
  - Currently, logical expressions with global variables define OR forks & selection policies
  - In future, timers & waiting places may be covered
- Scenario definitions consist of …
  - Name of scenario (scenarios may be grouped for convenience)
  - Start point
  - Set of initial values assigned to global variables



Number Correct

Busy

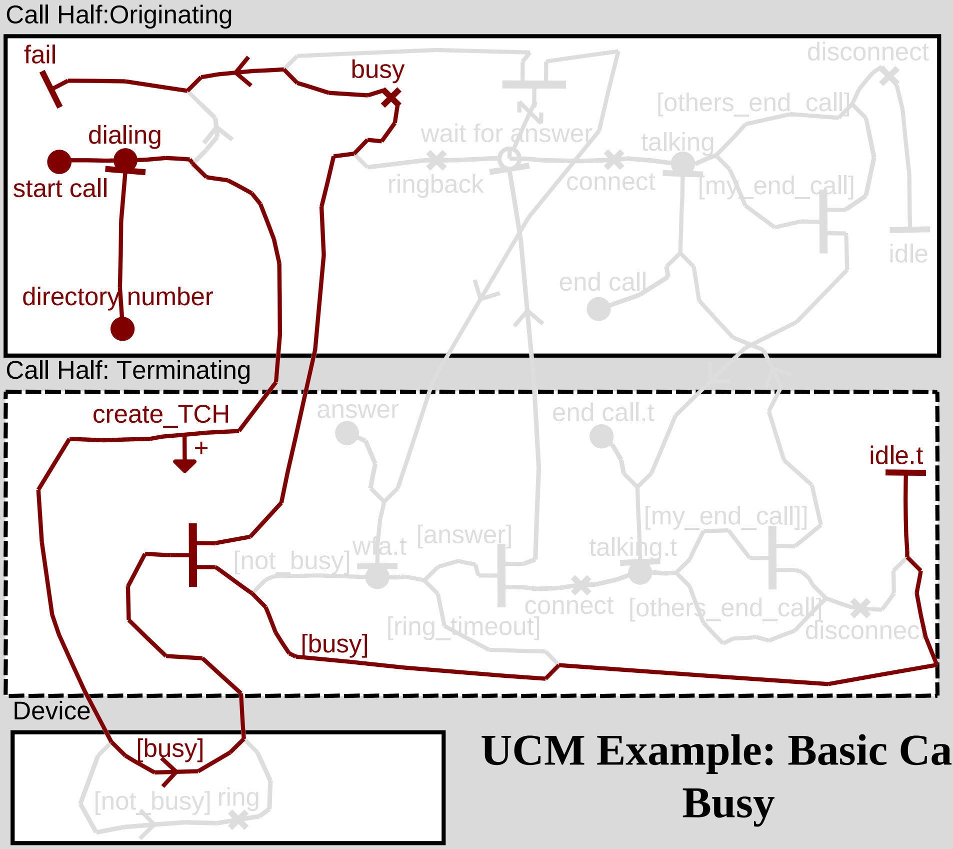
Answer

Ring Timeout

EndCall

EndCall.t

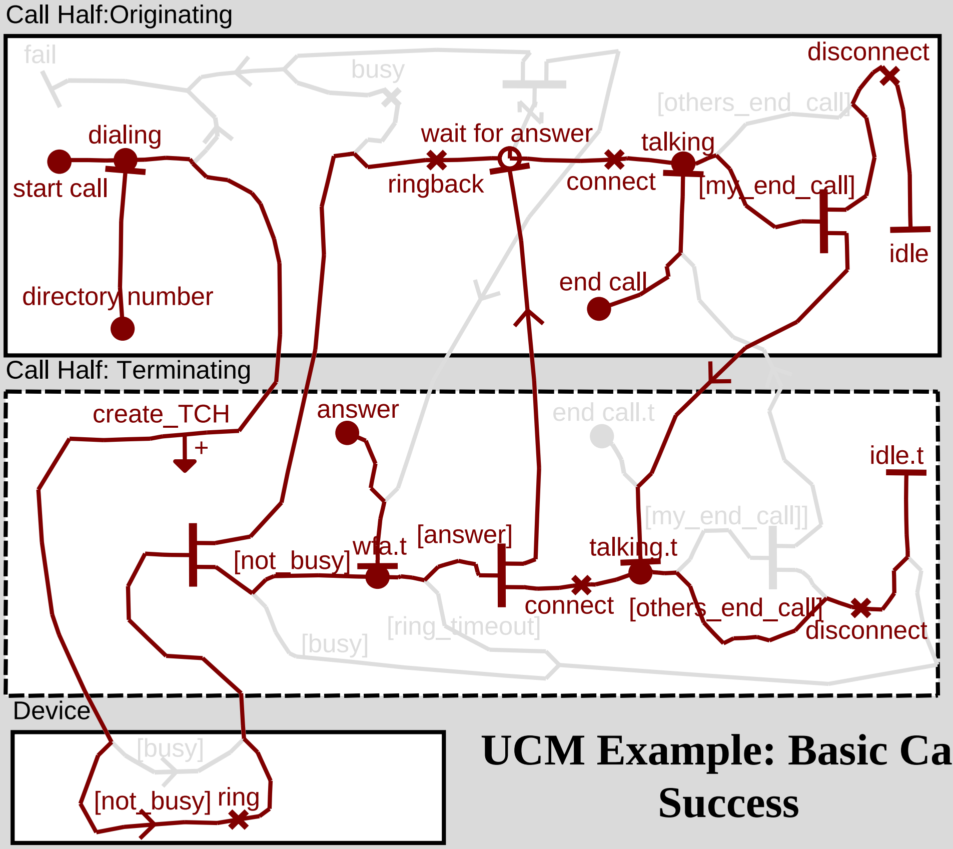
UCM Example: Basic Call



Number Correct = **Y**

Busy = **Y**

UCM Example: Basic Call Busy



Number Correct = **Y**

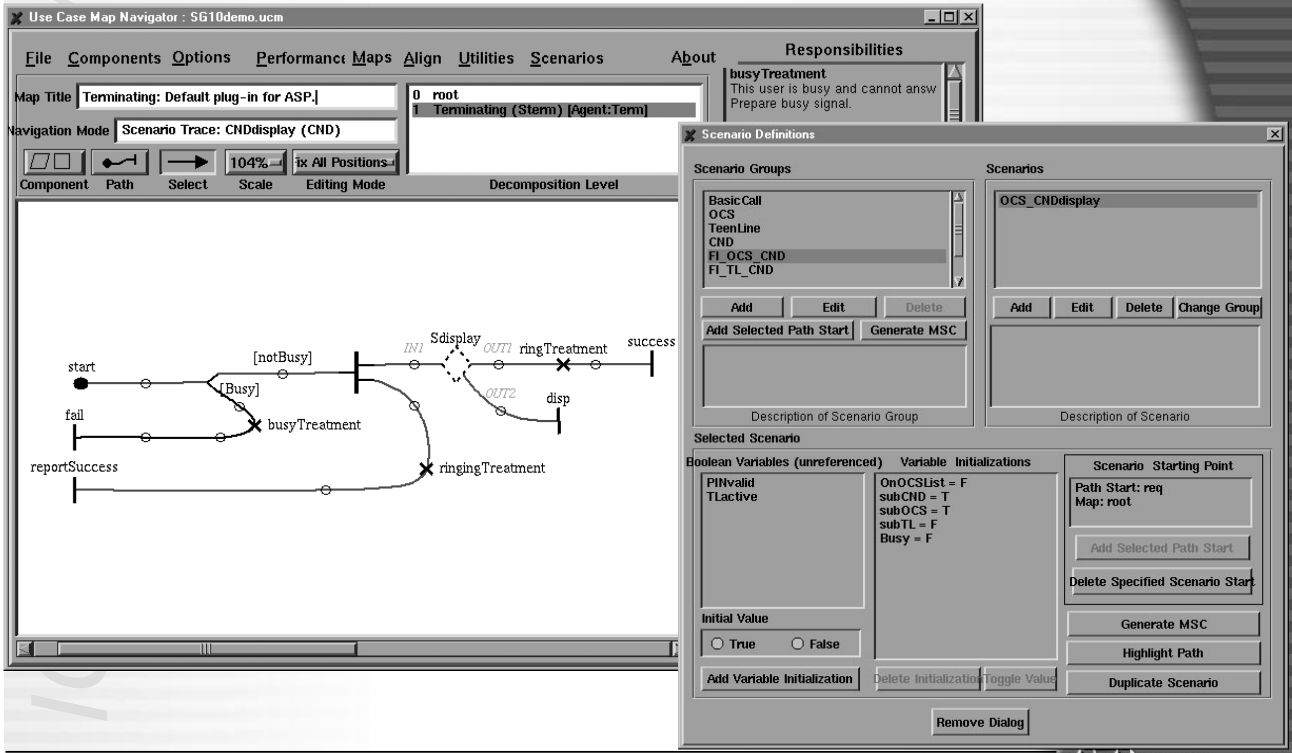
Busy = **N**

Answer = **Y**

EndCall = **Y**

UCM Example: Basic Call Success

Scenario Highlight (UCMNAV 2)



Map Title: Terminating: Default plug-in for ASP.

Navigation Mode: Scenario Trace: CNDdisplay (CND)

0 root
1 Terminating (Stem) [Agent:Term]

Responsibilities

busyTreatment
This user is busy and cannot answ
Prepare busy signal.

Scenario Groups

BasicCall
OCS
TeenLine
CND
FI_OCS_CND
FI_TL_CND

Scenarios

OCS_CNDdisplay

Selected Scenario

Boolean Variables (unreferenced)

PINvalid
TLactive

Initial Value: ○ True ○ False

Variable Initializations

OnOCSList = F
subCND = T
subOCS = T
subTL = F
Busy = F

Scenario Starting Point

Path Start: req
Map: root

Key Points - Scenario Definitions

- Path data model is not a problem domain data model
- Improves understanding of scenarios
- Scenario definitions are the foundation for more advanced functionality such as …
  - Highlighting of a scenario … available
  - Animation of a scenario … just a small step
  - Generation of MSCs … available
  - Generation of test cases … just a small step
  - Detection of feature interactions … early results
  - Execution of UCMs … future research

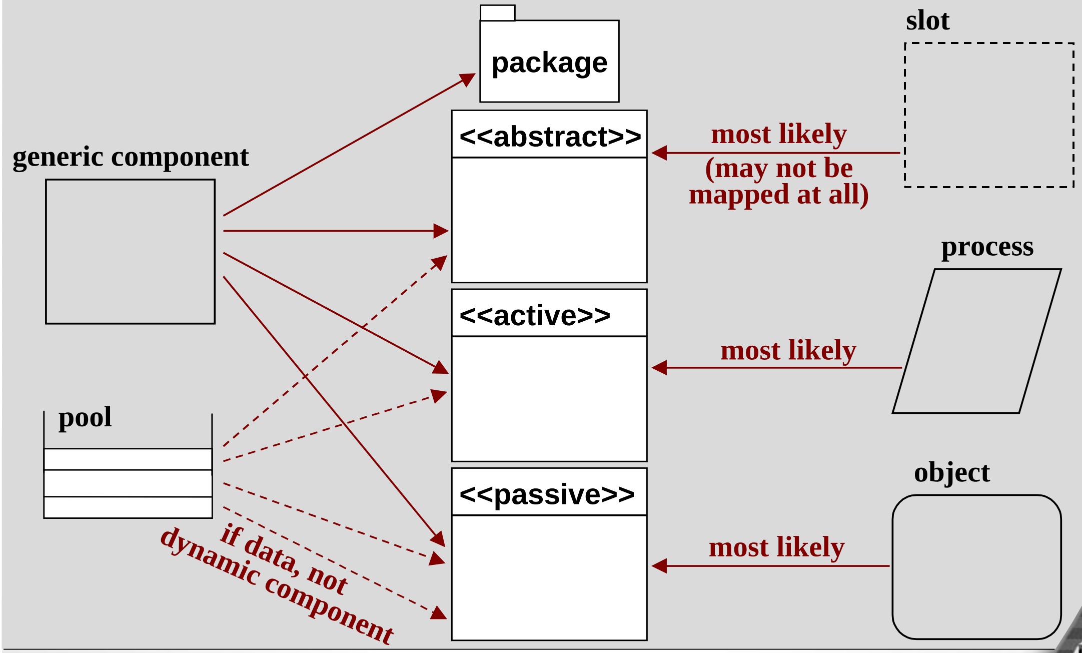
Table of Contents

- Introduction to Use Case Maps (UCMs) and the UCM Notation
- Dynamic Aspects of the UCM Notation
- Enhancing UCMs with Formal Scenario Definitions
- **Derivation of Structural Specifications from UCMs**
- Validation of Use Case Map Specifications
- Generation of Message Sequence Charts From UCMs
- Generation of Performance Models from UCMs
- Tool Demonstration: UCMNAV - The UCM Navigator
- Use Case Maps Exercise
- Use Case Map Styles for Small and Large Systems
- UCM Puzzles

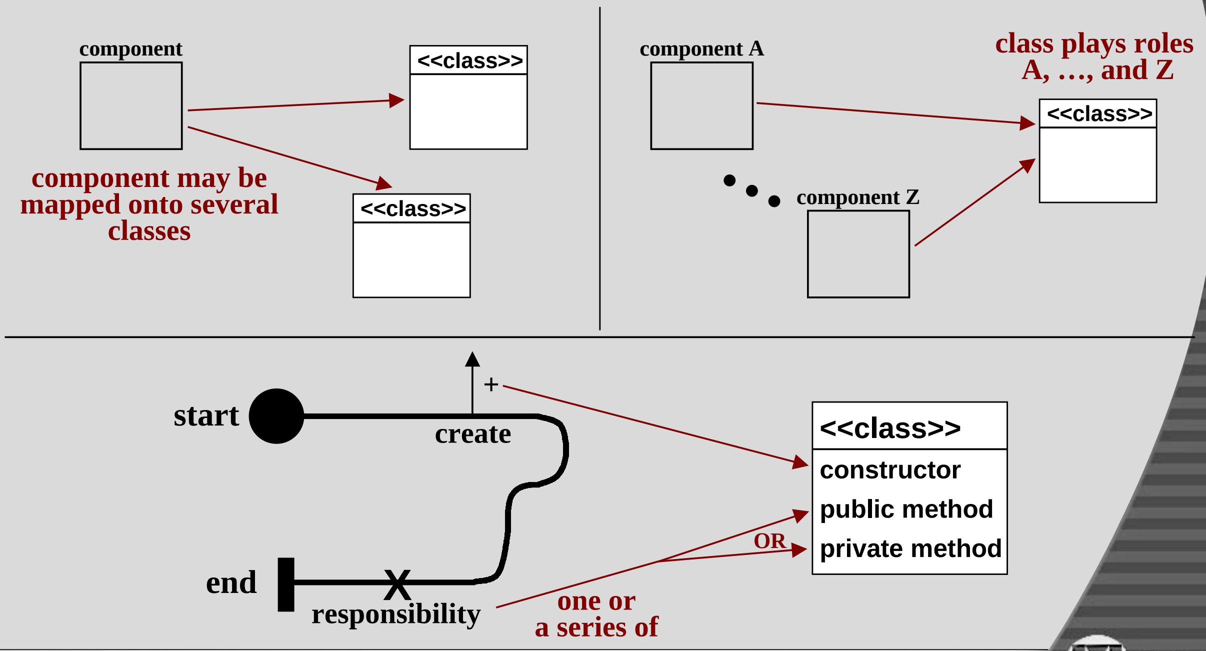
Derivation of Structure

- Simplest assumption: component in UCM is a role an instance of a class plays
- From roles to class diagrams at problem domain (conceptual) level or interface (specification) level, not implementation level
- There is a high degree of freedom for the mapping
- Therefore, the following slides are only guidelines
  - Look at all maps where component appears
  - Map components to classes anywhere in inheritance hierarchy
  - N-to-M mapping between UCM components and classes

Derivation of Structure



Derivation of Structure



component

component may be mapped onto several classes

<<class>>

<<class>>

component A

component Z

class plays roles A, …, and Z

<<class>>

start

create

+

end

responsibility

one or a series of

OR

<<class>>
constructor
public method
private method

Table of Contents

Generic Problem with Scenarios

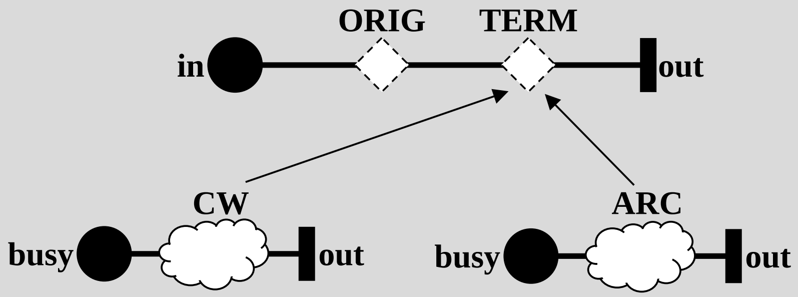
- Given a set of scenarios capturing informal (functional) requirements
- Specify (formally) the integration of scenarios
  - Undesirable emergent behaviour may result…
- Validate, i.e. look for logical errors and check against informal requirements
- Numerous tools and techniques can be applied (e.g.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

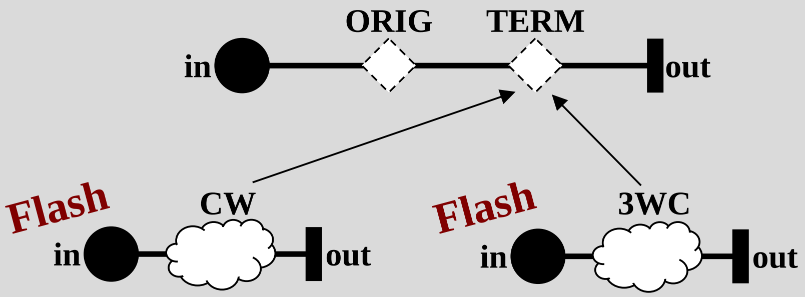
Feature Interaction

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



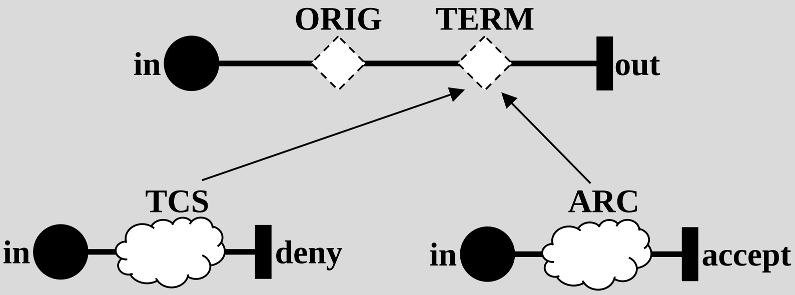
Feature Interaction

- Conflict between selected plug-ins (both plug-ins are triggered by the same event)
  - Flash key pressed with active call waiting (CW) and three way calling (3WC)



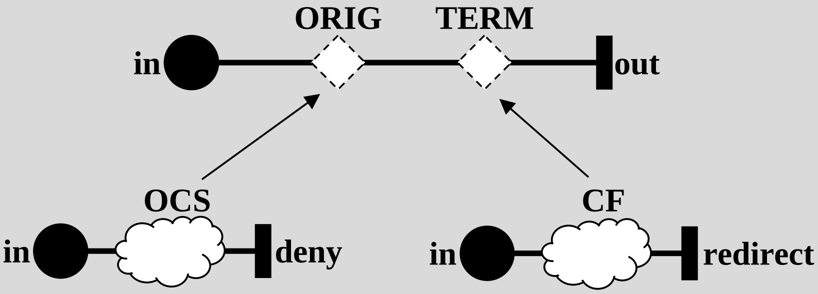
Feature Interaction

- Inconsistent behavior among different selected plug-ins for the same stub (postconditions of plug-ins are not the same)
  - Terminating call screening (TCS) denies call whereas automatic re-call (ARC) accepts call



Feature Interaction

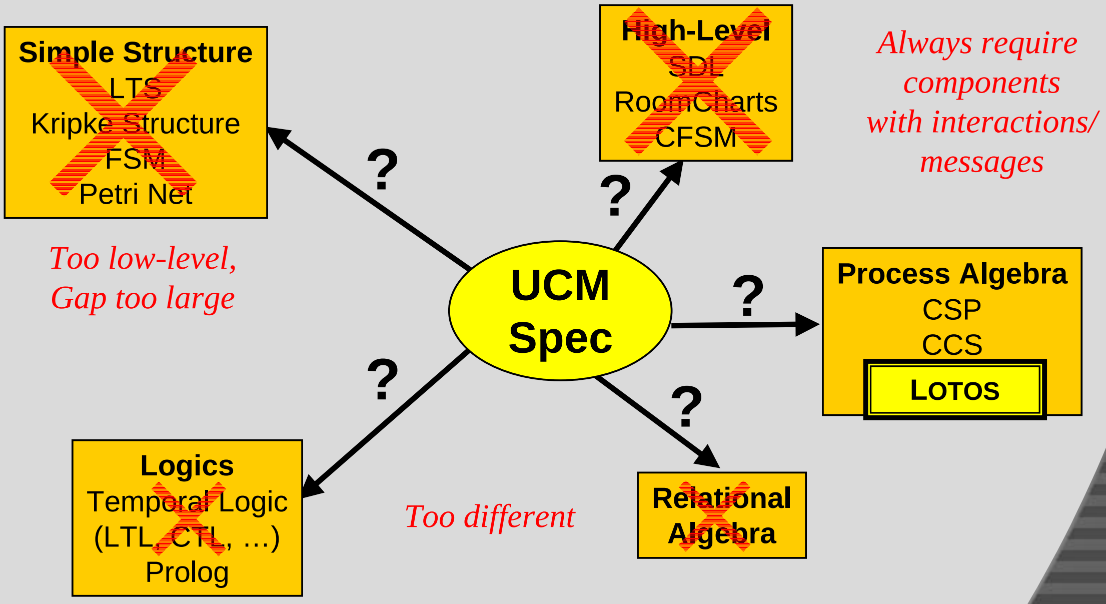
- Unexpected behavior among different selected plug-ins for different stubs (postconditions of plug-ins are not the same)
  - Originating call screening (OCS) denies call whereas call forward (CF) redirects call to screened number



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

Q1. Candidate Target Languages



Too low-level, Gap too large

Always require components with interactions/ messages

Too different

Why LOTOS?

- Formal syntax and semantics
- International standard (ISO 8807)
- Executable
- Good theories and tools for validation and verification
  - Testing
  - Model checking
  - Equivalence checking
  - Symbolic execution
  - …

Complementary Yet Compatible!

Use Case Maps

Scenario notation, readable, abstract, scalable, loose, relatively effortless to learn

LOTOS

Mature formal language, good theories and tools for V&V and completeness & consistency checking.

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

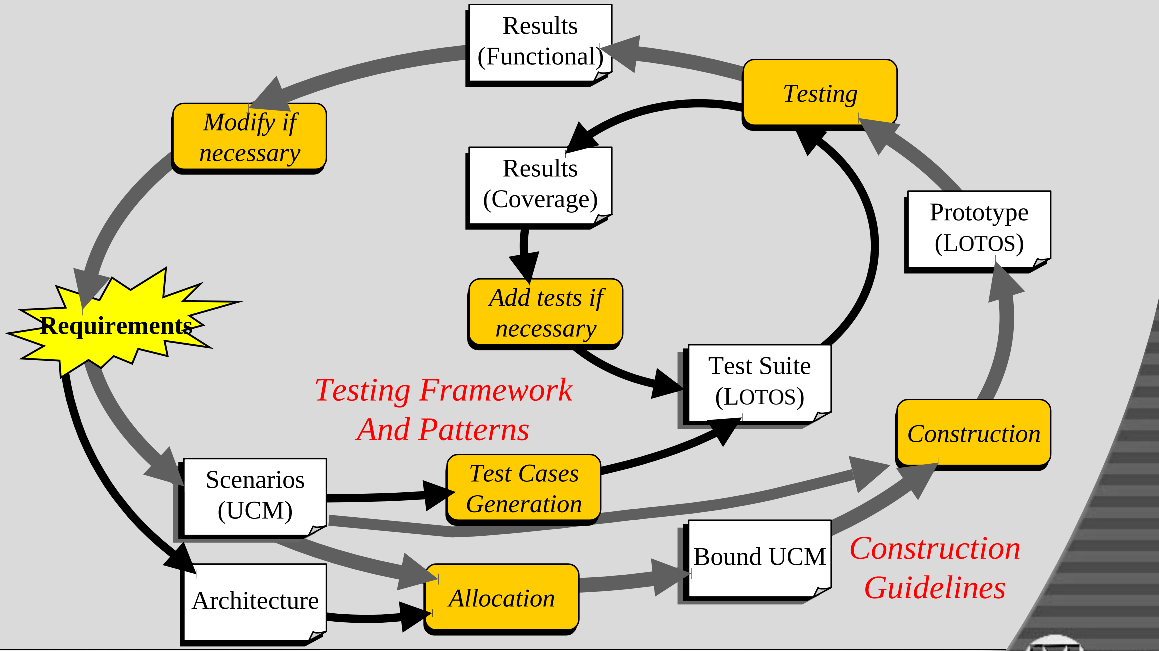
From UCMs to LOTOS

Start/end points	(Visible) gates
Responsibilities	(Hidden) gates
Agents/components	Processes
Stubs	Processes (implement selection policies)
Plug-ins	Processes
Inter-path causality	Inter-process synchronization
Data, conditions	Abstract Data Types

Q2. Construction Strategies

	Analytic	**Synthetic**
Benefits	–No formal source model required –Exploit richness of source and target languages –Take into consideration NFRs and implementation constraints	–Automated –Correctness "ensured" by construction –Verification "not required" –Very quick construction in one iteration
Drawbacks	–Manual transformation, prone to errors –Verification required –Time-consuming iterations	–Formal and detailed source model required –Restricted use of languages –May result in improper scenario integrations –Cannot take into consideration NFRs and implementation constraints

Specification-Validation Approach with LOTOS and UCMs

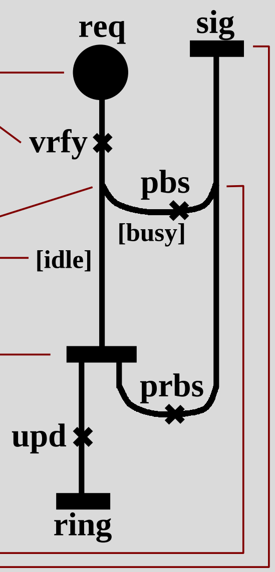


Construction Guidelines

- Paths
  - Interaction points and responsibilities
  - Causal paths (inside components)
  - Stubs and plug-ins
  - Other path elements
- Structure
  - Component topology
  - Multiple unrelated paths in a component
  - Inter-component causality
- Data
  - Component identifiers, conditions, pools/DBs

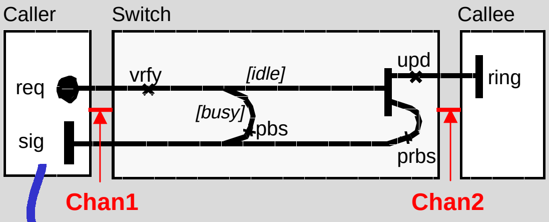
Application to Unbound UCMs

```
(* Announcement ADT definition… *)
req;
vrfy;
(
    [busy] ->
        ( pbs;  exit  !busyTone )
    []
    [idle] ->
        (
            prbs;  exit  !ringBackTone
            |||
            upd; ring;  exit  !any:Announcement
        )
)
>>  accept  signalTone: Announcement in
  sig!signalTone;  stop
```



Application to Bound UCMs

The structure is mapped onto a set of processes composed through channels or shared events.



Each component becomes a process that implements all the paths that cross it (possibly from multiple scenarios).

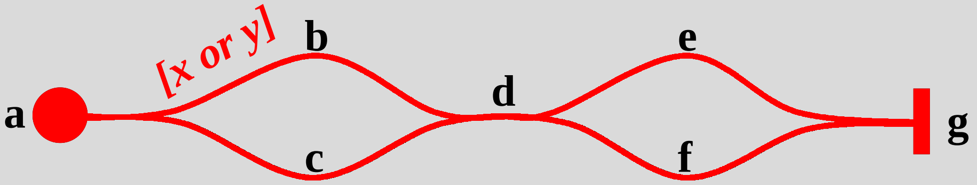
```
specification System[req, sig, ring] :noexit
(* Abstract Data Types here... *)
behaviour
   hide Chan1, Chan2 in
      ( Caller[req, sig, Chan1] ||| Callee[ring, Chan2] )
      |[ Chan1, Chan2 ]|
      Switch[Chan1, Chan2](idle, idle)
where
   (* Component processes here... *)
endspec (* System *)
```

```
process Caller[req, sig, Chan1] : noexit :=
   req ?calleeNum:PhoneNumber;
      Chan1 !request !calleeNum;
         Caller[req, sig, Chan1]
   []
   Chan1 ?ann:Announcement;
      sig !ann;
         Caller[req, sig, Chan1]
endproc (* Caller *)
```

UCM-Oriented Testing Pattern Language

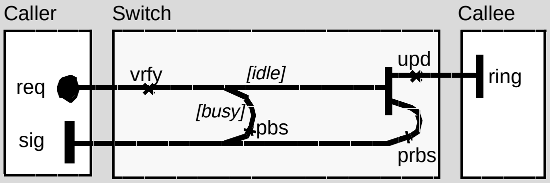
- Testing language connects testing patterns
  - UCM paths + testing patterns + target coverage = **test goals**
- Testing patterns
  - Alternatives
  - Concurrent Paths
  - Loops
  - Multiple Start Points
  - Single Stubs and Plug-ins
  - Causally Linked Stubs

Testing Pattern for Alternatives



- All results (end points): <a,b,d,e,g>
- All path segments: <a,b,d,e,g>,<a,c,d,f,g>
- All paths:
  <a,b,d,e,g>, <a,b,d,f,g>, <a,c,d,e,g>, <a,c,d,f,g>
- All combinations of sub-conditions
  (one with *x*, another with *y*…)

Application to Acceptance and Rejection Test Cases



Coverage: All path segments

Patterns: Alternative and Parallel

Abstract seq. 1: *<req, *vrfy, *pbs, sig>*

Abstract seq. 2: *<req, *vrfy, *upd, *prbs, ring, sig>*

Generation of **acceptance** & **rejection** tests for each abstract sequence

(* = not visible)

```
process Test1A[req, sig, ring, accept] : noexit :=
   req; sig!busyTone; accept; stop
endproc (* Test1A *)

process Test1R[req, sig, ring, reject] : noexit :=
   req; sig!ringBackTone; reject; stop
endproc (* Test1R *)

process Test2A[req, sig, ring, accept] : noexit :=
   req; ring; sig!ringBackTone; accept; stop
endproc (* Test2A *)

process Test2R[req, sig, ring, reject] : noexit :=
   req; ring; sig! busyTone; reject; stop
endproc (* Test2R *)
```

Complementary Strategies

- From test goals to test cases
  - Test goal + mapping = LOTOS test case
- Strategies for value selection
- Strategies for rejection test cases
  - Forbidden scenarios
  - Testing patterns
  - Incomplete conditions
  - Off-by-one value/result

Tool Output

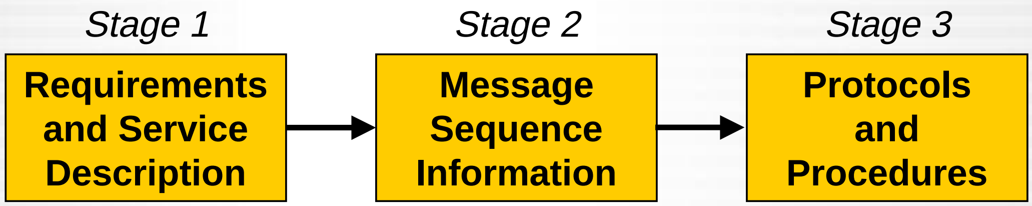
- LOLA (LOtos LAboratory, U. of Madrid)
- For each test case:
  - **Verdict**: Must pass, May pass (undesirable non-determinism), or Reject
  - Acceptance tests: expect Must pass
  - Rejection tests: expect Reject
  - Logical or mapping error (in test or in specification), feature interaction
- Structural coverage
  - Unreachable code in formal model (incorrect specification)
  - Incomplete test suite

Key Points - UCM Validation

- Scenario integration leads to potential problems (e.g. feature interaction) that require validation
- Formal specifications can be constructed from UCMs
  - Brings executability
  - Enables validation and automated feature interaction detection
  - Reuses existing tools, techniques and practices
- Analytic approach: SPEC-VALUE
  - UCM and LOTOS (dynamic duo!)
  - Bound UCMs require more design decisions
  - Validation testing

Table of Contents

Common Methodologies



Stage 1

Requirements and Service Description

Stage 2

Message Sequence Information

Stage 3

Protocols and Procedures

Informal requirements?
Use Cases?
UCM!

MSC or
UML interaction diagrams

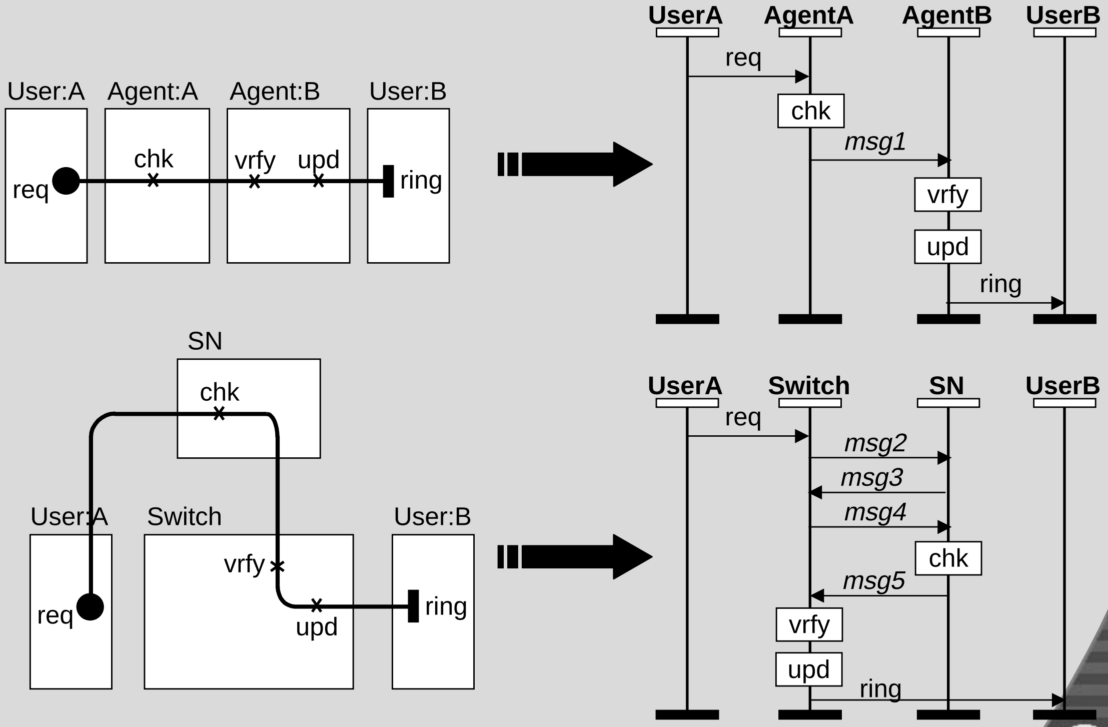
SDL or
UML Statechart diagrams

- Common design and standardisation methodologies already use scenarios
- Need improvement to cope with new realities of complex, dynamic, and evolving systems

Motivation for Transformation

- UCMs are good for… ***(Stage 1)***
  - Describing multiple scenarios abstractly
  - For analysing architectural alternatives
- MSCs are better for… ***(Stage 2)***
  - Developing and presenting the details of interactions
  - Describing lengthy sequences of messages in scenarios
  - Providing access to well-developed methodologies and tools for analysis and synthesis
- UCM-to-MSC transformation helps to further bridge the gap between Stage 1 descriptions (**requirements**) and Stage 2 descriptions (**design**).

Refining UCM with Messages



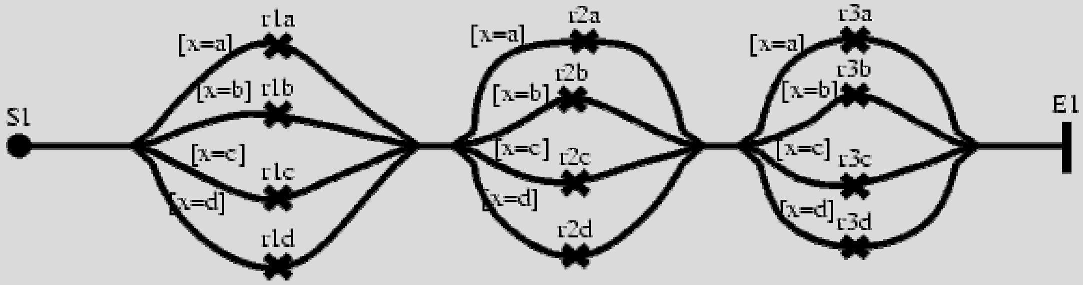
From UCM to MSC

- UCM component → MSC instance
- UCM path crossing from one component to another → abstract MSC message ("implements" causal flow)
- UCM start (or end) point → abstract MSC message
- UCM pre/post-condition → MSC condition
- UCM responsibility → MSC action
- UCM OR-fork or dynamic stub with multiple plug-ins → multiple basic MSCs
- UCM AND-fork → MSC parallel inline box
- UCM loop → MSC loop box
- UCM timer → MSC timer

Approaches to Transformation

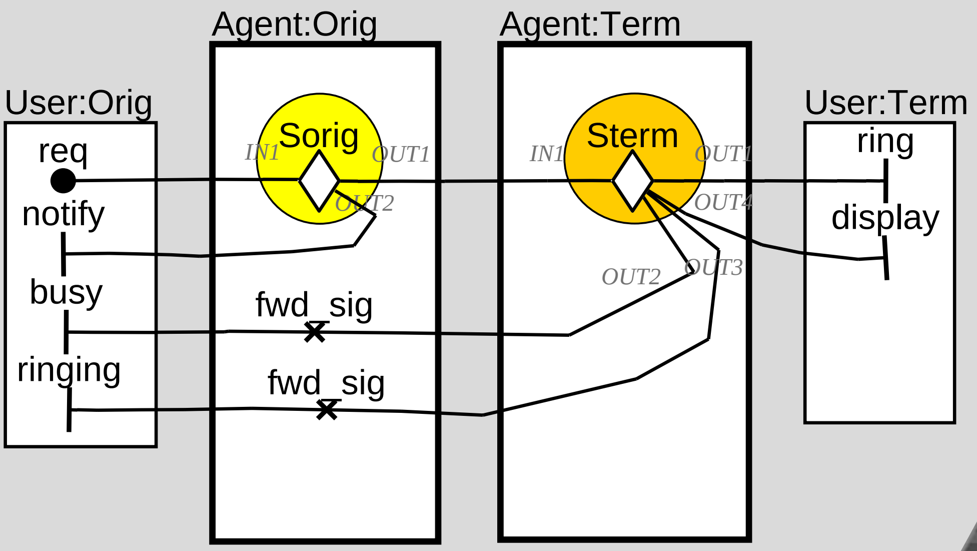
- Intermediate Formalism
  - Extract formal (LOTOS, SDL…) model from UCM
  - Generate MSC from model
  - High quality and realistic MSC, but much effort
- Direct transformation
  - Requires less effort, but MSC of lower quality
  - Often enough for early analysis and refinement
- In both cases, UCM descriptions need to be supplemented by a path data model

Need for Path Data Model

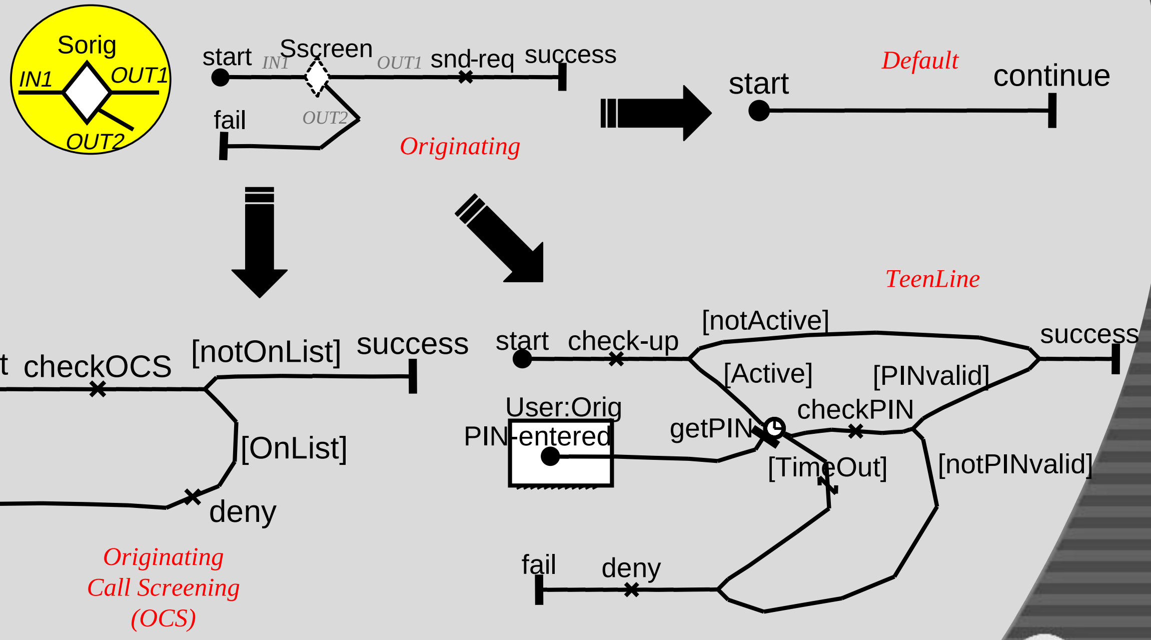


- 64 potential combination of end-to-end paths
- Reduced to 4 when conditions are taken into account
- Similar problem with combinations of plug-ins in dynamic stubs (e.g. embedded or in sequence)
- Path data model can help identify specific scenarios

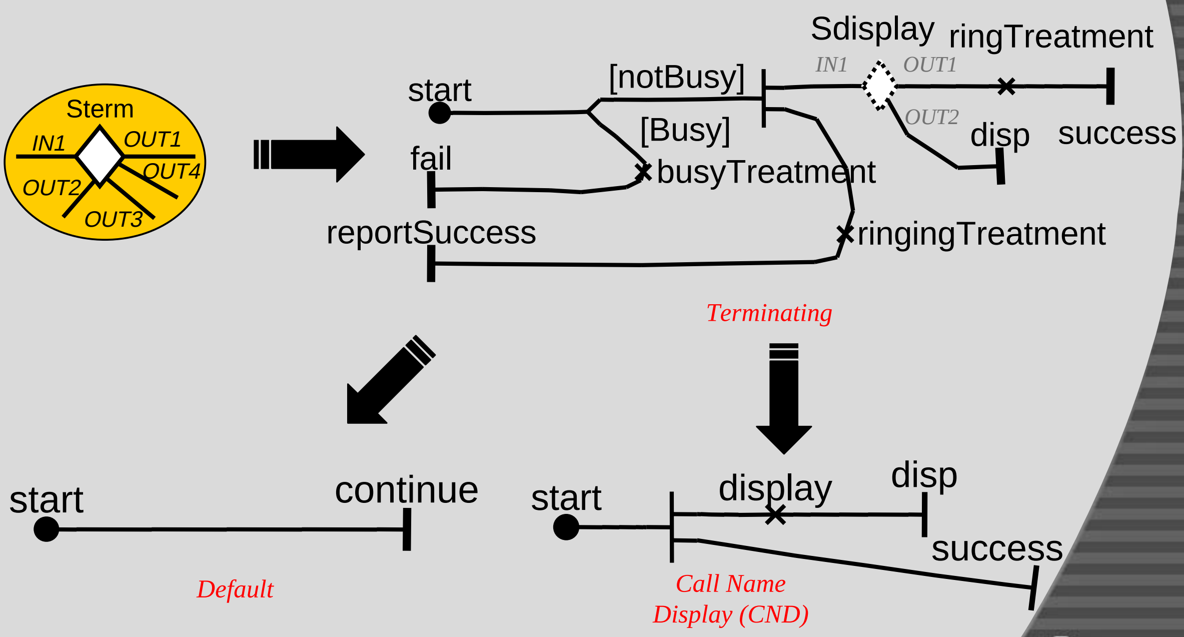
Agent-Based Basic Call with Three Features



Plug-ins for Sorig and Sscreen



Plug-ins for Sterm and Sdisplay



MSC Generation - Example

- From the example UCM specification:
  - Seven scenario variables are required
    - 3 for feature subscription, 4 for user conditions
    - Potential of 2^7 = 128 scenarios (assuming all choice points are guarded and deterministic)
  - 15 MSC scenarios can be generated:

Basic Call:	2 (success or busy)
CND:	1 (display)
OCS:	3 (success, busy, or denied)
TeenLine:	6 (active, valid PIN, timeout, busy…)
CND-OCS:	1 (success/display)
CND-TeenLine:	2
OCS-TeenLine:	0 (same as OCS alone!)
CND-OCS-TeenLine:	0 (same as CND-OCS!)

OCS, Successful Call

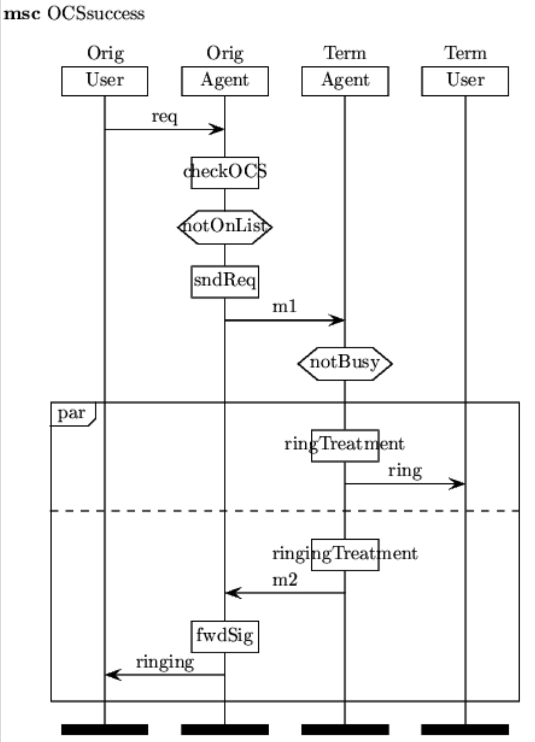
- subOCS = T
- subCND = F
- subTL = F
- Busy = F
- OnOCSList = F
- PINvalid = X
- TLactive = X
- Start point = req

To MSC (Z.120)

```
mscdocument OCSsuccess;
msc OCSsuccess;
User[Orig]:  instance;
Agent[Orig]: instance;
Agent[Term]: instance;
User[Term]:  instance;
User[Orig]:  out req,1 to Agent[Orig];
Agent[Orig]: in req,1 from User[Orig];
             action 'checkOCS';
             condition [notOnList];
             action 'snd_req';
             out m1,2 to Agent[Term];
Agent[Term]: in m1,2 from Agent[Orig];
             condition [notBusy];
all: par begin;
   Agent[Term]: action 'ringTreatment';
                out ring,3 to User[Term];
   User[Term]:  in ring,3 from Agent[Term];
all: par;
   Agent[Term]: action 'ringingTreatment';
                out m2,4 to Agent[Orig];
   Agent[Orig]: in m2,4 from Agent[Term];
                action 'fwd_sig';
                out ringing,5 to User[Orig];
   User[Orig]:  in ringing,5 from Agent[Orig];
all: par end;
Agent[Term]: endinstance;
Agent[Orig]: endinstance;
User[Orig]:  endinstance;
User[Term]:  endinstance;
endmsc;
```

OCS, Successful Call

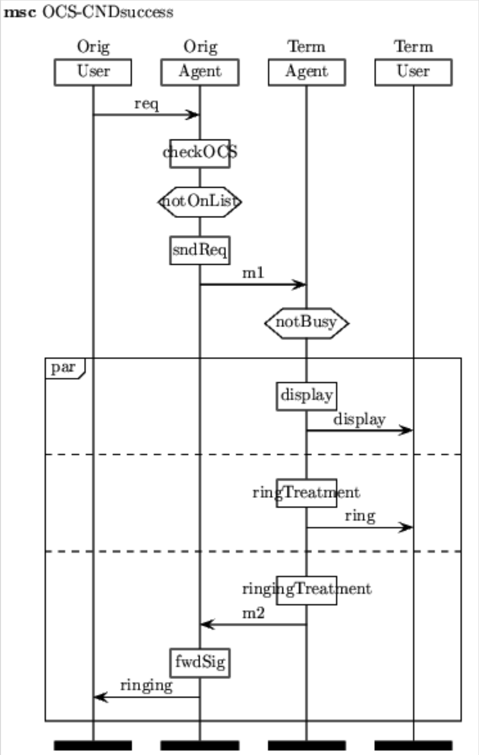
- subOCS = T
- subCND = F
- subTL = F
- Busy = F
- OnOCSList = F
- PINvalid = X
- TLactive = X
- Start point = req



OCS - CND Interaction

- subOCS = T
- subCND = T
- subTL = F
- Busy = F
- OnOCSList = F
- PINvalid = X
- TLactive = X
- Start point = req

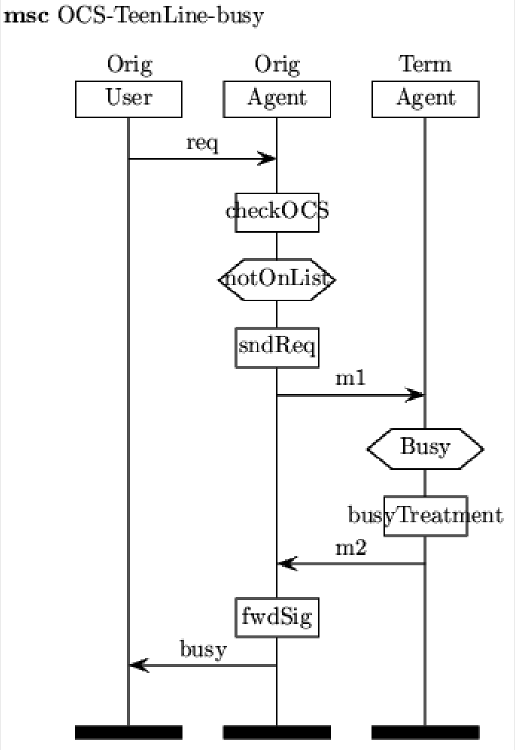
Desirable interaction!



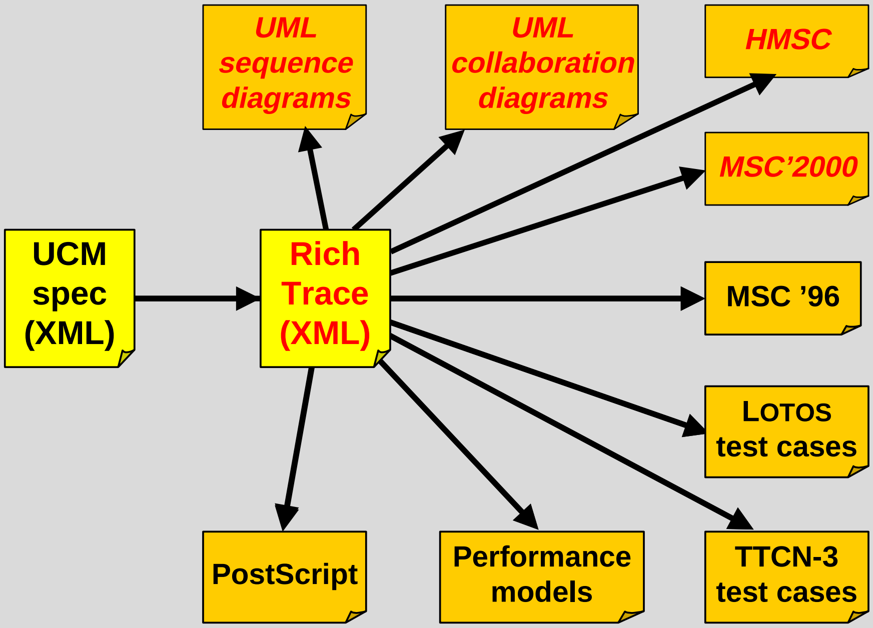
OCS - TeenLine Interaction

- subOCS = T
- subCND = F
- subTL = T
- Busy = T
- OnOCSList = F
- PINvalid = F
- TLactive = T
- Start point = req

Undesirable interaction!
TeenLine prevented by OCS, even when subscribed and active



Why Stop at MSCs?



UCM spec (XML)

Rich Trace (XML)

UML sequence diagrams

UML collaboration diagrams

HMSC

MSC'2000

MSC '96

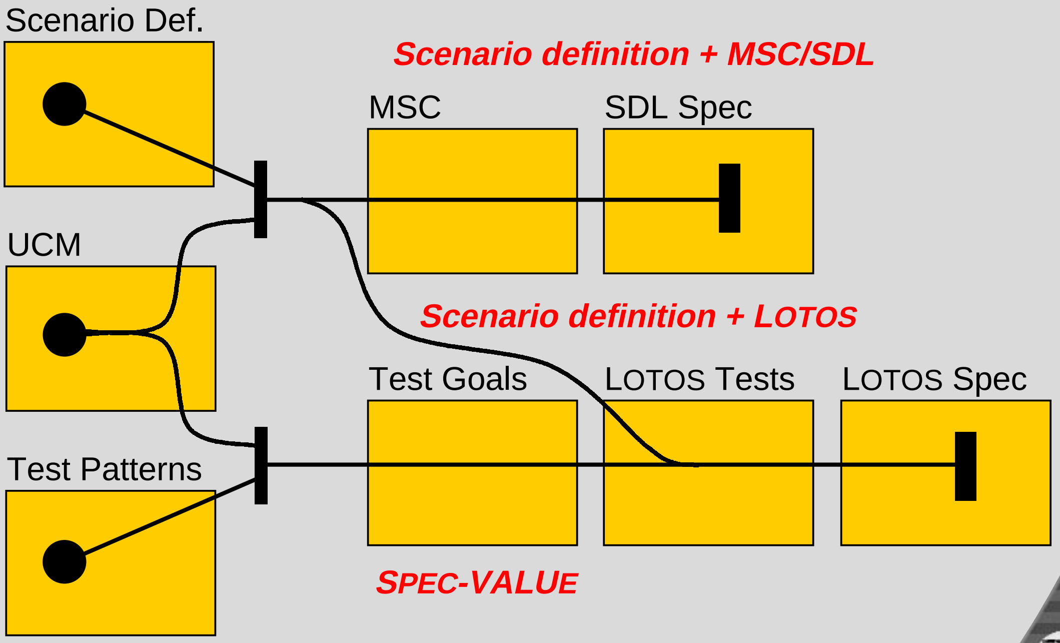
LOTOS test cases

PostScript

Performance models

TTCN-3 test cases

Revisiting Test Case Generation: UCM/Requirements Validation



Scenario Def.

UCM

Test Patterns

Scenario definition + MSC/SDL

MSC

SDL Spec

Scenario definition + LOTOS

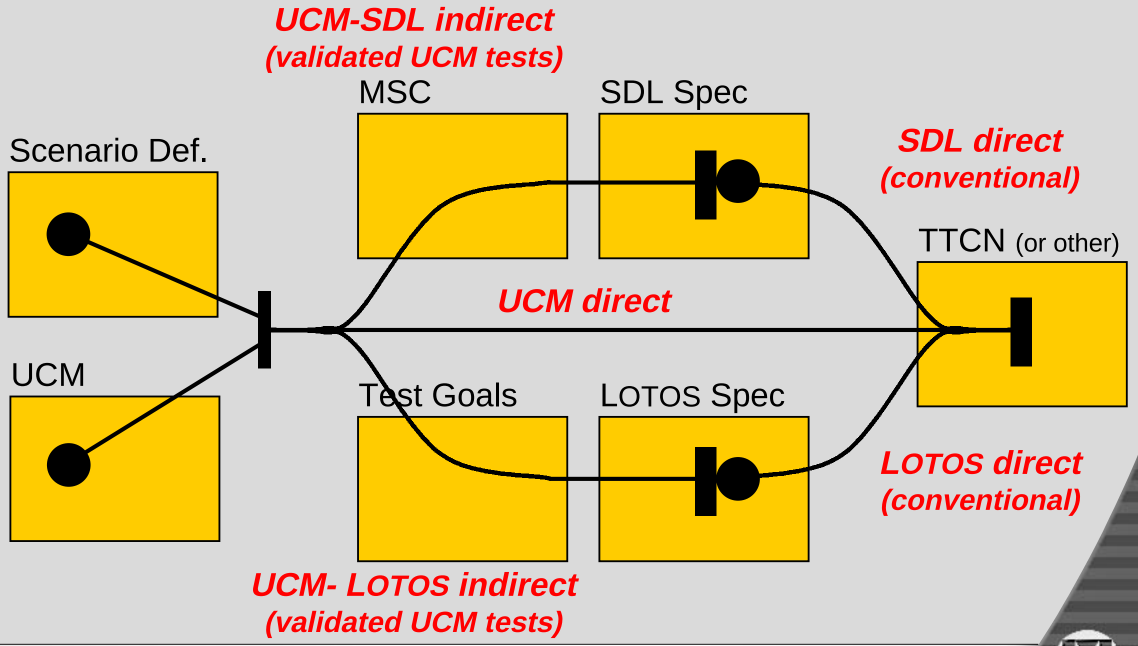
Test Goals

LOTOS Tests

LOTOS Spec

SPEC-VALUE

Revisiting Test Case Generation: Conformance Testing



UCM-SDL indirect
(validated UCM tests)

MSC

SDL Spec

Scenario Def.

SDL direct
(conventional)

TTCN (or other)

UCM direct

UCM

Test Goals

LOTOS Spec

LOTOS direct
(conventional)

UCM- LOTOS indirect
(validated UCM tests)

Key Points - MSC Generation

- Much value in a tool-supported translation
  - Effortless (push of a button)
  - MSCs in-sync with UCMs, forward traceability
  - Basis for further refinement
    - Synthetic abstract message may be refined into more concrete protocol messages…
  - Help to bridge the requirements/design gap
  - Other applications of path-traversal mechanism
- Need for path data model and scenario specifications

Table of Contents

Early Performance Evaluation

- Requires additional information to be added to UCM paths
  - Device characteristics (for processors, disks, …)
  - Response-time requirements for path segments (delay value and percentage of responses which must complete within that delay)
  - Arrival characteristics for start points
  - Device demand parameters for responsibilities (amount of service required from devices) and data access modes for responsibilities
  - Relative weights for OR forks to select branches
  - Allocation of components to processors

Early Performance Evaluation - PERFECT

- Specify behavior and concurrency architecture by annotating UCMs with performance information
- PERFormance Evaluation by Construction Tool (PERFECT)
  - Automatically translates annotated UCMs into representation of simulation tool (PARASOL)
  - Uses a simulated virtual implementation of the specification and heuristics for scheduling to evaluate the feasibility of software concurrency architectures
  - Reports the percentage of deadlines achieved, resource utilizations, and overhead costs

Early Performance Evaluation - LQN

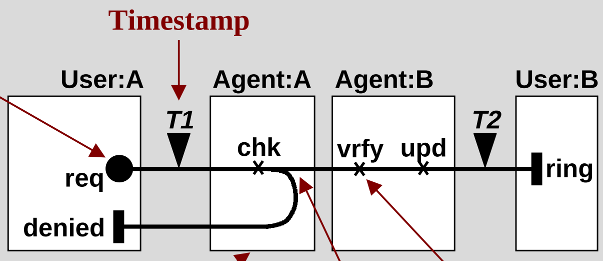
- ◆ Specify behavior and concurrency architecture by annotating UCMs with performance information
- ◆ Automatically translate annotated UCMs into LQN representation
- ◆ Layered Queuing Network (LQN)
  - – Provides a client-server performance view of systems
  - – Uses LQN Solver (LQNS) to solve LQN models
  - – Typical results are throughputs, response time, and utilization
  - – Used to determine and eliminate bottlenecks and for capacity planning

---

Performance Annotations

Arrival Characteristics
- Exponential, or
- Deterministic, or
- Uniform, or
- Erlang, or
- Other

Device Characteristics



Response Time Requirement
- From **T1** to **T2**
- Name
- Response time
- Percentage

Components
- Allocated responsibilities
- Processor assignment

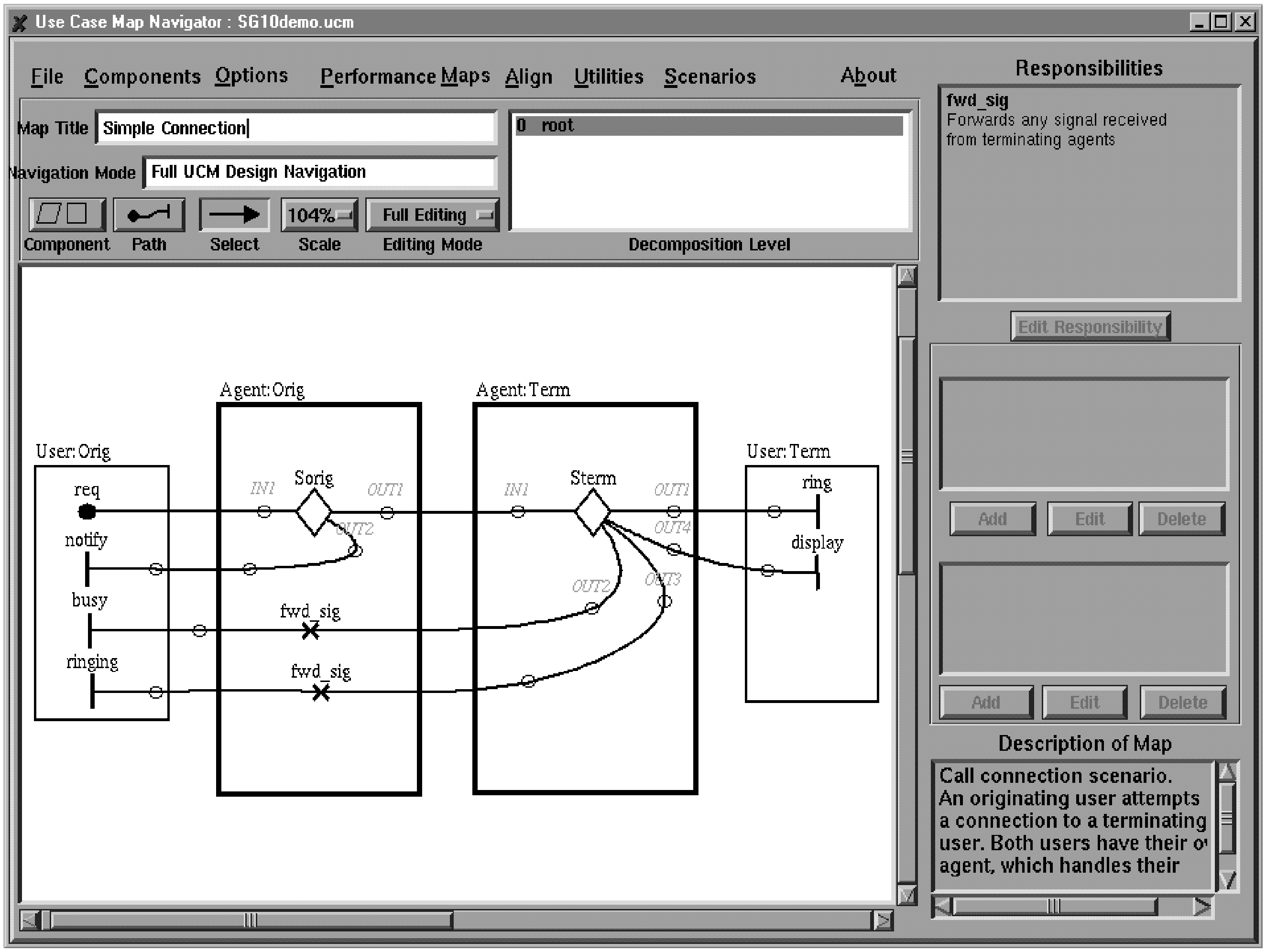
OR Forks
- Relative weights

Responsibilities
- Data access modes
- Device demand parameters

Table of Contents

UCMNAV

- Developed by Andrew Miga (Carleton U.) since 1997
- Editing and navigating of UCMs
- Supports UCM path and component notations
- Maintains bindings
  - Plug-ins to stubs, responsibilities to components, sub-components to components, etc.
- Editing is transformation-based
  - Operations maintain syntactic correctness and enforce some static semantics constraints



Use Case Map Navigator : SG10demo.ucm

Map Title: Simple Connection

Navigation Mode: Full UCM Design Navigation

Responsibilities

fwd_sig
Forwards any signal received from terminating agents

Description of Map

Call connection scenario. An originating user attempts a connection to a terminating user. Both users have their o agent, which handles their

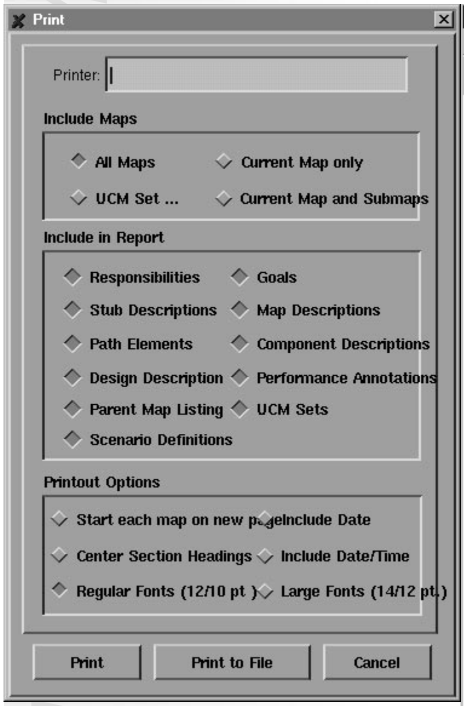
UCMNAV Facts

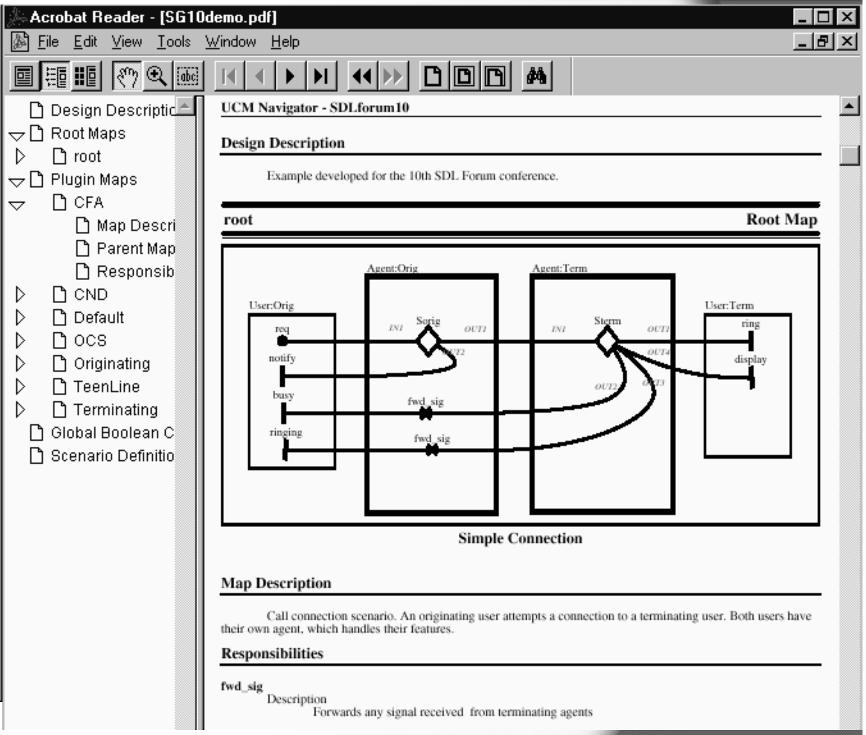
- Load/save/import/export in XML
- Developed in C++, GUI in Xforms
- Requires an X-server
- Multiple platforms are currently supported
  - Solaris, Linux (Intel and Sparc), HP/UX, and Windows (95, 98, 2000 and NT)
- Current stable version: 1.13.4
  - Freely available at http://www.UseCaseMaps.org
- Beta version: 2.0.0 (MSC generation)

UCM Documents

- XML (conforms to UCM DTD)
- Export of UCM figures
  - Encapsulated PostScript (EPS)
  - Maker Interchange Format (MIF)
  - Computer Graphics Metafile (CGM)
    - Used extensively in this tutorial
- Flexible report generation
  - Content options
  - PostScript, with PDF hyperlink information

Report Generation in PS/PDF





UCM Navigator - SDLforum10

Design Description

Example developed for the 10th SDL Forum conference.

root Root Map

Simple Connection

Map Description

Call connection scenario. An originating user attempts a connection to a terminating user. Both users have their own agent, which handles their features.

Responsibilities

fwd_sig
Description
Forwards any signal received from terminating agents

Table of Contents

Elevator Control System - Problem Description

- For each elevator, there are
  - **A set of elevator buttons.** A user presses a button to select a destination.
  - **A corresponding set of elevator lamps.** Indicate the floors to be visited by the elevator.
  - **An elevator motor.** Controlled by commands to move up, move down, and stop.
  - **A set of elevator doors.** Controlled by commands to open and close a door.
- For each floor, there are
  - **Up and down floor buttons.** A user presses a button to request an elevator.
  - **A corresponding pair of floor lamps.** Indicate the directions that have been requested.
- At each floor, and for each elevator, there is a pair of direction lamps to indicate whether an arriving elevator is heading in the up or down direction. For the top and bottom floors, there is only one floor button, one floor lamp, and (for each elevator) one direction lamp. There is also an arrival sensor at each floor in each elevator shaft to detect the arrival of an elevator at the floor.
- The hardware characteristics of the I/O devices are that the elevator buttons, floor buttons, and arrival sensors are asynchronous; that is, an interrupt is generated when there is an input from one of these devices. The other I/O devices are all passive. The elevator and floor lamps are switched on by the hardware, but must be switched off by the software. The direction lamps are switched on and off by the software.

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.

Elevator Control System - Actors and Use Cases

- The *Elevator Control System* has two actors: one representing the *Elevator User* who wishes to use the elevator and the second representing the *Arrival Sensor*. The *Elevator User* interacts with the system via the elevator buttons and the floor buttons.
- The *Elevator User* actor initiates two use cases, the *Select Destination* use case and the *Request Elevator* use case:
  - **Select Destination.** The user in the elevator presses an elevator button, either for a floor above or below the current position, to select a destination floor to which to move.
  - **Request Elevator.** The user at the floor presses an up or down floor button to request an elevator.

Elevator Control System - Select Destination Use Case

- **Actors:** Elevator User (primary), Arrival Sensor
- **Precondition:** User is in the elevator.
- **Description:**
  - User presses an elevator button for a floor above the current position. The elevator button sensor sends the user request to the system, identifying the destination floor the user wishes to visit.
  - The new request is added to the list of floors to visit. If the elevator is stationary, the system determines in which direction the system should move in order to service the next request. The system commands the elevator door to close. When the door has closed, the system commands the motor to start moving the elevator, either up or down.
  - As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.
  - If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user. Eventually, the elevator arrives at the destination floor selected by the user.
- **Alternatives:**
  - User presses an elevator button for a floor below the current position to move down. System response is the same as for the main sequence.
  - If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator has arrived at the destination floor selected by the user.

Elevator Control System - Request Elevator Use Case

- **Actors:** Elevator User (primary), Arrival Sensor
- **Precondition:** User is at a floor and wants an elevator.
- **Description:**
  - User presses an up floor button. The floor button sensor sends the user request to the system, identifying the floor number.
  - The system selects an elevator to visit this floor. The new request is added to the list of floors to visit. If the elevator is stationary, the system determines in which direction the system should move in order to service the next request. The system commands the elevator door to close. After the door has closed, the system commands the motor to start moving the elevator, either up or down.
  - As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.
  - If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user. Eventually, the elevator arrives at the floor in response to the user request.
- **Alternatives:**
  - User presses down floor button to move down. System response is the same as for the main sequence.
  - If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator has arrived at the floor in response to user request.

Elevator Control System - Exercise

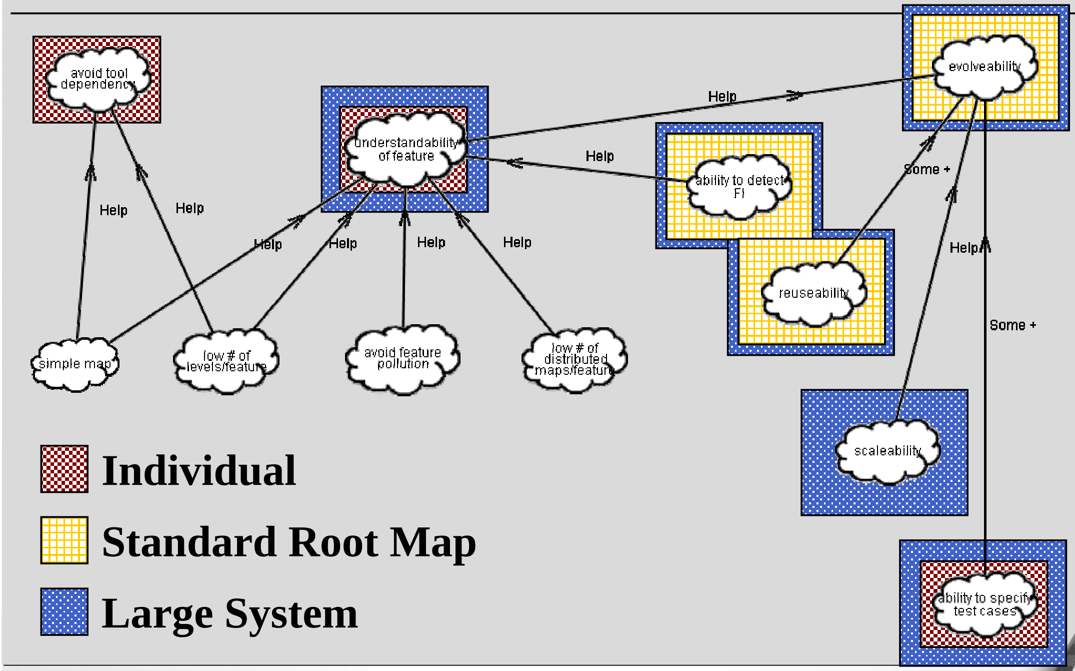
- Create UCMs for each use case
- Use the following components
  - User, Arrival Sensor (i.e. the actors)
  - Elevator Control System
- Do not attempt to use any of the following notational elements
  - Pools, Slots, Dynamic responsibilities, Timers

Solution

Table of Contents

- Introduction to Use Case Maps (UCMs) and the UCM Notation
- Dynamic Aspects of the UCM Notation
- Enhancing UCMs with Formal Scenario Definitions
- Derivation of Structural Specifications from UCMs
- Validation of Use Case Map Specifications
- Generation of Message Sequence Charts From UCMs
- Generation of Performance Models from UCMs
- Tool Demonstration: UCMNAV - The UCM Navigator
- Use Case Maps Exercise
- **Use Case Map Styles for Small and Large Systems**
- UCM Puzzles

Use Case Map Styles



▨ **Individual**

▦ **Standard Root Map**

▩ **Large System**

UCM Style - Individual

- Describe each feature individually (hierarchy of UCMs may be used for a feature)
- Each UCM belongs only to one feature and is not reused for any other feature
- Mainly concerned with …
  - Understandability of single features, ability to specify test cases, tool dependency (non-authors)
- Not concerned with …
  - Scaleability, feature interaction, evolveability, reuseability across features

---

UCM Style - Standard Root Map

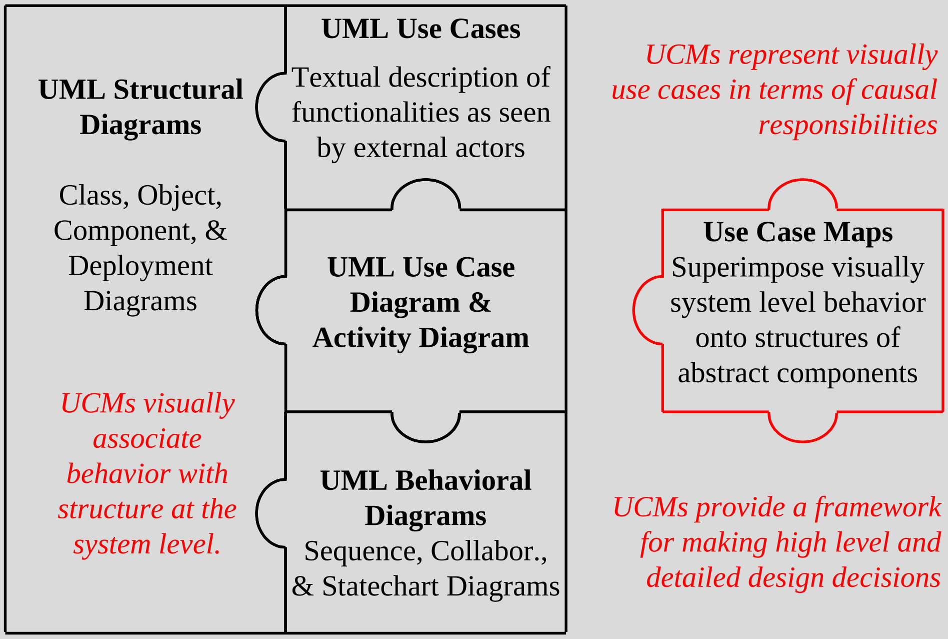
- Describe base feature on stub-rich root map with default plug-ins (hierarchy inherent)
- Describe variations of the base feature (i.e. other features) with the base root map and one or more different plug-ins to the stubs
- Mainly concerned with …
  - Feature interaction, evolveability, reuseability across features
- Not concerned with …
  - Scaleability

UCM Style - Large Systems

- Describe each feature on a separate root map (hierarchy of UCMs may be used for a feature)
- Integrate features by explicitly defining interaction points such as locations where …
  - a variation of the feature occurs or the system is ready to deal with an event that may not be related to the feature
- Mainly concerned with …
  - Scaleability, feature interaction, evolveability, reuseability across features, understandability of single features, ability to specify test cases
- Not concerned with …
  - Tool dependency (non-authors)

Table of Contents

UCM - UML Puzzle



UML Structural Diagrams

Class, Object, Component, & Deployment Diagrams

UCMs visually associate behavior with structure at the system level.

UML Use Cases

Textual description of functionalities as seen by external actors

UML Use Case Diagram & Activity Diagram

UML Behavioral Diagrams

Sequence, Collabor., & Statechart Diagrams

UCMs represent visually use cases in terms of causal responsibilities

Use Case Maps

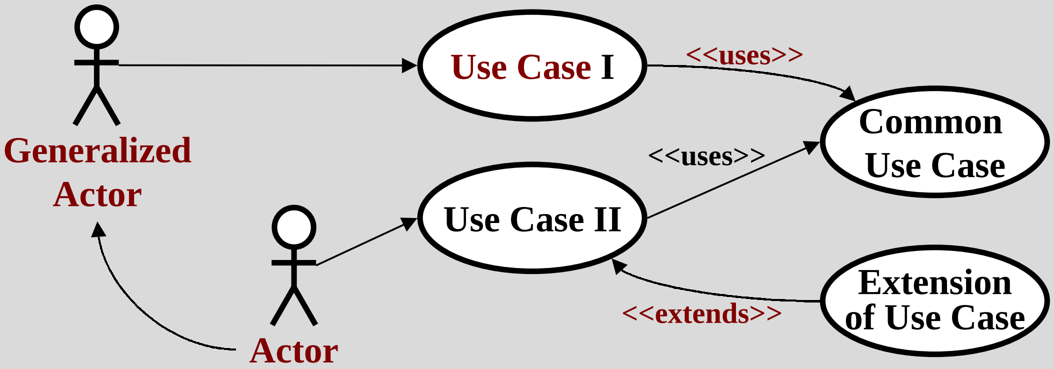
Superimpose visually system level behavior onto structures of abstract components

UCMs provide a framework for making high level and detailed design decisions

---

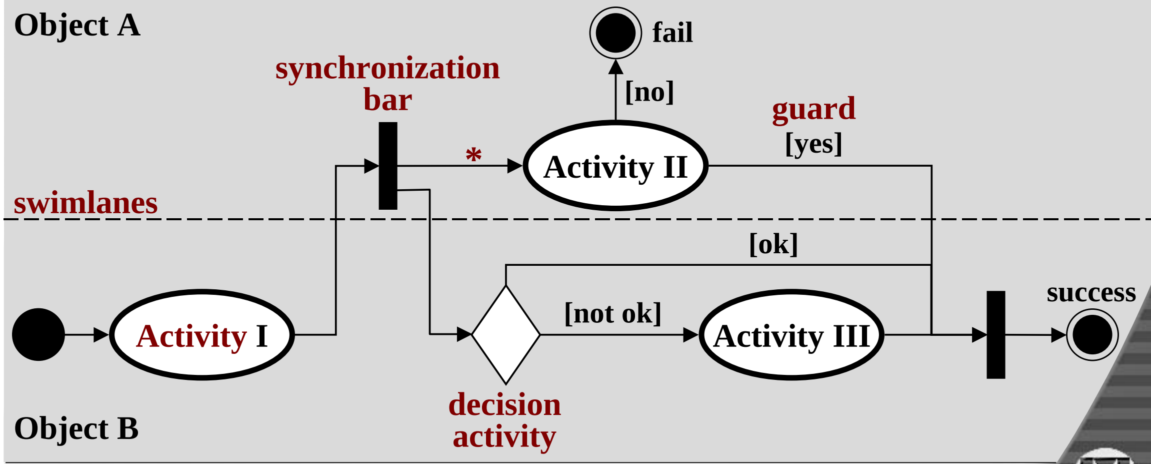
UML: Use Case Diagram

- Shows actors, use cases, and their relationships
- Shows system functionality from user's point of view
- Shows several use cases, variations and common parts of use cases, and generalization of actors



UML: Activity Diagram

- Shows dynamic behavior in terms of activities
- Shows behavior of many objects across many use cases



UCM - UML

- UCMs express almost all concepts of use case diagrams and all of activity diagrams
- Use case diagrams
  - UCMs express all concepts except actors and actor generalization; these concepts, however, could be modeled after minor changes to UCMs
  - UCMs show more precisely the location of and circumstances for extensions to use cases and for common use cases
  - How useful are use case diagrams?

UCM - UML

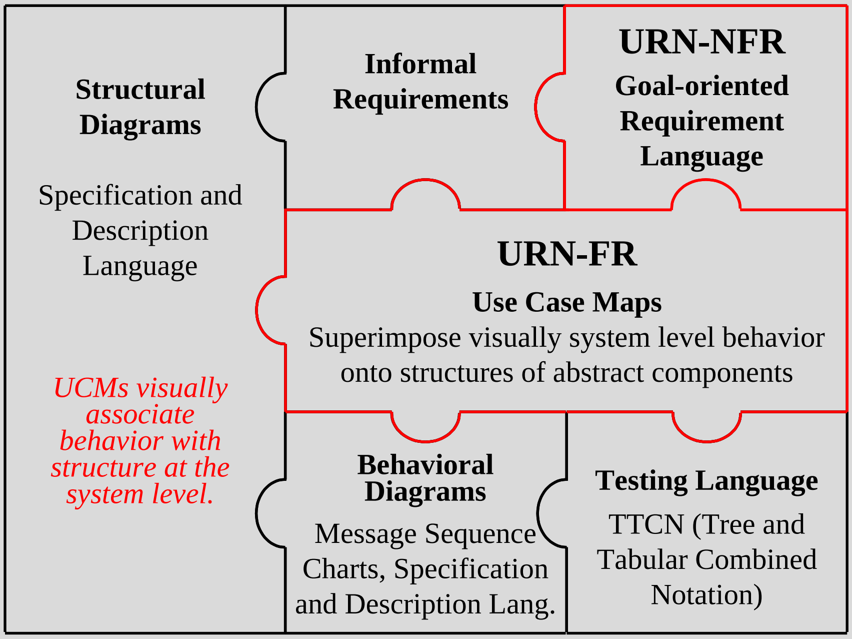
- Activity diagrams
  - UCMs provide dynamic stubs & selection policies
    - Selection policies specify which plug-in(s) to choose dynamically depending on preconditions and whether the plug-ins' execution occurs concurrently or sequentially
  - UCMs provide powerful dynamic capabilities
  - UCMs provide timers and timeout paths
  - UCMs allow several start points per map and several in-paths per stub
  - UCMs have a much more flexible and expressive mapping of structure to behavior

---

UCM - UML

- UCMs are at a higher abstraction level than sequence and collaboration diagrams
  - Causality instead of messages
- UCMs are at a higher abstraction level than statechart diagrams
  - Not focussed on component states and messages
- UCM components may be mapped to UML structural diagrams

ITU-T Study Group 10 Puzzle



Structural Diagrams

Specification and Description Language

UCMs visually associate behavior with structure at the system level.

Informal Requirements

URN-NFR

Goal-oriented Requirement Language

URN-FR

Use Case Maps

Superimpose visually system level behavior onto structures of abstract components

Behavioral Diagrams

Message Sequence Charts, Specification and Description Lang.

Testing Language

TTCN (Tree and Tabular Combined Notation)

UCMs represent visually use cases in terms of causal responsibilities

UCMs link to operationalizations (tasks) in NFR graphs

UCMs provide a framework for making high level and detailed design decisions

---

About ITU-T

- The International Telecommunication Union - Telecommunication Standardization Sector (ITU-T)
- ITU is a United Nation organization (189 members)
- 14 Study Groups in ITU-T
- SG10: Languages and software aspects for telecommunication systems (to become SG17)
- 13 questions for study in SG10 (MSC, SDL, UML…)
- Q12/10: **URN: User Requirements Notation**
  - Create a standard by end of 2003

URN - Initial Requirements

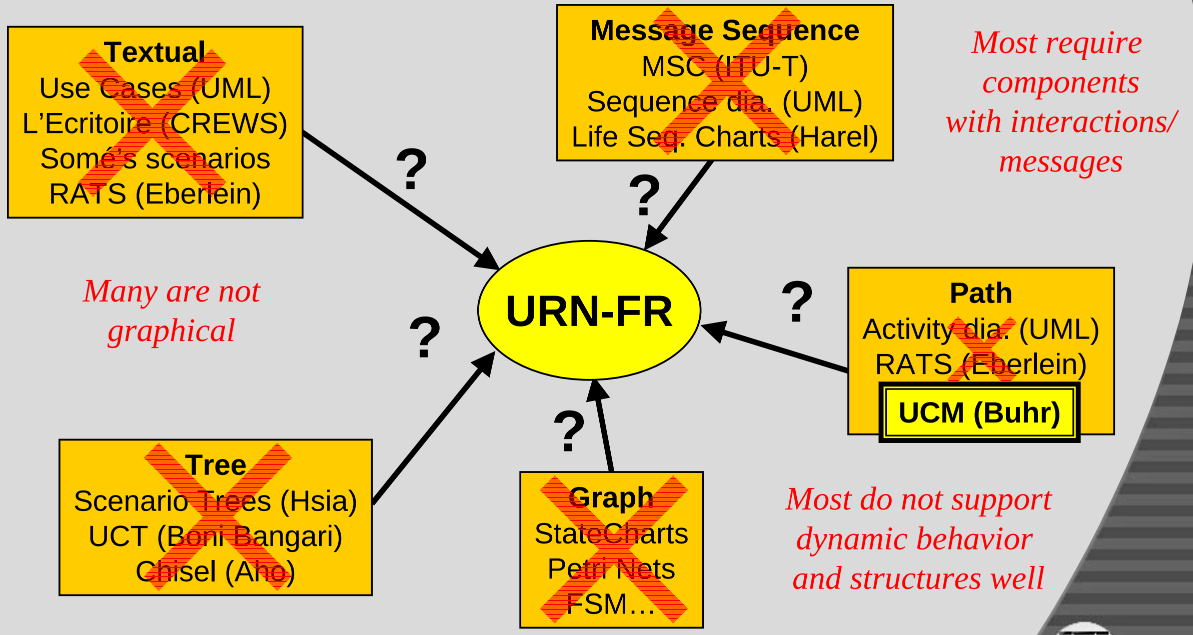
- Focus on early stages of design, with scenarios
- Capture user requirements when little design detail is available
- No messages, components, or component states required
- Reusability of scenarios and allocation to components
- Dynamic refinement capabilities
- Modelling of agent systems, early performance analysis, and early detection of undesirable interactions

---

URN - Additional Requirements

- Express, analyse and deal with non-functional requirements (NFRs)
- Express the relationship between business objectives/goals and system requirements
- Capture reusable analysis (argumentation) and design knowledge (patterns) for addressing non-functional requirements
- Connect to other ITU-T languages

Candidate Scenario Notations For URN-FR



~~Textual~~
~~Use Cases (UML)~~
~~L'Ecritoire (CREWS)~~
~~Somé's scenarios~~
~~RATS (Eberlein)~~

~~Message Sequence~~
~~MSC (ITU-T)~~
~~Sequence dia. (UML)~~
~~Life Seq. Charts (Harel)~~

Most require components with interactions/ messages

URN-FR

?

?

?

?

?

Many are not graphical

Path
~~Activity dia.~~ (UML)
~~RATS (Eberlein)~~
UCM (Buhr)

~~Tree~~
~~Scenario Trees (Hsia)~~
~~UCT (Boni Bangari)~~
~~Chisel (Aho)~~

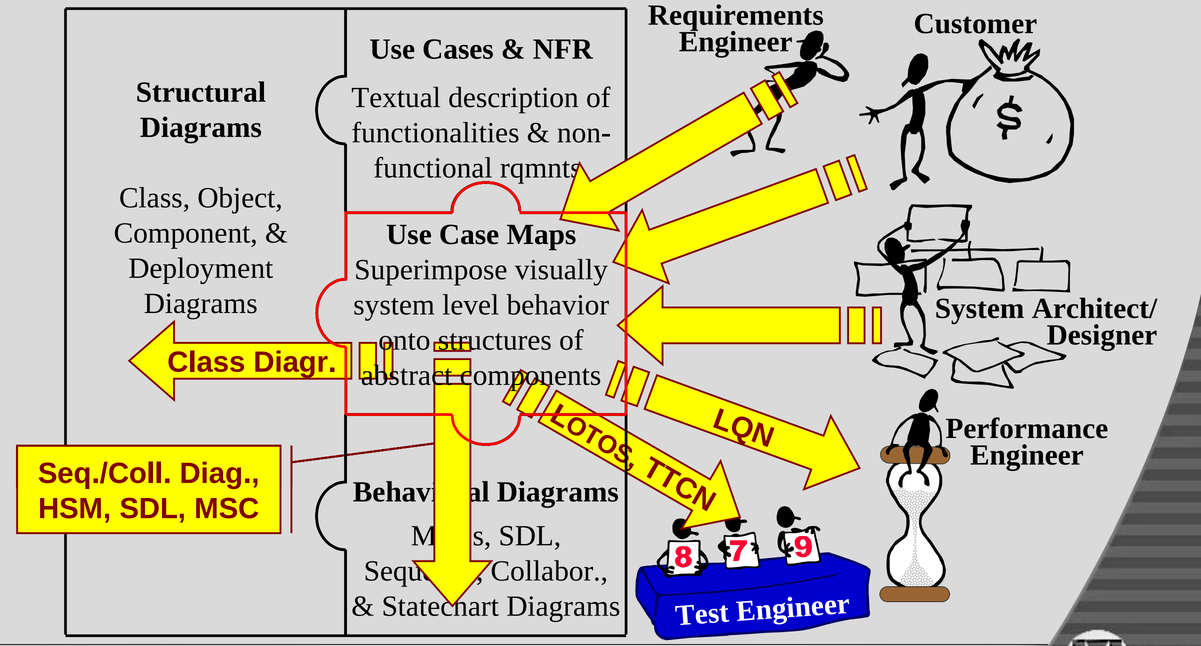
~~Graph~~
~~StateCharts~~
~~Petri Nets~~
~~FSM…~~

Most do not support dynamic behavior and structures well

Current Proposal for URN

- Combined use of two notations
- **Goal-oriented Requirement Language** (**GRL**) for Non-Functional Requirements
  - http://www.cs.toronto.edu/km/GRL/
- **Use Case Maps** (**UCM**) for Functional Requirements
  - http://www.UseCaseMaps.org/

Scope of Use Case Maps



Structural Diagrams

Class, Object, Component, & Deployment Diagrams

Use Cases & NFR

Textual description of functionalities & non-functional rqmnts

Use Case Maps

Superimpose visually system level behavior onto structures of abstract components

Behavi al Diagrams

M s, SDL, Sequ , Collabor., & Statechart Diagrams

Key Points - UCM Puzzles

- UCMs offer more capabilities than use case diagrams and activity diagrams
- UCMs, as part of the User Requirements Notation (URN), propose to fill a void in methodologies based on ITU-T languages
- Compared to other scenario notations, UCMs are graphical, do not require components with interactions/messages, and support dynamic behavior and structures well

Key Points - UCM Puzzles

- UCMs fit well into scenario-based software development methodologies
- UCMs become the focal point for early activities in software development, bringing together stakeholders with expertise in many different areas
- UCMs provide a good basis for design-time feature interaction detection and for model construction (tests, performance, MSC, LOTOS, and others)

Main References

- Web site http://www.UseCaseMaps.org/

  - Amyot, D. and Logrippo, L., *Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System*, In:Computer Communications 23(8), 2000.
  - Amyot, D. and Mussbacher, G., *On the Extension of UML with Use Case Maps Concepts*, UML2000, York, UK, October 2000.
  - Buhr, R.J.A., *Use Case Maps as Architectural Entities for Complex Systems*, In: Transactions on Software Engineering, IEEE, Vol. 24, No. 12, December 1998, pp. 1131-1155.
  - Buhr, R.J.A. and Casselman, R.S., *Use CASE Maps for Object-Oriented Systems*, Prentice Hall, 1996.
  - Cameron, D. et al., *Draft Specification of the User Requirements Notation*, Canadian contribution CAN COM 10-12 to ITU-T, November 2000.
  - Miga, A., Amyot, D., Bordeleau, F., Cameron, D., and Woodside, M., *Deriving Message Sequence Charts from Use Case Maps Scenario Specifications*, 10th SDL Forum, Copenhagen, Denmark, June 2001.
  - Scratchley, W.C., *Evaluation and Diagnosis of Concurrency Architectures*, Ph.D. thesis, Carleton University, Canada, November 2000.