

22 March, 2011

# The Aspect-oriented User Requirements Notation (AoURN): Aspects, Goals, and Scenarios

Gunter Mussbacher

SCE, Carleton University, Canada ◀▶ [gunter@sce.carleton.ca](mailto:gunter@sce.carleton.ca)

# Acknowledgements

- As this tutorial builds on several others, a special acknowledgement is due to the following people:
  - Daniel Amyot  
(University of Ottawa, Canada)
  - João Araújo, Ana Moreira  
(Universidade Nova de Lisboa, Portugal)
  - Jon Whittle  
(Lancaster University, UK)

# Table of Contents

- Overview of the User Requirements Notation (URN)
- Introduction to the Goal-oriented Requirement Language (GRL)
- Introduction to Use Case Maps (UCMs)
- Beyond the Basic Notations of URN

---

- Motivating Example

---

- Introduction to Aspect-oriented Requirements Engineering (AORE)

---

- The Aspect-oriented User Requirements Notation (AoURN) in a Nutshell
- Aspect-oriented Use Case Maps (AoUCM)
- Aspect-oriented GRL (AoGRL)

---

- Aspect-oriented Analysis and Validation with AoURN

---

- The jUCMNav AoURN Tool

---

- Summary, References, and Appendix (Notation Overview)





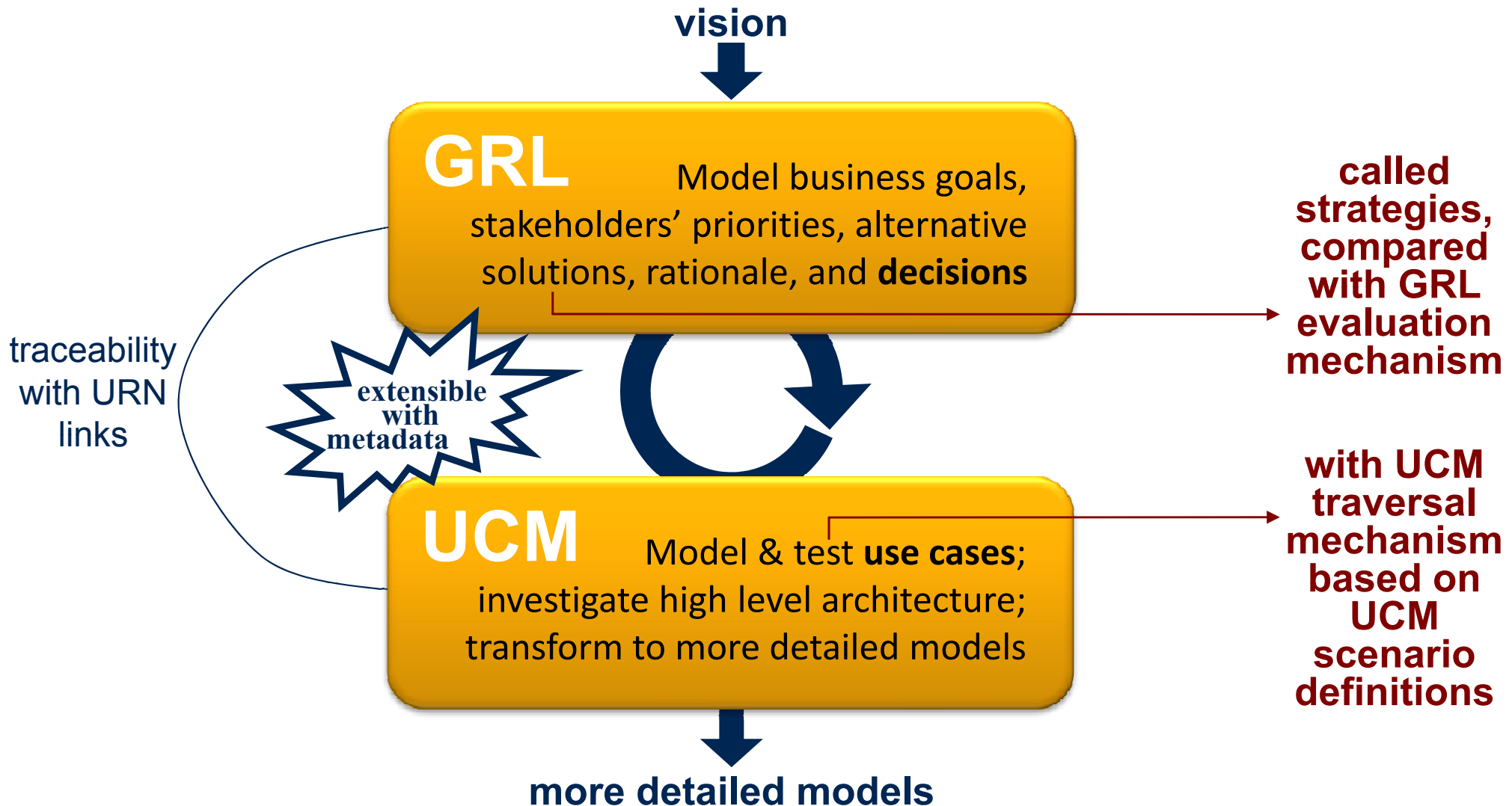
# Overview of the User Requirements Notation (URN)



# User Requirements Notation (URN): Overview (1)

- URN is a semi-formal, lightweight graphical language for modeling and analyzing requirements in the form of goals and scenarios
- Combines two existing notations
  - **Goal-oriented Requirement Language (GRL)**  
(based on i\* & NFR Framework)
  - **Use Case Maps (UCMs)**
- Support for the elicitation, analysis, specification, and validation of requirements
- Allows systems/software/requirements engineers to discover and specify requirements for a proposed or an evolving system, and analyse such requirements for correctness and completeness
- URN models can be used to specify and analyze various types of reactive systems, business processes, and telecommunications standards
- jUCMNav: URN editor & analysis tool, Eclipse plugin, open source project

# User Requirements Notation (URN): Overview (2)



- A GRL / UCM model visually communicates business objectives and constraints / high-level functional requirements to all stakeholders

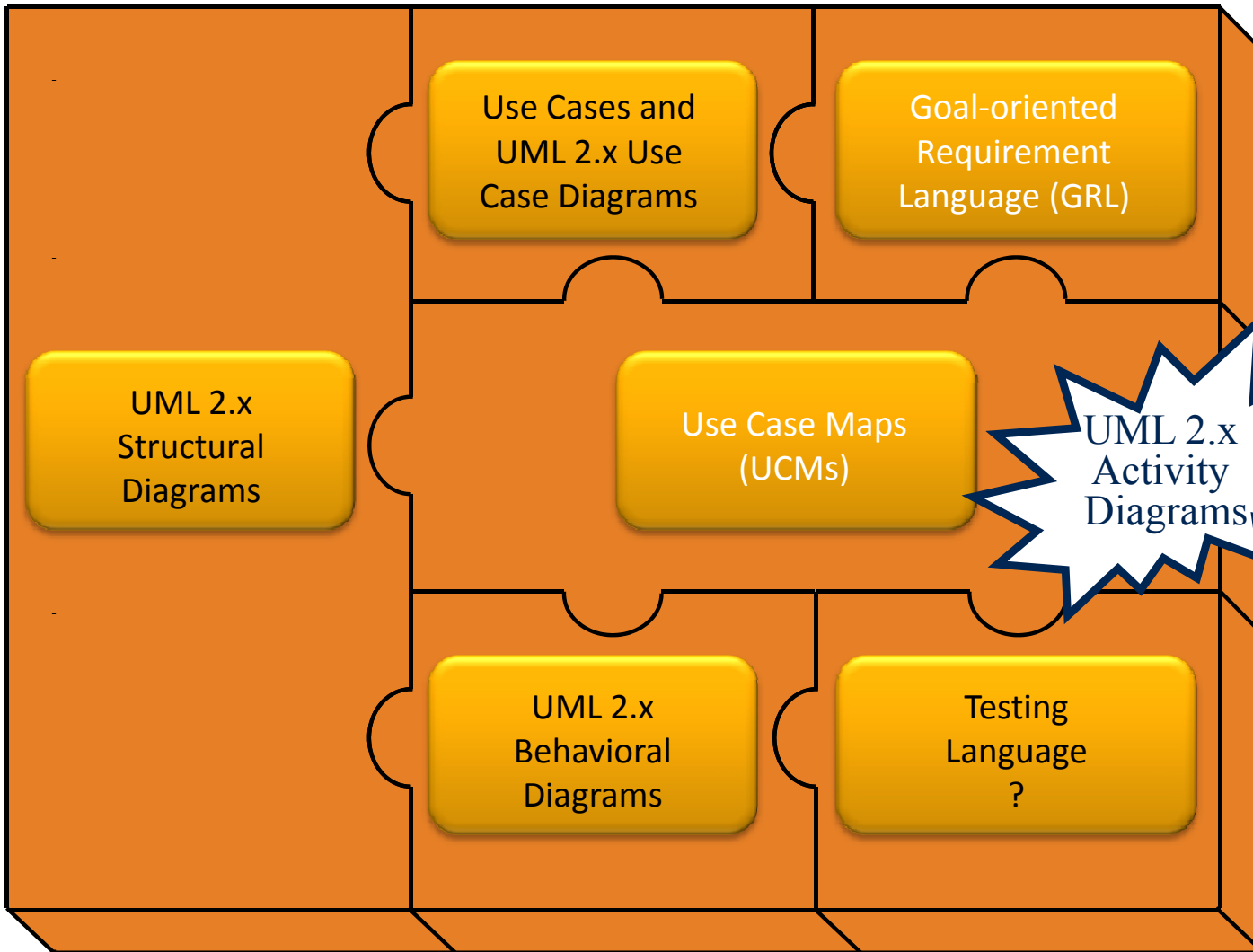


# ITU-T Z.151: URN – Language Definition

- URN is the **first** and **currently only** standard which explicitly addresses goals (non-functional requirements with GRL) in addition to scenarios (functional requirements with UCMs) in a **graphical** way in one unified language
  - International Telecommunication Union (ITU-T Z.150 series)
  - ITU-T Z.150 (02/03):  
User Requirements Notation (URN) - Language requirements and framework
  - ITU-T Z.151 (11/08):  
User requirements notation (URN) - Language definition
- Part of the ITU family of languages: SDL, MSC, TTCN-3, ASN.1...
- Definition of URN in Recommendation Z.151 (approved November 2008)
  - Metamodel, abstract/concrete syntaxes, semantics...



# User Requirements Notation: UML Puzzle



*GRL captures stakeholders' business goals, alternatives, decisions, and rationales.*

*Alternatives in GRL are linked with more detailed models described as UCMs.*

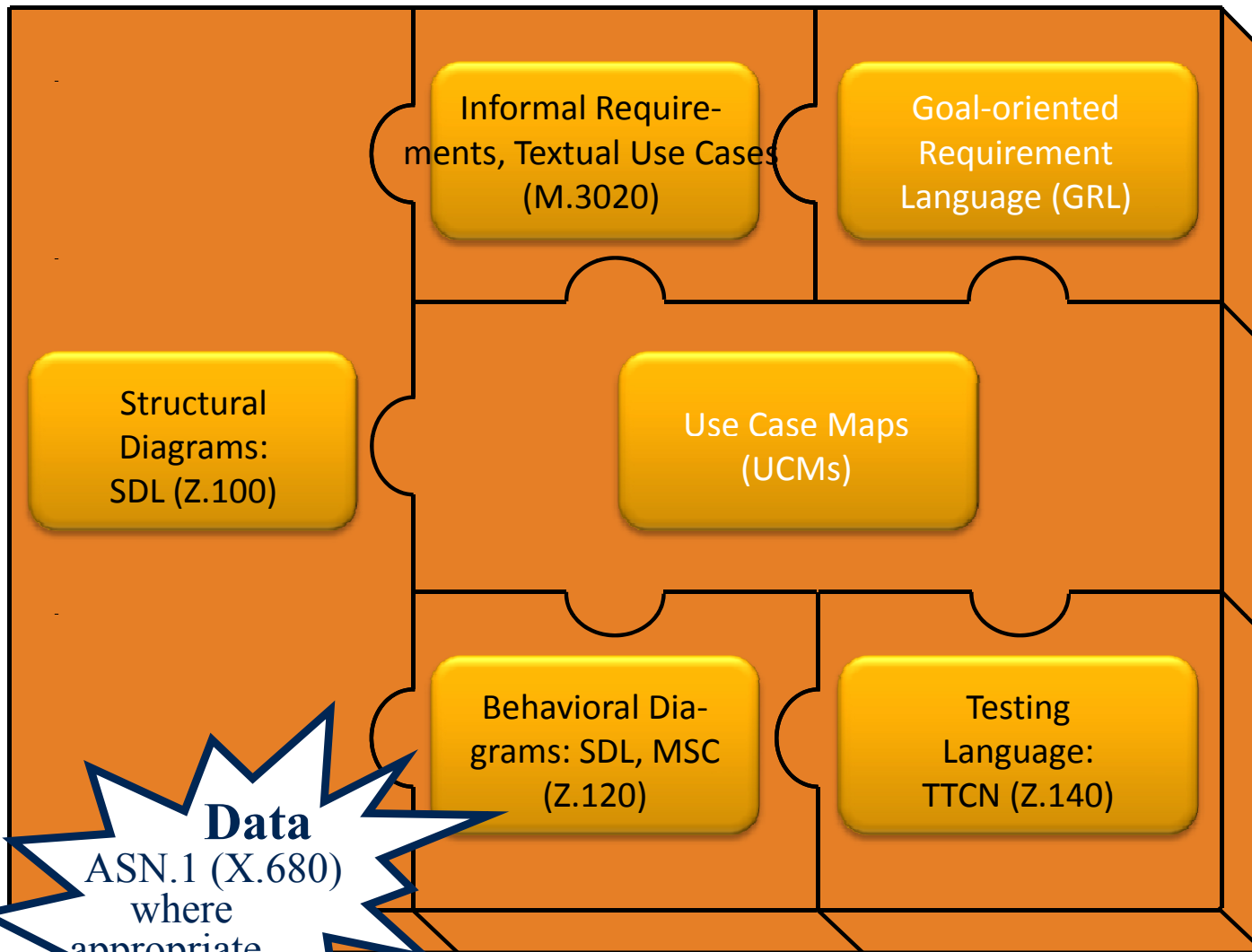
*UCMs represent visually use cases in terms of causal responsibilities.*

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

*UCMs visually associate behavior with structure at the system level.*

*UCMs enable functional testing and allow the generation of test purposes.*

# User Requirements Notation: ITU Puzzle



*GRL captures stakeholders' business goals, alternatives, decisions, and rationales.*

*Alternatives in GRL are linked with more detailed models described as UCMs.*

*UCMs represent visually use cases in terms of causal responsibilities.*

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

*UCMs visually associate behavior with structure at the system level.*

*UCMs enable functional testing and allow the generation of test purposes.*



# Introduction to the Goal-oriented Requirement Language (GRL)

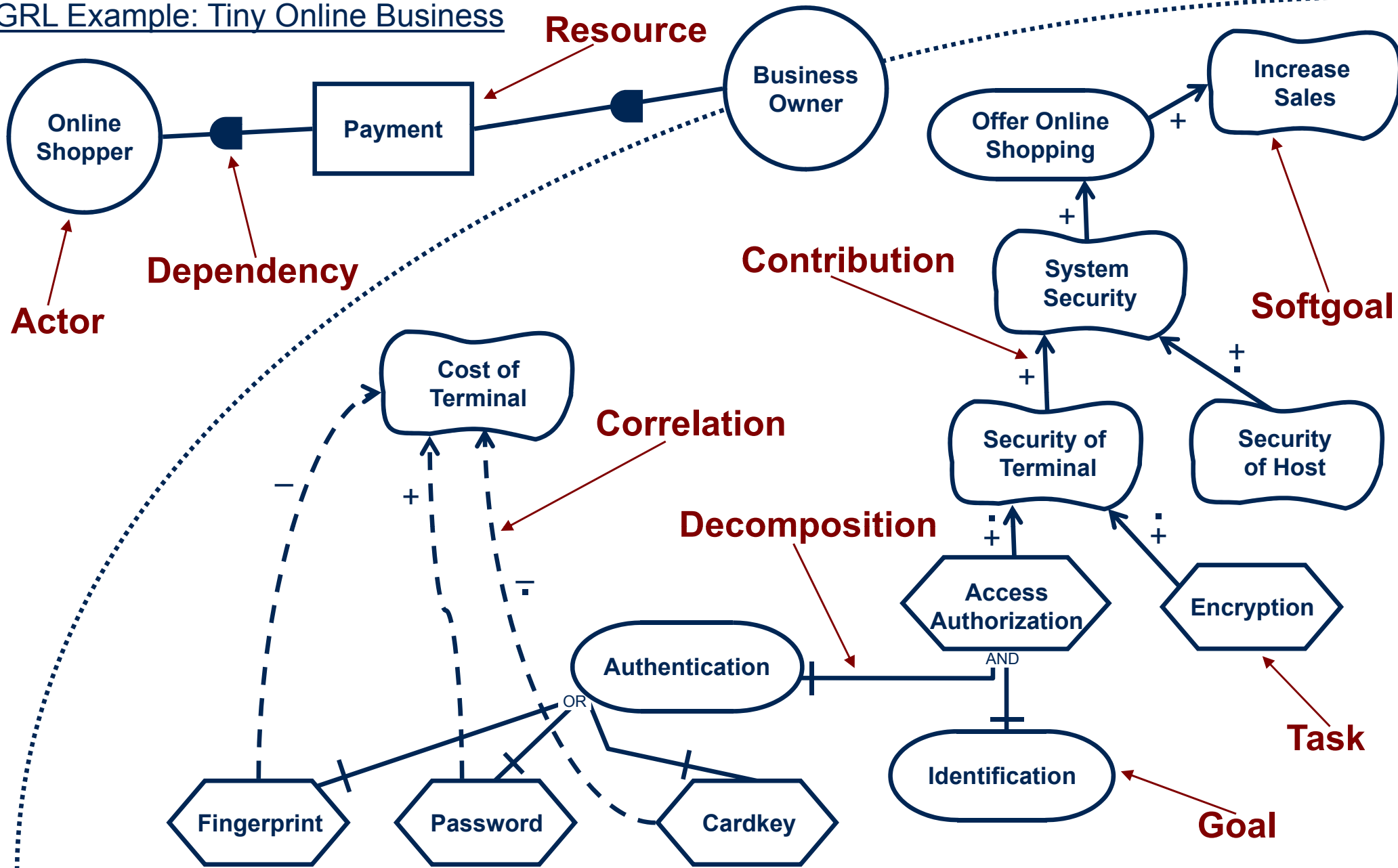


# Goal-oriented Requirement Language: Overview

- The Goal-oriented Requirement Language (GRL) is based on ...
  - i\* (concepts / syntax)
  - NFR Framework (evaluation mechanism)
- Model **goals** and other intentional concepts – mainly for non-functional requirements, quality attributes, rationale documentation, and reasoning about alternatives and tradeoffs
- GRL is used to ...
  - Visually describe business goals, stakeholders' priorities, alternative solutions, rationale, and decisions
  - Decompose high-level goals into alternative solutions called tasks (this process is called **operationalization**)
  - Model positive and negative influences of goals and tasks on each other
  - Capture dependencies between actors (i.e. stakeholders)








# Goal-oriented Requirement Language: Notation (1)

GRL Example: Tiny Online Business





# Goal-oriented Requirement Language: Notation (2)

- Intentional elements
    - Softgoal, goal, task ,resource, belief
  - Achievement of softgoal is qualifiable but not measurable; it is quantifiable for goals
  - Task is a proposed solution that achieves a goal or satisfies (**satisfices**) a softgoal
  - Belief captures rationale
- GRL Contributions Types:  
(qualitative)
- |   |   |
|---|---|
|  |  |
| Break   | Make  |
|  |  |
| Some-   | Some+   |
|  |  |
| Hurt  | Help  |
- Contribution and Correlation Links
    - Contribution describes desired impact, correlation shows side effects
    - Qualitative or quantitative contribution types are used for these links
  - Dependency Link
    - An actor (the depender) depends on another actor (the dependee) for something (the dependum), e.g. the business owner depends on the online shopper for payment (the dependum is optional)
-  Unknown





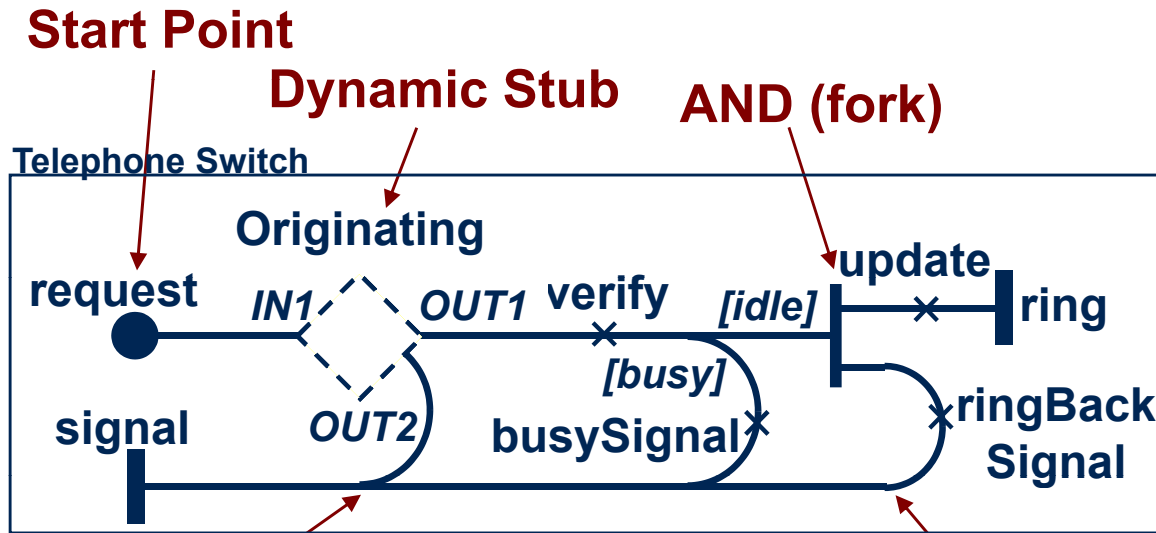
# Introduction to Use Case Maps (UCMs)

# Use Case Maps: Overview

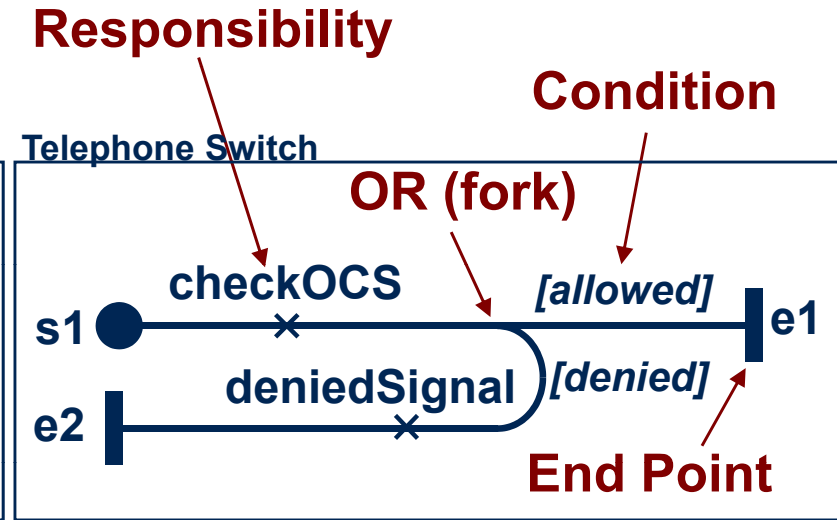
- Model **scenario concepts** – mainly for operational requirements, functional requirements, and performance and architectural reasoning
- Use Case Maps (UCMs) provide ...
  - Visual description of behavior **superimposed** over entities (from software architecture to actors to hardware)
  - Easy graphical manipulation of use cases/scenarios
  - Single use case/scenario view
  - **Combined system view**
  - Enhanced consistency and completeness
  - Smooth transition to design models (e.g. message sequence charts)
  - Connections to performance models and testing models

# Use Case Maps: Notation (1)

## UCM Example: Tiny Telephone System



**OR (join)** a) Basic Call map **Path**



b) *OCS* plug-in map



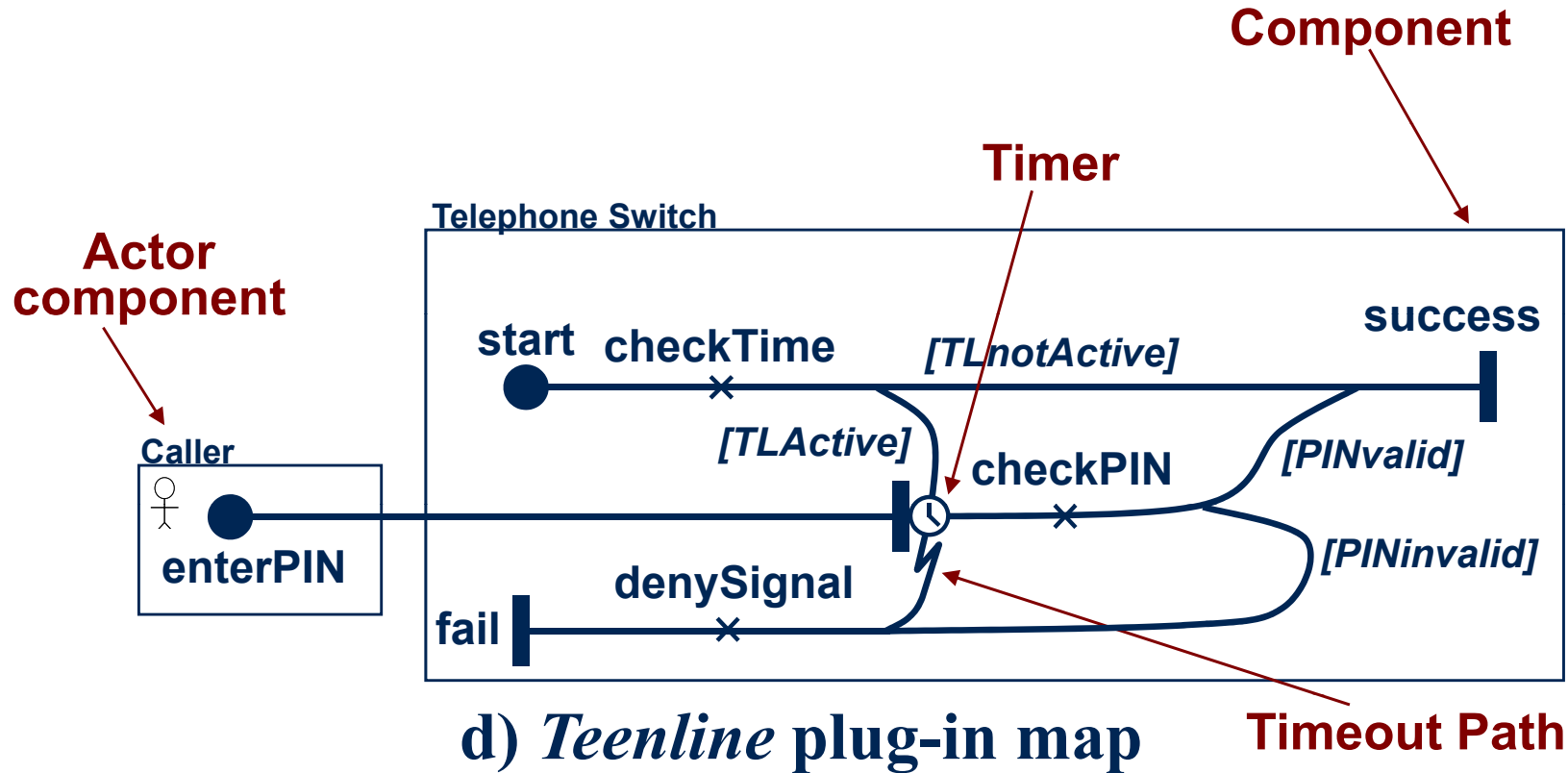
c) default plug-in map





# Use Case Maps: Notation (2)

## UCM Example: Tiny Telephone System

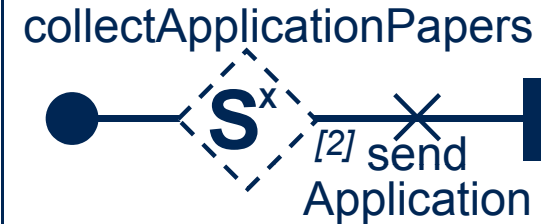
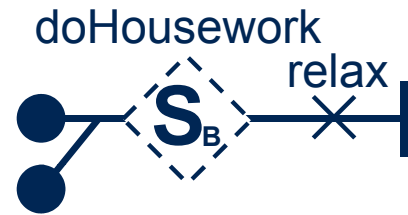
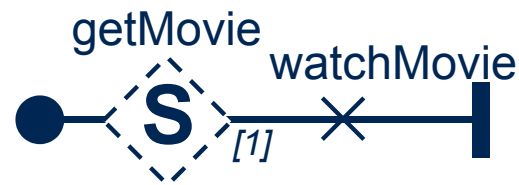


● waiting place

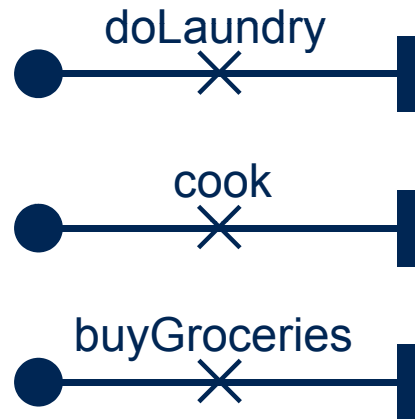
# Use Case Maps: Notation (3)

**RECENT**

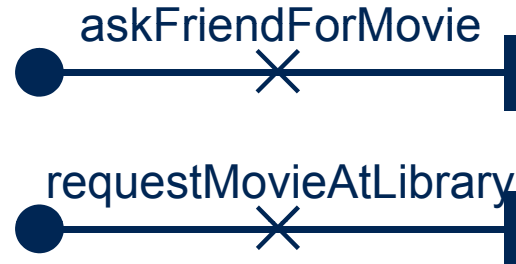
## UCM Examples: Advanced Types of Stubs



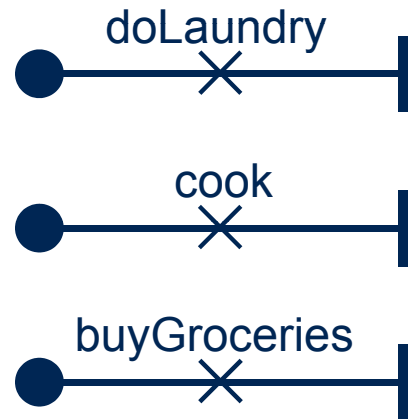
### Plug-in Maps:



### Plug-in Maps:



### Plug-in Maps:



### Plug-in Map: (three times)



**Synchronizing Stub  
with synchronization threshold**

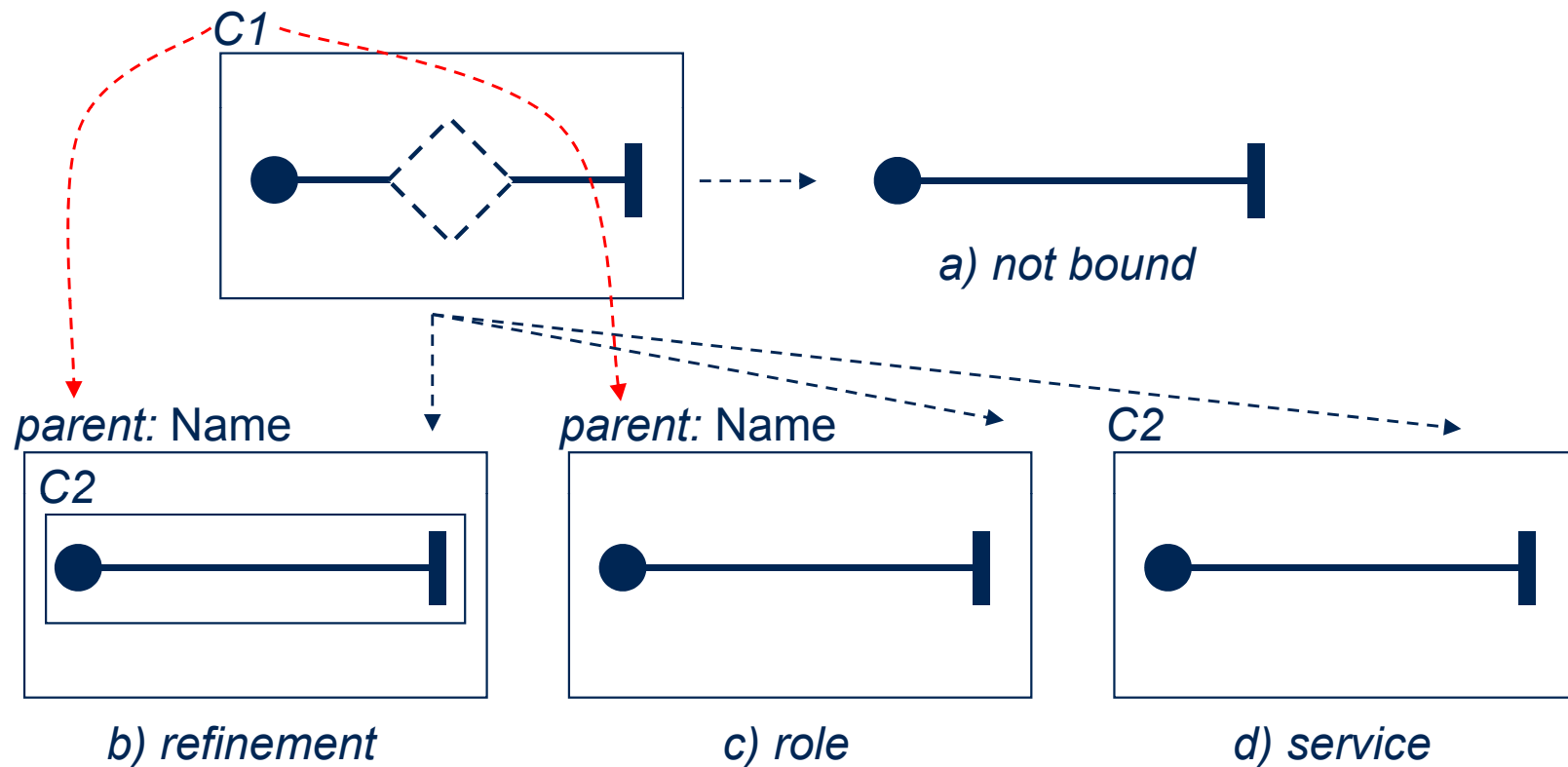


**Blocking Stub  
with replication indicator**

# Use Case Maps: Notation – Component Plug-in Bindings

RECENT

- Establishes a relationship of a component on a parent map with a component on a plug-in map (in addition to bindings of in/out-paths of a stub with start/end points on a plug-in map)

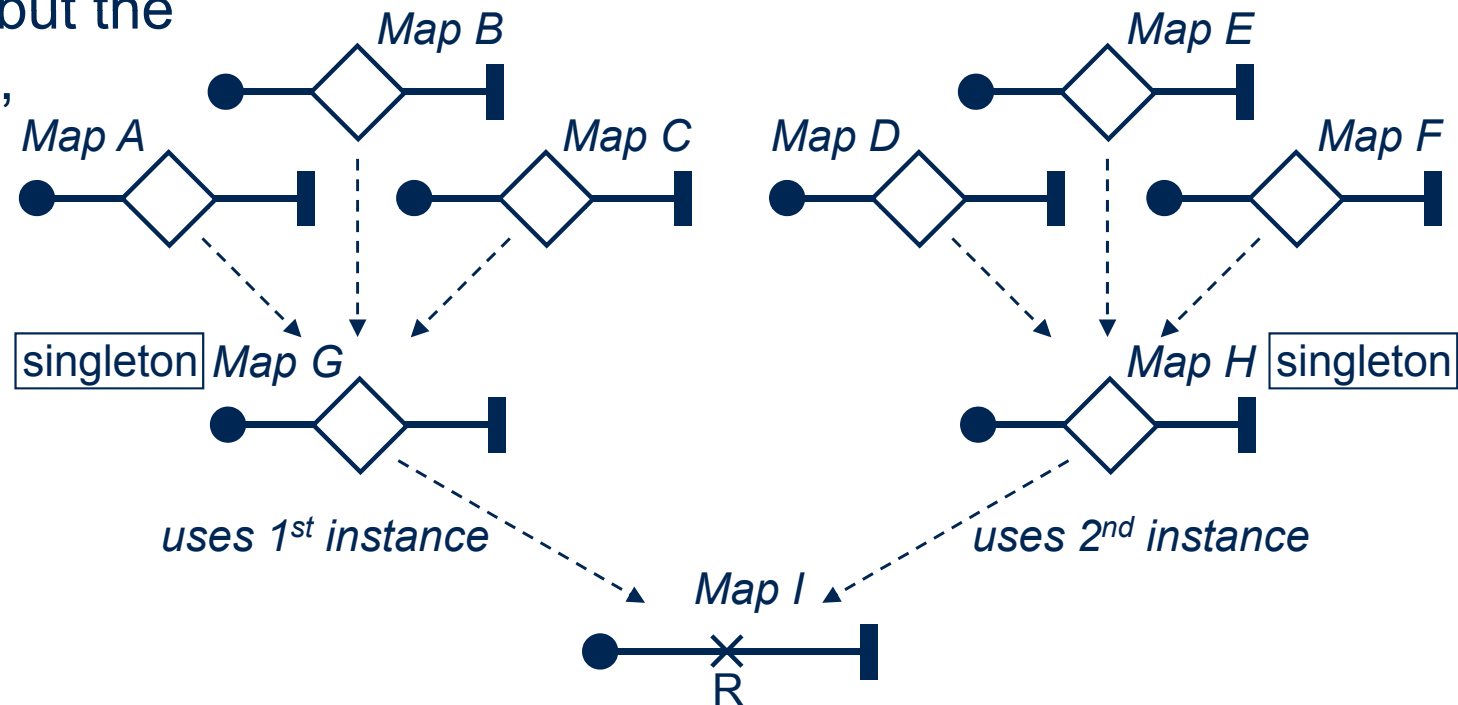


- Option c: the parent component plays a role e.g. in an architectural or behavioral pattern



# Use Case Maps: Notation – Singleton Maps

- Case 1: Map G is a singleton and therefore the stubs on Map A, B, and C use the same instance of Map G
- Case 2: Map I is not a singleton and therefore the stubs on Map G and Map H use different instances of Map I
- Case 3: A groups of stubs may want to use the same instance of a plug-in map but a different instance than other stubs. Achieved with an intermediate layer (e.g., the stubs on Map A, B, and C use the same instance of Map I, but the stubs on Map D, E, and F use a different instance of Map I)

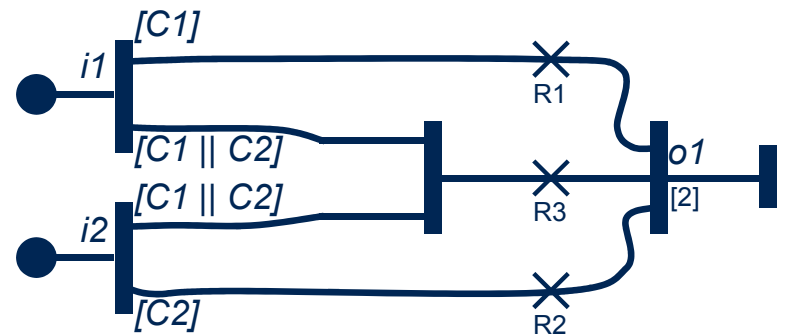
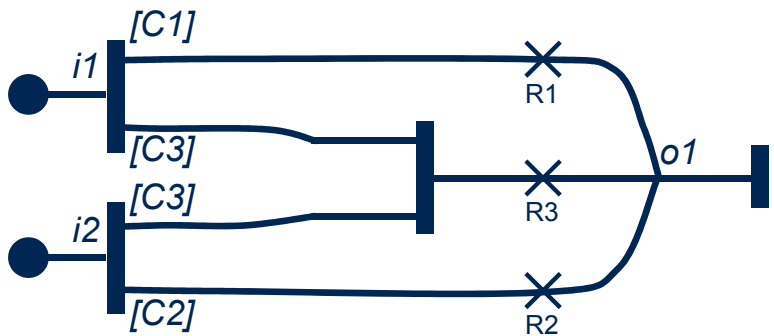
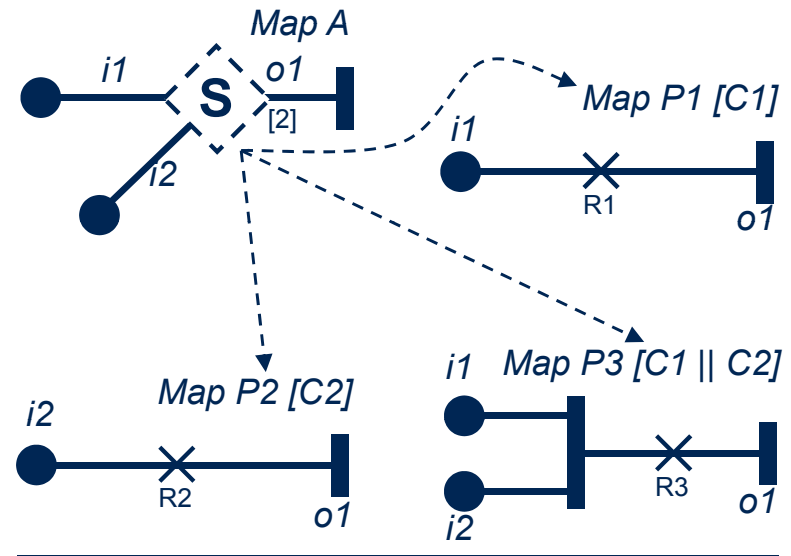
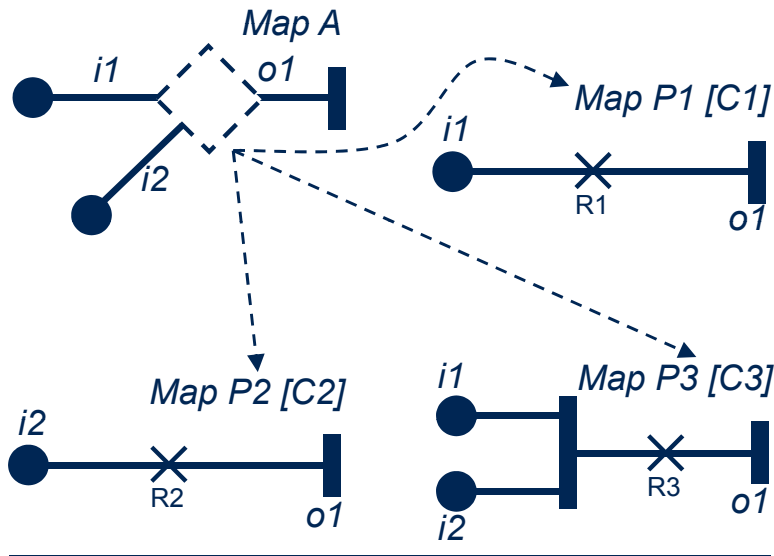


RECENT

# Use Case Maps: Semantics (1)



- OR-fork: exclusive OR
- AND-fork: always all paths
- Protected component: only one active path at a time
- Dynamic and synchronizing stubs:

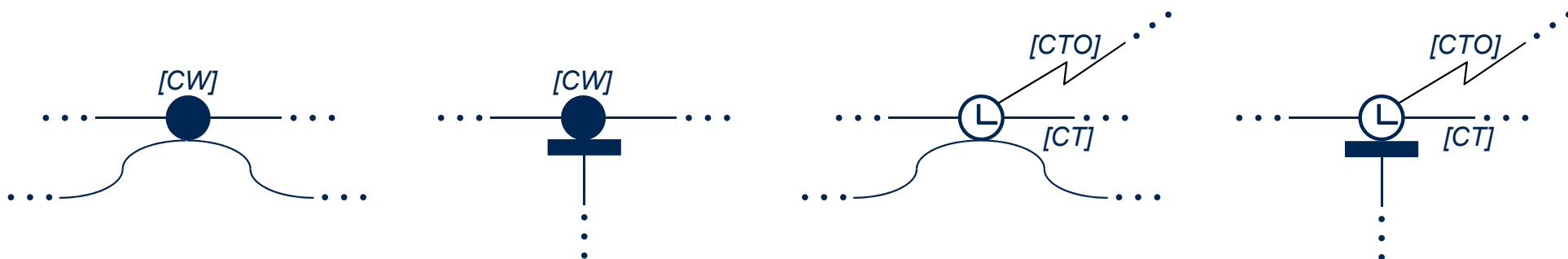




# Use Case Maps: Semantics (2)

RECENT

- Waiting place and timer
  - Transient = a trigger is counted only if a scenario is already waiting
  - Persistent = all triggers are counted (i.e., remembered)
- Waiting place
  - Scenario is allowed to continue if  $CW = \text{true}$  or the trigger counter  $> 0$
  - Warning is issued if  $CW = \text{false}$  and the trigger counter = 0
- Timer: Scenario is allowed to continue along ...
  - Regular path if  $CT = \text{true}$
  - Regular path if  $CT = \text{false}$  and  $CTO = \text{false}$  and trigger counter  $> 0$
  - Timeout path if  $CT = \text{false}$  and  $CTO = \text{true}$
  - Timeout path if  $CT = \text{false}$  and  $CTO = \text{false}$  and trigger counter = 0



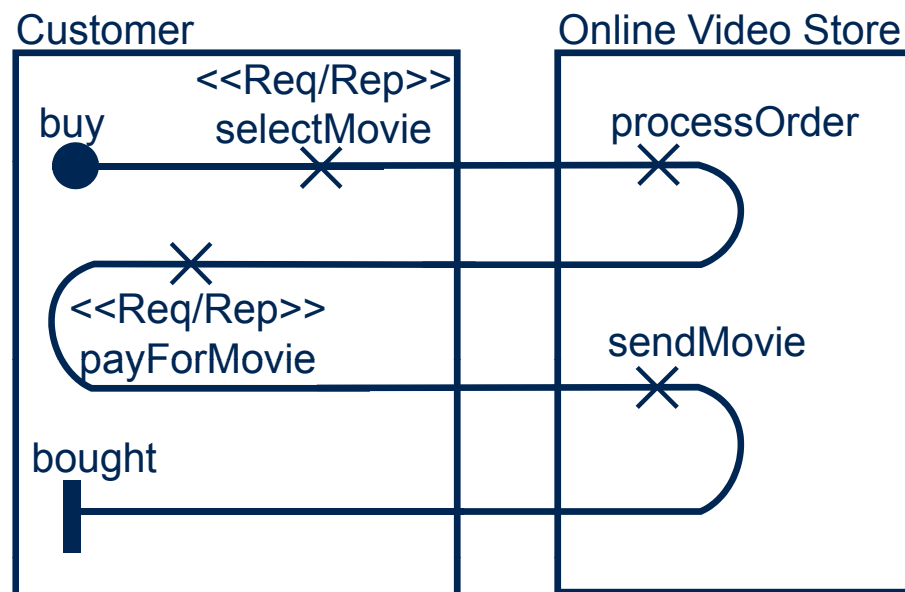




# Beyond the Basic Notations of URN

# Extensibility: Metadata

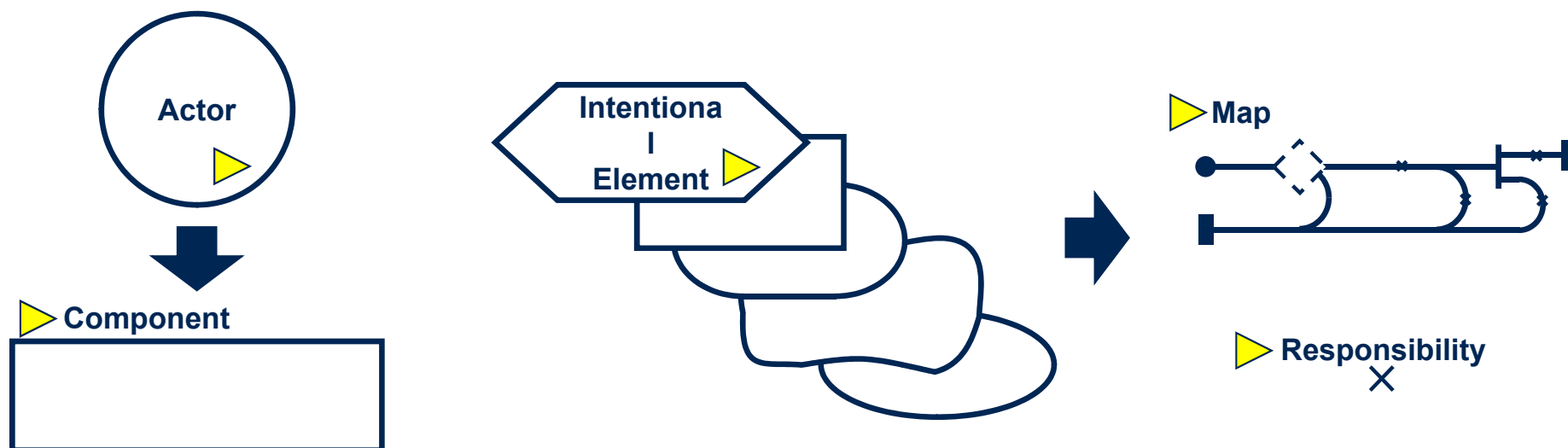
- URN can be **extended** with metadata (name/value pairs)
  - Can be attached to any URN model element and exploited by specialized tools



- For example, <<Req/Rep>> indicates that the communication between the Customer and the Online Video Store adheres to the Request/Reply pattern
- The metadata definition (name = value) is: Communication = Req/Rep

# Traceability: URN Links

- URN allows **traceability** relationships to be established with URN links
  - Typed links
  - Connect any pair of URN model elements
  - Most frequently, URN links are used to trace ...
    - Actors in GRL models to components in UCM models
    - Tasks in GRL models to maps or responsibilities in UCM models
- jUCMNav also allows other intentional elements to be linked to responsibilities or maps - other links are currently restricted in the tool



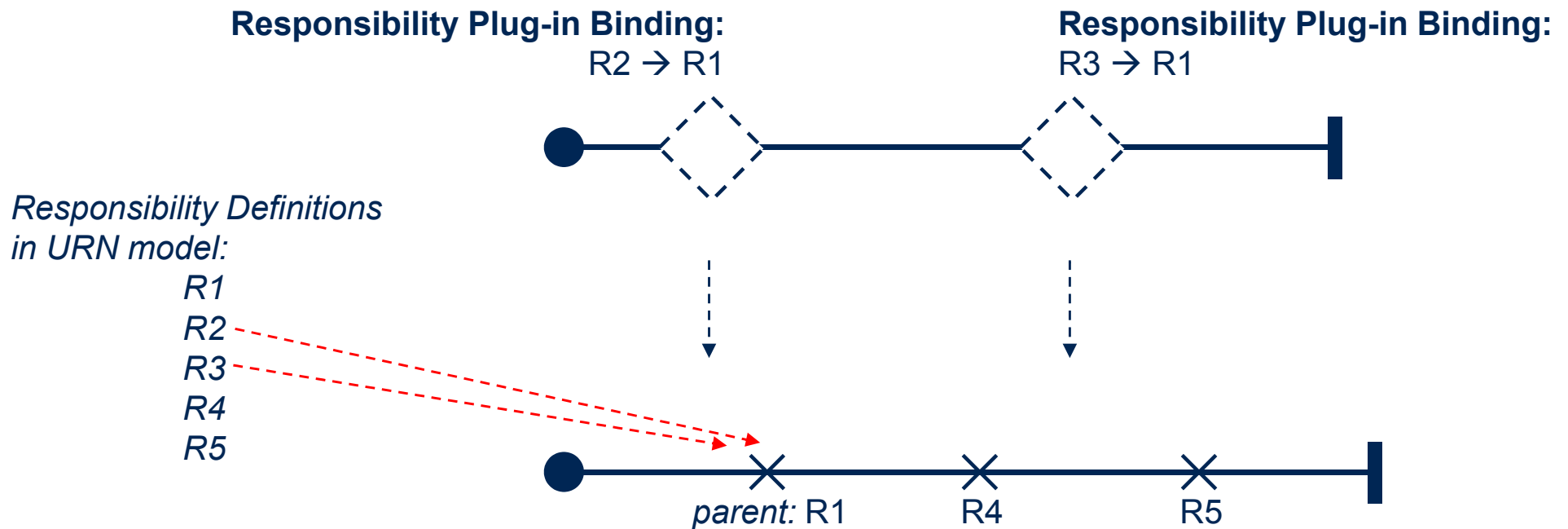
URN link: ►



# Use Case Maps: Notation – Responsibility Plug-in Bindings

- Establishes a relationship of a responsibility in the URN model with a responsibility on a plug-in map (in addition to bindings of in/out-paths of a stub with start/end points on a plug-in map and in addition to component plug-in bindings)

**EXTENSION**



# Use Case Maps: Notation – Failures and Aborts (1)

- **Explicit** approach with Failure Points

- Indicates location of failure on scenario path
- Failure condition
- Sets failure variable to indicate which failure occurred

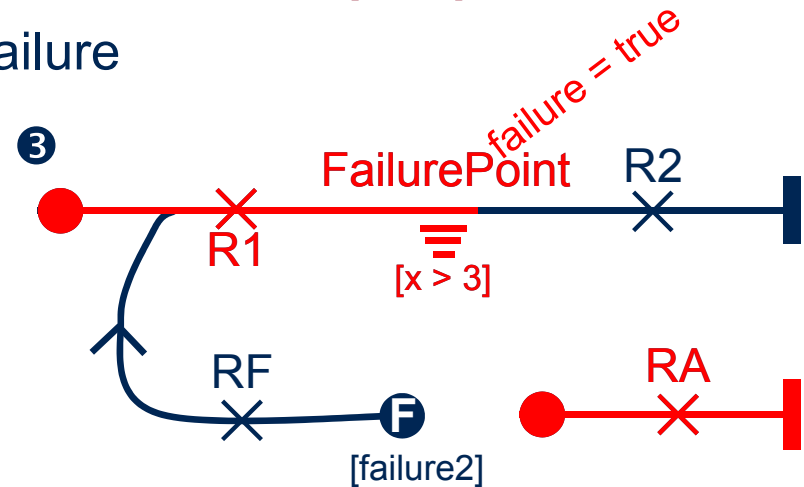
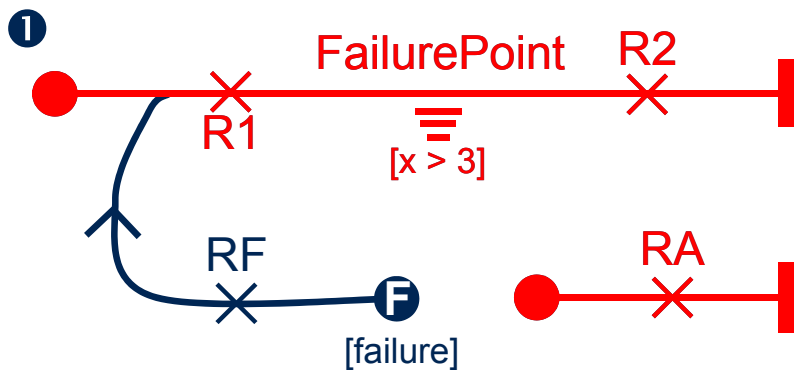
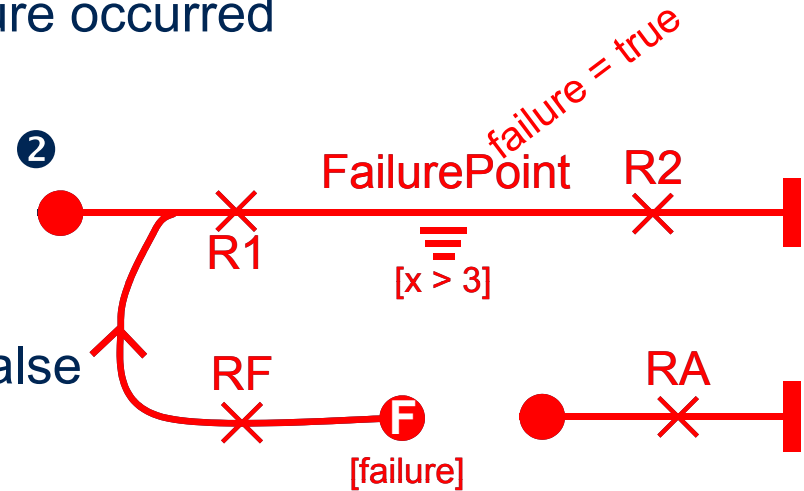
*extension*

- Failure Start Point

- Activated if guard evaluates to true

- Examples

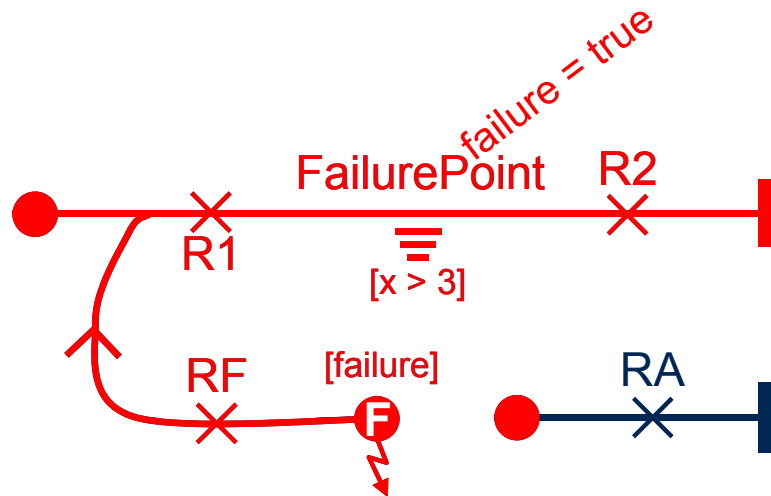
- RF sets X to 0; failure = false; failure2 = false
- ❶ ... x = 1; ❷ ... x = 5; ❸ ... x = 5;
- The path with RA is not impacted by the failure



# Use Case Maps: Notation – Failures and Aborts (2)

- Abort Start Points
  - Activated if guard evaluates to true
  - Aborts all other paths in the abort scope (all concurrent branches that are active on the same or lower level maps)

*EXTENSION*





# Use Case Maps: Notation – Failures and Aborts (3)

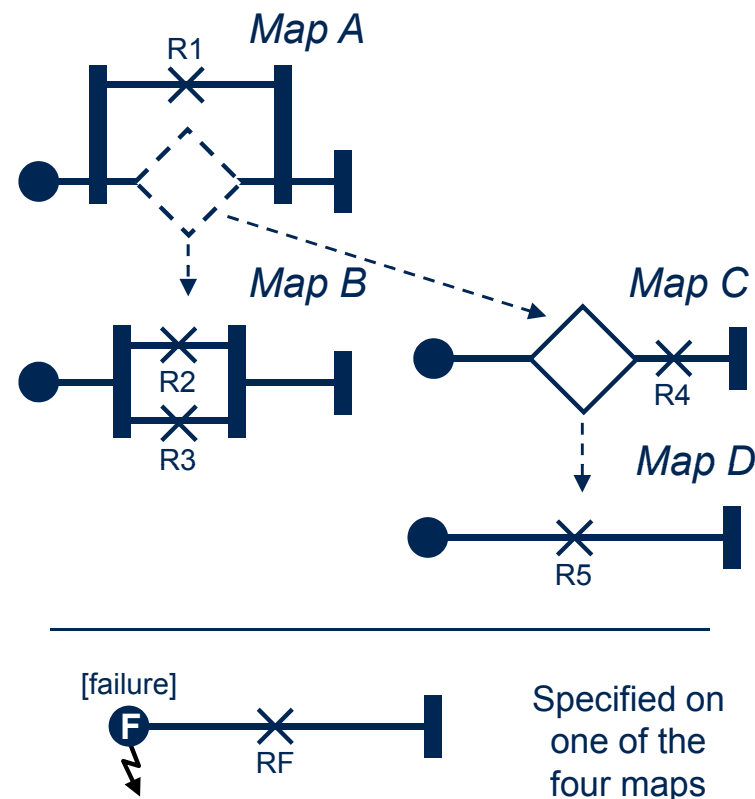
*EXTENSION*

- **Implicit** approach without Failure Points

- Location and failure variable defined by scenario definition
- Greater flexibility in defining location (path node, map, component reference, component/responsibility definition)

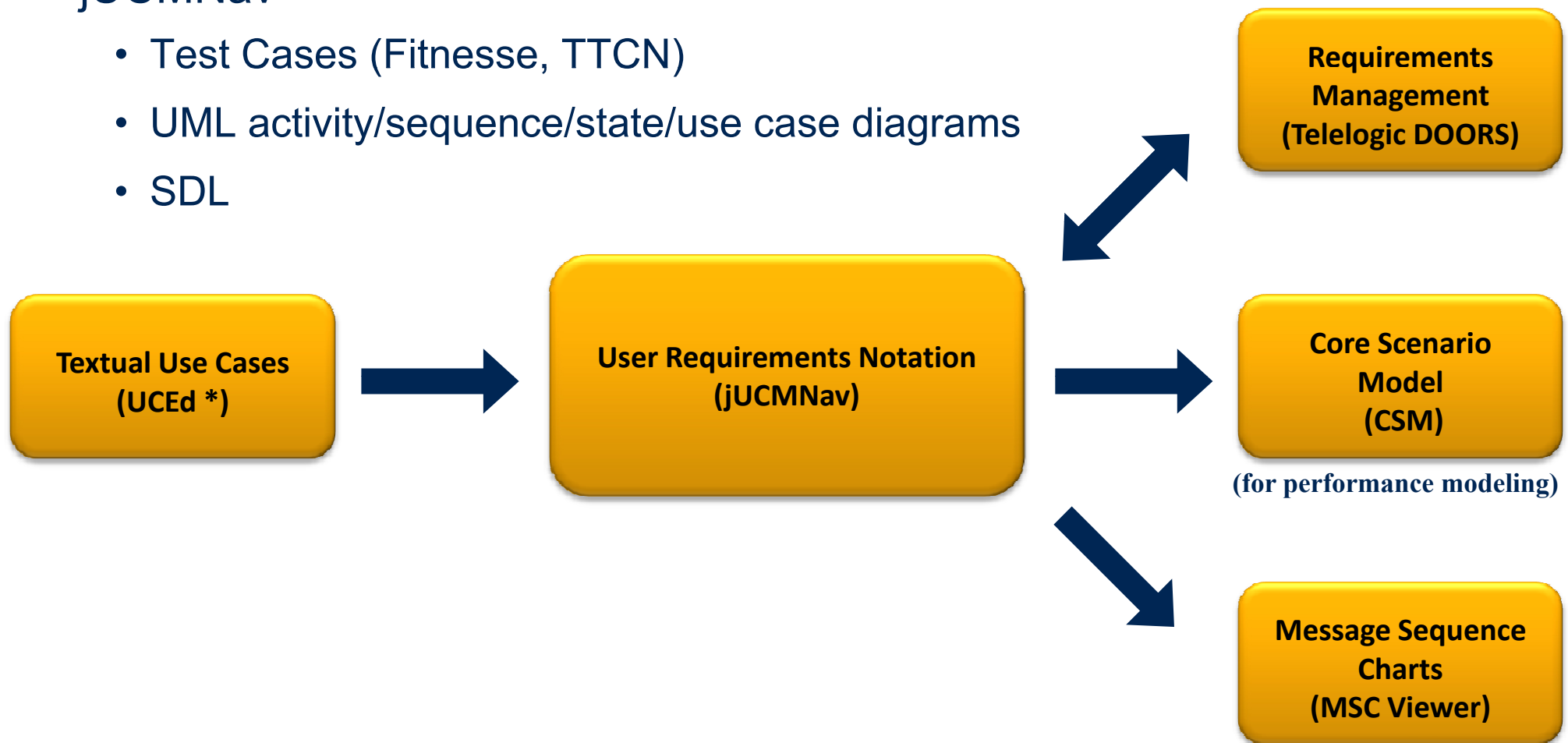
- **Example**

- Failure occurs above map with the abort path → not caught, scenario terminates with error
- Failure occurs on or below map with abort path
  - Scenario ends with end point of abort path (unless end point is bound to out-path of a stub)
  - Abort path specified on map A → all paths on all maps are aborted
  - Specified on B → only the two concurrent paths on B are aborted
  - Specified on C → all paths on C and D are aborted; Specified on D → only the path on D is aborted



# User Requirements Notation: Transformations

- Transformations connect URN models to other models
- Some transformations discussed in publications or implemented in jUCMNav's predecessor UCMNav have not yet been implemented in jUCMNav
  - Test Cases (Fitsesse, TTCN)
  - UML activity/sequence/state/use case diagrams
  - SDL



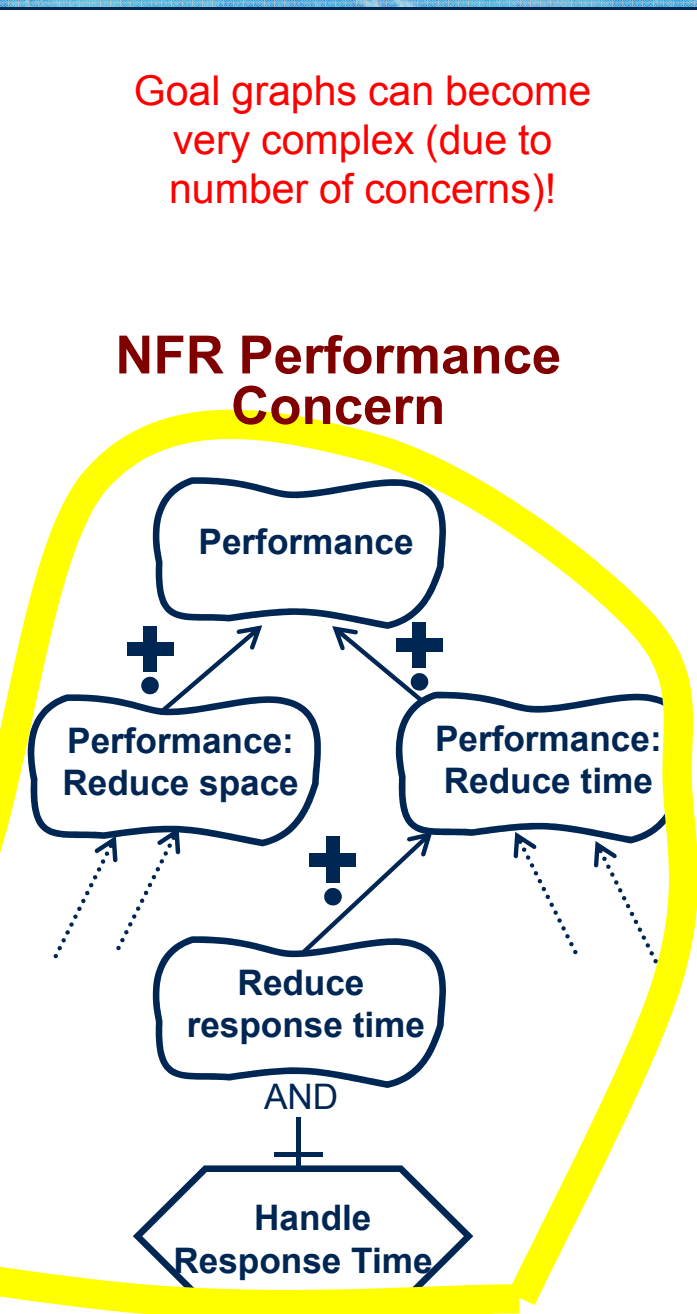
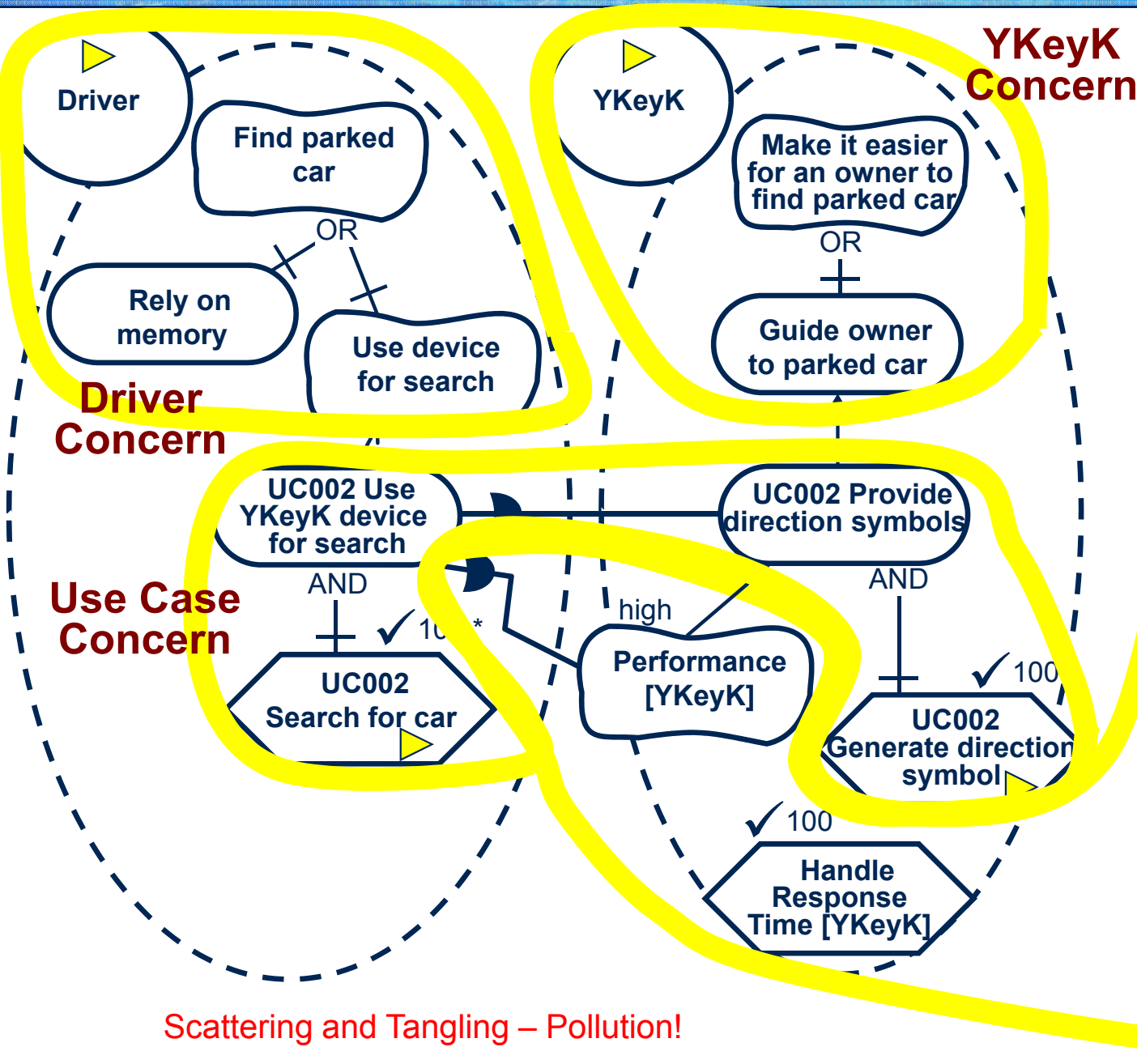
\* Stéphane Somé's UCed: <https://sourceforge.net/projects/uced/>



# Motivating Example

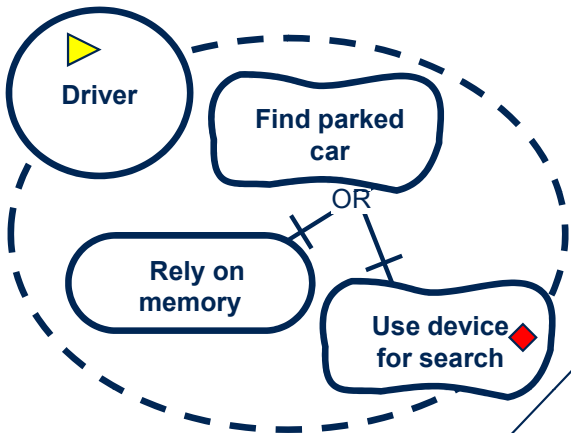


# Goal-oriented Requirement Language: Concerns

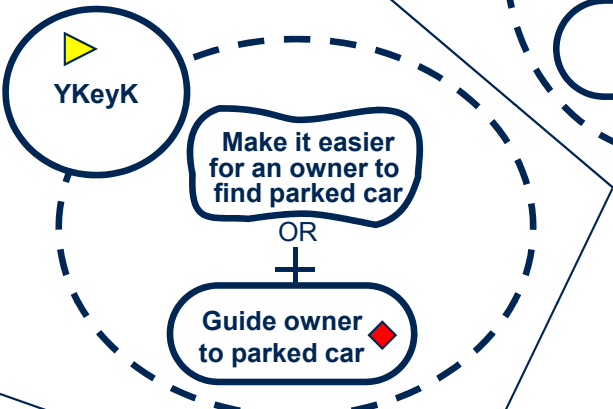


# Aspect-Oriented GRL (1)

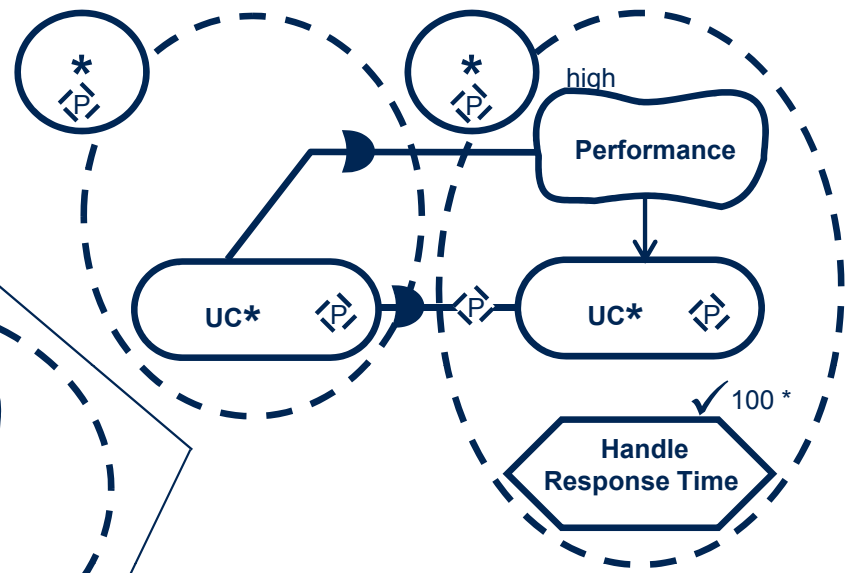
DRIVER



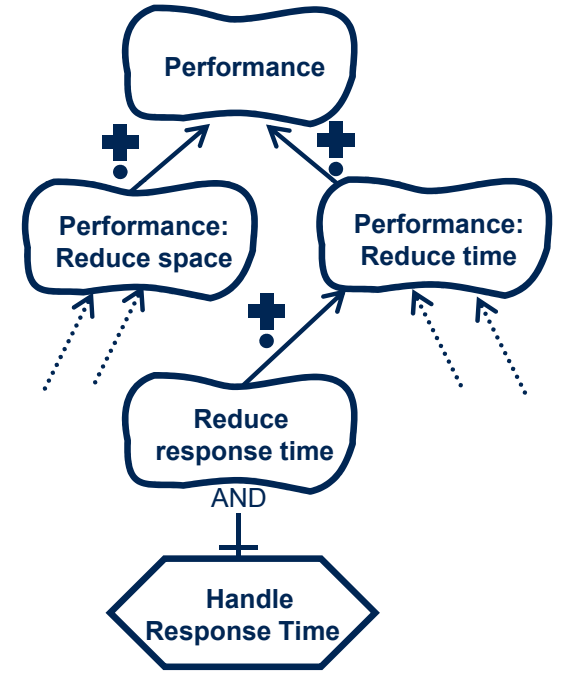
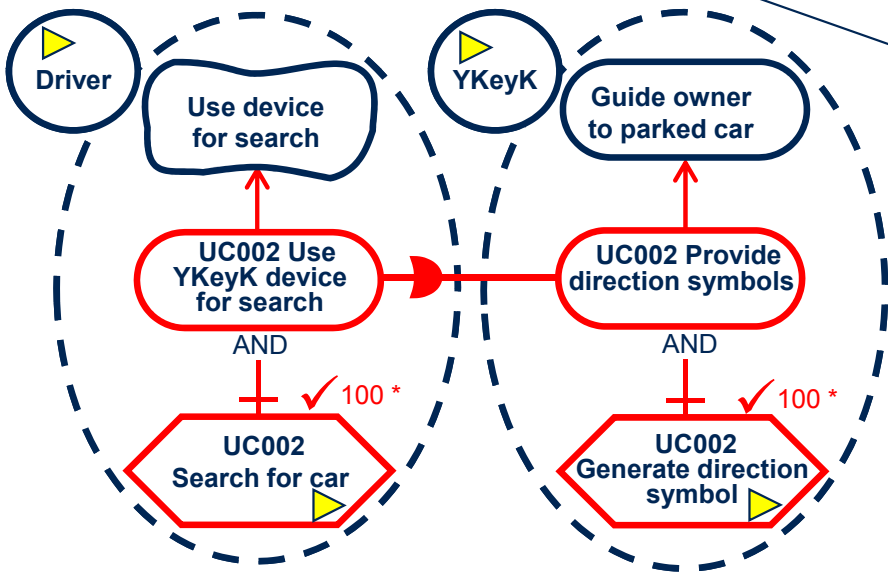
YKEYK



PERFORMANCE



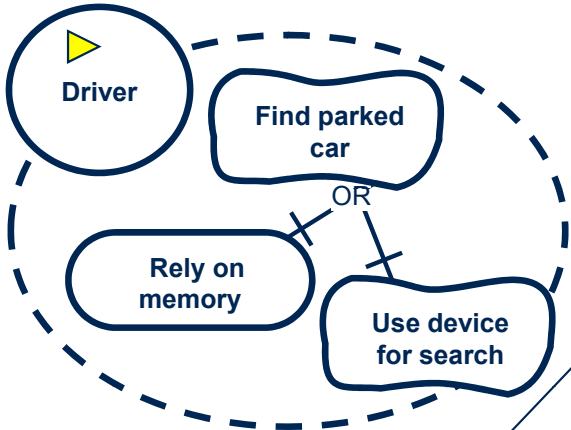
USE CASE



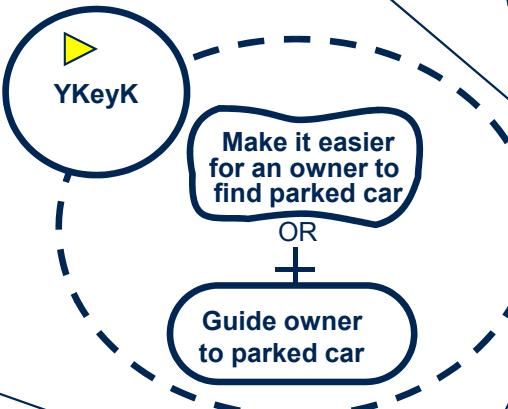


# Aspect-Oriented GRL (2)

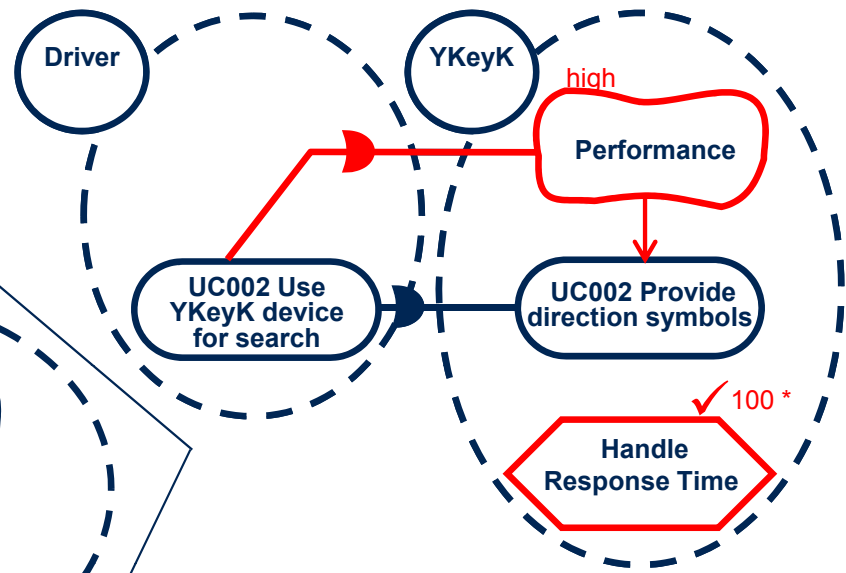
DRIVER



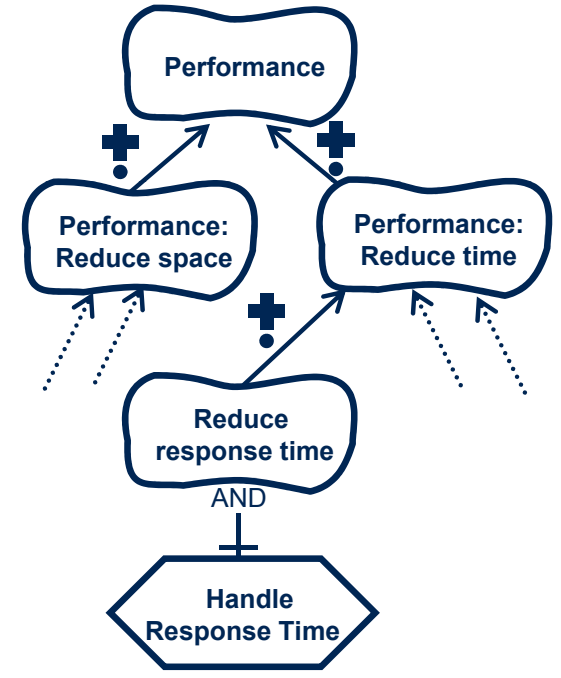
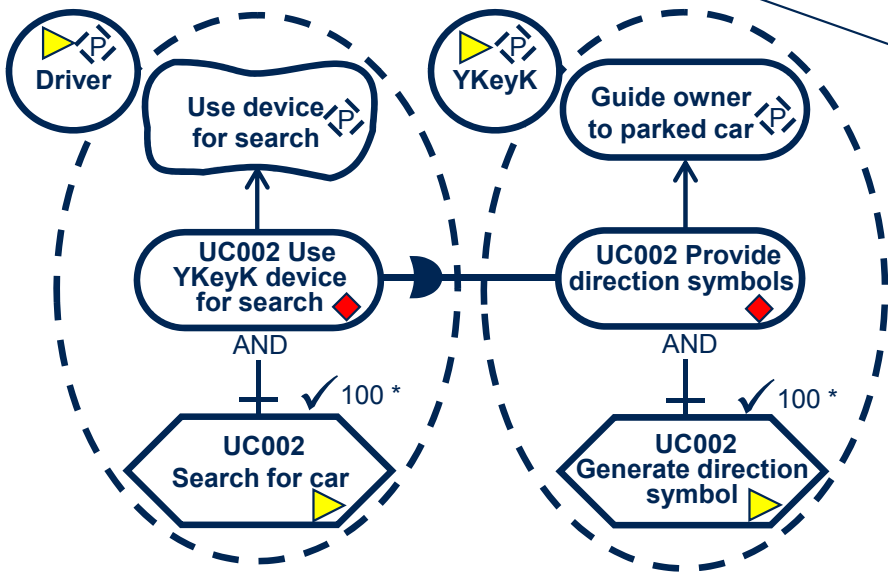
YKEYK



PERFORMANCE



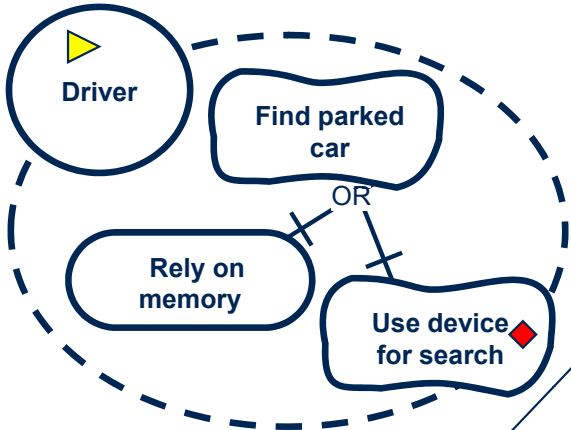
USE CASE



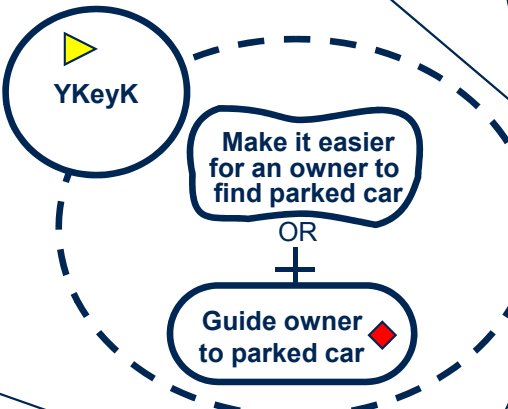


# Aspect-Oriented GRL (3)

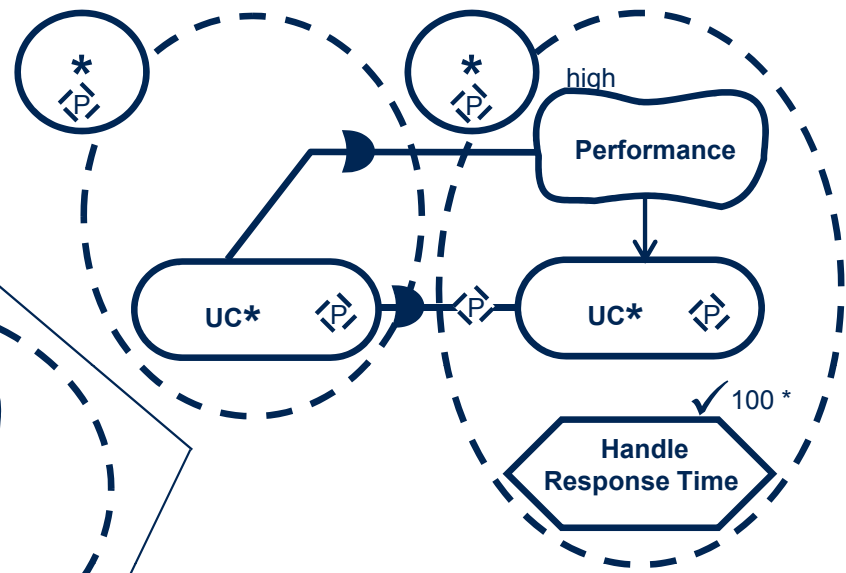
DRIVER



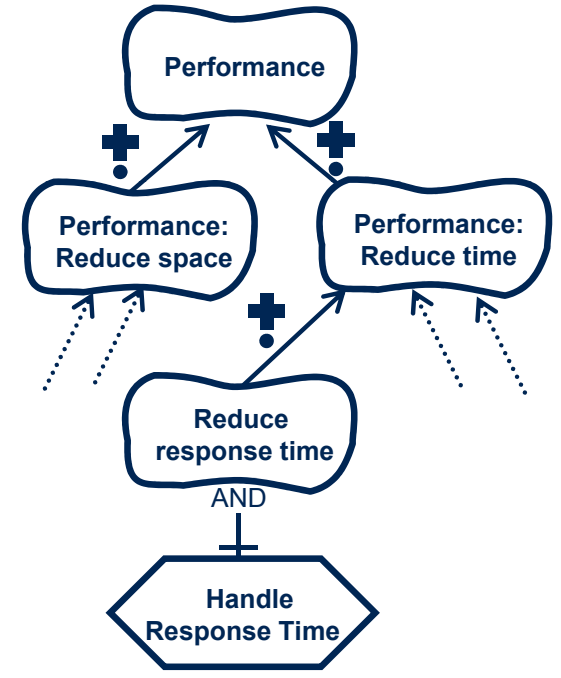
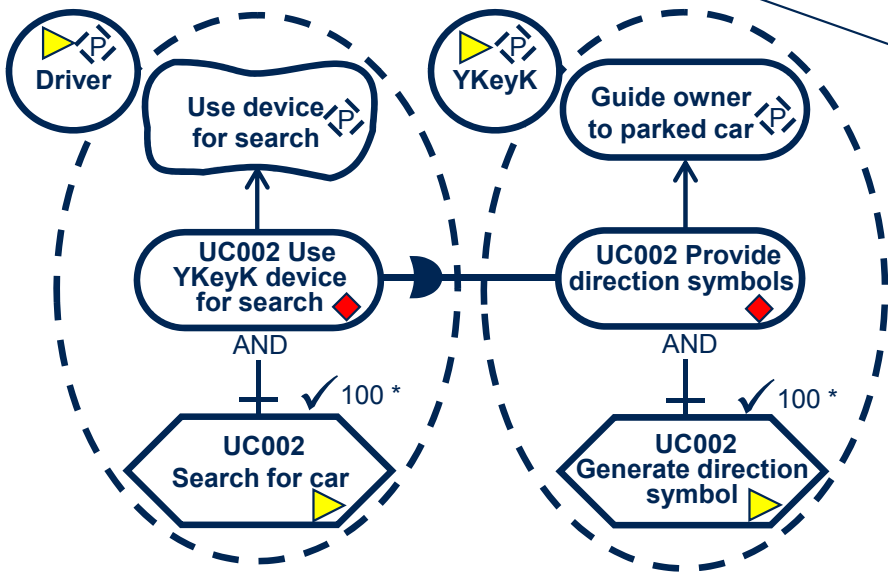
YKEYK



PERFORMANCE



USE CASE



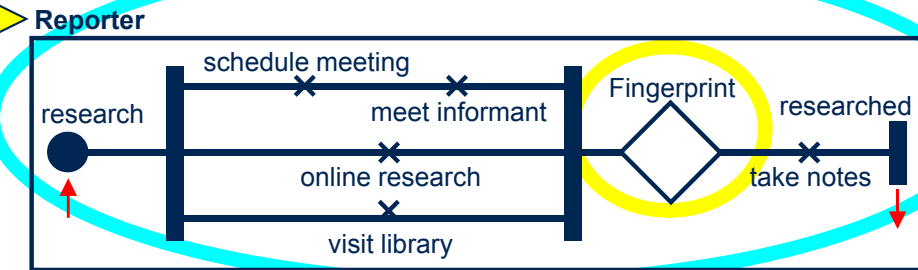
# Use Case Maps: Concerns

## Root UCM



Maps can become very complex (due to number of NFRs, features, and fail cases)!

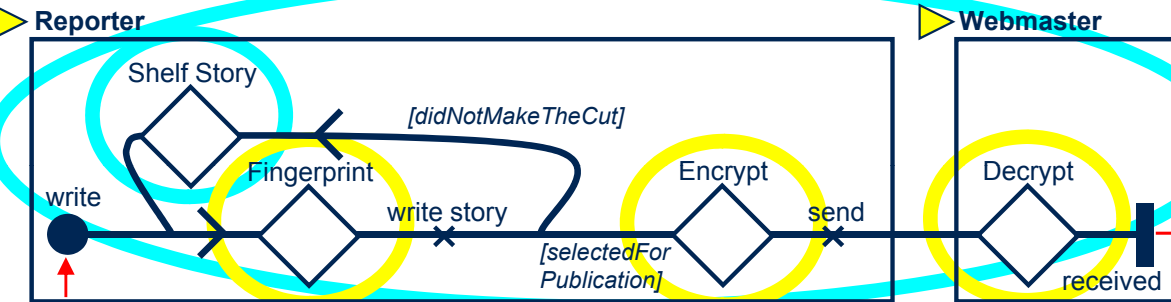
## Research Story UCM



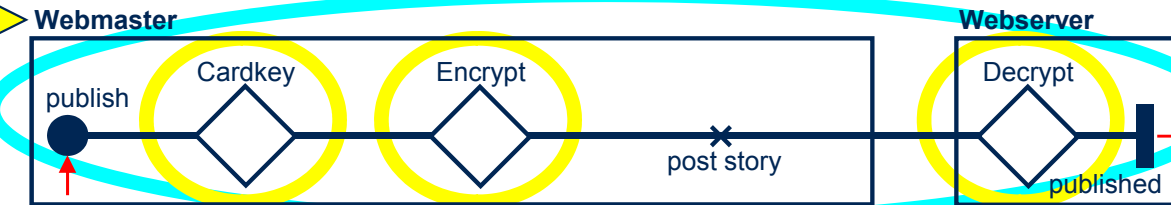
Use Case Concerns

NFR Security Concerns

## Write Story UCM

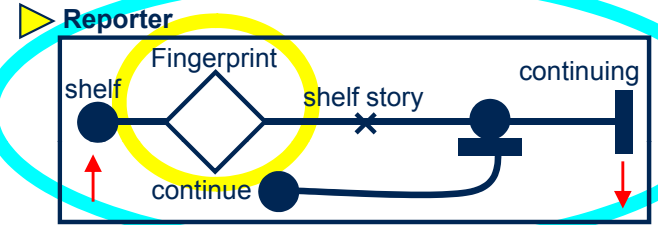


## Publish Story UCM

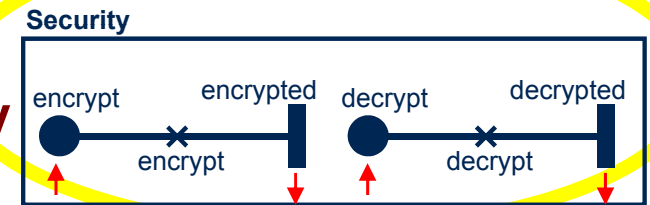


Scattering and Tangling – Pollution!

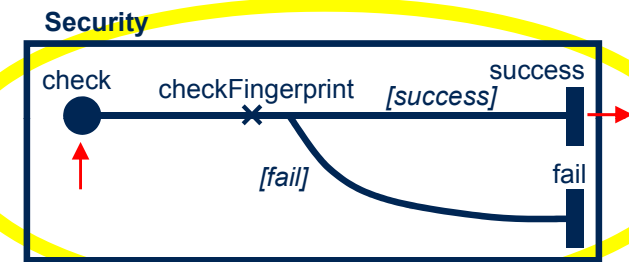
## Shelf Story UCM



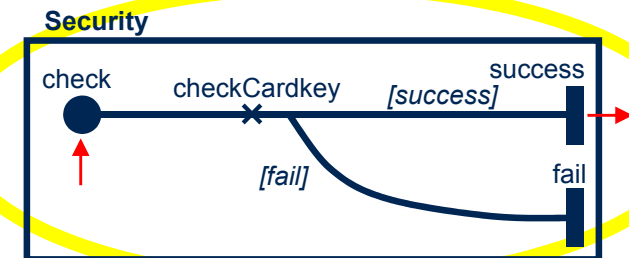
## Encrypt/Decrypt UCM



## Authentication UCM: Fingerprint



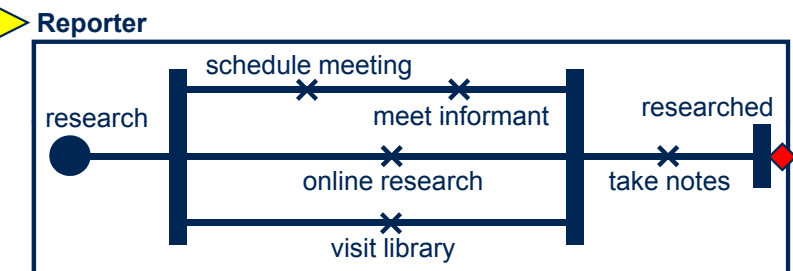
## Authentication UCM: Cardkey



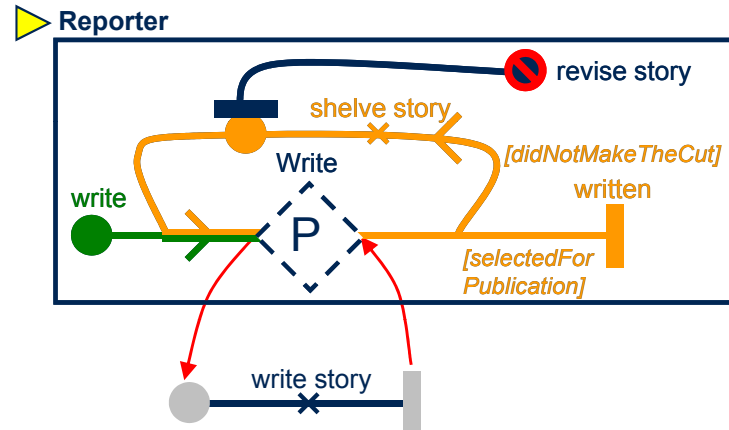


# Aspect-Oriented Use Case Maps (1)

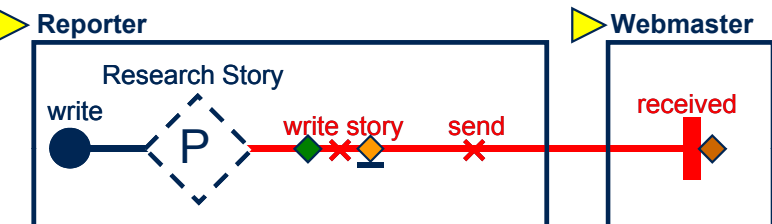
## Research Story UCM



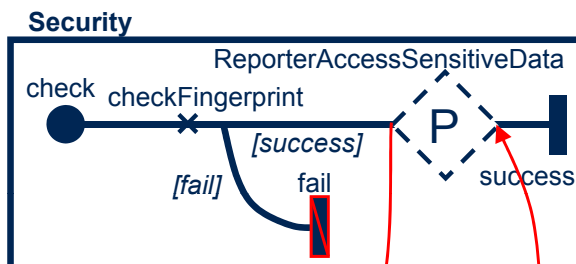
## Shelve Story AoUCM



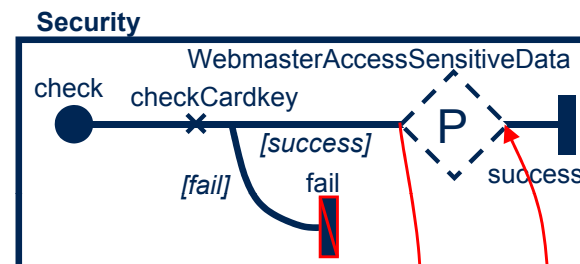
## Write Story AoUCM



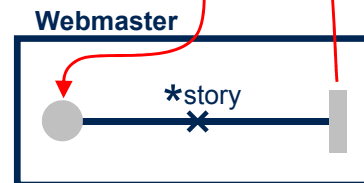
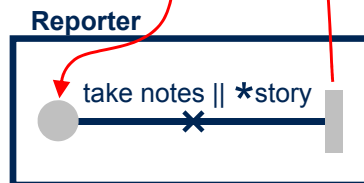
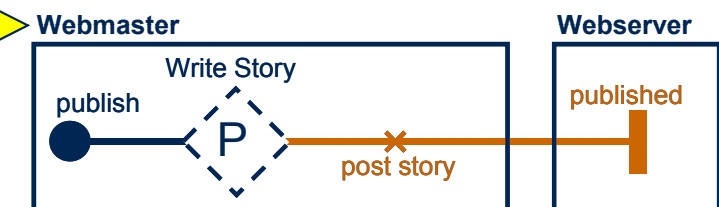
## Authentication AoUCM: Fingerprint



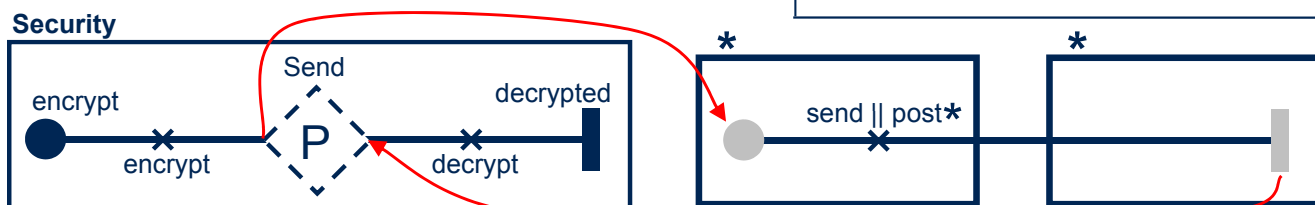
## Authentication AoUCM: Cardkey



## Publish Story AoUCM



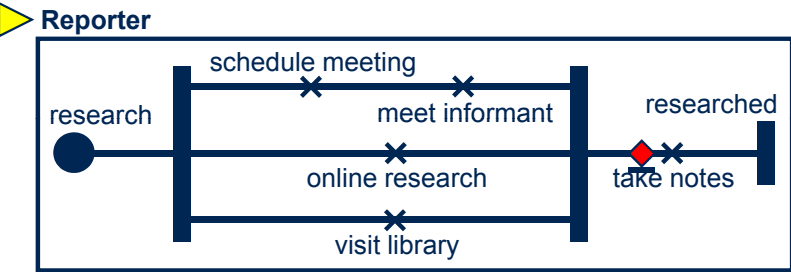
## Encrypt/Decrypt AoUCM



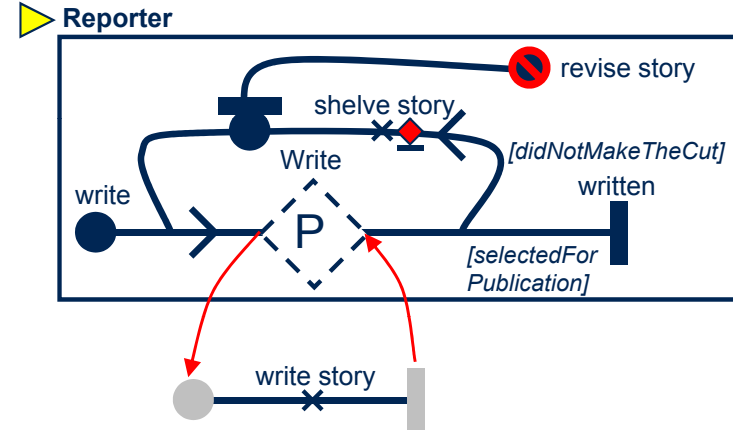


# Aspect-Oriented Use Case Maps (2)

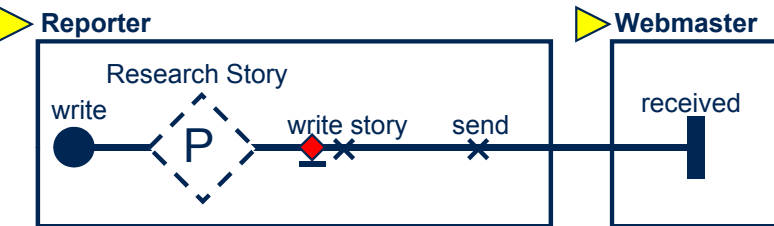
## Research Story UCM



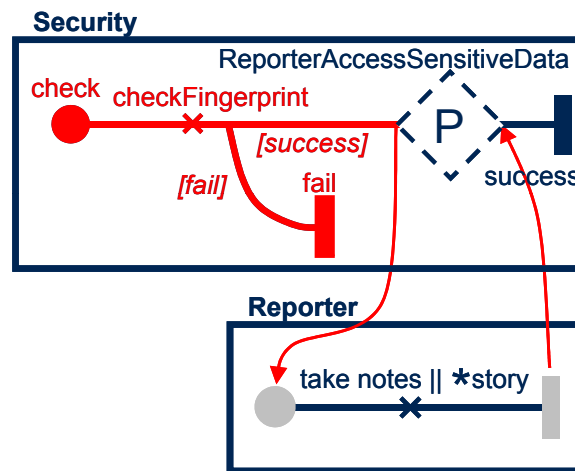
## Shelve Story AoUCM



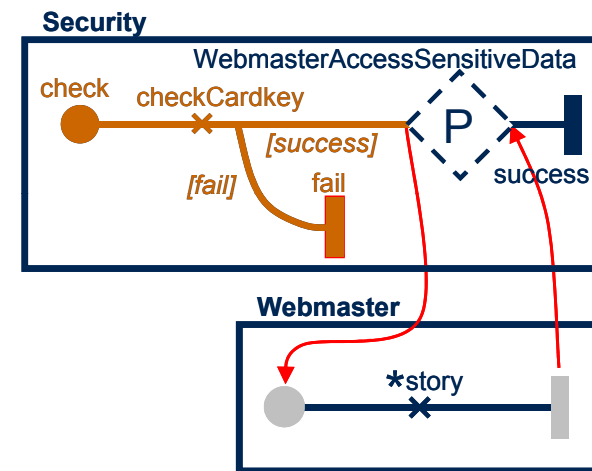
## Write Story AoUCM



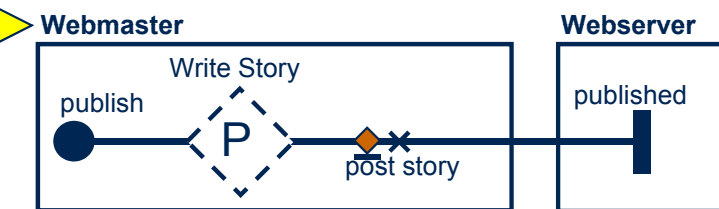
## Authentication AoUCM: Fingerprint



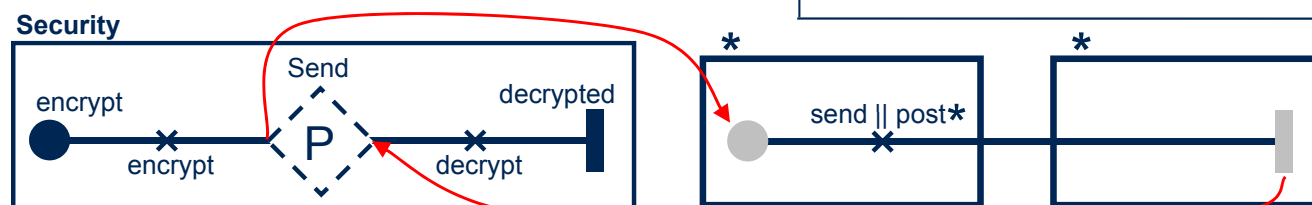
## Authentication AoUCM: Cardkey



## Publish Story AoUCM

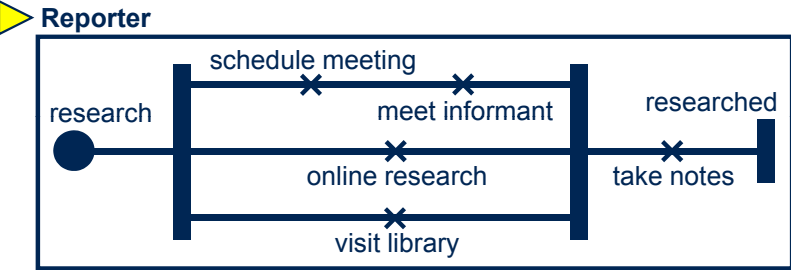


## Encrypt/Decrypt AoUCM

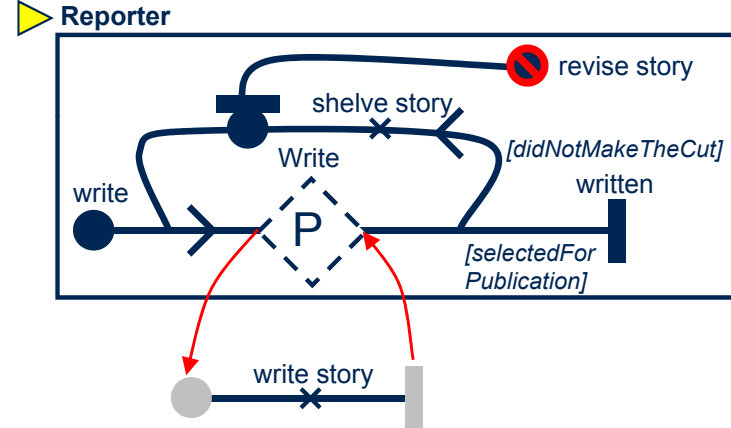


# Aspect-Oriented Use Case Maps (3)

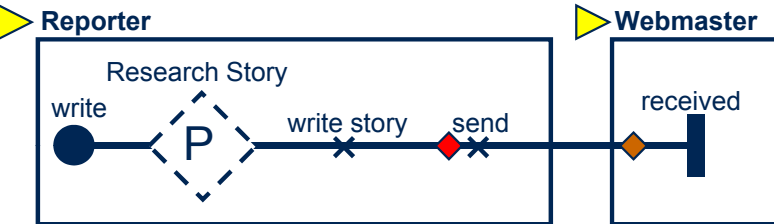
## Research Story UCM



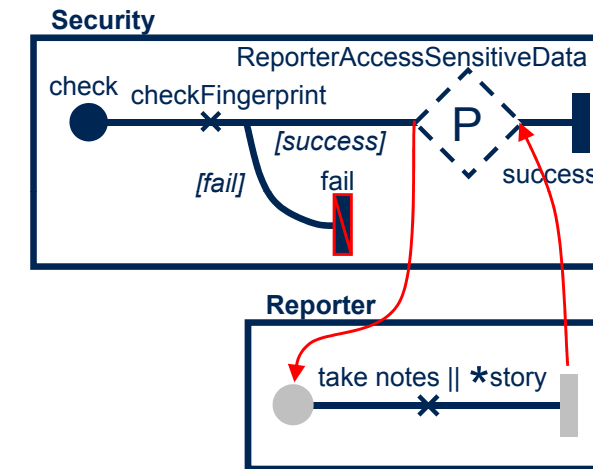
## Shelve Story AoUCM



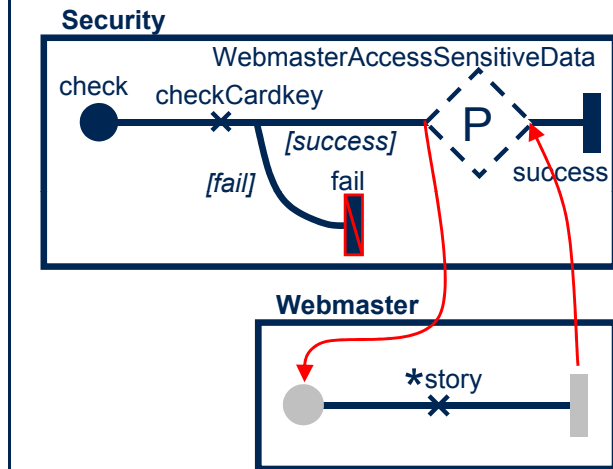
## Write Story AoUCM



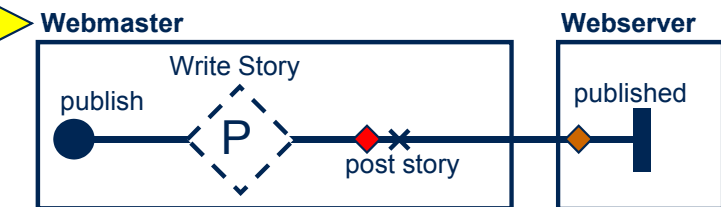
## Authentication AoUCM: Fingerprint



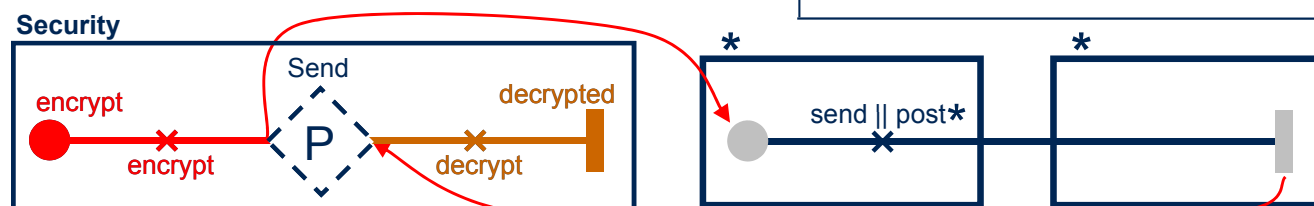
## Authentication AoUCM: Cardkey



## Publish Story AoUCM



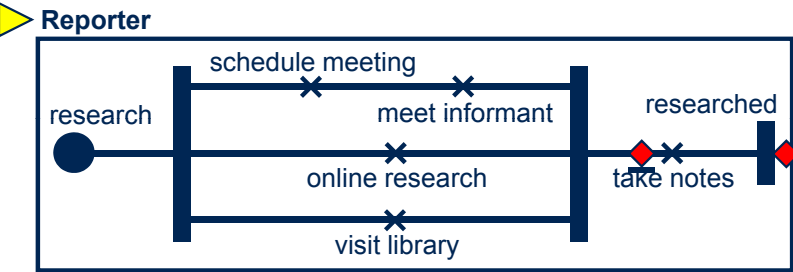
## Encrypt/Decrypt AoUCM



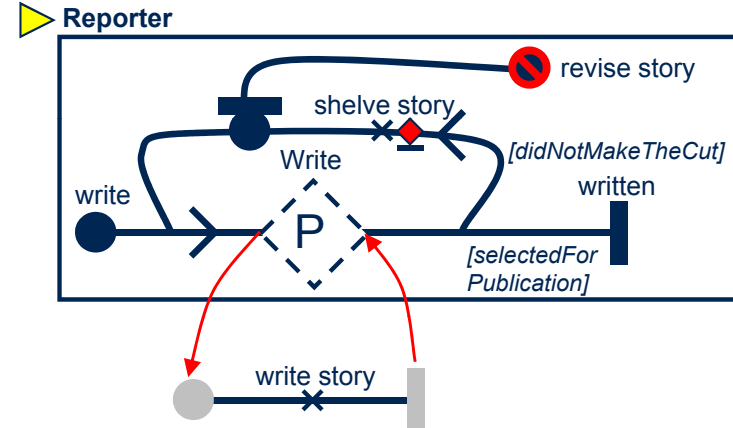


# Aspect-Oriented Use Case Maps (4)

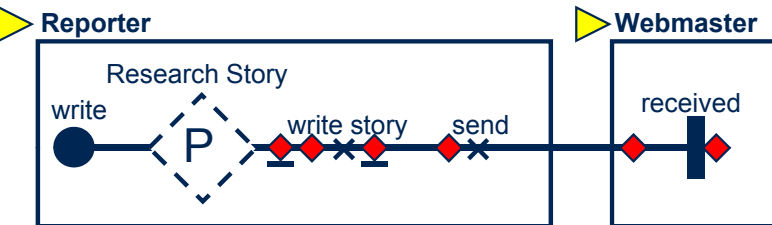
## Research Story UCM



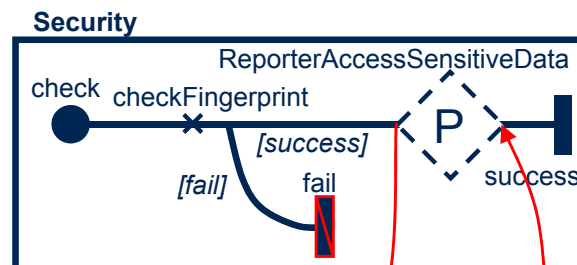
## Shelve Story AoUCM



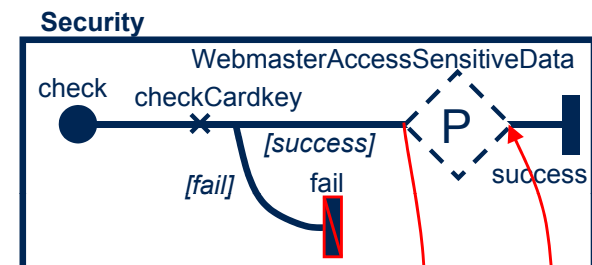
## Write Story AoUCM



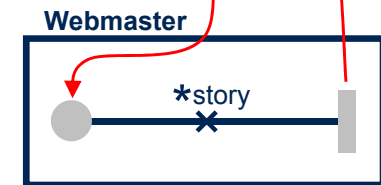
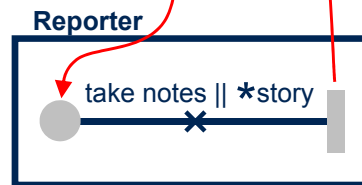
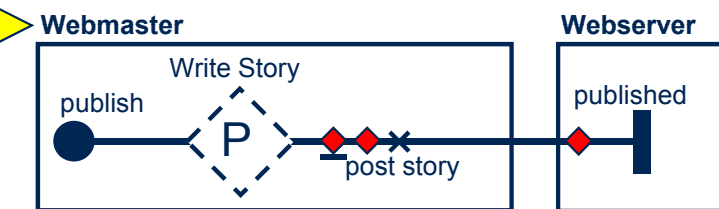
## Authentication AoUCM: Fingerprint



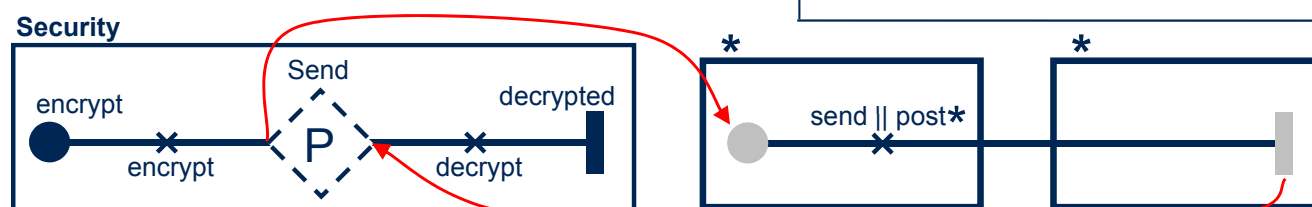
## Authentication AoUCM: Cardkey



## Publish Story AoUCM



## Encrypt/Decrypt AoUCM



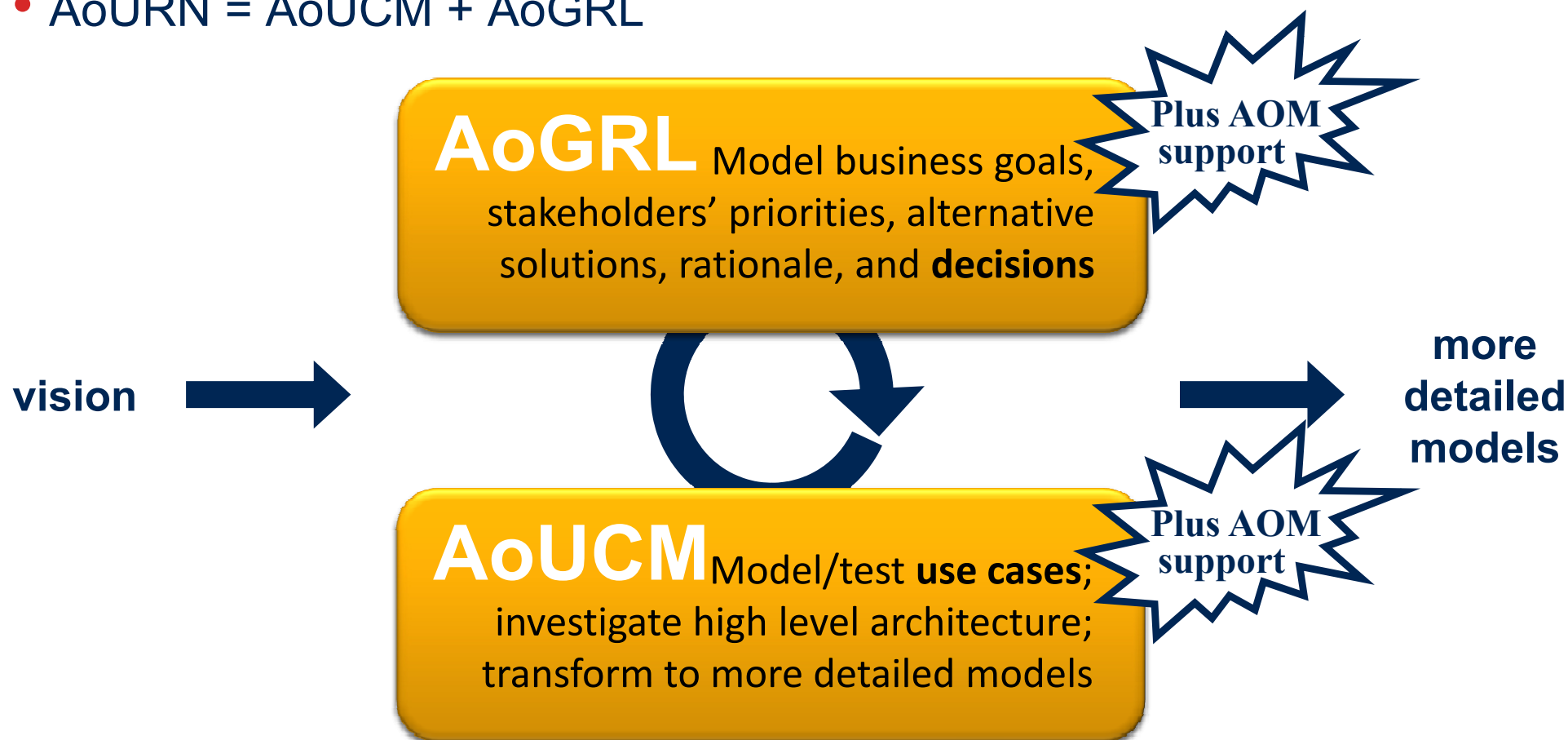


# Motivation

- Aspects have the potential to significantly influence the future of software development like no other software engineering concept since the emergence of object-oriented techniques
- The User Requirements Notation (URN) is the **first** and **currently only** standard (ITU-T Z.151) which explicitly combines goals (non-functional requirements – GRL) with scenarios (functional requirements – UCMs)
- Aspects can **improve** the modularity, reusability, scalability, maintainability and other properties of URN models
- Aspects can help **bridge** the gap between goals and operational scenarios
- Aspects can benefit from a **standardized** way of modeling NFRs and use cases with URN, considering the strong overlap between ...
  - Non-functional requirements (NFRs) and concerns encapsulated by aspects as well as use cases / stakeholder goals and concerns encapsulated by aspects

# Objective

- The objective is to deliver a notation and process which **unify** URN concepts and aspect concepts in one framework in order to encapsulate concerns from the **early requirements** stage in the software development life cycle
- $\text{AoURN} = \text{AoUCM} + \text{AoGRL}$





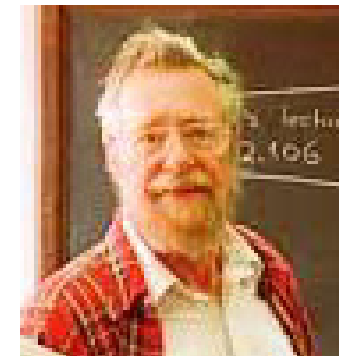
# Introduction to Aspect-oriented Requirements Engineering (AORE)



# Separation of Concerns

This is what I mean by focusing one's attention upon a certain aspect; it does not mean completely ignoring the other ones, but temporarily forgetting them to the extent that they are irrelevant for the current topic.

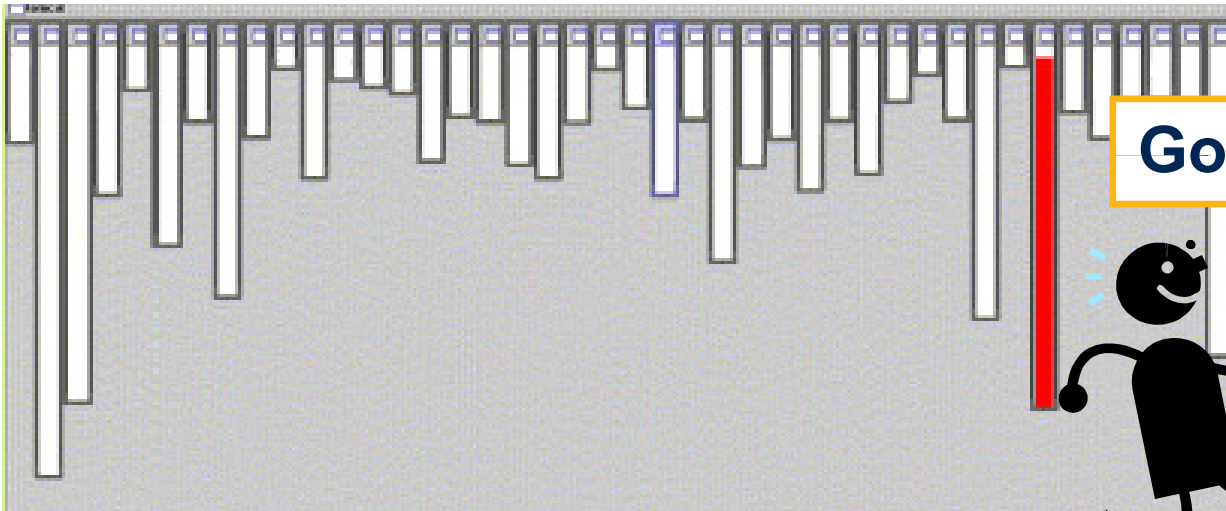
Such a separation, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts that I know of. I usually refer to it as **"a separation of concerns" [...]."**



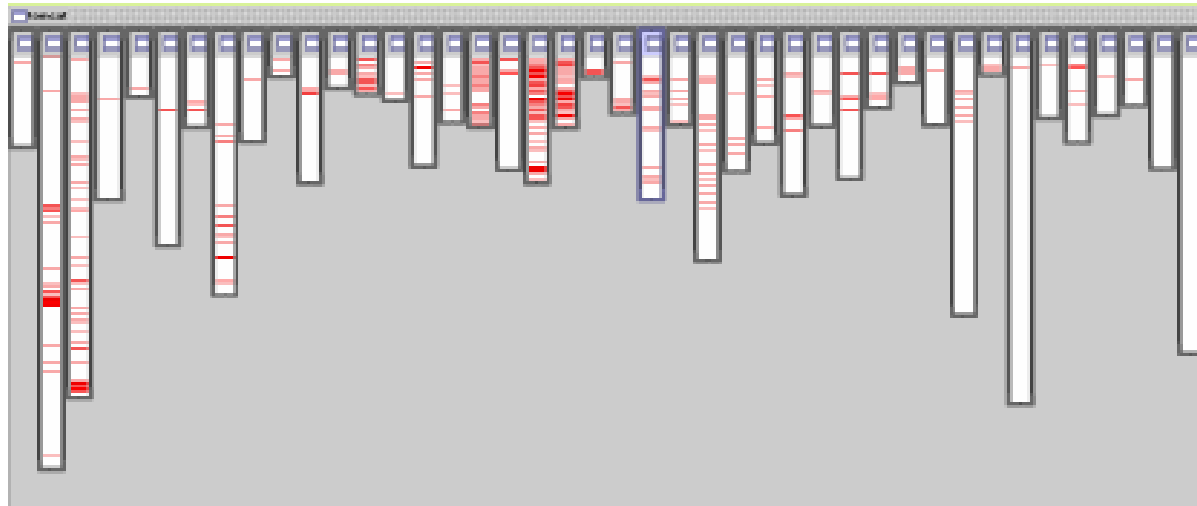
Edsger Dijkstra 1930-2002

Source: E. Dijkstra, A Discipline of Programming, Prentice Hall, 1976, pp. 210

# Crosscutting Concerns Affect Modularization



[XML parsing in org.apache.tomcat]



[logging in org.apache.tomcat]



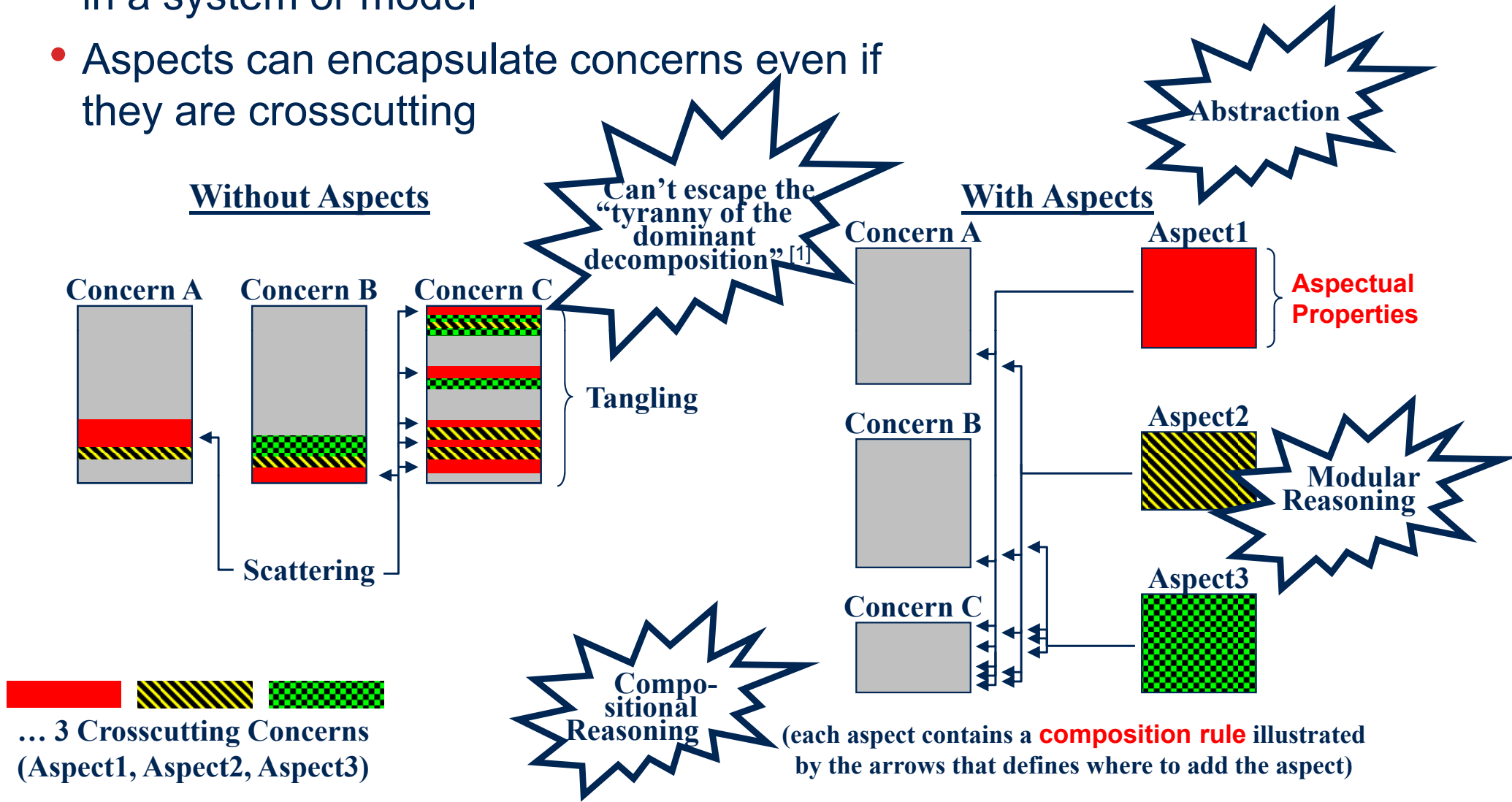
# Problems Detected

- Poor modularization
  - Scattering and Tangling
  - Limits modular reasoning
- Lack of composability
  - Missing systematic mechanism to describe concern influences
- Lack of crosscutting concern identification
- Lack of impact analysis
  - Evaluation mechanisms for goal models offer some support for high level goals (often NFRs) though



# Overview of Aspect-oriented Modeling (AOM)

- Aspects address the problem of one concern **crosscutting** other concerns in a system or model
- Aspects can encapsulate concerns even if they are crosscutting



[1] Tarr, P., Ossher, H., Harrison, W., and Sutton, S.M.: N degrees of separation: Multidimensional separation of concerns. ICSE 99

# Main Value of Aspect-Orientation

- **Abstraction**: abstract away from the details of how crosscutting concerns, or aspects, might be scattered and tangled with the functionality of other modules in the system
- **Modularization**: keep crosscutting concerns separated regardless of how they affect or influence various other modules in the system, so that we can reason about each module in isolation – **Modular Reasoning**
- **Composition**: the various modules need to relate to each other in a systematic and coherent fashion so that one may reason about the global or emergent properties of the system – **Compositional Reasoning**

Source: A. Rashid, A. Moreira, Domain Models are NOT Aspect Free, Proceedings of MoDELS Conference 2006, Springer.

# What is AORE About?

- Leverage the benefits of aspect-orientation in terms of
  - Abstraction
  - Modularity
  - Composability
- To improve modular and compositional reasoning about **stakeholders' requirements**, and
- To address scattering, tangling, and composability issues during **requirements engineering**



# Aspect-Oriented Requirements Engineering

- Improved support for separation of crosscutting functional and non-functional properties during requirements engineering
  - A better means to identify and manage conflicts arising due to tangled representations
- Identify the mapping and influence of requirements-level aspects on artefacts at later development stages
  - Establish critical trade-offs even before the architecture is derived
- Trace aspectual requirements and their trade-offs to architecture and subsequently all the way to implementation

**Improved understanding of the problem and ability to reason about it**

# Managing Crosscutting Concerns in Requirements (1)

- To find crosscutting in requirements, look for behavioural terms or concepts that are mentioned in more than one location
1. Pay interest of a certain percent on each account making sure that the transaction is fully completed and an audit history is kept.
  2. Allow customers to withdraw from their accounts, making sure that the transaction is fully completed and an audit history is kept.



## Managing Crosscutting Concerns in Requirements (2)

- Separate each of those concepts into a section of their own

1. *Pay interest* of a certain percent on each account

2. Allow customers to *withdraw* from their accounts

3. To *fully complete a transaction*...

4. To *maintain an audit history*...



## Managing Crosscutting Concerns in Requirements (3)

- Composition specifies the crosscutting relationship, showing how crosscutting concerns affect base concerns

1. *Pay interest* of a certain percent on each account

*Fully complete pay interest and withdraw*

3. *To fully complete a transaction...*

2. Allow customers to *withdraw* from their accounts

*Audit pay interest and withdraw*

4. *To maintain an audit history...*

# Modeling Techniques for Aspect-oriented Requirements

- Theme/DOC
  - Identifies themes in textual requirements documents and their potentially crosscutting relationships
- MATA
  - Transformation-based approach
  - May be applied to any modeling technique with a well-defined meta-model
- AOSD with Use Cases
  - A methodology that allows use cases to be encapsulated throughout the whole development process with the help of aspect-oriented techniques
- and many more...

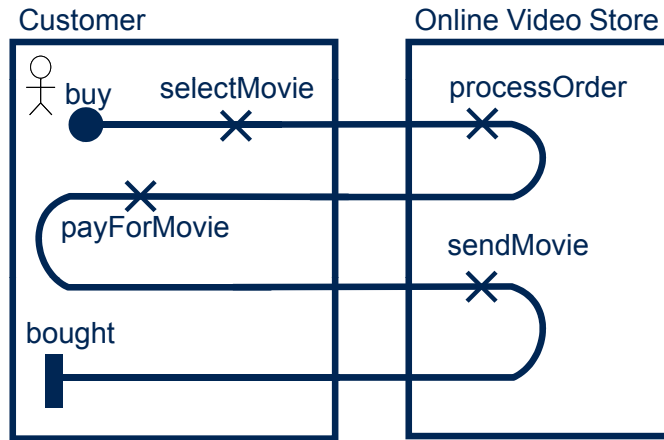


# The Aspect-oriented User Requirements Notation (AoURN) in a Nutshell

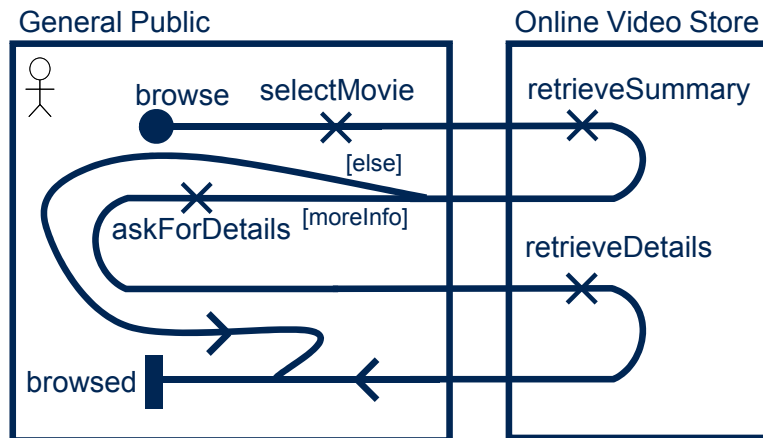


# AoURN: Learning by Example (1)

- Buy Movie Concern

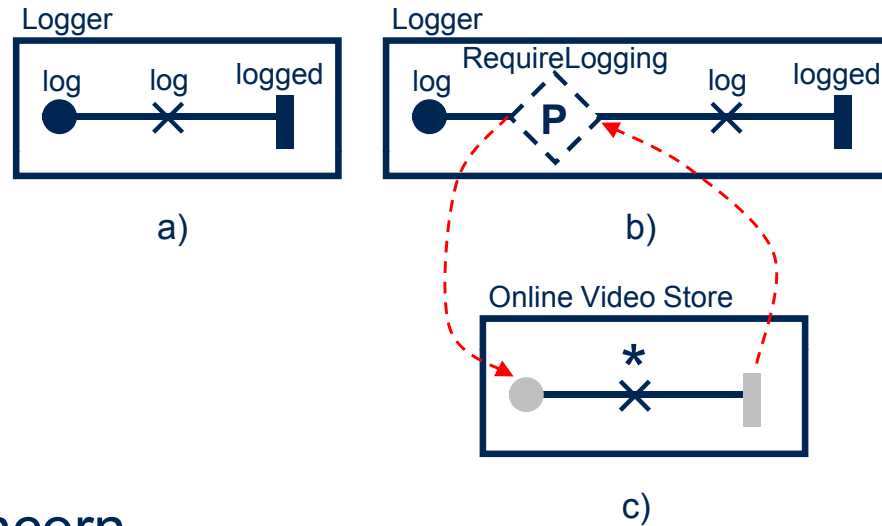


- Browse Movie Concern

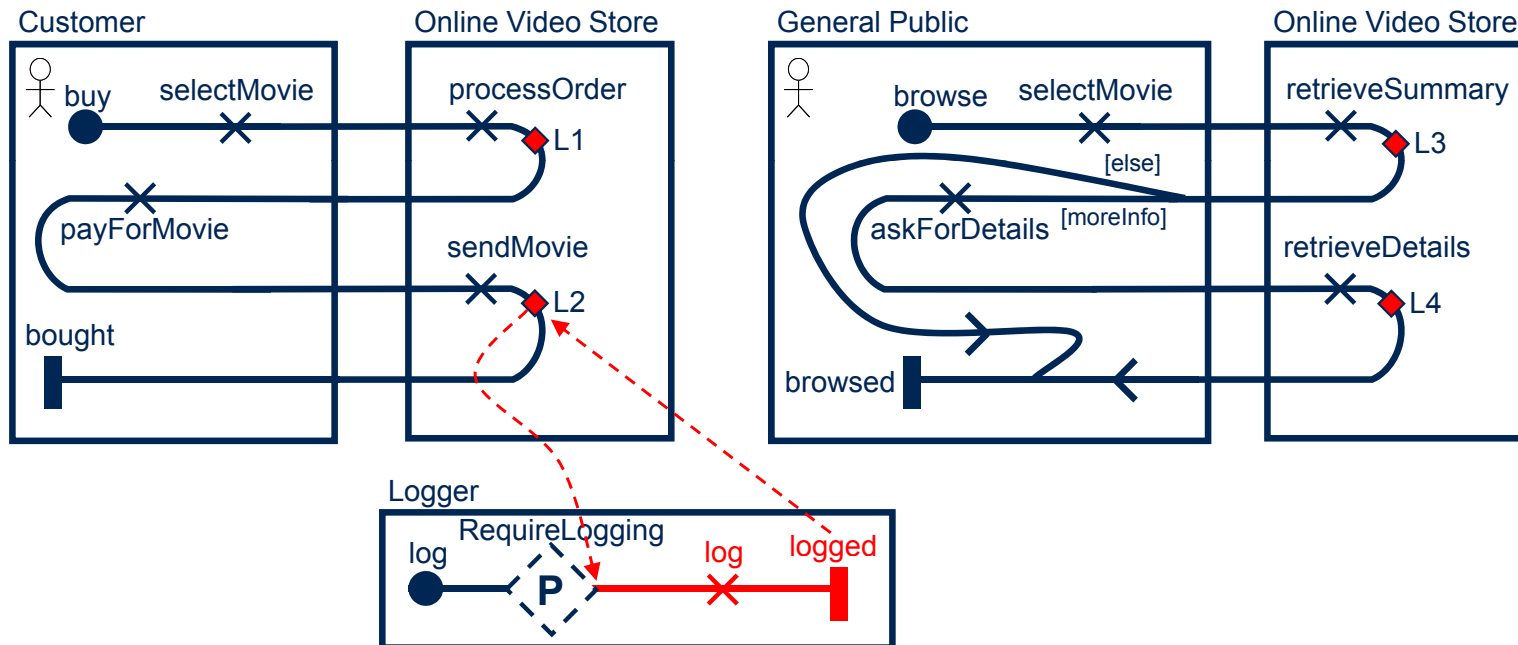


# AoURN: Learning by Example (2)

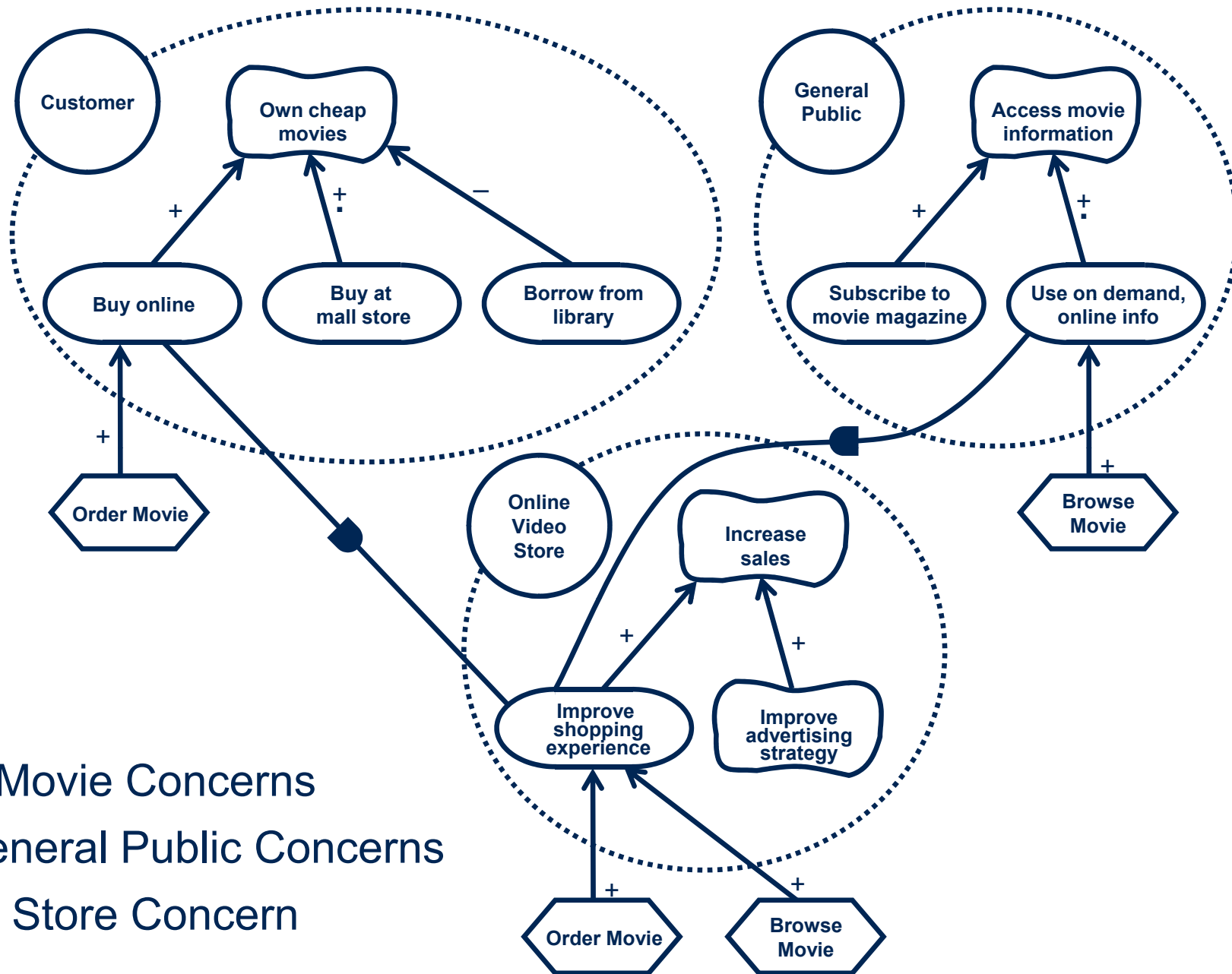
- Logging Concern (b + c)



- Applied Logging Concern



# AoURN: Learning by Example (3)

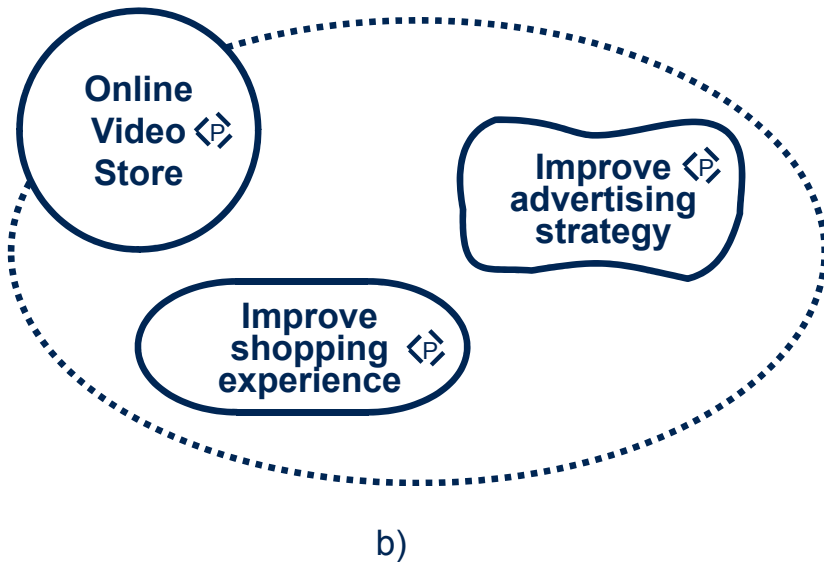
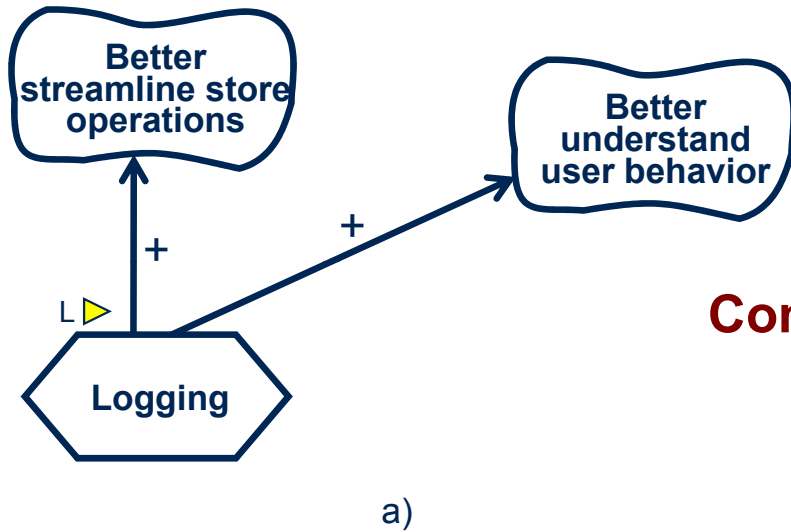


- Buy/Browse Movie Concerns
- Customer/General Public Concerns
- Online Video Store Concern

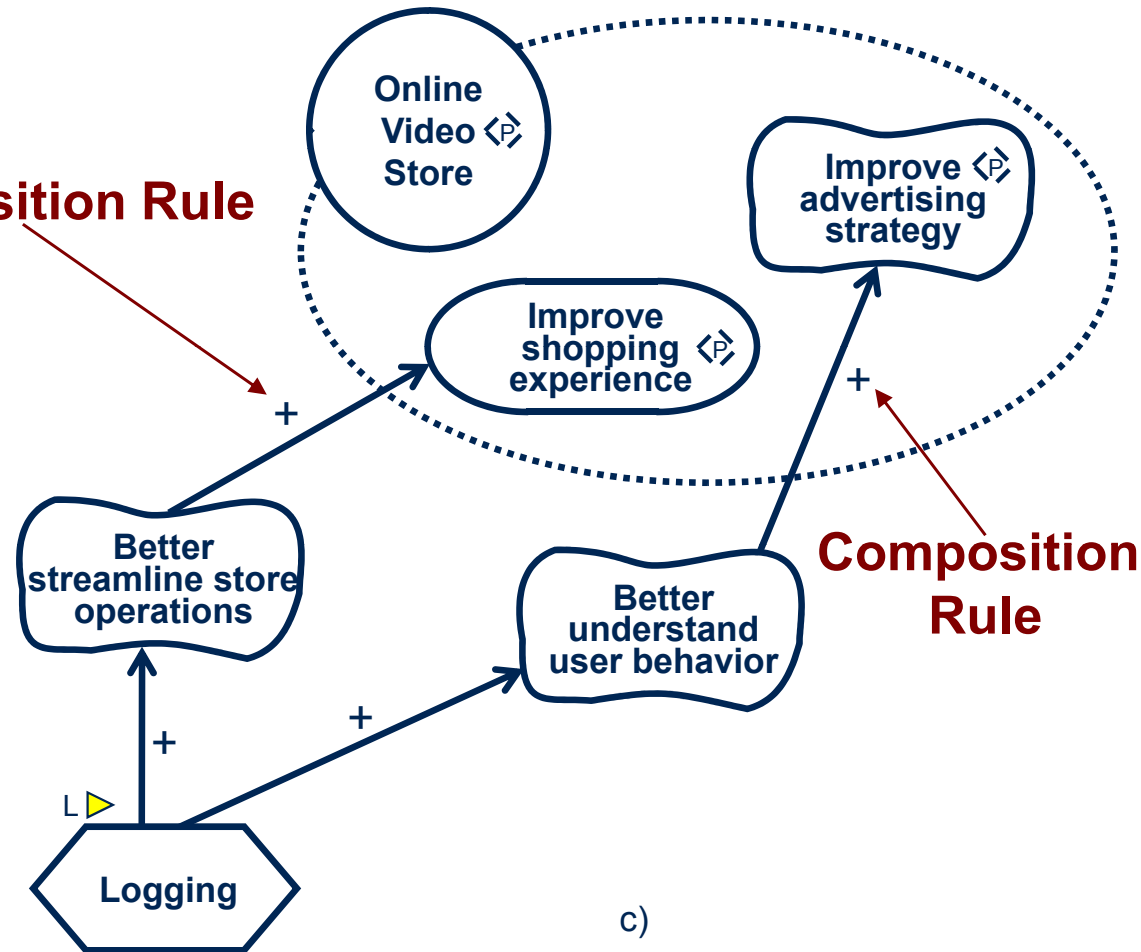


# AoURN: Learning by Example (4)

- Logging Concern (c)



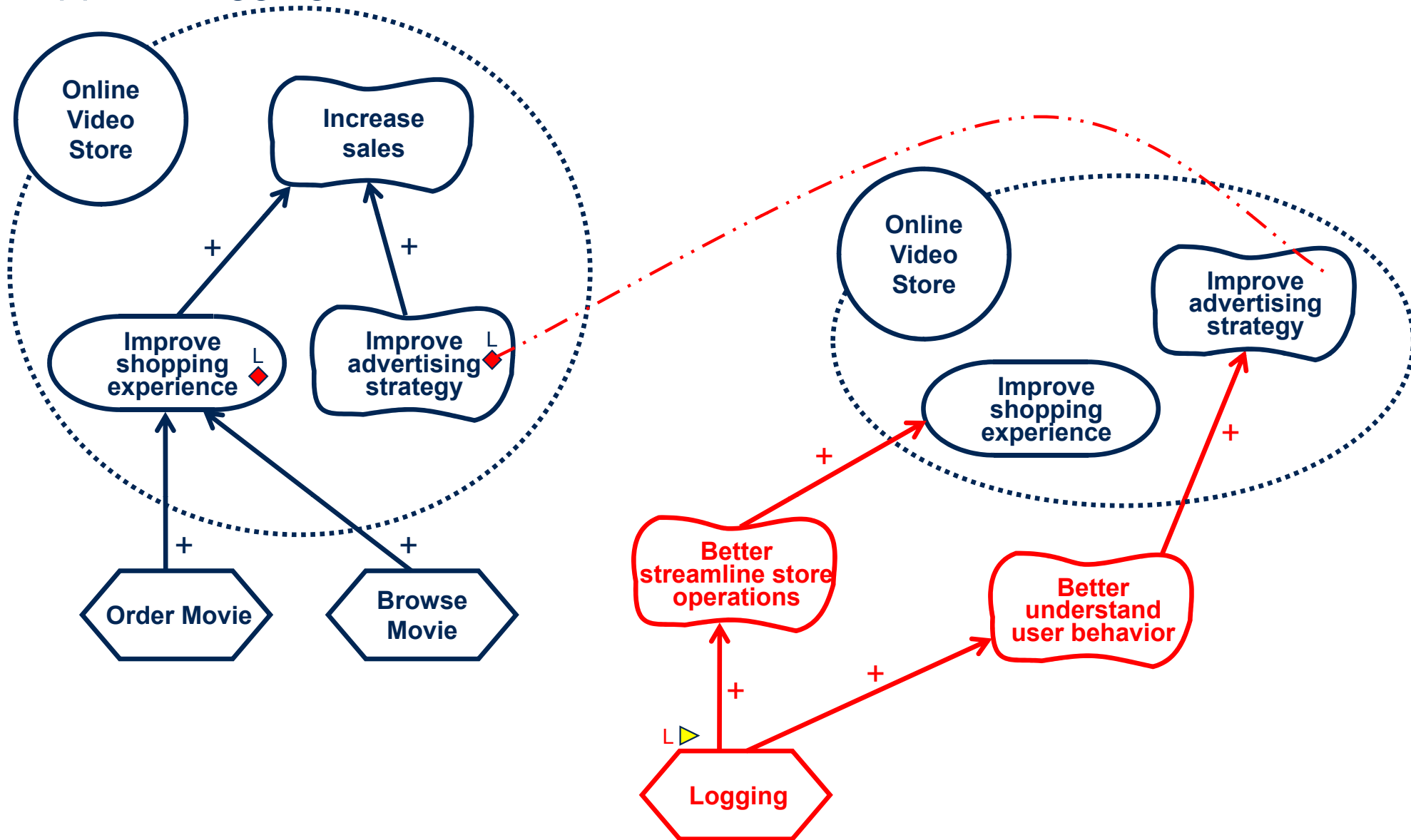
**Composition Rule**



**Composition Rule**

# AoURN: Learning by Example (5)

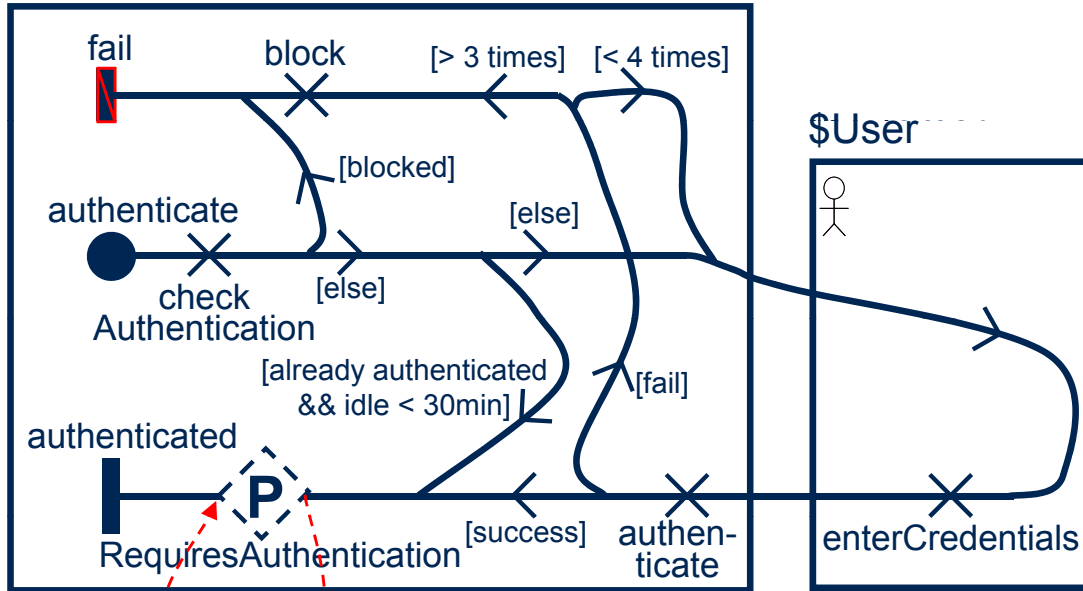
- Applied Logging Concern



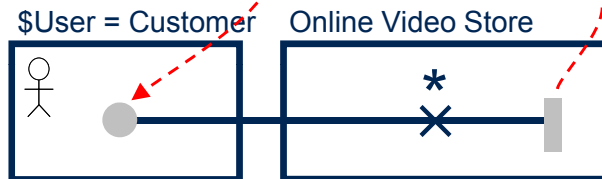
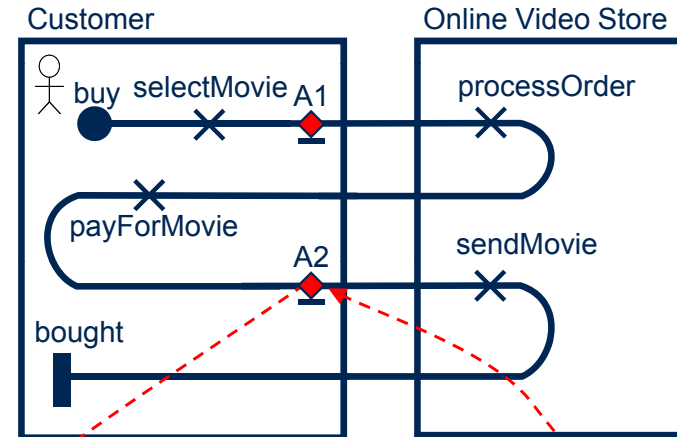


# AoURN: Learning by Example (6)

## Security Server

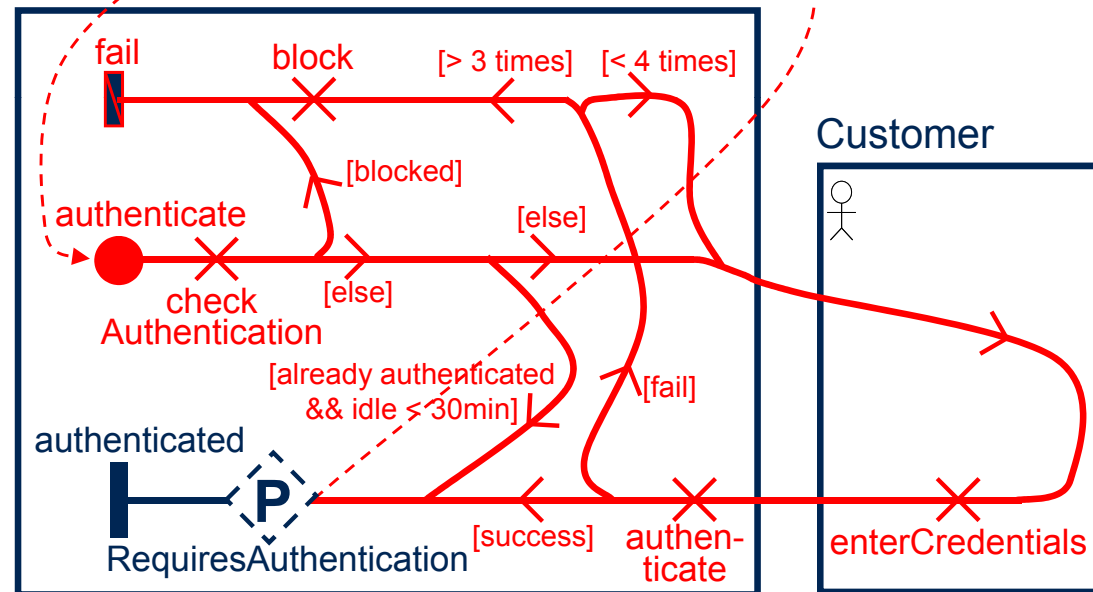


- Applied Authentication Concern



- Authentication Concern

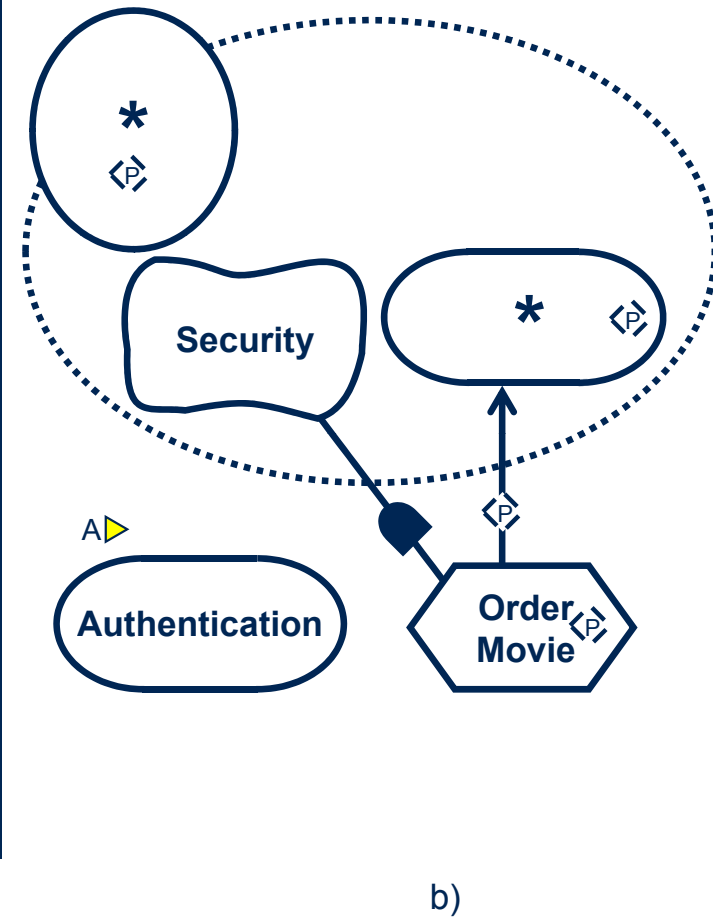
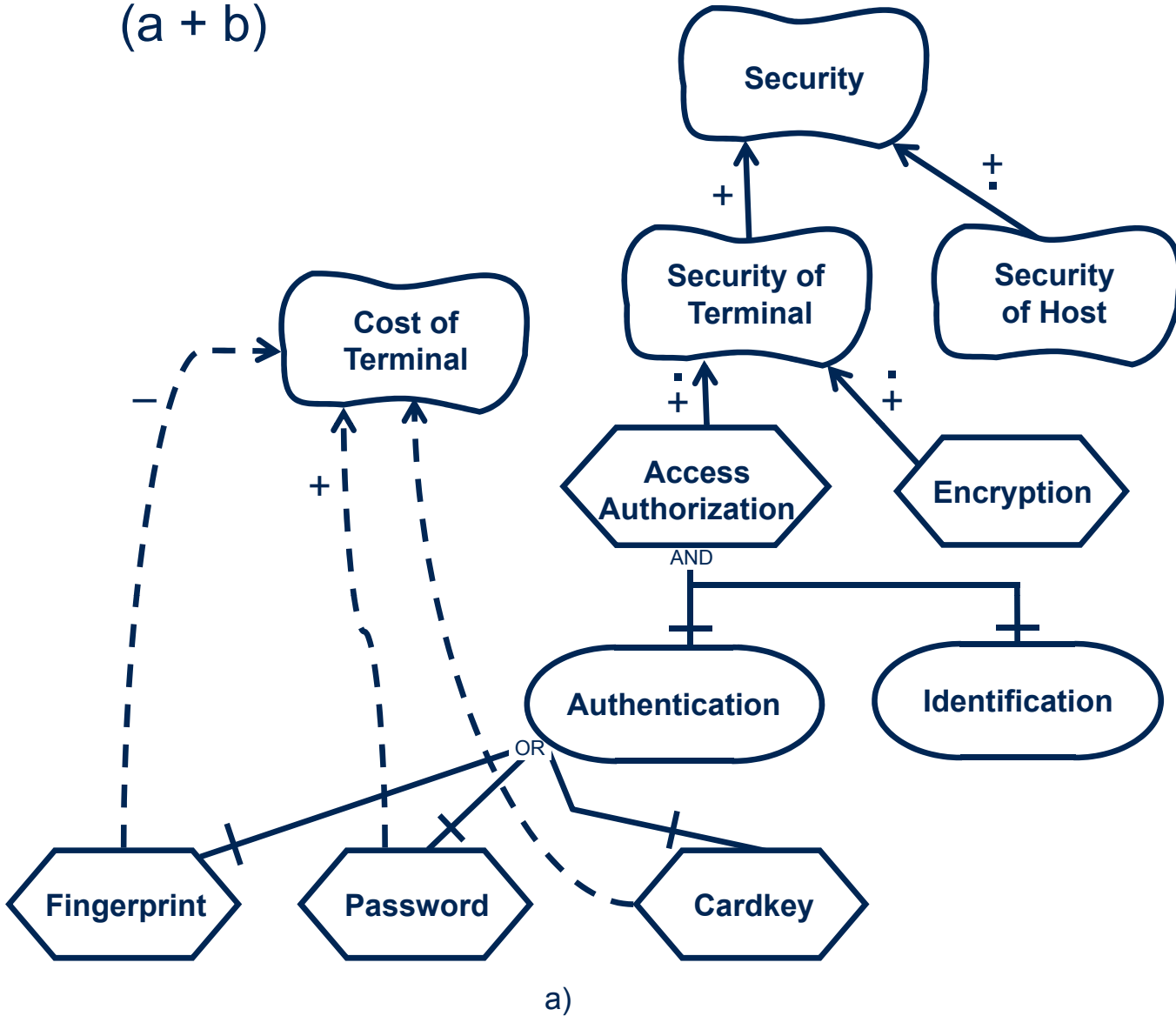
## Security Server





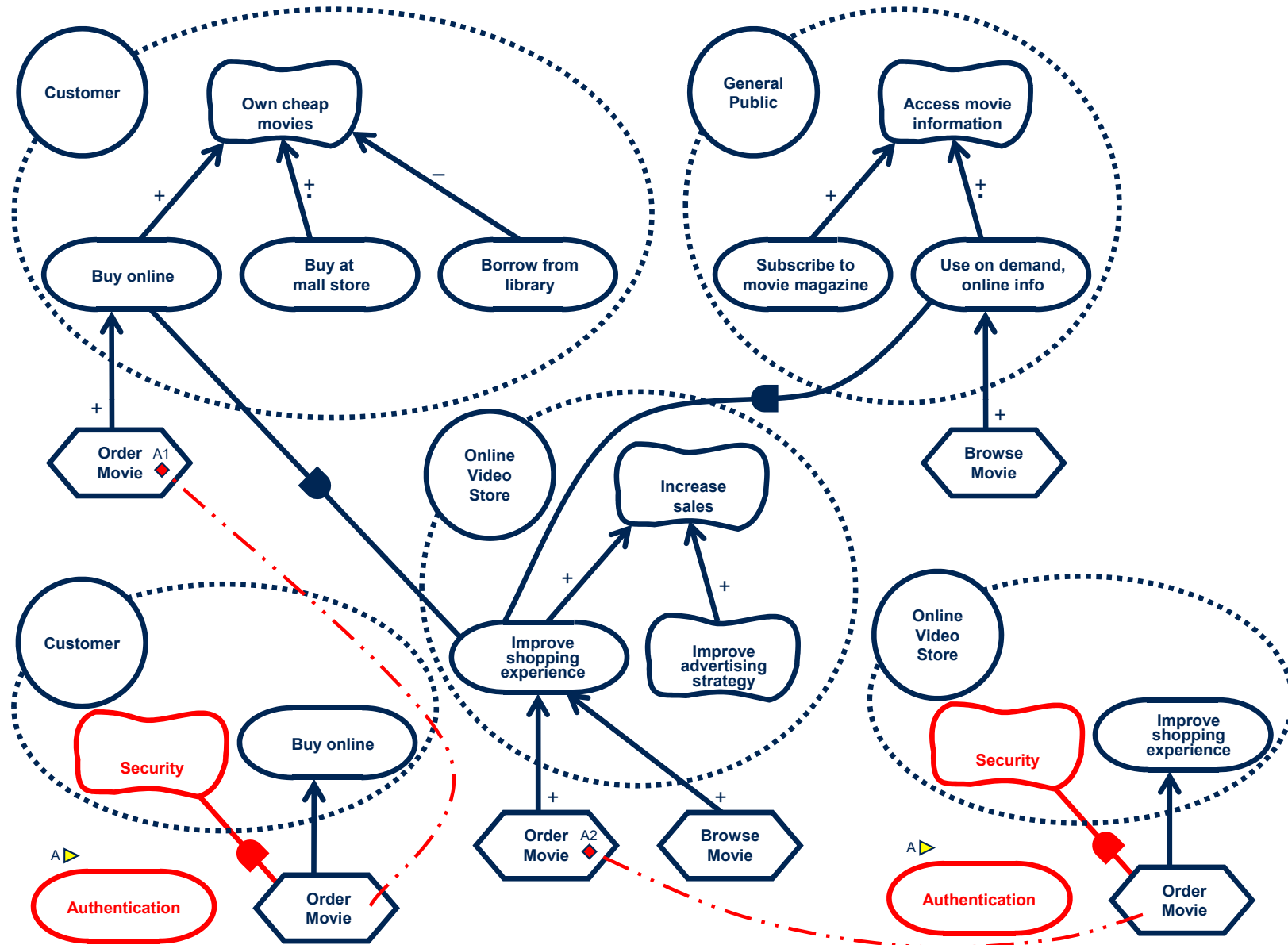
# AoURN: Learning by Example (7)

- Authentication Concern (a + b)



# AoURN: Learning by Example (8)

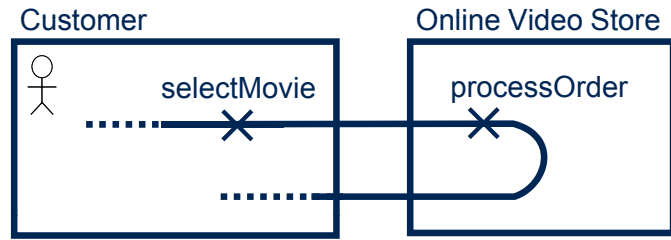
- Applied Authentication Concern



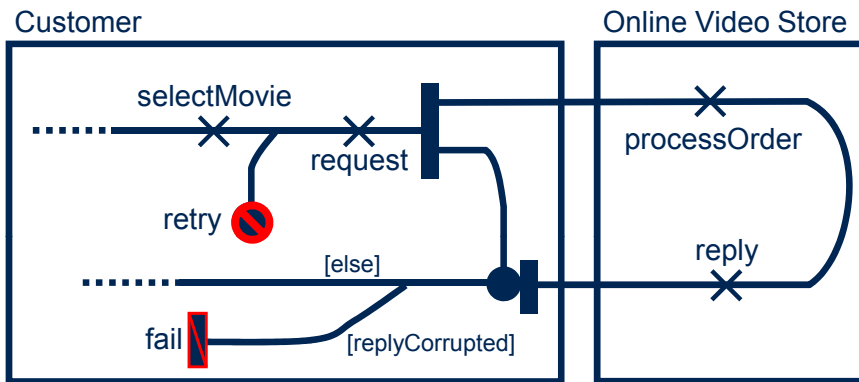


# AoURN: Learning by Example (9)

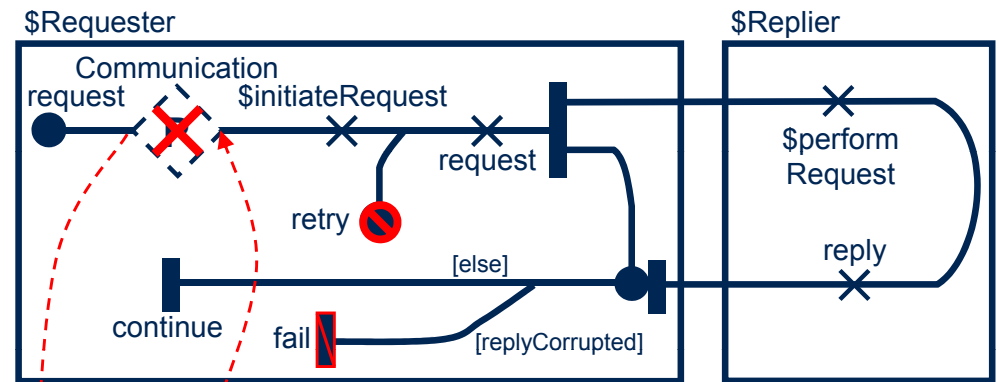
- Communication Concern (c)



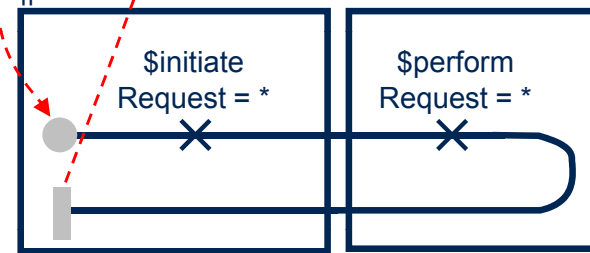
a)



b)



`$Requester = Customer` `$Replier = Online Video Store`  
 || General Public

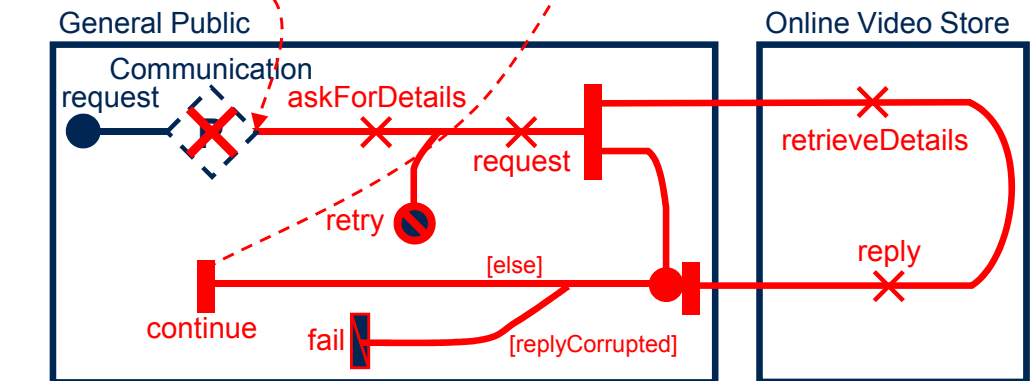
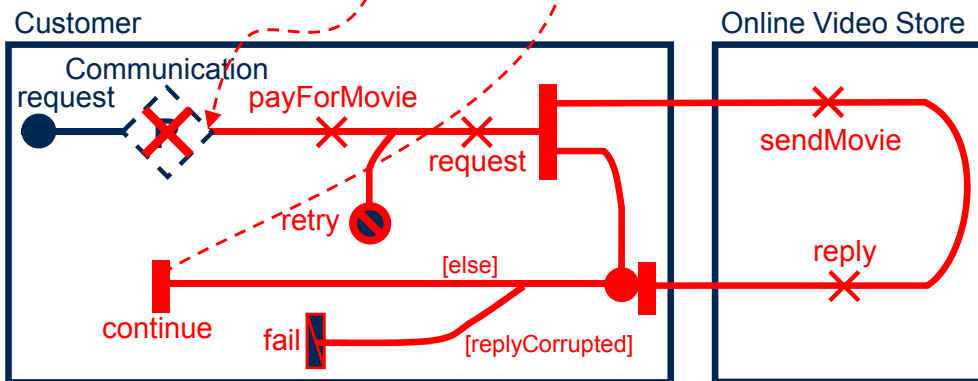
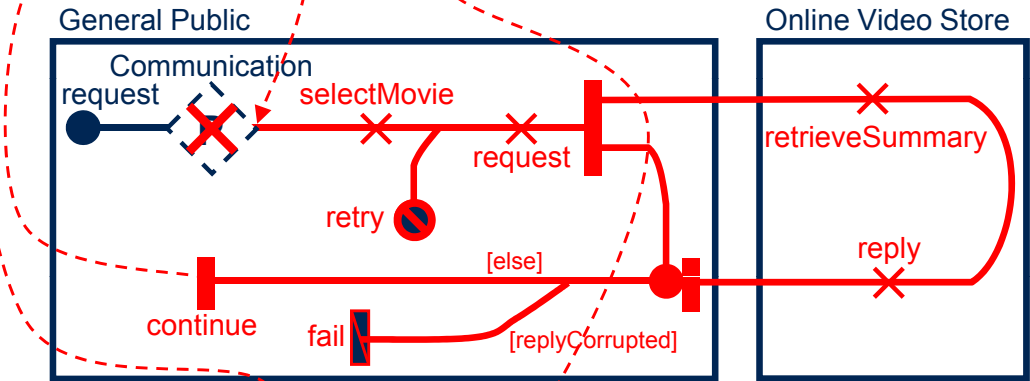
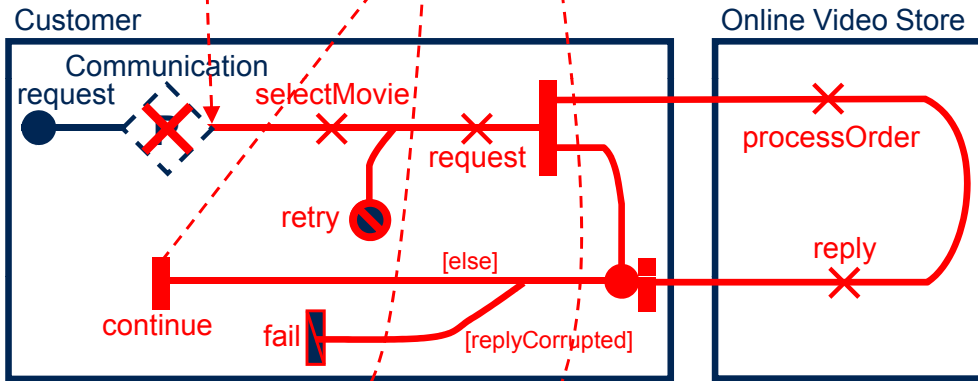
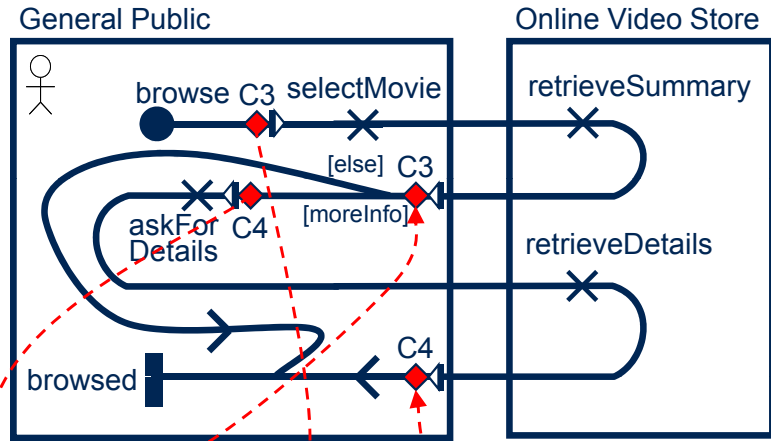
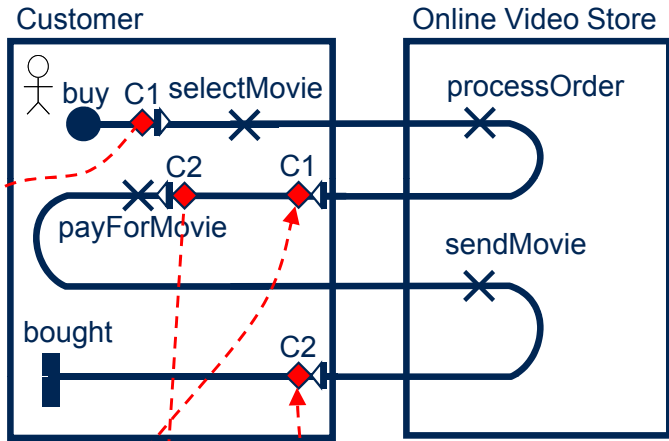


c)

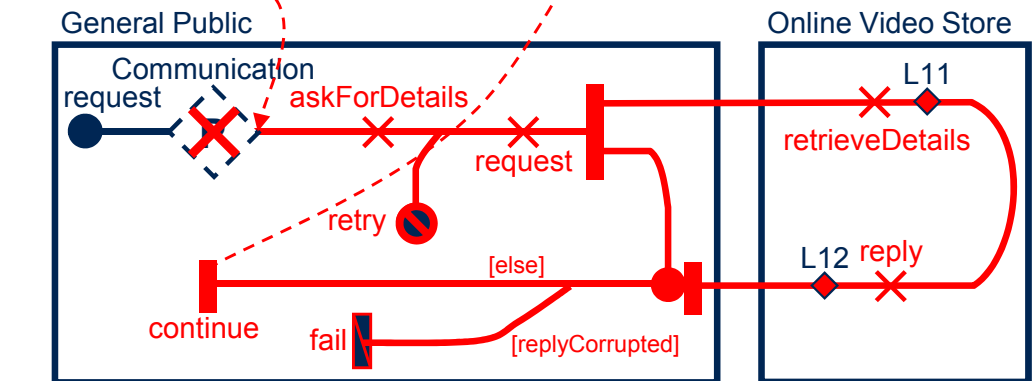
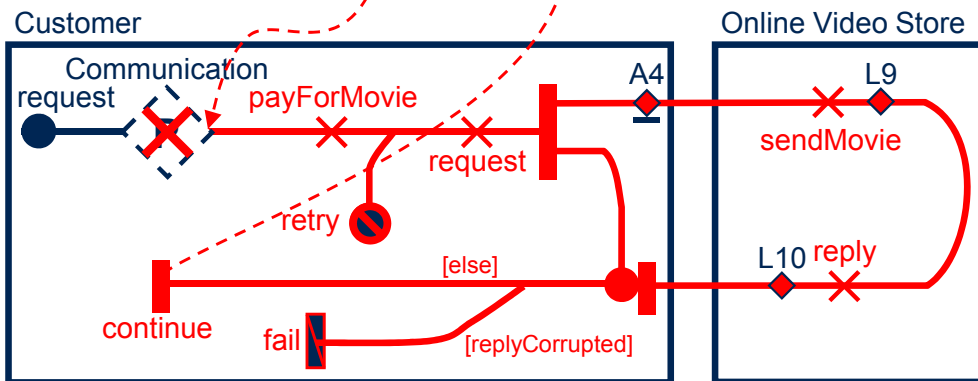
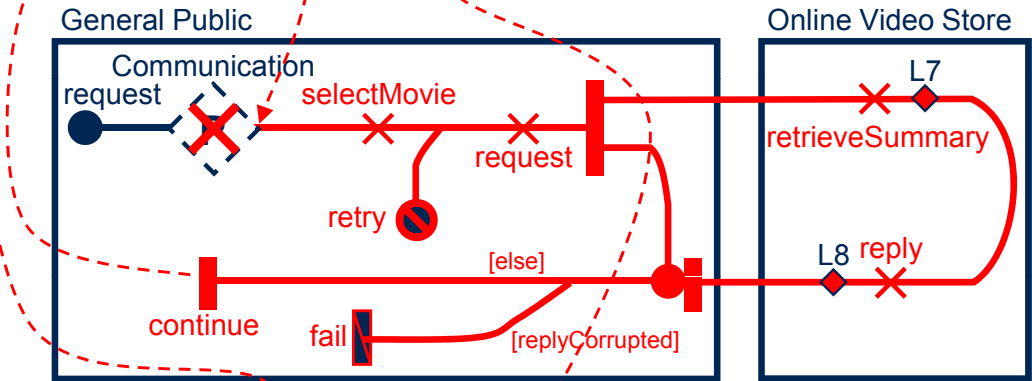
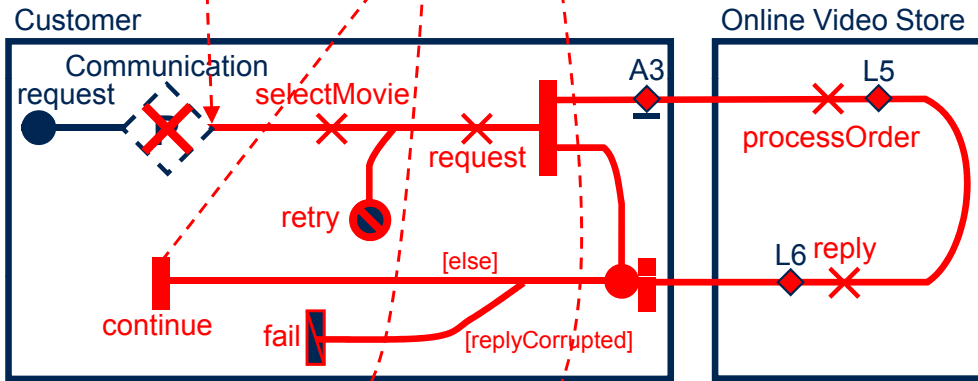
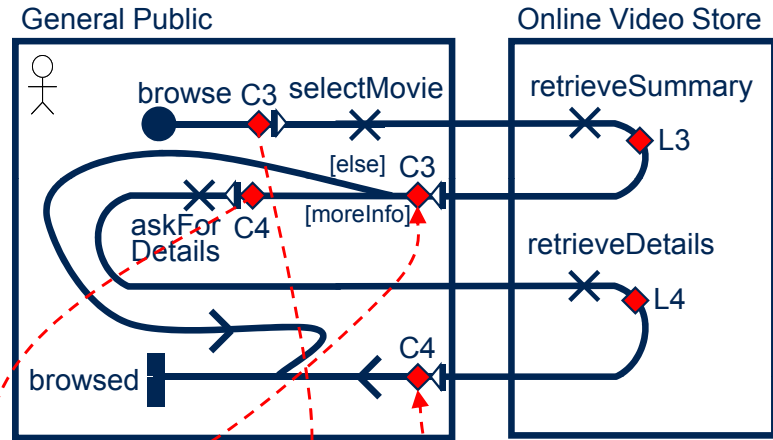
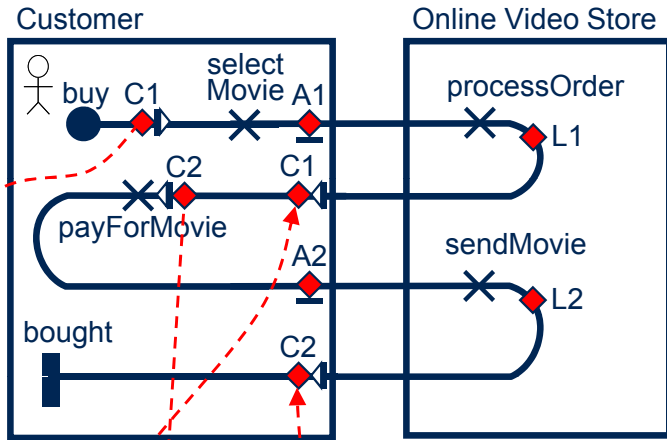
- Applied Communication Concern (on next slide)
- Applied Logging / Authentication / Communication Concern (on slide after next slide)



# AoURN: Learning by Example (10)



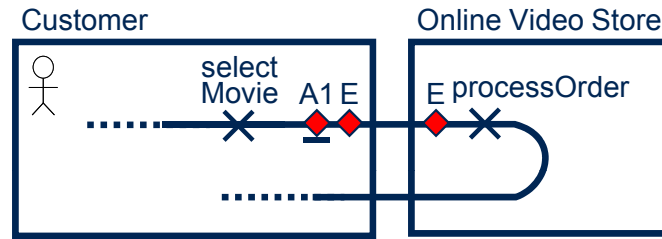
# AoURN: Learning by Example (11)





# AoURN: Learning by Example (12)

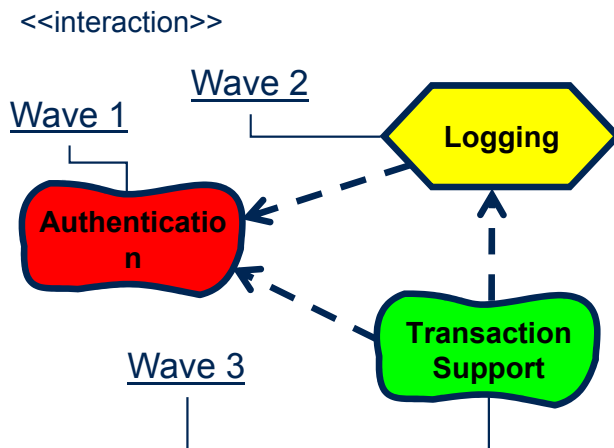
- Encryption Concern



- Concern Interaction Graph



- Waves



First Wave (add authentication before/after join point)



Second Wave (add logging before/after join point)



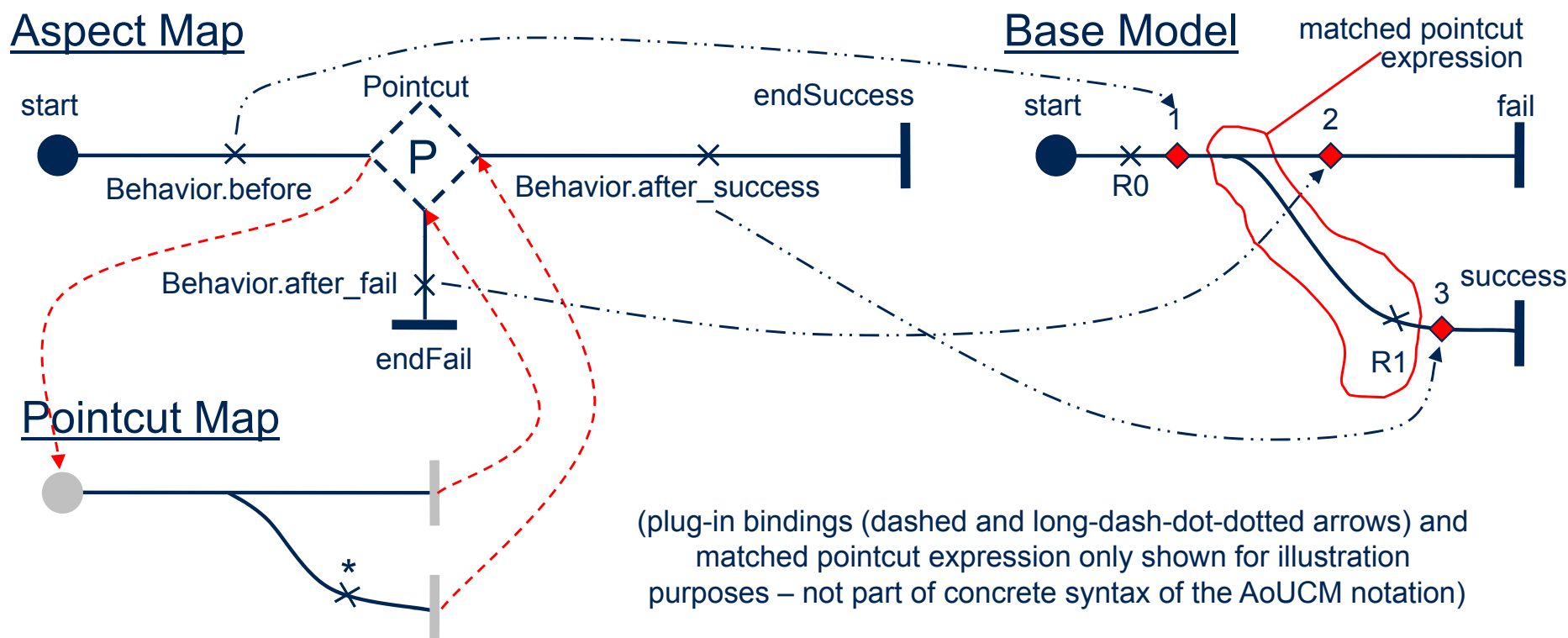
Third Wave (add transaction support before/after join point)





# Aspect-oriented Use Case Maps: In a Nutshell (1)

- An aspect defines its structure/behavior (= aspect map) and a pattern called pointcut expression (= pointcut map) for its composition rule stating where the aspect is to be applied in a model



Pointcut stub:



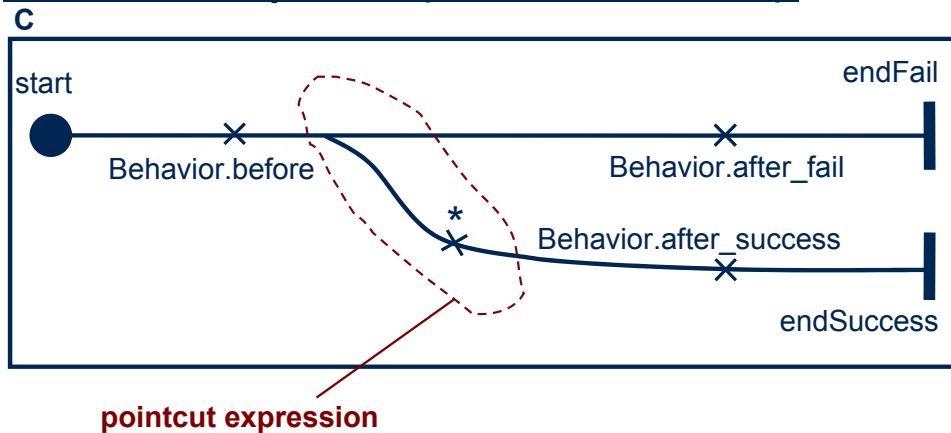
Aspect marker:



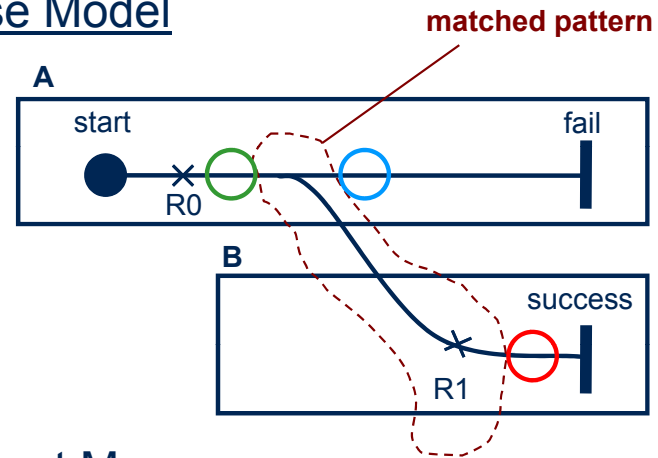
# Aspect-oriented Use Case Maps: In a Nutshell (2)

- All-in-One style not used because it hinders separate reuse of aspectual properties and pointcut expressions
- All-in-One style leads to duplication if the same aspect needs to be applied at various locations that cannot be described with one pattern

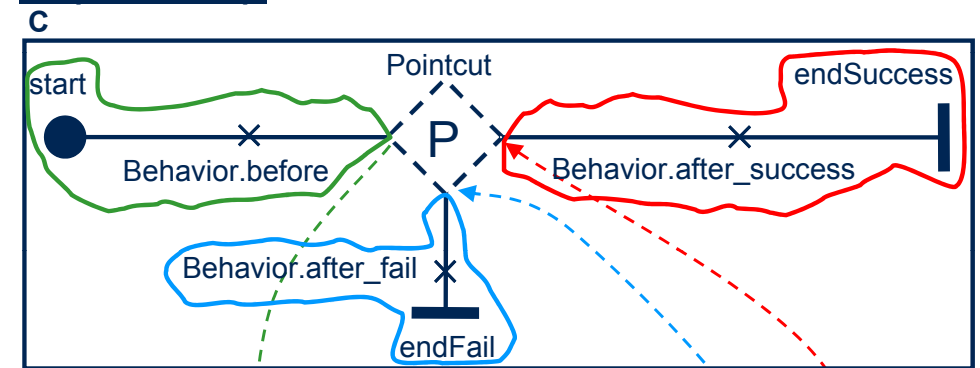
All-in-One Style - Aspect/Pointcut Map



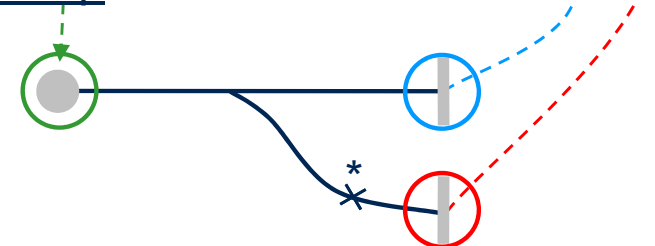
Base Model



Aspect Map



Pointcut Map

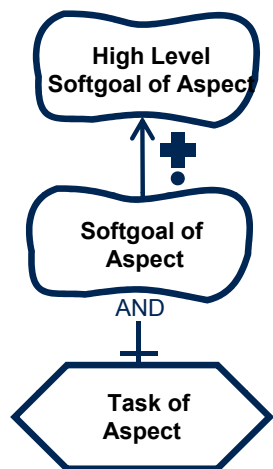




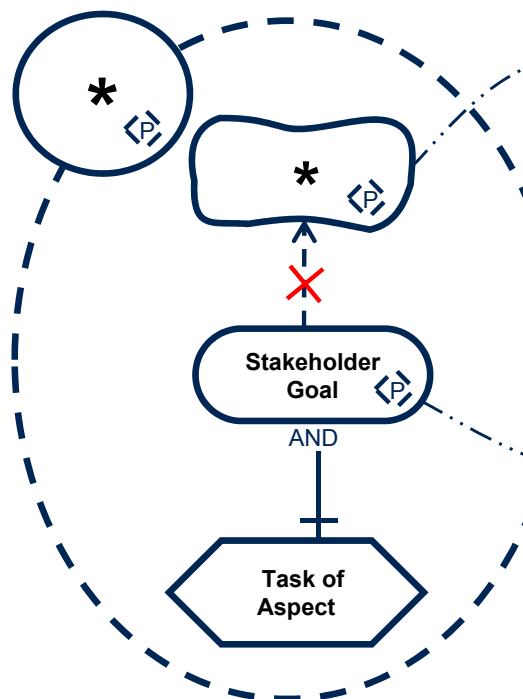
# Aspect-oriented GRL: In a Nutshell

- An aspect defines its structure/behavior (= aspect graph + non-marked pointcut graph) and a pattern called pointcut expression (= marked pointcut graph) for its composition rule stating where the aspect is to be applied in a model

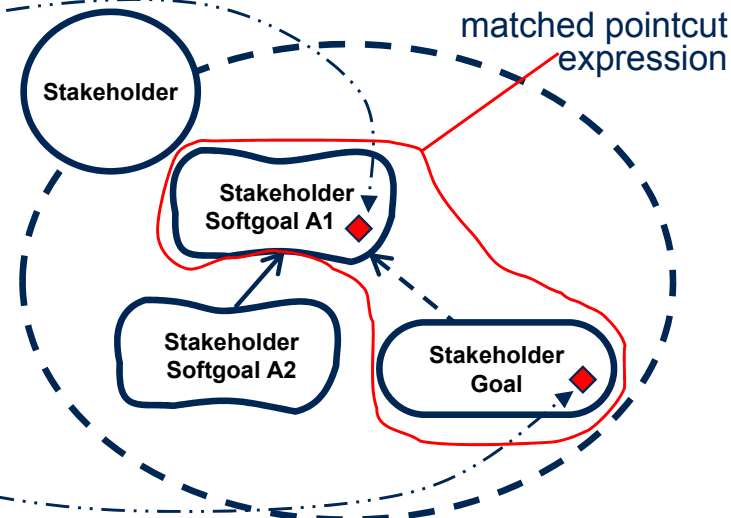
Aspect Graph



Pointcut Graph



Base Model



Pointcut marker:



Pointcut deletion

marker:



Aspect marker:



(mapping of pointcut expression to base model (long-dash-dot-dotted arrows) and matched pointcut expression only shown for illustration purposes – not part of AoGRL notation)

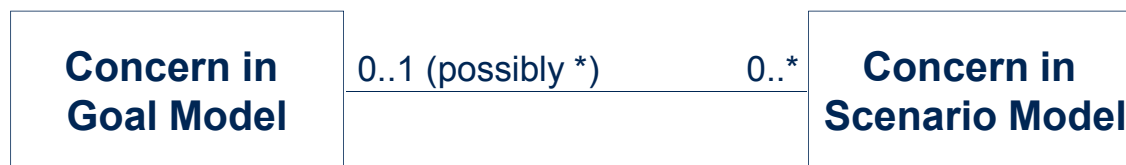


# Aspect-oriented URN: Introduction (1)

- AoURN (Aspect-oriented User Requirements Notation)
  - **Unifies** goal-oriented, scenario-based, and aspect-oriented concepts in a **scalable** framework
  - **Extends** the abstract syntax, the concrete syntax, and the semantics of URN with aspect-oriented concepts
  - Requires almost no changes to the **familiar** URN notation (syntax remains virtually the same but the existing semantics are clarified and extended)
- AoURN models **each** use case, **each** stakeholder's goals, and **each** NFR as an aspect
  - NFRs fundamentally are aspectual in nature
  - Most use cases are not aspectual but are peers (aspects, however, can also be used to sufficiently separated such concerns)
  - Most stakeholder goals are not aspectual but are peers (aspects, however, can also be used to sufficiently separated such concerns)
- AoURN models all concerns as aspects and therefore follows a more **symmetrical, multi-dimensional** approach to aspects

# Aspect-oriented URN: Introduction (2)

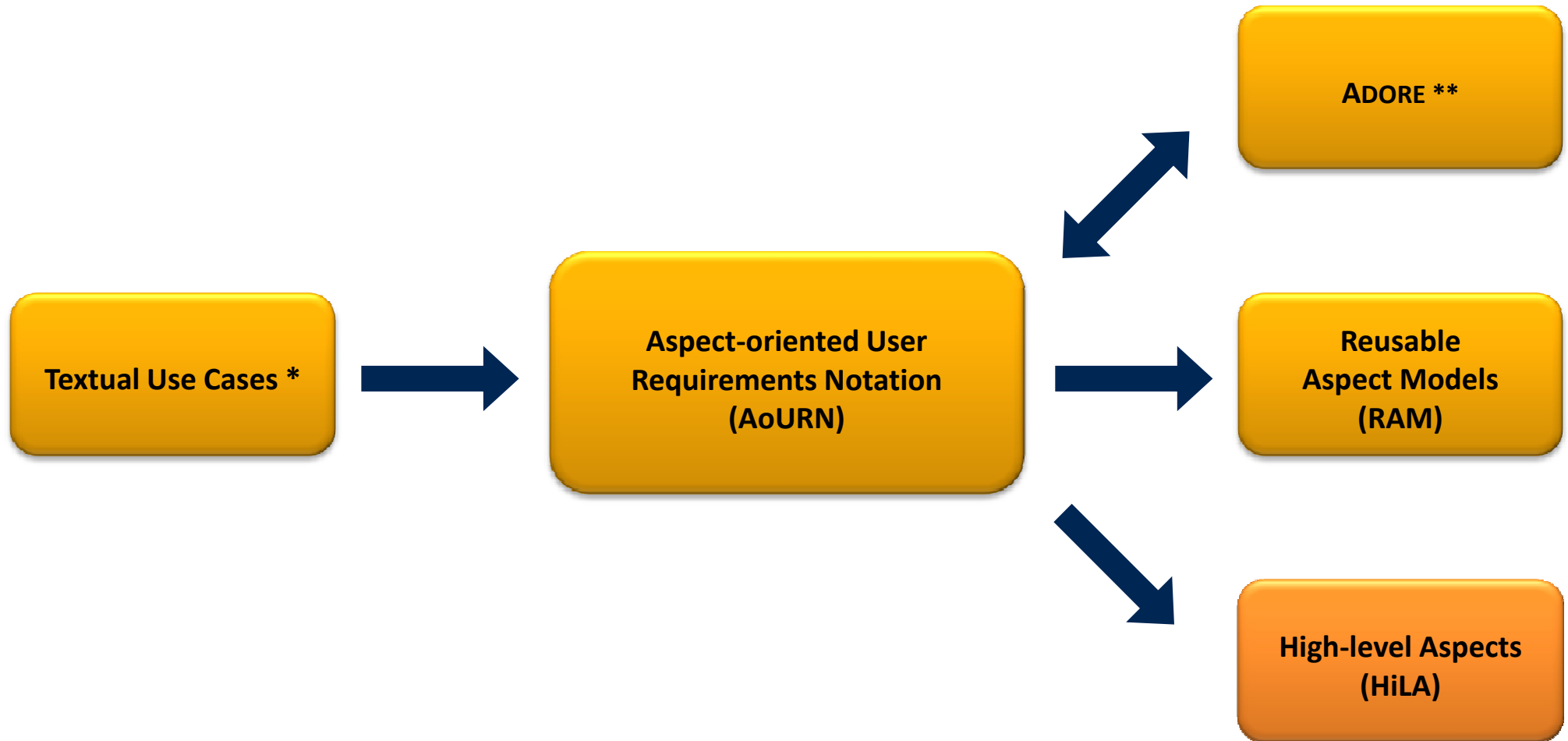
- Features of AoURN
  - Aspects (including pointcut expressions) are fully described in a **graphical** way
  - **Exhaustive** composition of aspects is only limited by the expressive power of URN itself (as opposed to a particular pointcut or composition language)
  - Aspectual properties and pointcut expressions are defined **separately**
- AoURN defines for each aspect at least a goal model or a scenario model (or both)
  - Behavioral/structural dimensions of an aspect are modeled with AoUCM
  - Reasons for an aspect are modeled with AoGRL
  - Aspects in GRL models can be **traced** to aspects in UCM models





# Aspect-oriented URN: Transformations

- Transformations to other models are being investigated for AoURN models



\* Stéphane Somé's UCED: <https://sourceforge.net/projects/uced/>

\*\* see presentation at AOSD'11



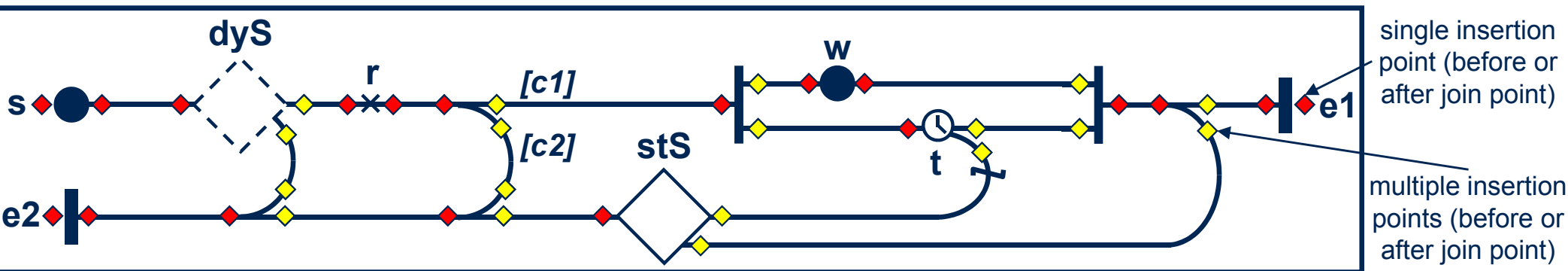


# Aspect-oriented Use Case Maps (AoUCM)

# Introduction to AoUCM: Join Point Model

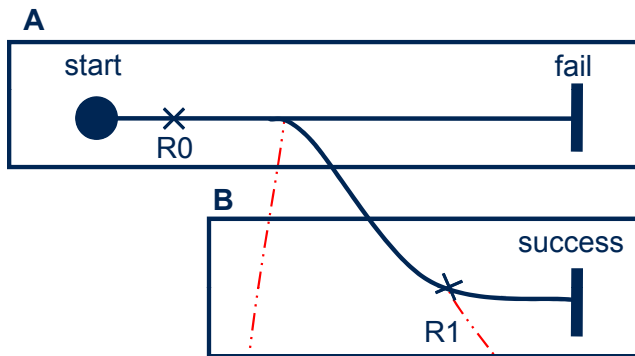
- In order to unify aspect concepts with UCM concepts, **join points**, **aspects**, and **pointcuts** have to be defined in UCM terms
- In AOP, a join point is a point in the dynamic flow of a program
  - Method call, execution of a method, initialization of object, set method, get method, exception handling...
  - A join point has a behavioral and structural dimension
- Advice occurs before/after/around something at the granularity of methods
- In UCM terms, translates to responsibility optionally bound to component
- Responsibilities are just one kind of path node (like many other)
  - Thus, the AoUCM **join point model** includes **all path nodes** (i.e., any (un)bound location on a path) except for purely visual nodes (e.g., direction arrow)

c

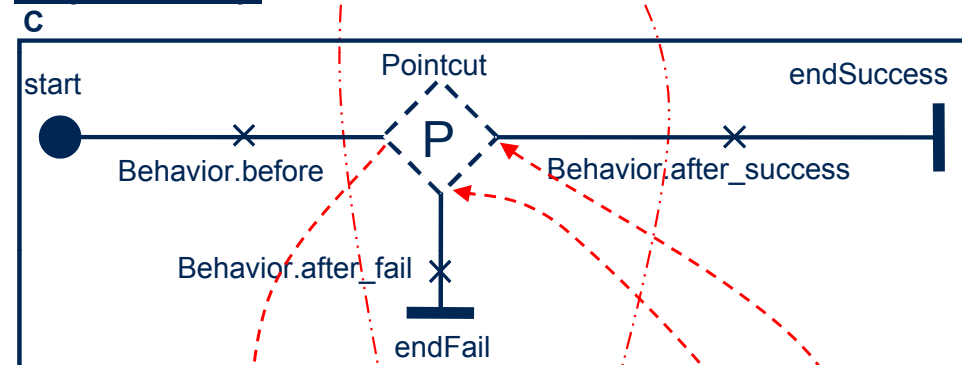


# Introduction to AoUCM: Aspect and Pointcut Maps

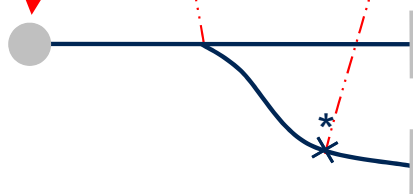
## Base Model



## Aspect Map



## Pointcut Map



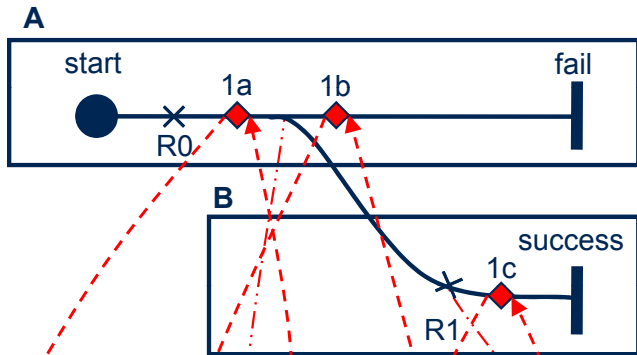
- An aspect is a group of UCMs (some may be aspect maps)
- An **aspect map** visually describes the aspectual properties
- An aspect map contains one (zero) or more pointcut stubs (other than that it is a standard UCM)
- A **pointcut stub** contains one or more pointcut maps (other than that it is a standard stub)
- A **pointcut map** visually describes a pointcut expression (other than that it is a standard UCM)
- A pointcut map is matched against the base model in order to identify the join points that are affected by the aspect

(mapping of pointcut expression to base model (long-dash-dot-dotted lines without arrowheads) and plug-in bindings (dashed arrows) only shown for illustration purposes – not part of AoUCM notation)

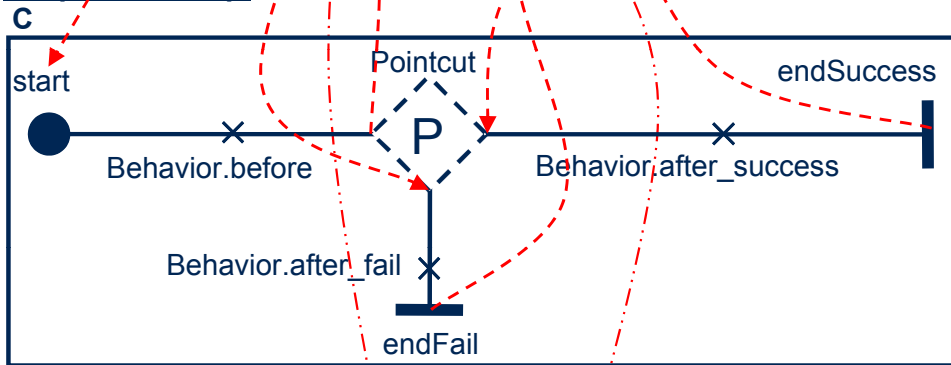


# Introduction to AoUCM: AoView

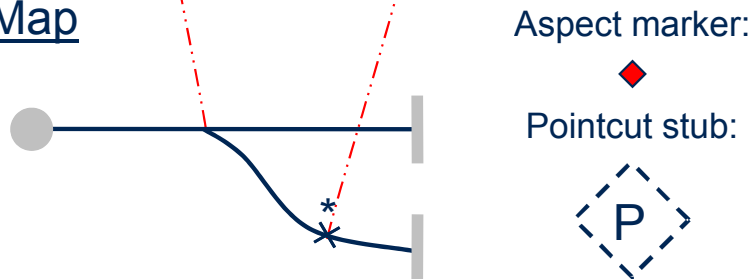
## Base Model



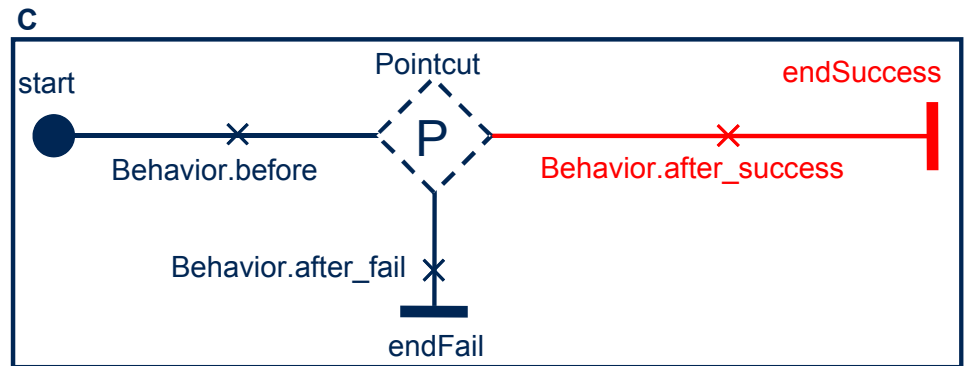
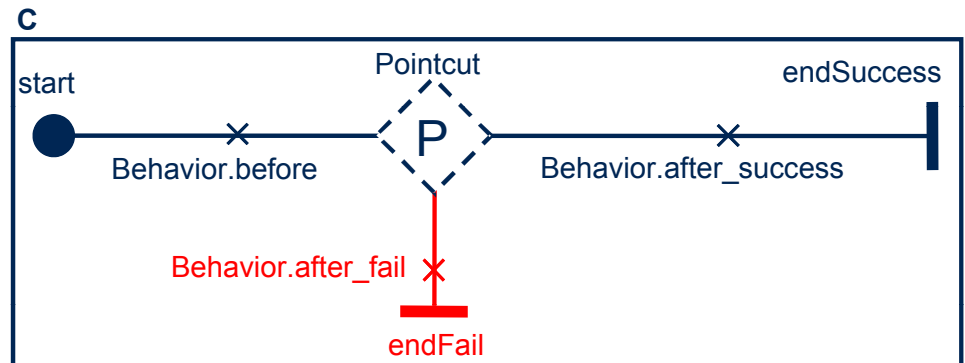
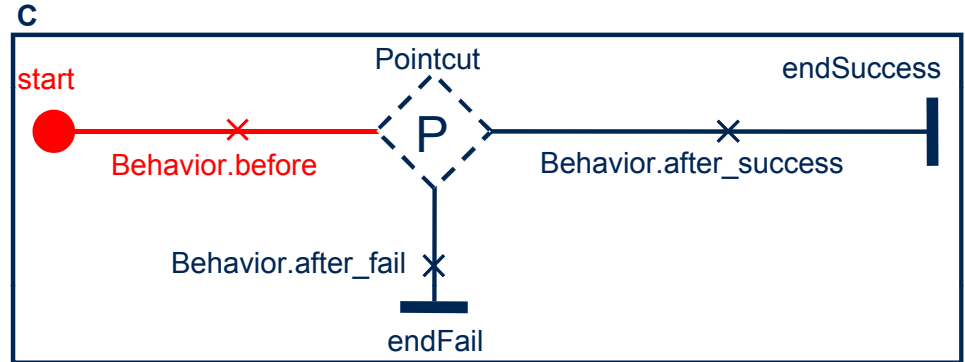
## Aspect Map



## Pointcut Map



## AoViews



# Introduction to AoUCM: Pointcut Expressions (1)

- A pointcut defines a set of join points in a **parameterized** way
  - Defines structural context and behavioral context for the execution of aspect
  - A pointcut map represents a partial map which identifies join points when matched against all other maps in the UCM model
- Semantics of wildcard \*, logical expressions, and empty start/end points
  - A named element on a pointcut map may contain a \* or logical expressions
  - Operators for logical expressions are AND (&&), OR (||), and NOT (!)
  - Start/end points without a name (indicated by gray color) denote only the start/end of the partial map and are therefore not matched against the base model
- **Variables** for responsibilities or components (indicated by \$)



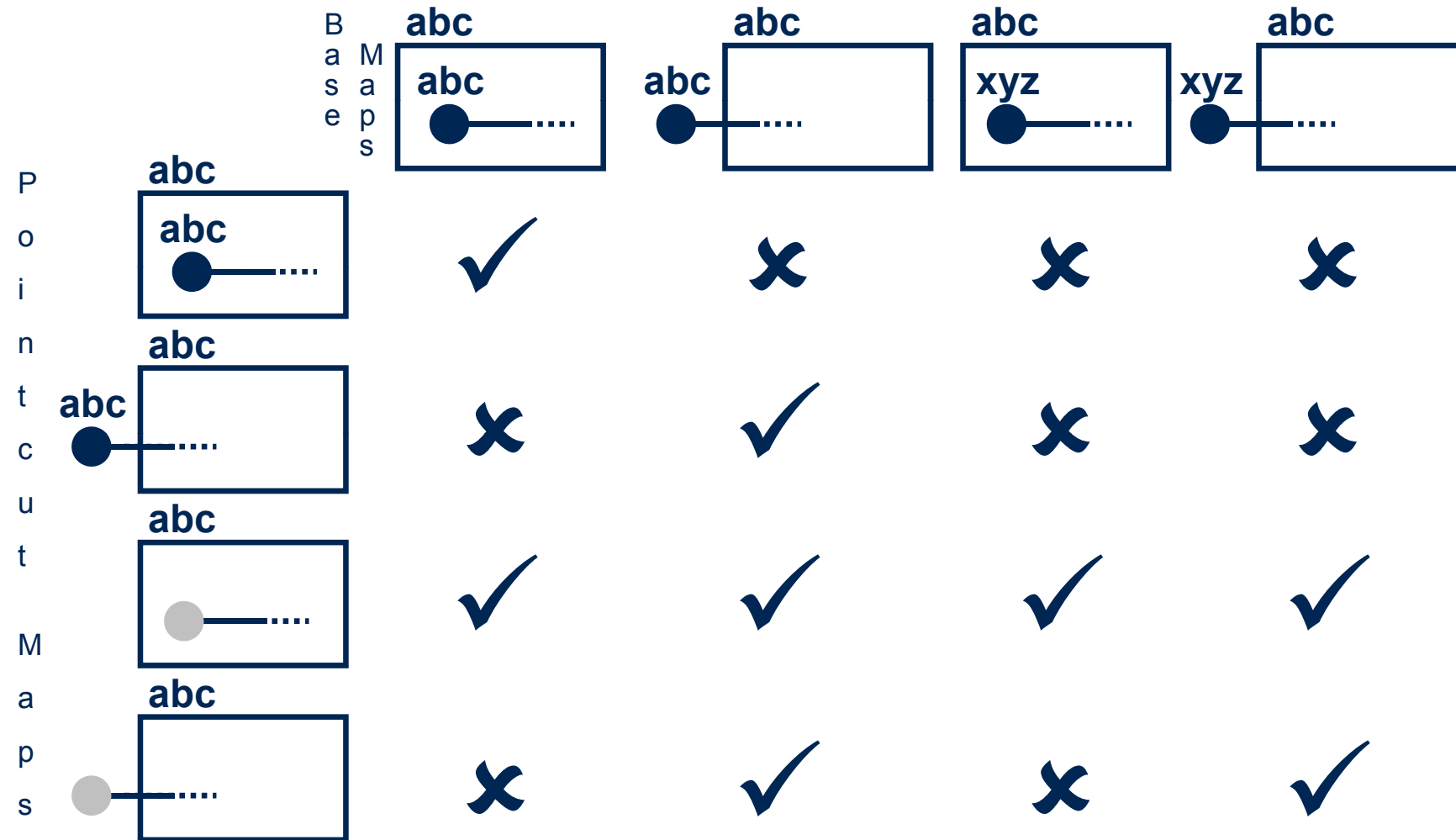
all start points starting with s; all responsibilities; all waiting places named ready or starting with w; all components starting with A  
*note: naming conventions are crucially important*



unnamed (gray) start or end points are not matched against the base model

# Introduction to AoUCM: Pointcut Expressions (2)

- The location of start (end) points **relative** to components is important for the meaning of the pointcut map
- Boundary crossings of a path are preserved when matching a UCM



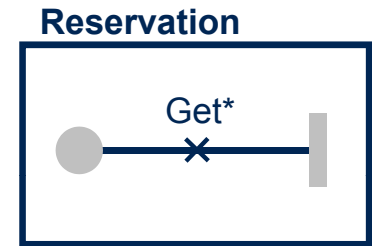
Note: The dotted parts of the base maps and pointcut maps match.



# Introduction to AoUCM: Pointcut Expressions (3)

- Example 1

- Matches all responsibilities with a name starting with “Get” and bound to the component “Reservation”



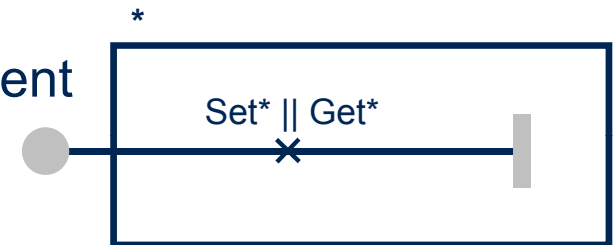
- Example 2:

- Matches all responsibilities with a name starting with “Confirm”, immediately followed by an end point called “confirmed”, both bound to any or no component



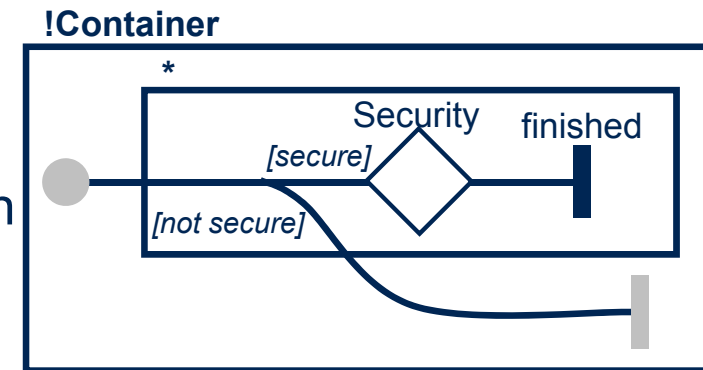
- Example 3:

- Matches all responsibilities with a name starting with “Set” or “Get” and bound to any component as the first path element



- Example 4:

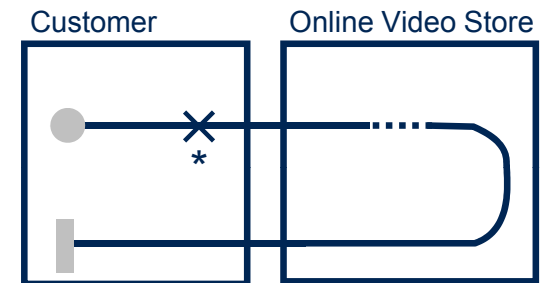
- Matches all OR-forks bound to any component not inside component “Container” as the first path element, immediately followed on one branch by a stub (“Security”) and an end point (“finished”), and followed by nothing on the other branch before exiting the component



# Introduction to AoUCM: Pointcut Expressions (4)

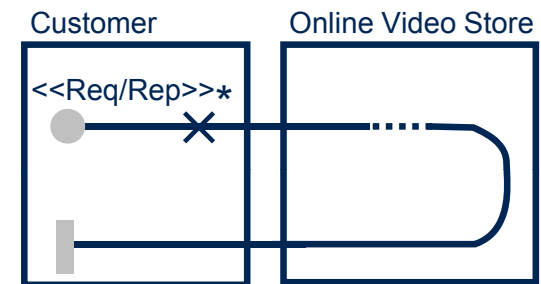
- Example 5 – Anything Pointcut Element:

- Matches all responsibilities bound to the component “Customer”, immediately followed by any sequence of path elements bound to the component “Online Video Store” before the path returns to the “Customer” component



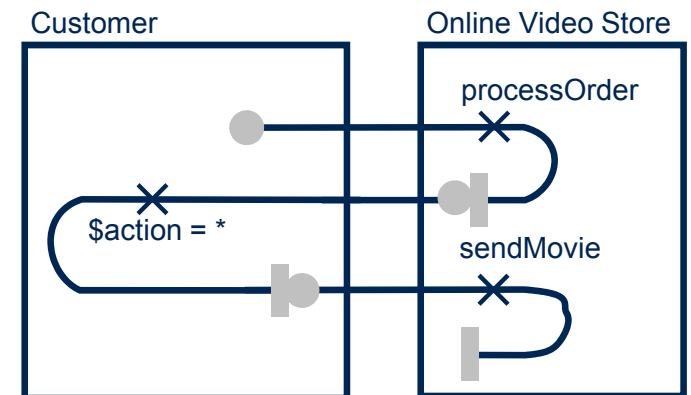
- Example 6 – Metadata Matching:

- The same as in example 5 but the responsibilities must additionally be annotated with <<Req/Rep>>



- Example 7 – Ignored End/Start Point & Variables:

- Matches a path segment where the path first must cross from component “Customer” to component “Online Video Store”, immediately followed by a series of three responsibilities: “processOrder” bound to component “Online Video Store”, variable “\*” bound to component “Customer”, and “sendMovie” bound to component “Online Video Store” (see interleaved composition)





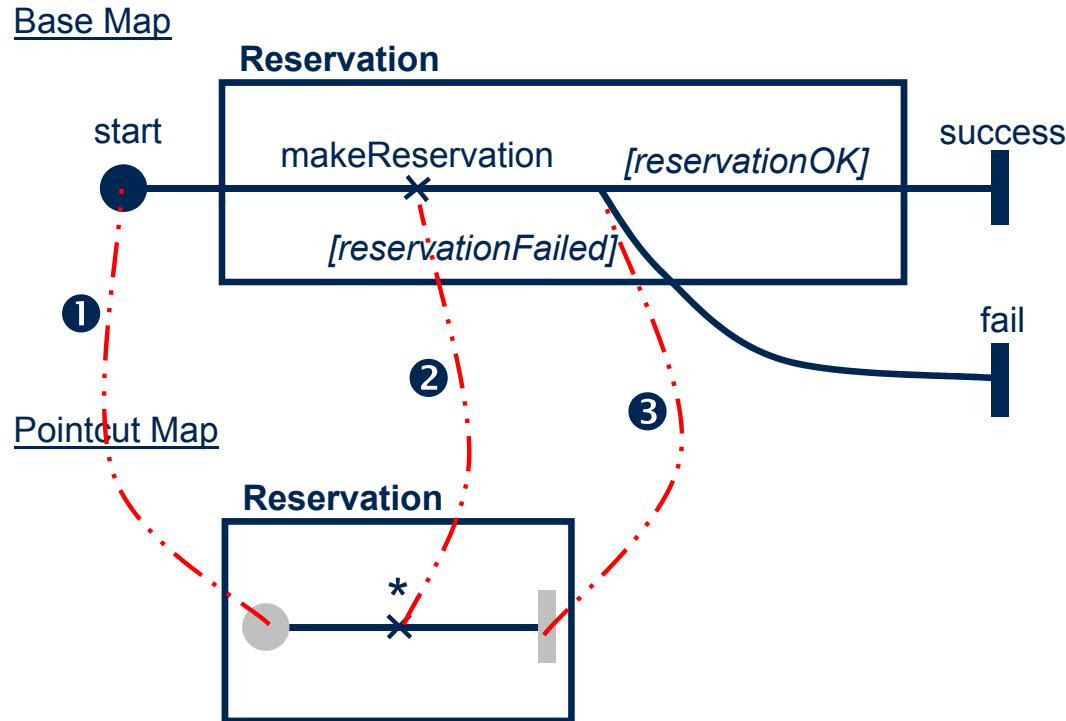
# Matching of AoUCM: Criteria

- Criteria for matching AoUCM pointcut expression
  - Type of model element must match
  - Name of the model element must match
  - Names of conditions, if applicable, must match (if the pointcut expression does not specify conditions, any condition may be matched)
  - Path direction between model elements and type of connection between model elements (regular or timeout branch) must match
  - Location of the model element in its component must match (either first, last, or any location in the component)
  - Component hierarchy must be compatible
  - Metadata of the pointcut element must be a subset of the metadata of the base element

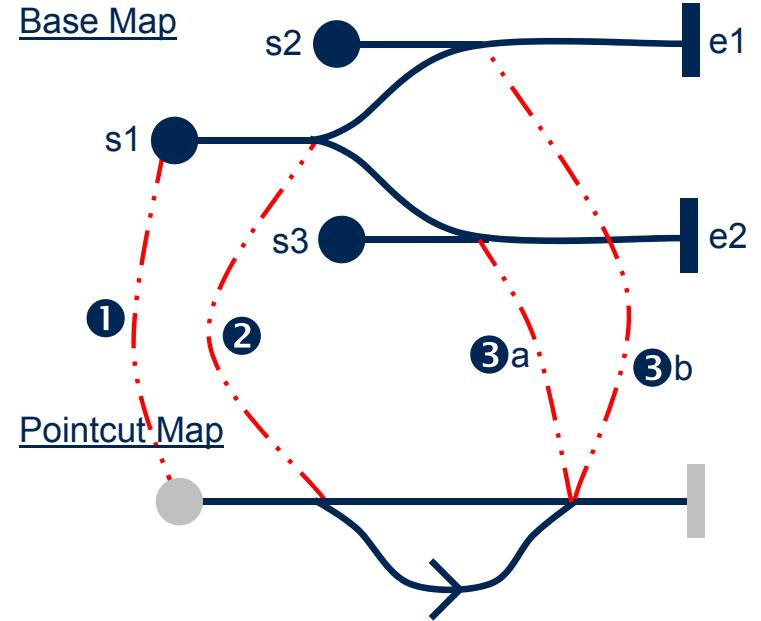


# Matching of AoUCM: Examples (1)

- Successful match



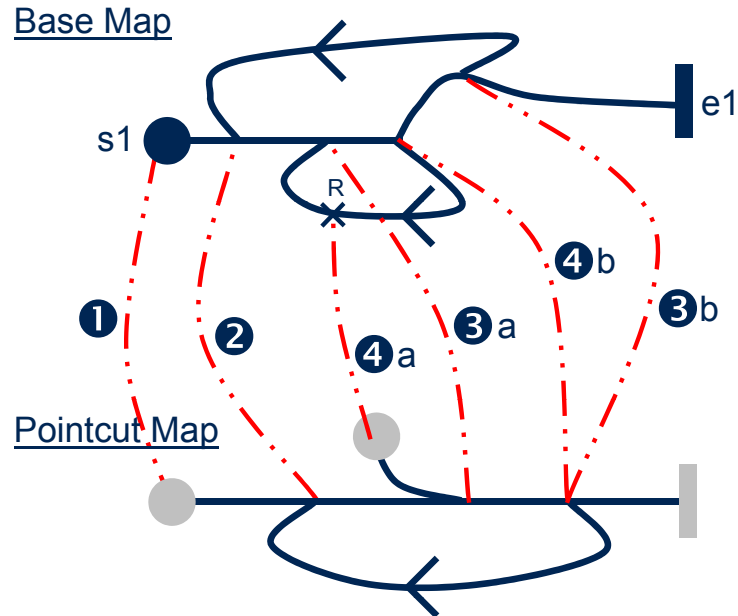
- Contradiction at same step



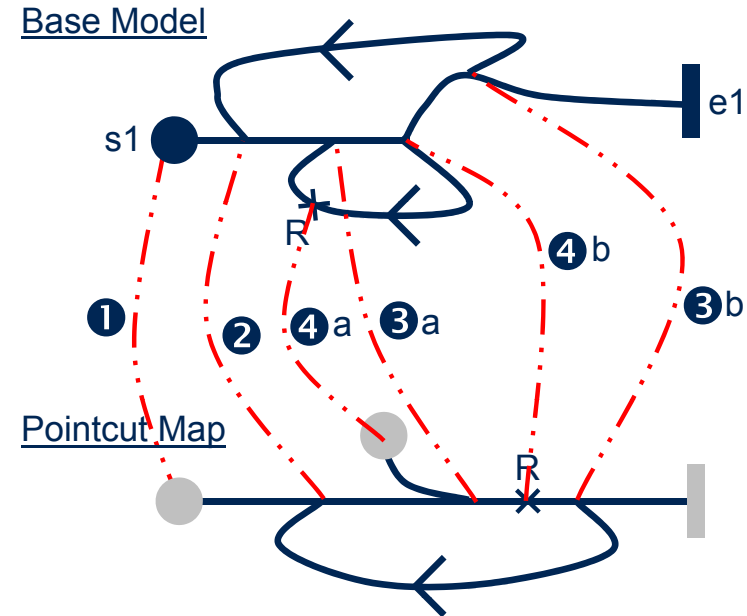
- The contradiction occurs at step 3 of the matching process because the same node cannot be matched to two different nodes
- The base map does therefore not match the pointcut expression and the aspect will not be applied to the base map

# Matching of AoUCM: Examples (2)

- Contradiction at different steps



- Failed Match (type mismatch)

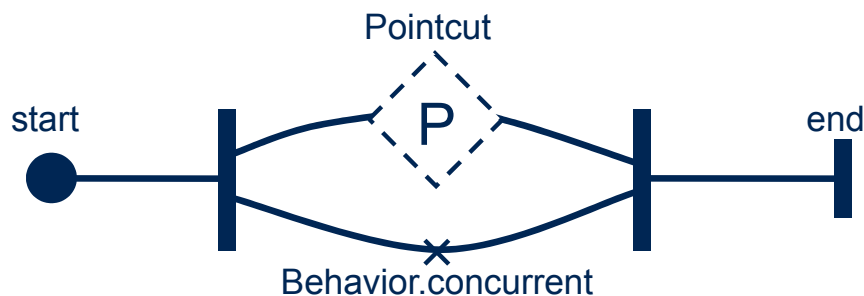
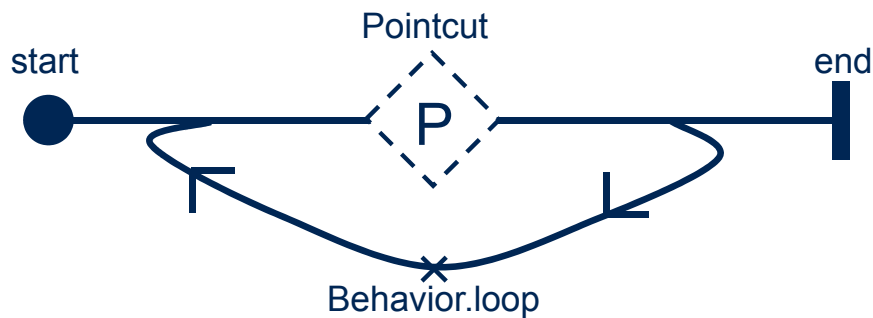
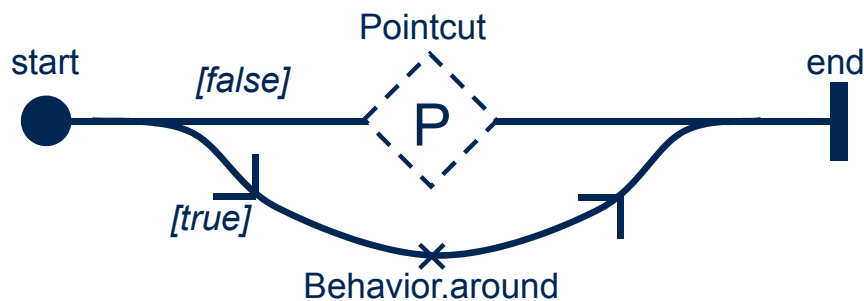


- The contradiction occurs between step 3b and 4b of the matching process because the same node cannot be matched to two different nodes
- The failed match occurs because the responsibility in the pointcut map cannot be matched against the OR-fork in the base map at step 4 of the matching process

# Composition of AoUCM: Composition Rules

- AoUCM can use **any** composition rule that can be described with the UCM notation, not just simple before and after composition rules
- The examples show replacement (around), loops, and concurrency

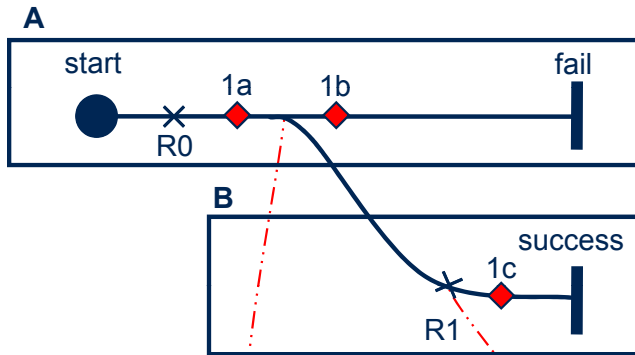
## Four Aspect Maps



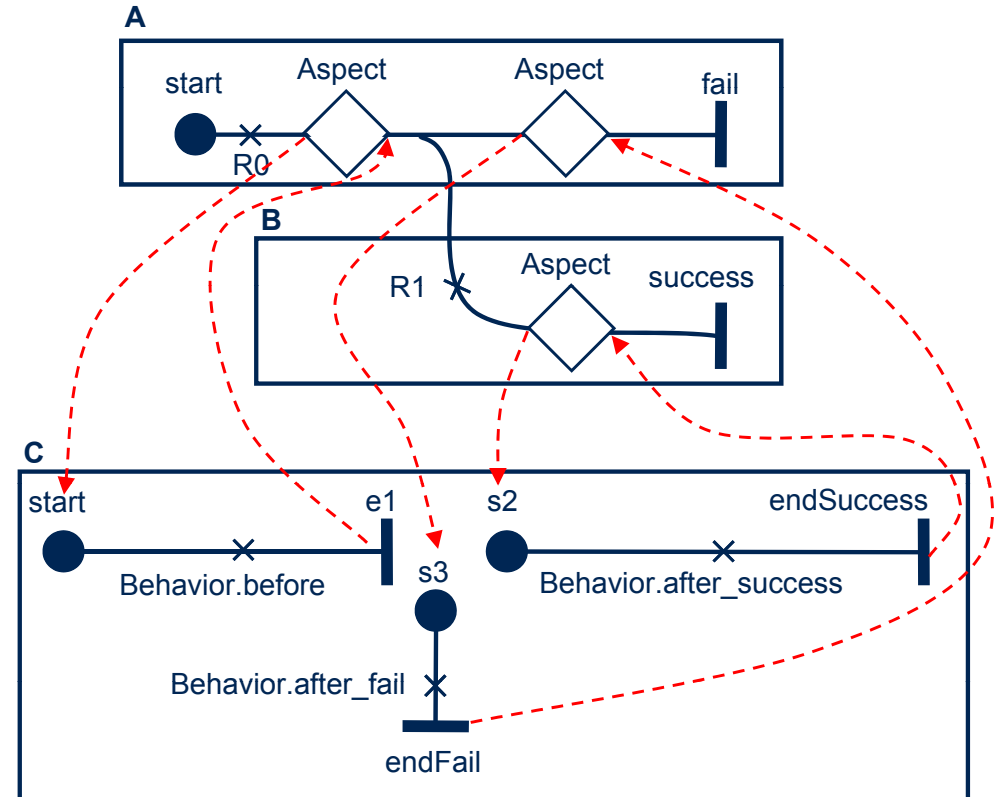


# Composition of AoUCM: AO or Traditional Stubs

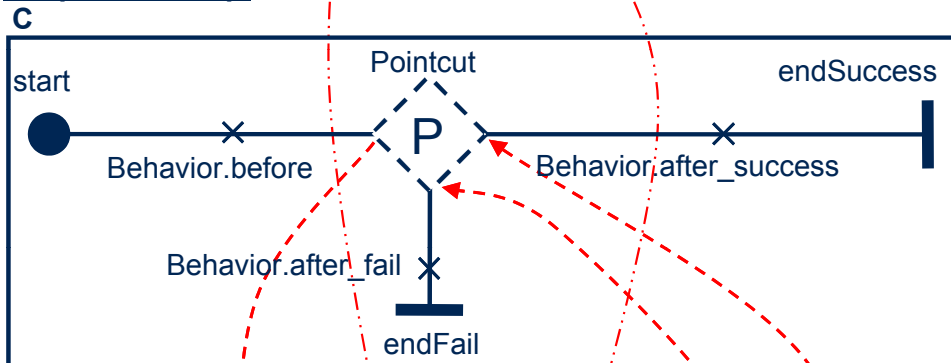
## Base Model



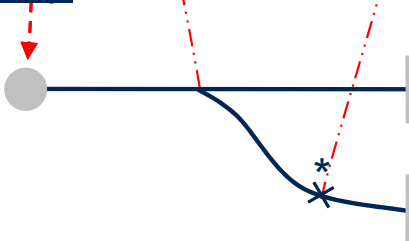
## Composed Traditional System



## Aspect Map



## Pointcut Map



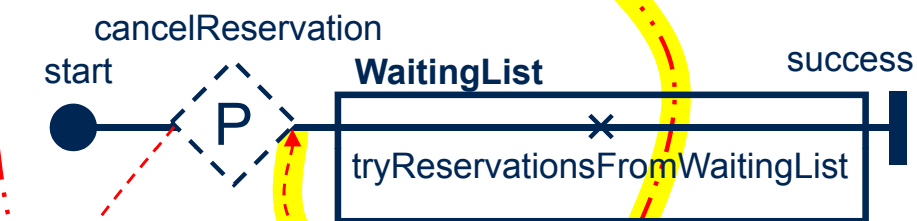
- Composition is achieved by inserting aspect markers or traditional stubs at the insertion points in the base model identified by the mapped pointcut expression

# Composition of AoUCM: Step 1

Base Map



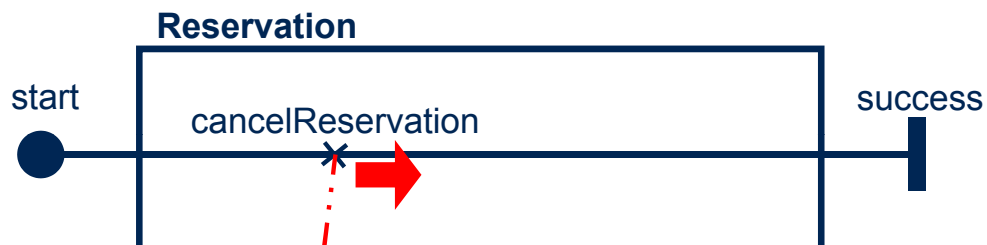
Waiting List Aspect (W)



Reservation



Base Map



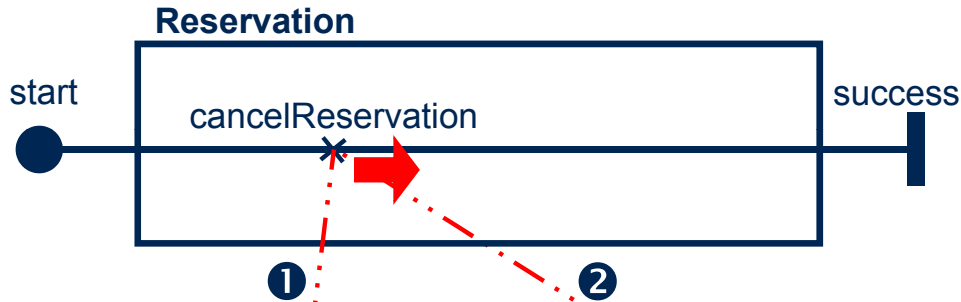
Waiting List Aspect (W)



- Composition Step 1
  - Do not consider empty path segments
  - Retain mappings
  - Retain **insertion point arrows**

# Composition of AoUCM: Steps 2 and 3

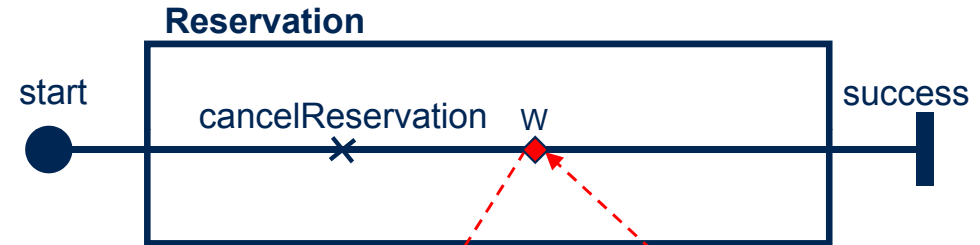
Base Map



Waiting List Aspect (W)



Base Map



Waiting List Aspect (W)



- Composition Step 2

- Scan for start/end points

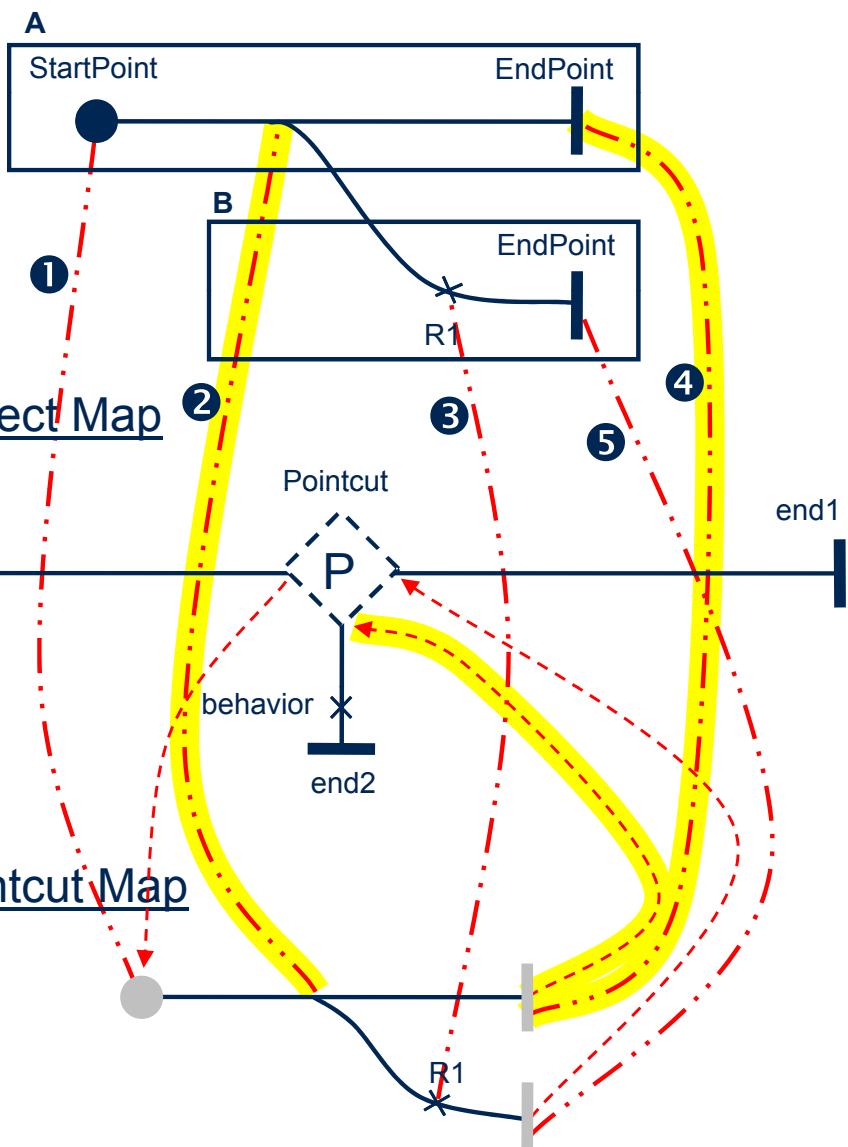
- Composition Step 3

- Insertion of **aspect marker** (**conditional aspect marker** if scenario may not continue)
- Convert mappings into plug-in bindings



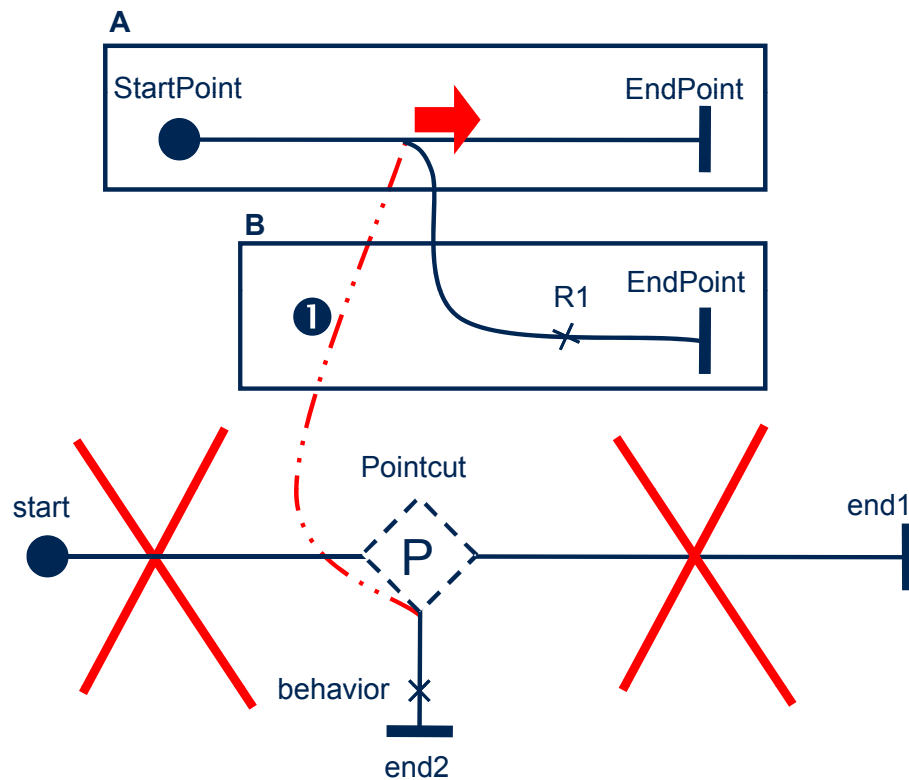
# Composition of AoUCM: Insertion Point Arrow

Base Map



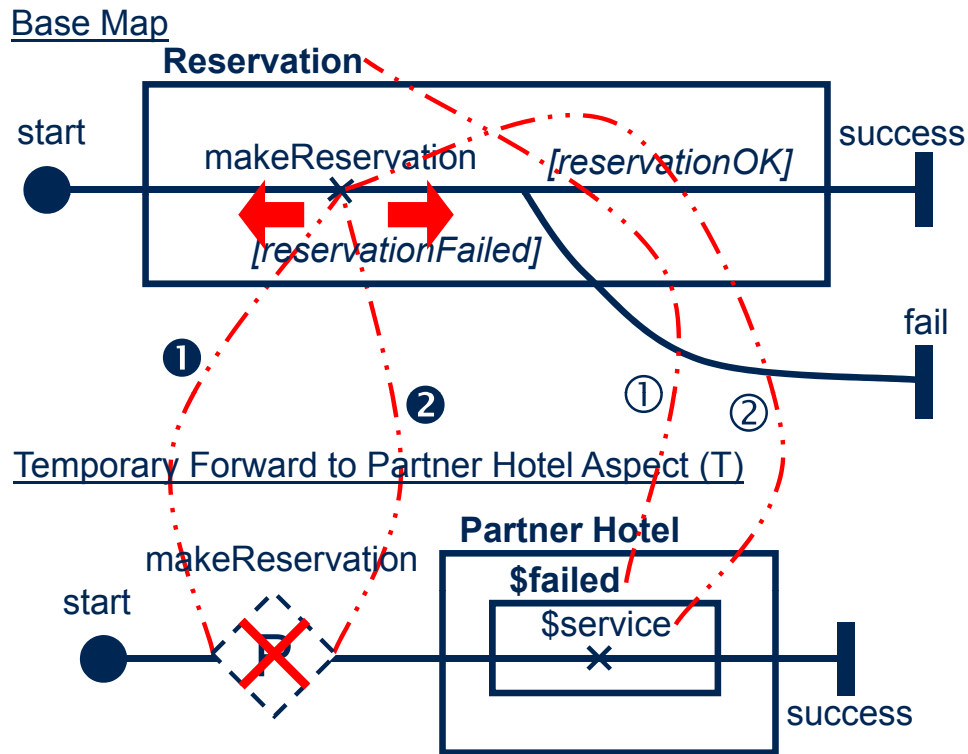
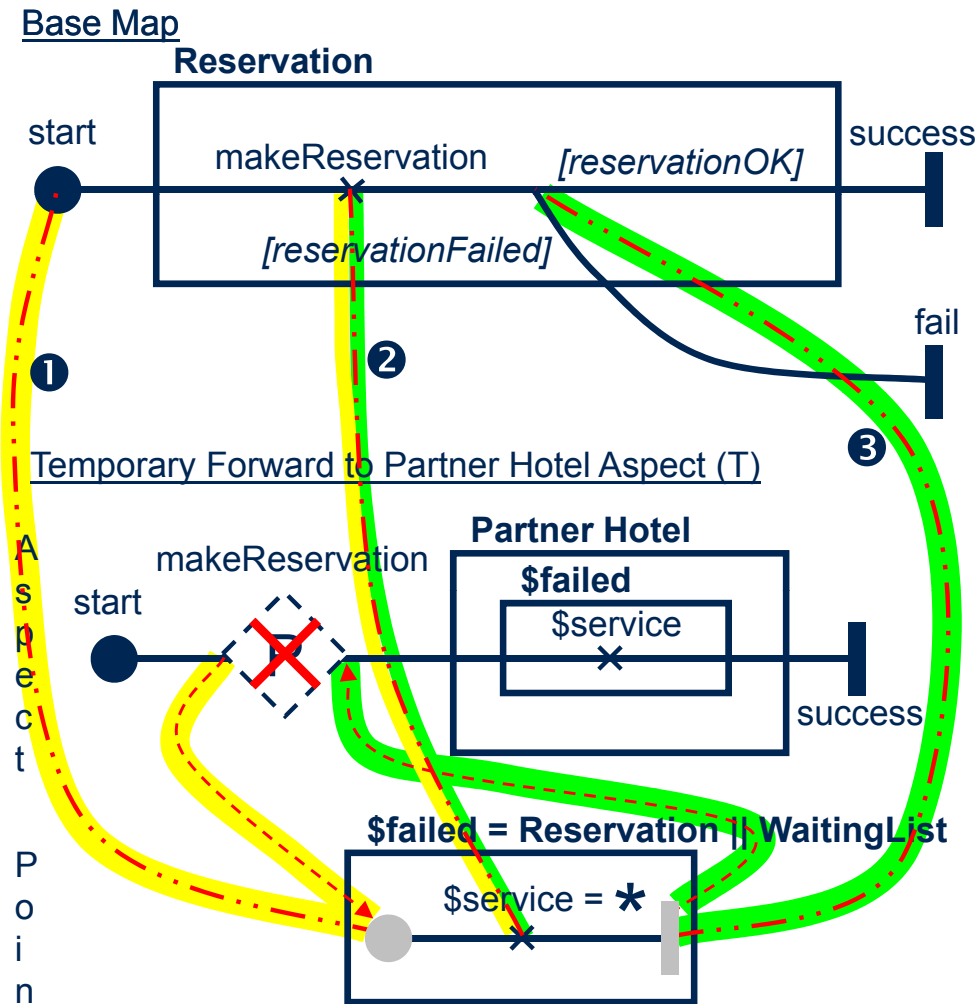
Aspect Map

Composition Step 1



- The insertion point arrow unambiguously identifies one of the insertion points of a join point

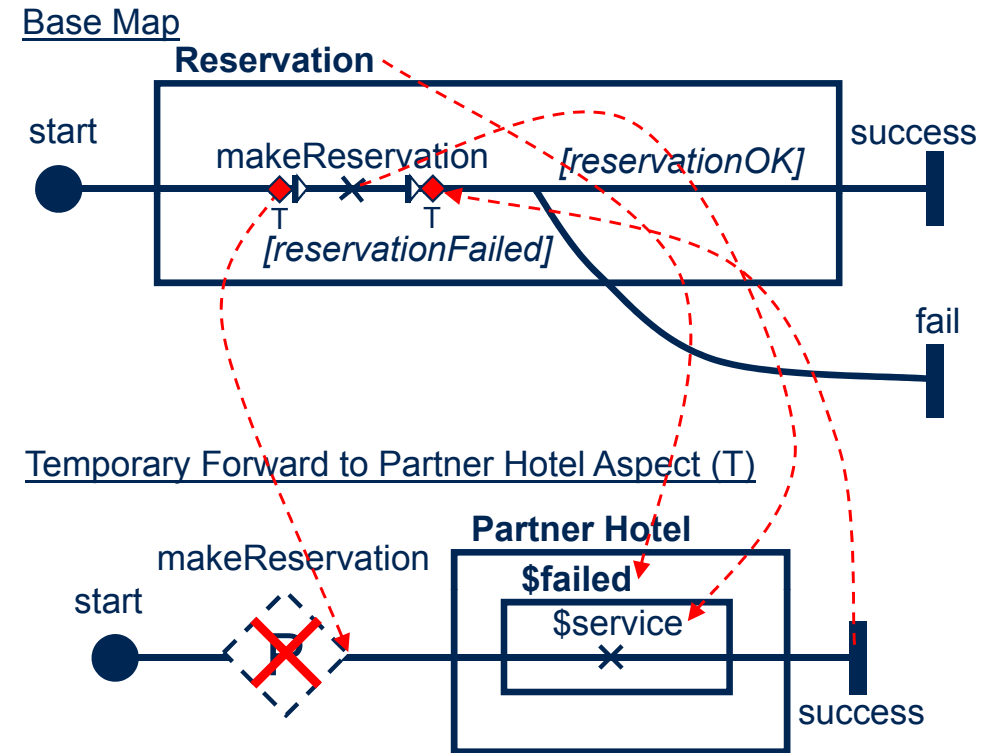
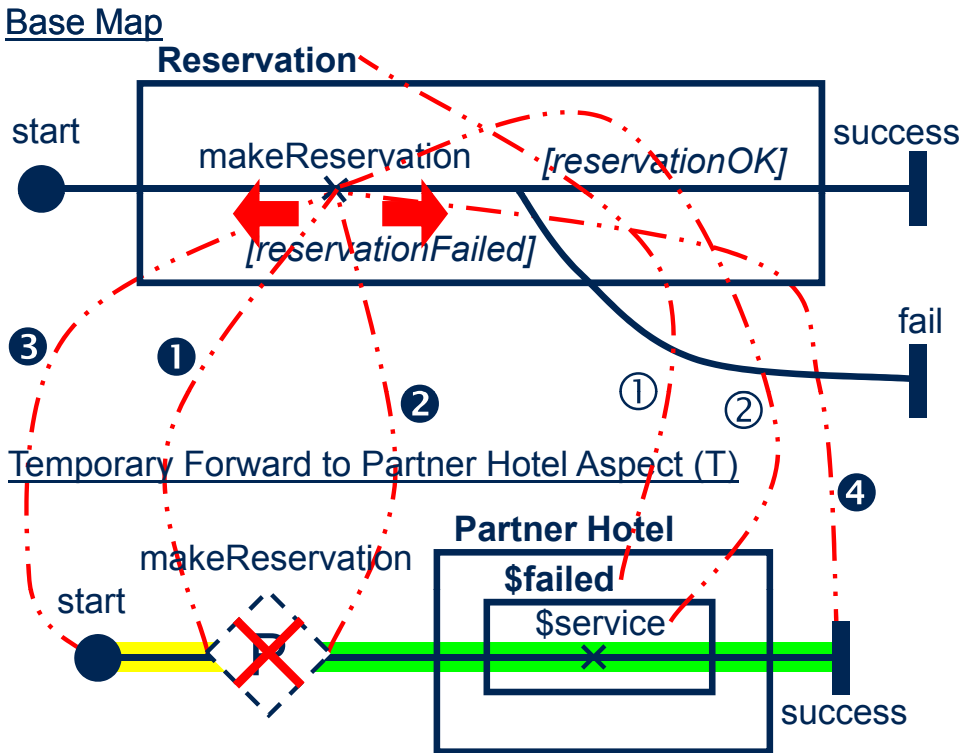
# Composition of AoUCM: Tunnels & Variables – Step 1



- Composition Step 1
  - Use empty path segments because of replacement
  - Retain mappings
  - Retain insertion point arrows



# Composition of AoUCM: Tunnels & Variables – Step 2 and 3



- Composition Step 2

- Scan for start/end points

- Composition Step 3

- Insertion of **tunnel entrance** and **tunnel exit aspect marker**
- Convert mappings into plug-in bindings
- Establish metadata

**responsibility plug-in binding (both T):**

makeReservation → \$service

**component plug-in binding (both T):**

Reservation → \$failed

**tunnel entrance T metadata:** 14 group 133

**tunnel exit T metadata:** 14 group 133

(<aspect map ID> group <number>)



# Composition of AoUCM: Interleaved

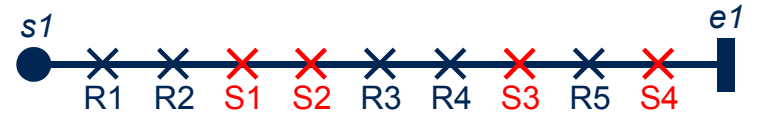
Scenario One



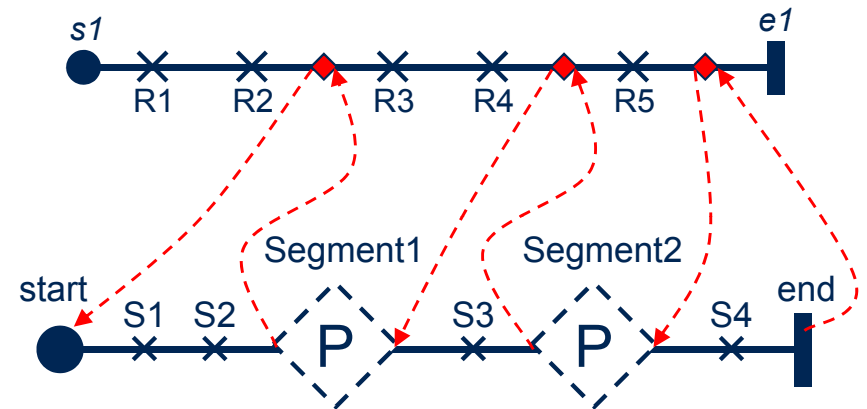
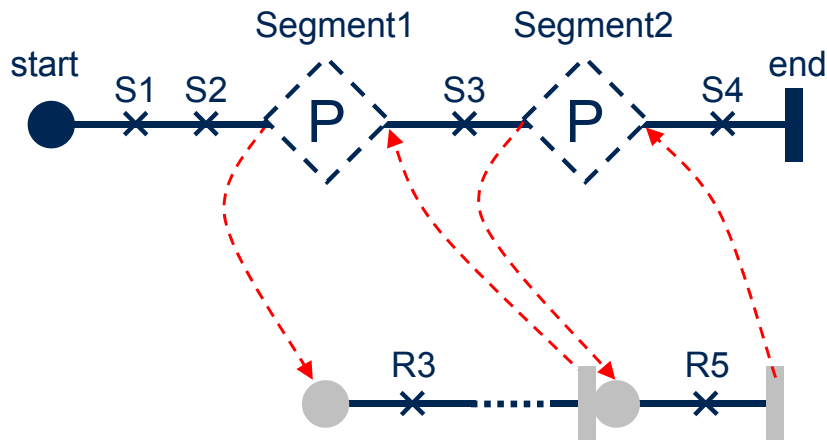
Scenario Two



Interleaved Scenario



Interleaved Aspect (I)



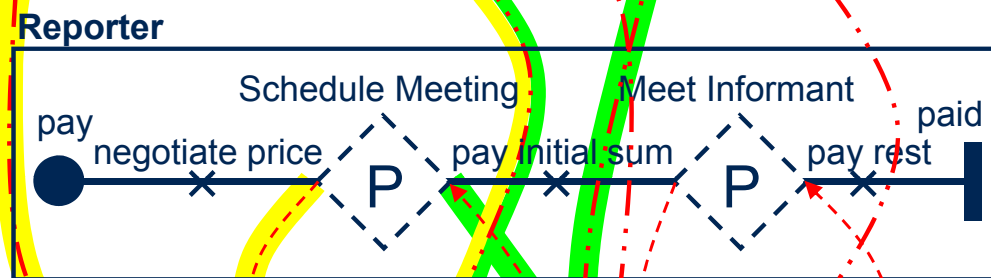
# Composition of AoUCM: Interleaved – Step 1

Base Map (Meet Informant)

Reporter



Pay Informant (P)  
Reporter



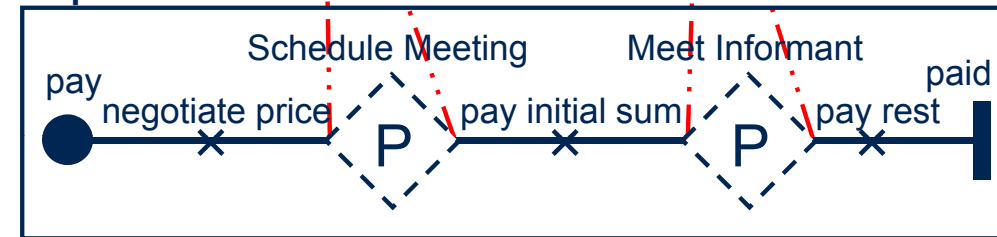
Base Map (Meet Informant)

Reporter



Pay Informant (P)

Reporter



- Prepare for Composition
  - Duplicate mappings for end/start point pair

- Composition Step 1
  - Do not consider empty path segments
  - Retain mappings
  - Retain insertion point arrows

# Composition of AoUCM: Interleaved – Step 2 and 3

Base Map (Meet Informant)

Reporter



5 1 2 3 4 6

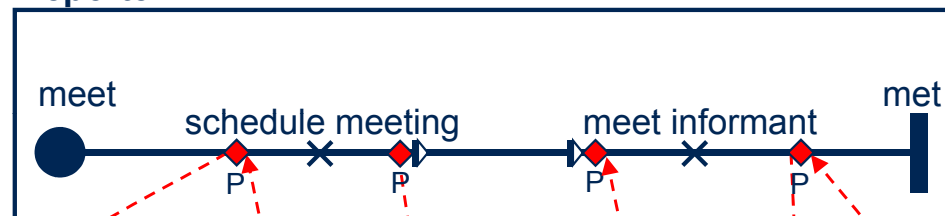
Pay Informant (P)

Reporter



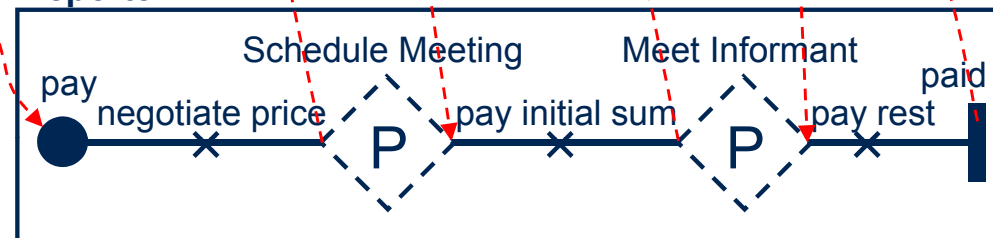
Base Map (Meet Informant)

Reporter



Pay Informant (P)

Reporter



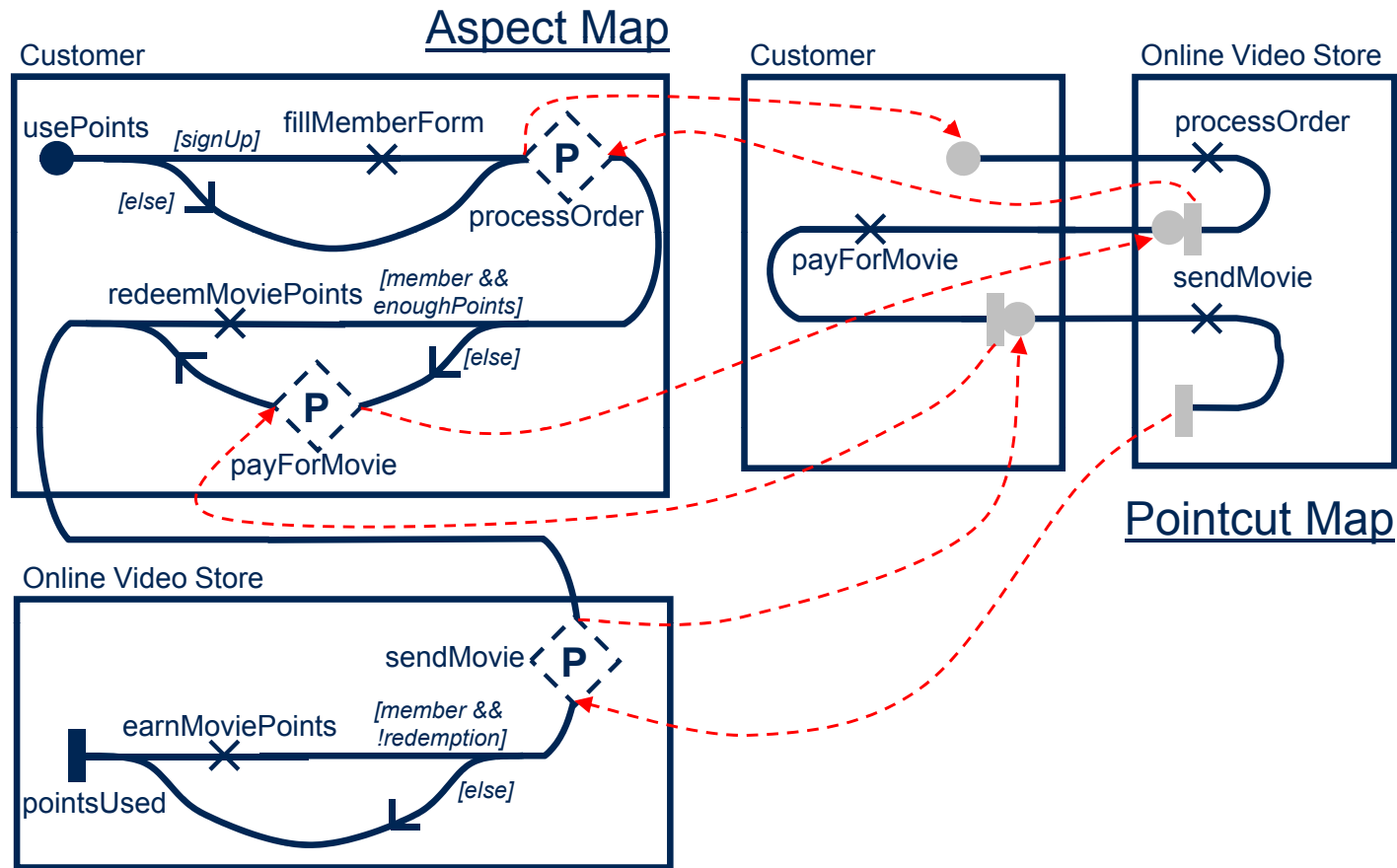
regular P metadata (both times): 89 group 14  
 tunnel entrance P metadata: 89 group 14  
 tunnel exit P metadata: 89 group 14

- Composition Step 2
  - Scan for start/end points

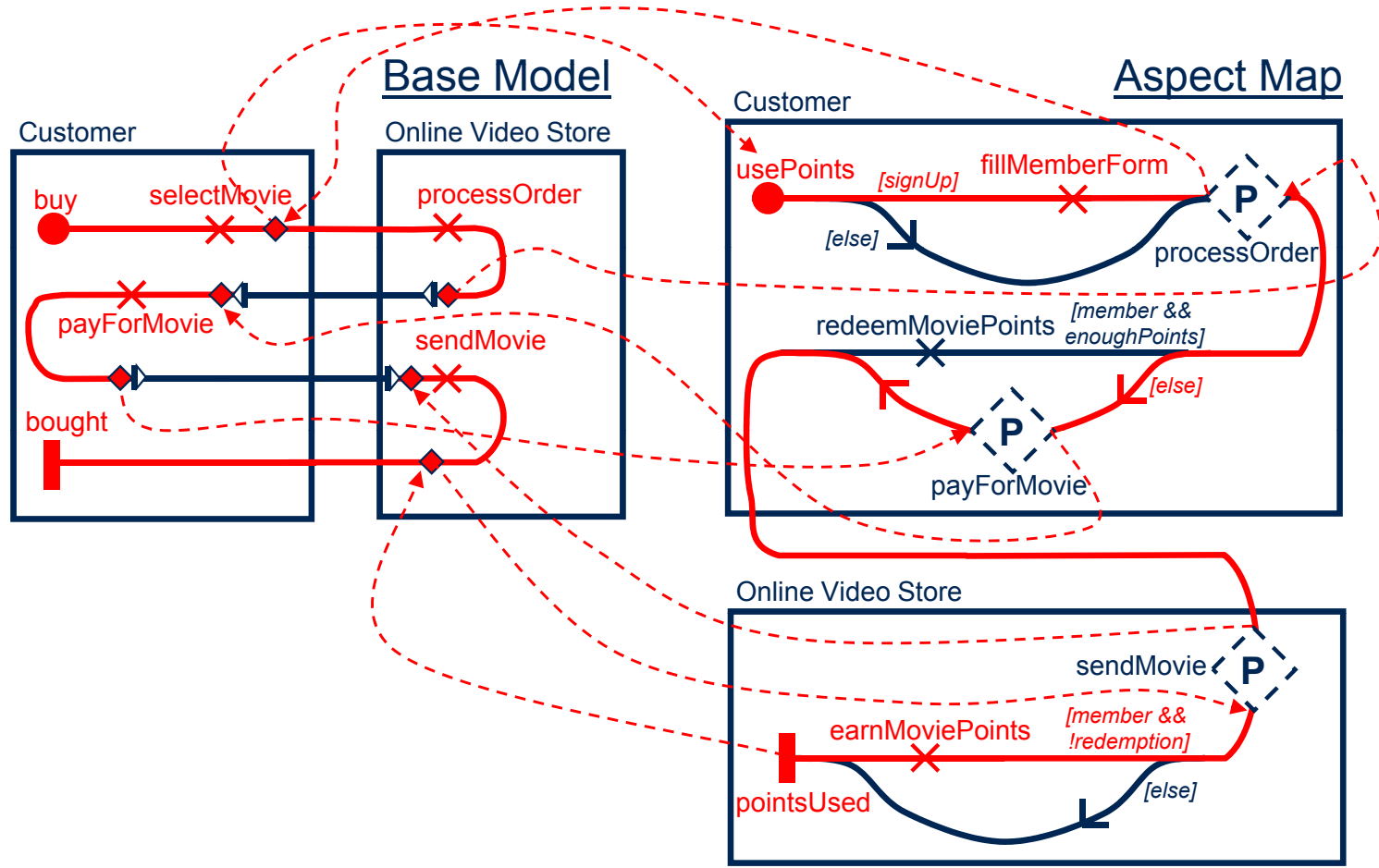
- Composition Step 3
  - Insertion of aspect markers
  - Convert mappings into plug-in bindings
  - Establish metadata



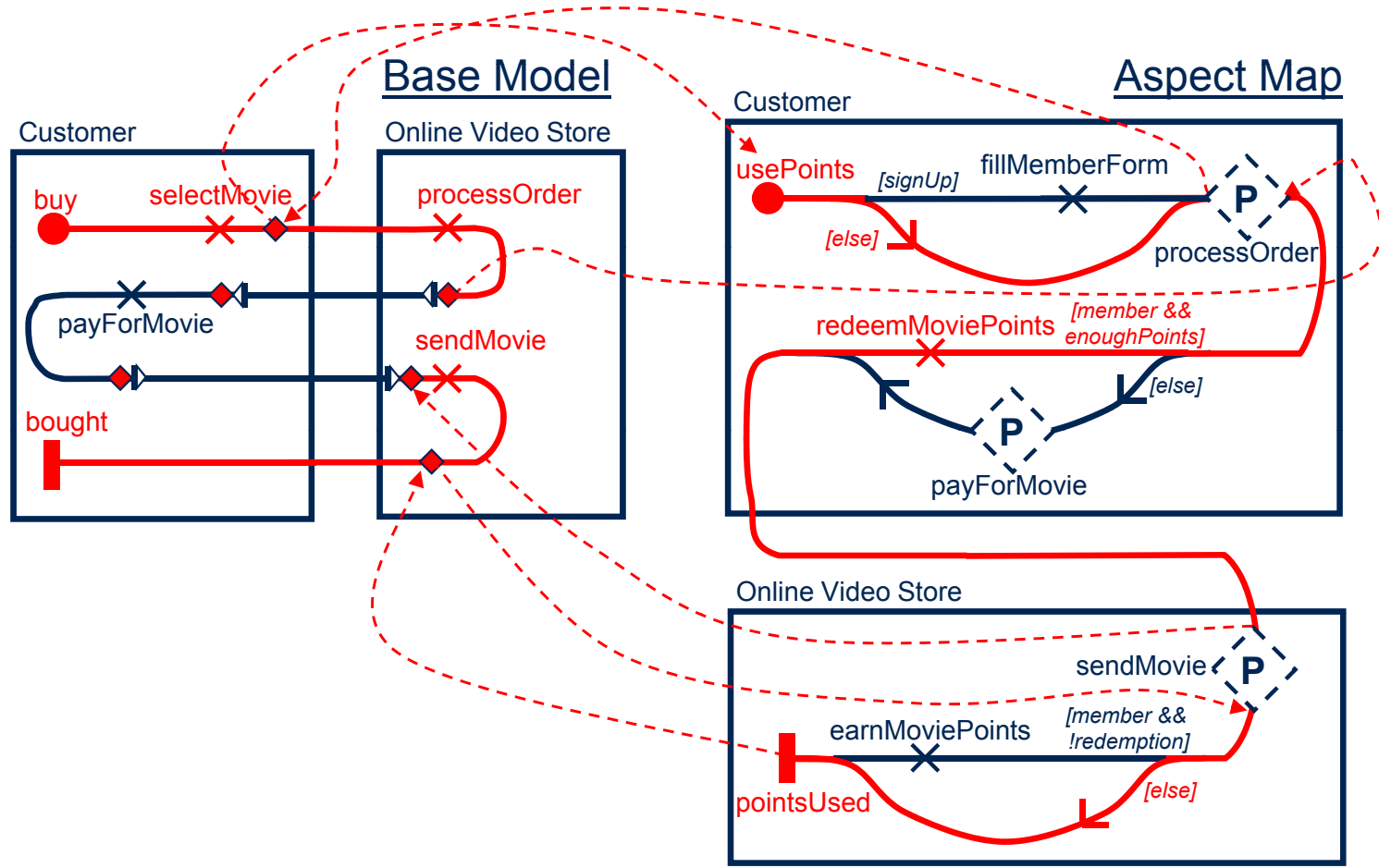
# Composition of AoUCM: Example – Interleaved (1)



# Composition of AoUCM: Example – Interleaved (2)



# Composition of AoUCM: Example – Interleaved (3)



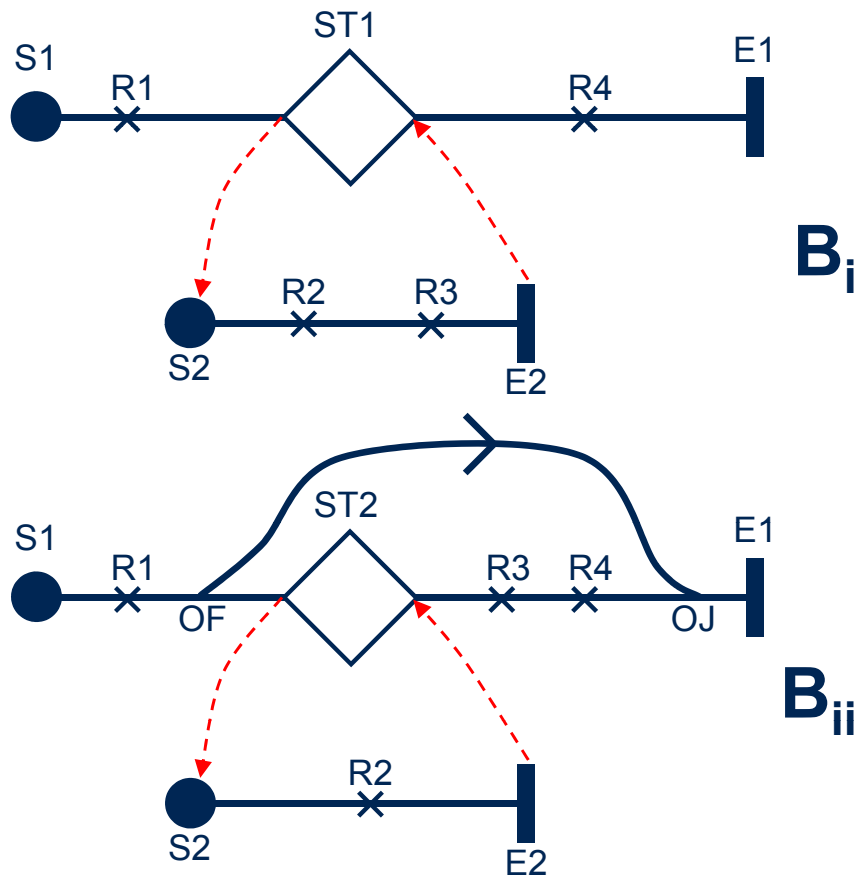


# Enhanced Matching Based on Semantics

- "Whitespace" in the base model is not matched
  - Empty point
  - Direction arrow
  - OR-join
  - End point connected to a start point without a condition
  - Static stub/aspect marker including the start and end points used in its plug-in bindings (in some cases, the stubs and start/end points are taken into account)
    - Start points must not have conditions
- Dynamic stubs, synchronizing stubs, and dynamic aspect markers are interpreted as their flattened equivalent
  - Allows AND-forks as well as OR/AND-joins to be matched against these model elements
- Waiting place with condition is equivalent to start point with condition

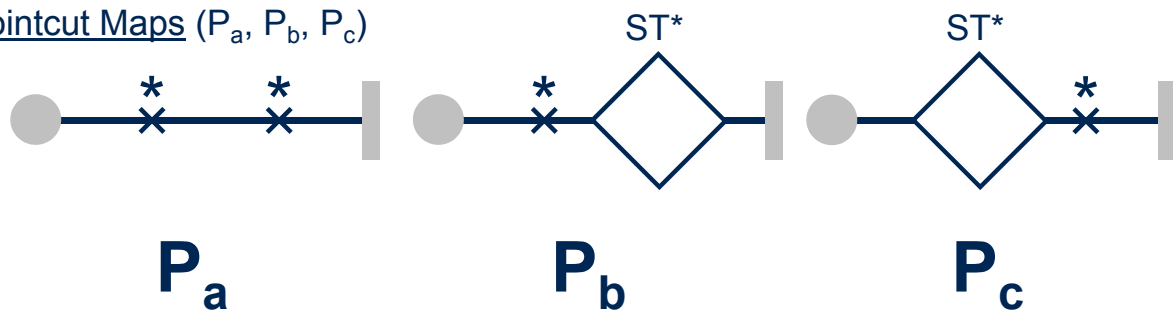
# Enhanced Matching Based on Semantics: Example

Base Maps ( $B_i, B_{ii}$ )



	$P_a$	$P_b$	$P_c$
$B_i$	S1-R1-R2-R3; S2-R2-R3-E2; R2-R3-R4-E1	S1-R1-ST1-R4	R1-ST1-R4-E1
$B_{ii}$	S2-R2-R3-R4; ST2-R3-R4-OJ	No match	OF-ST2-R3-R4

Pointcut Maps ( $P_a, P_b, P_c$ )

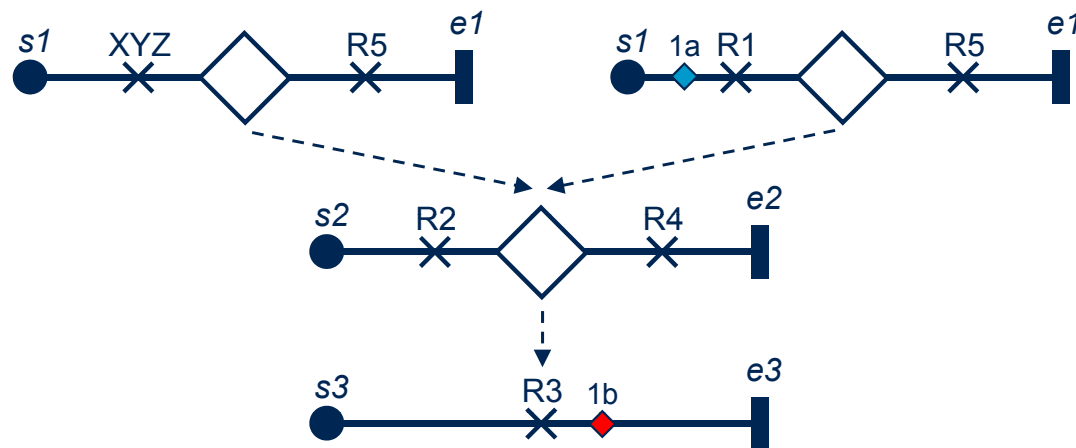


- If a stub without plug-in map exists on the pointcut map, then it must be matched with a stub in the base

# Enhanced Composition Based on Semantics: Issues

- Composition with shared plug-in maps

## UCM Model



## Aspect Map / AoView



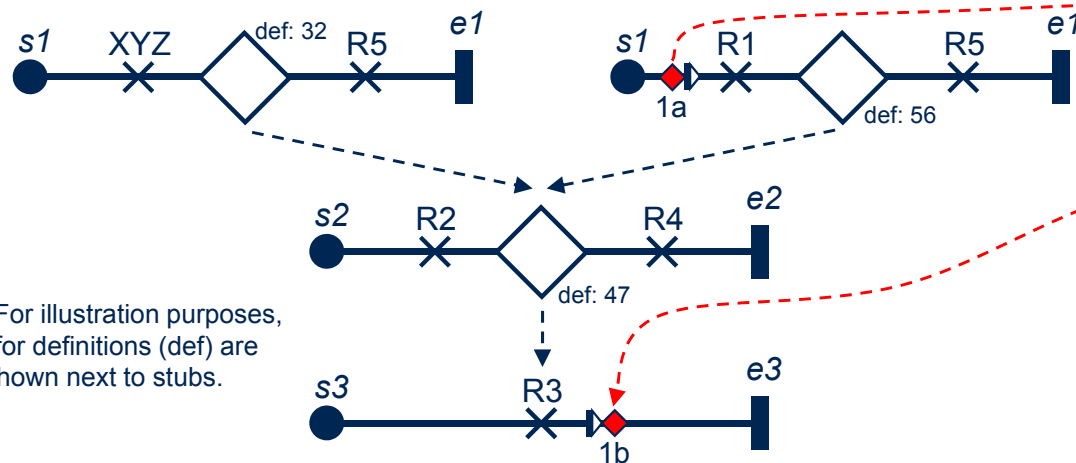
1a metadata: provides group 23  
1b metadata: requires group 23

## Pointcut Map



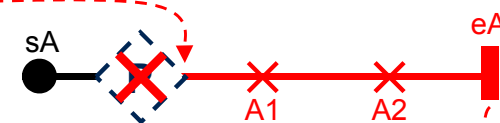
- Composition with lost hierarchies

## UCM Model



Note: For illustration purposes, IDs for definitions (def) are shown next to stubs.

## Aspect Map / AoView



1b metadata: context 56 47

## Pointcut Map

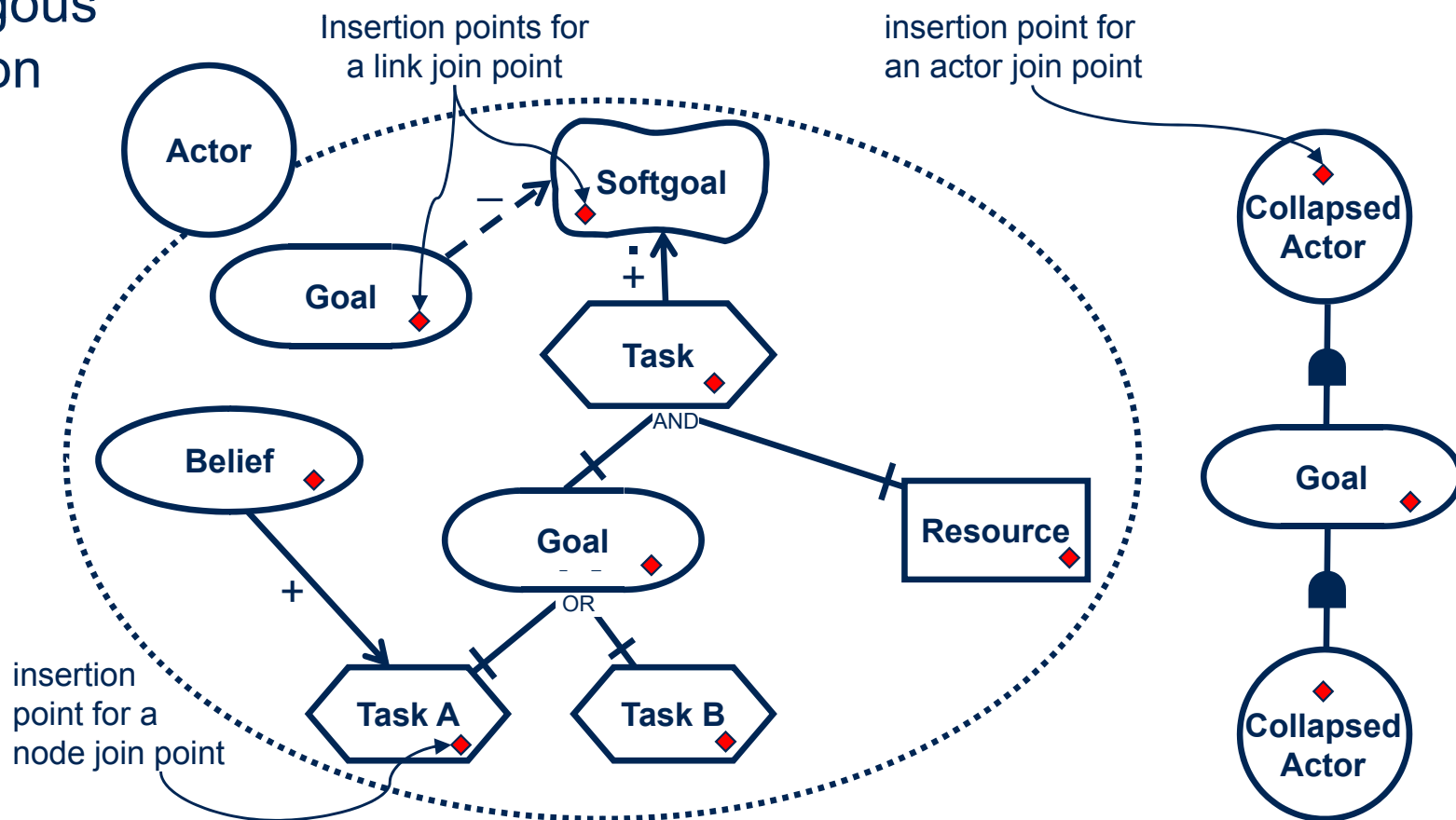




# Aspect-oriented GRL (AoGRL)

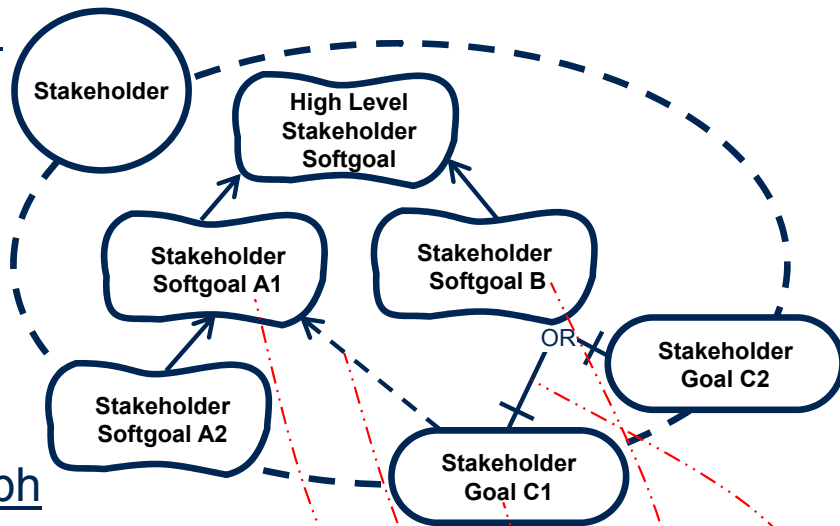
# Introduction to AoGRL: Join Point Model

- In order to unify aspect concepts with GRL concepts, **join points**, **aspects**, and **pointcuts** have to be defined in GRL terms
- The AoGRL **join point model** includes **all GRL nodes** (i.e. intentional elements) and **links** optionally residing within the boundary of an actor
- This is analogous to the definition of join points for AoUCM

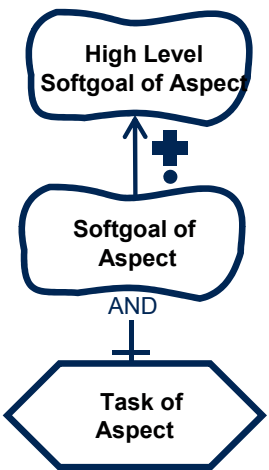


# Introduction to AoGRL: Aspect & Pointcut Graphs

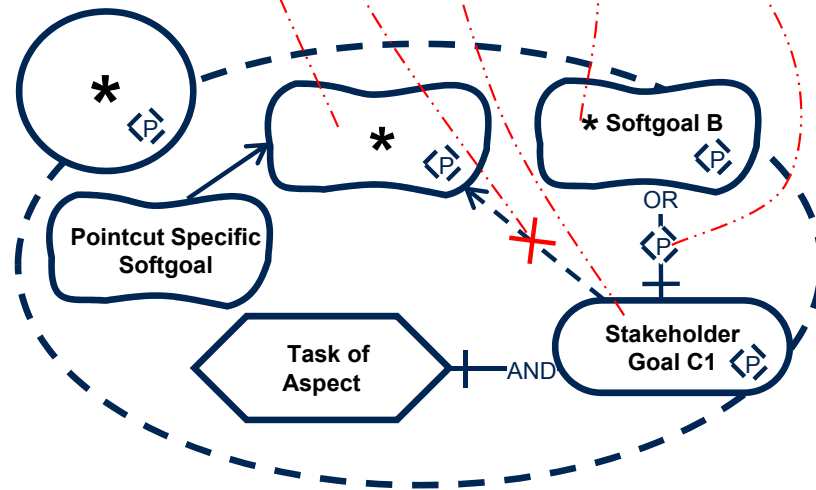
## Base Model



## Aspect Graph



## Pointcut Graph



Pointcut marker:  $\langle P \rangle$       Pointcut deletion marker:  $\times$

(mapping of pointcut expression to base model (long-dash-dot-dotted lines without arrowheads) only shown for illustration purposes – not part of AoGRL notation)

- An aspect is a group of goal graphs (some may be aspect graphs, some pointcut graphs)
- An **aspect graph** is a standard GRL graph that visually describes the aspectual properties
- An aspect graph focuses on one concern only (e.g. security)
- A **pointcut graph** is an annotated standard GRL graph that visually describes a parameterized pointcut expression
- The pointcut expression is matched against the base model in order to identify the join points that the aspect affects

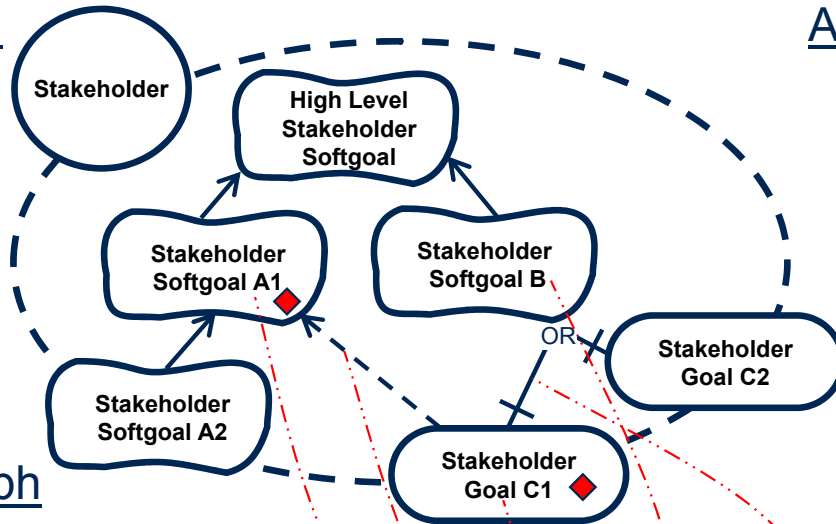


# Introduction to AoGRL: Pointcut Graphs vs. Maps

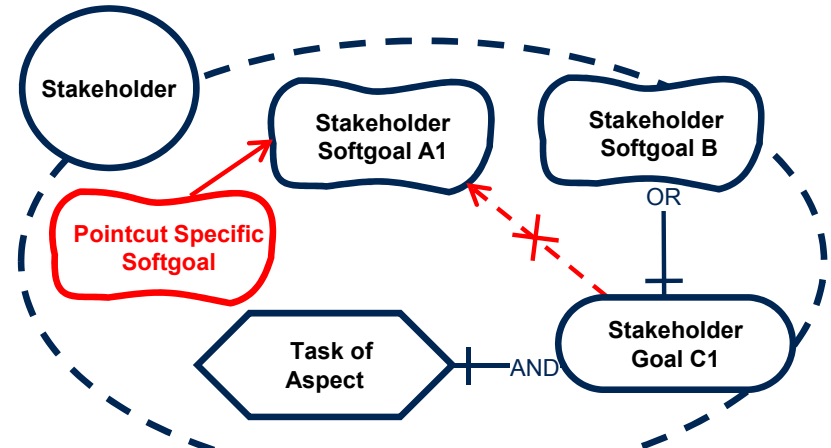
- Pointcut Map
  - Completely separate from aspect map – contains only pointcut expression
  - Allows for reuse of pointcut expression and aspectual properties
- Pointcut Graph
  - Contains pointcut expressions and some aspectual properties to specify the composition rule
  - Harder to reuse pointcut expression and aspectual properties
  - More inspired by graph transformation-based approaches such as MATA
  - Due to the nature of GRL models – highly interconnected

# Introduction to AoGRL: AoView

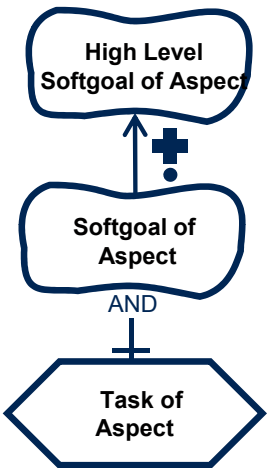
## Base Model



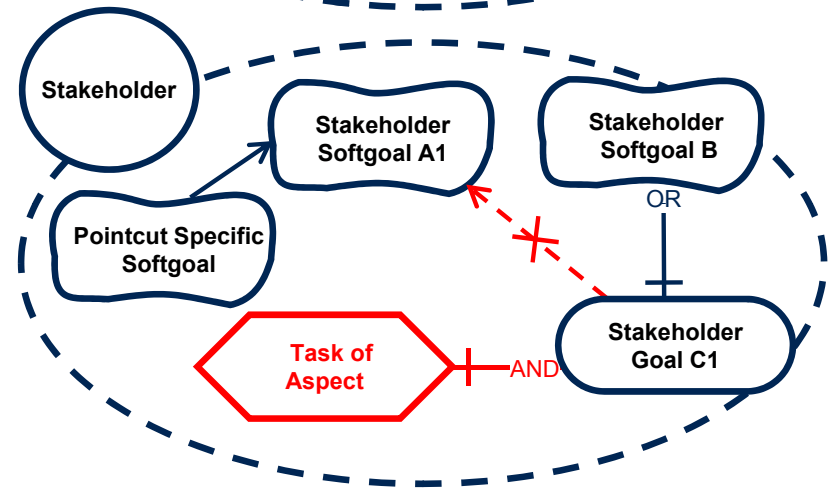
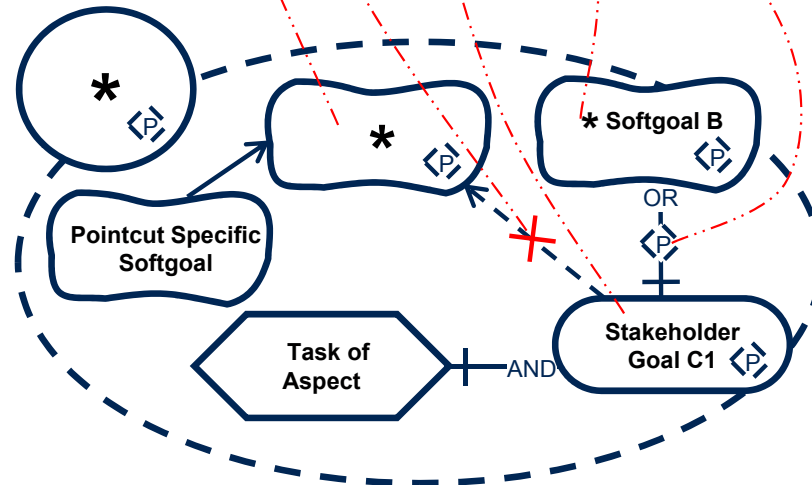
## AoViews



## Aspect Graph



## Pointcut Graph



Aspect marker: ◆ Pointcut marker: P Pointcut deletion marker: ✗



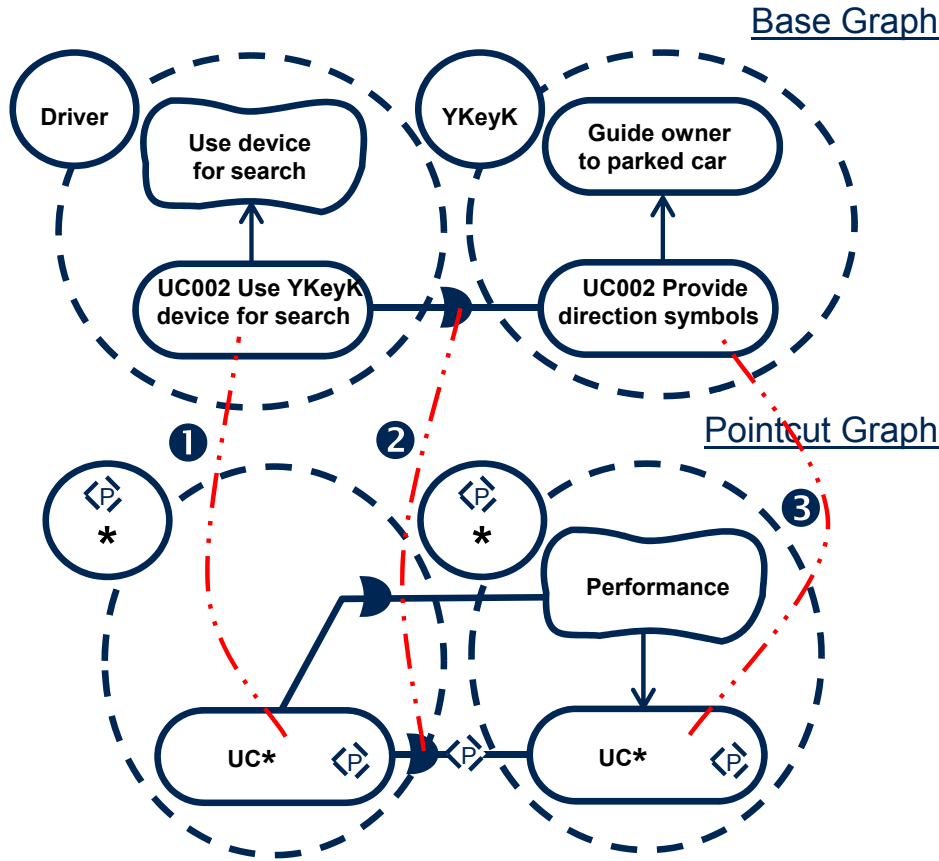
# Matching of AoGRL

- The matching of pointcut graphs with the base model is **analogous** to AoUCM
  - Parameterized pointcut expressions can contain the wildcard \* and logical operators for AND (&&), OR (||), and NOT (!)
  - More restrictive version of anything element in AoUCM
    - <<anytype>> element for intentional element or link
    - Decomposition chains are taken into account
- Criteria (also analogous to AoUCM)
  - Type of model element must match
  - Name of the model element must match
  - Type of link and link direction between model elements must match
  - GRL actor of the model element must be compatible
  - Metadata of the pointcut element must be a subset of the metadata of the base element

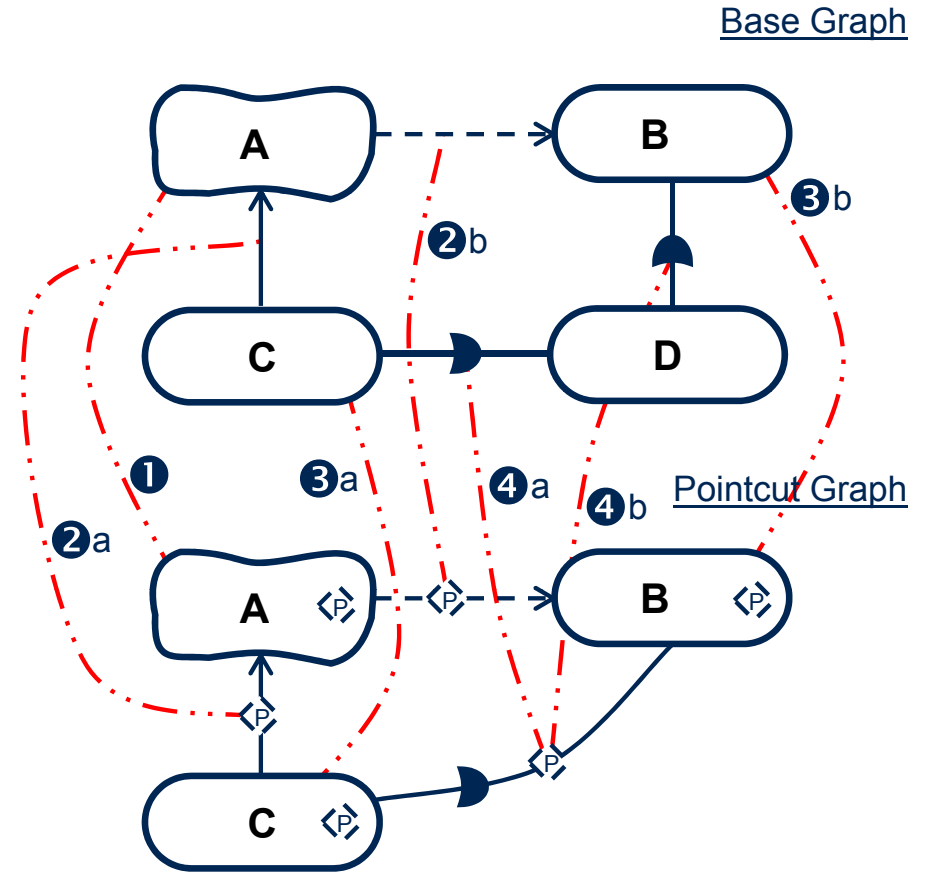


# Matching of AoGRL: Examples

- Successful match



- Contradiction at same step

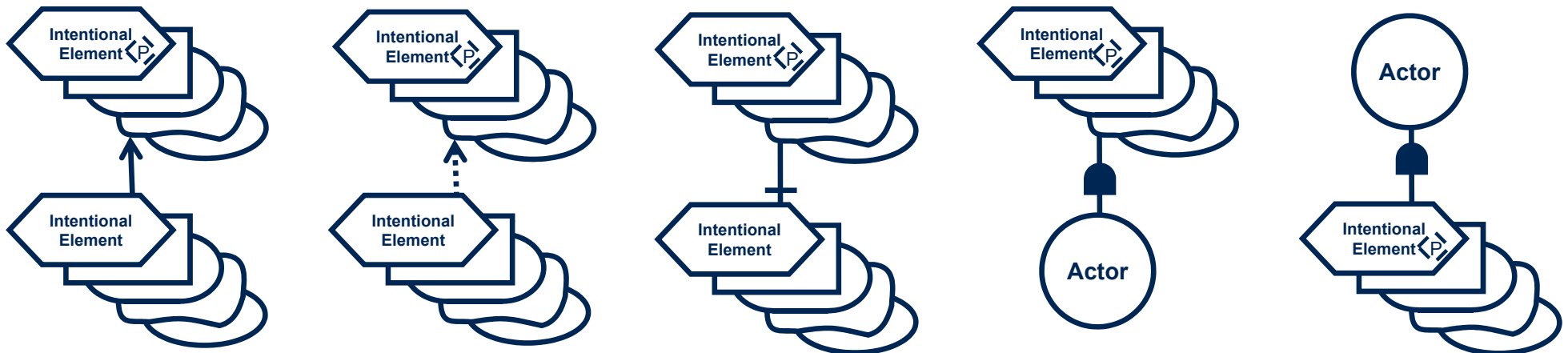


- The contradiction occurs at step 4 of the matching process because the same link cannot be matched to two different links

# Composition of AoGRL: Composition Rules

- AoGRL can use **any** composition rule that can be described with the GRL notation (i.e. all four link types – contribution, correlation, decomposition, and dependency – can be used in a composition rule)

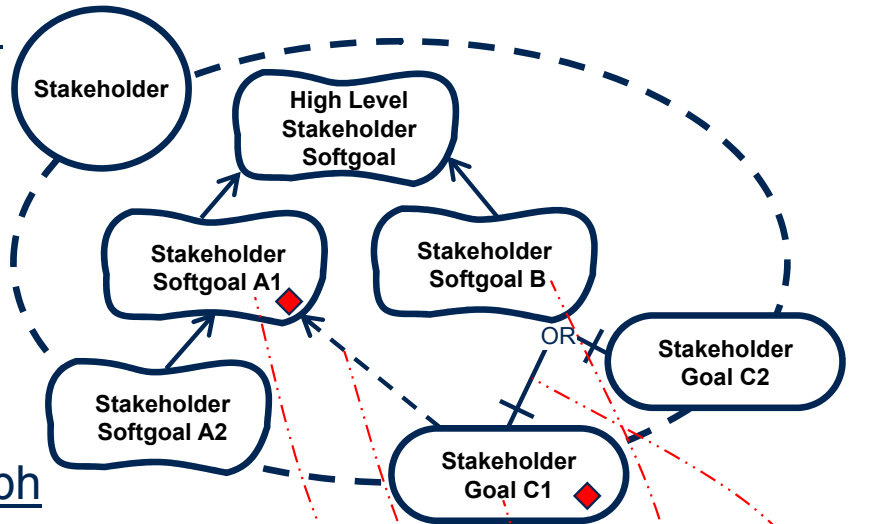
## Five Pointcut Graphs



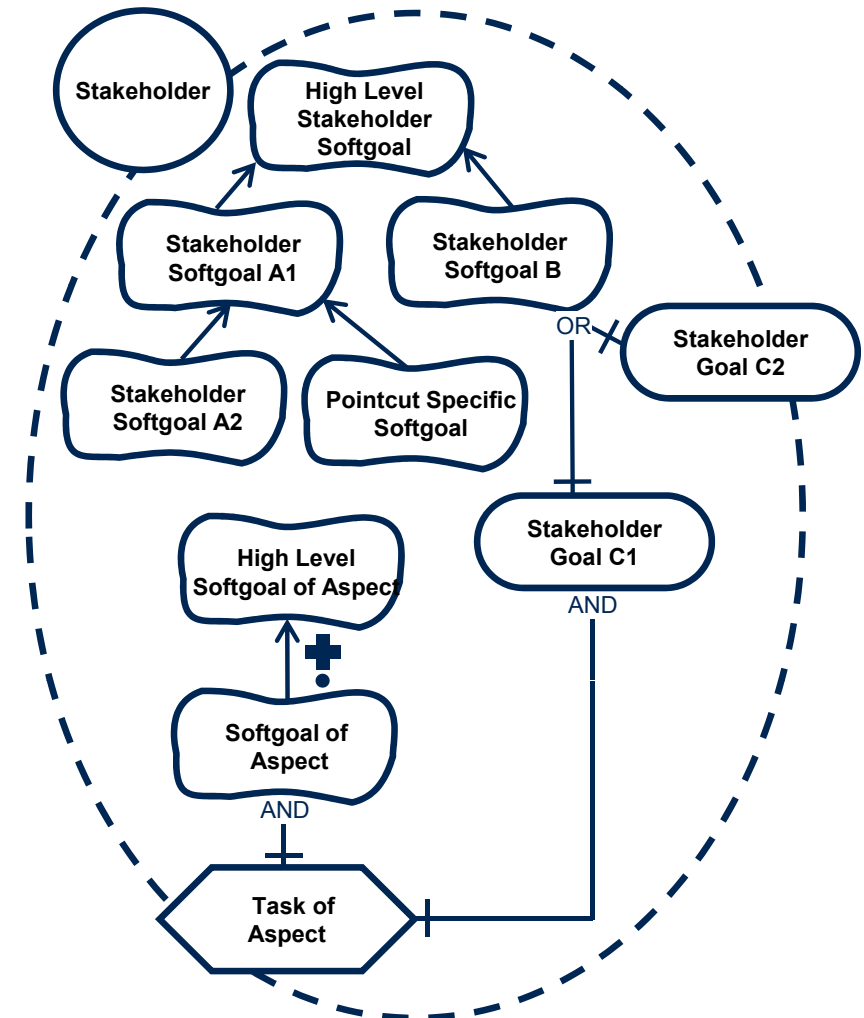


# Composition of AoGRL: AO or Traditional GRL

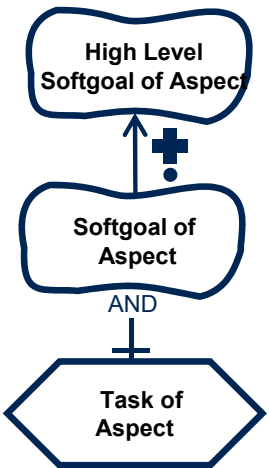
Base Model



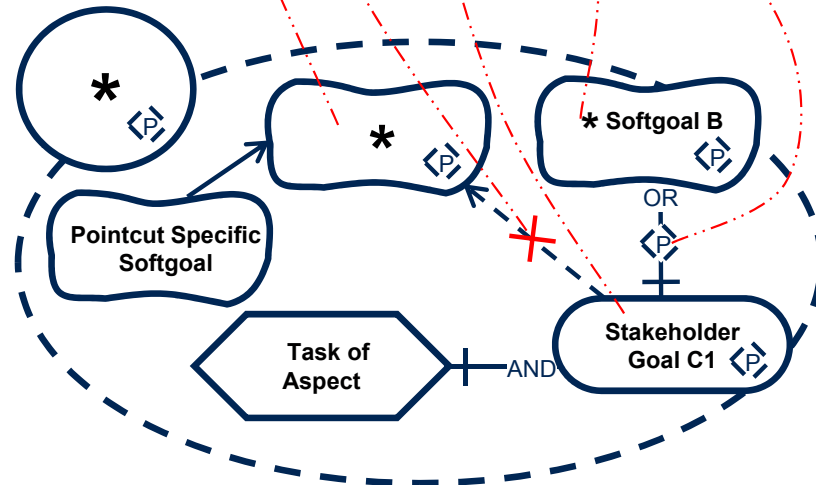
Composed Traditional System



Aspect Graph



Pointcut Graph

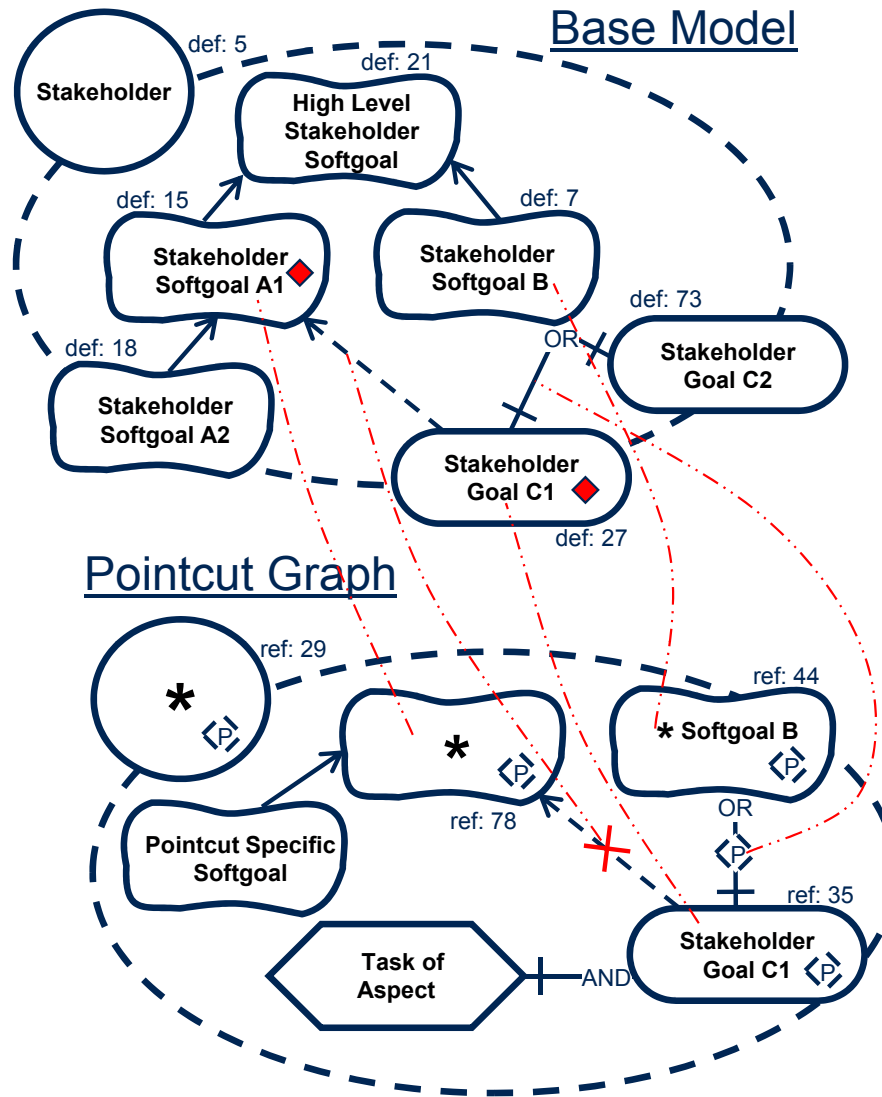


Aspect marker: ◆ Pointcut marker: P Pointcut deletion marker: ✗



# Composition of AoGRL: Step 1 and 2

- Result of composition is encoded with metadata



## Step 1

**Stakeholder Softgoal A1 metadata:**

aspect marker

**Stakeholder Goal C1 metadata:**

aspect marker

## Step 2

**Stakeholder Softgoal A1 metadata:**

aspect marker

to 78 match 1

**Stakeholder Goal C1 metadata:**

aspect marker

to 35 match 1

**Pointcut Graph metadata:**

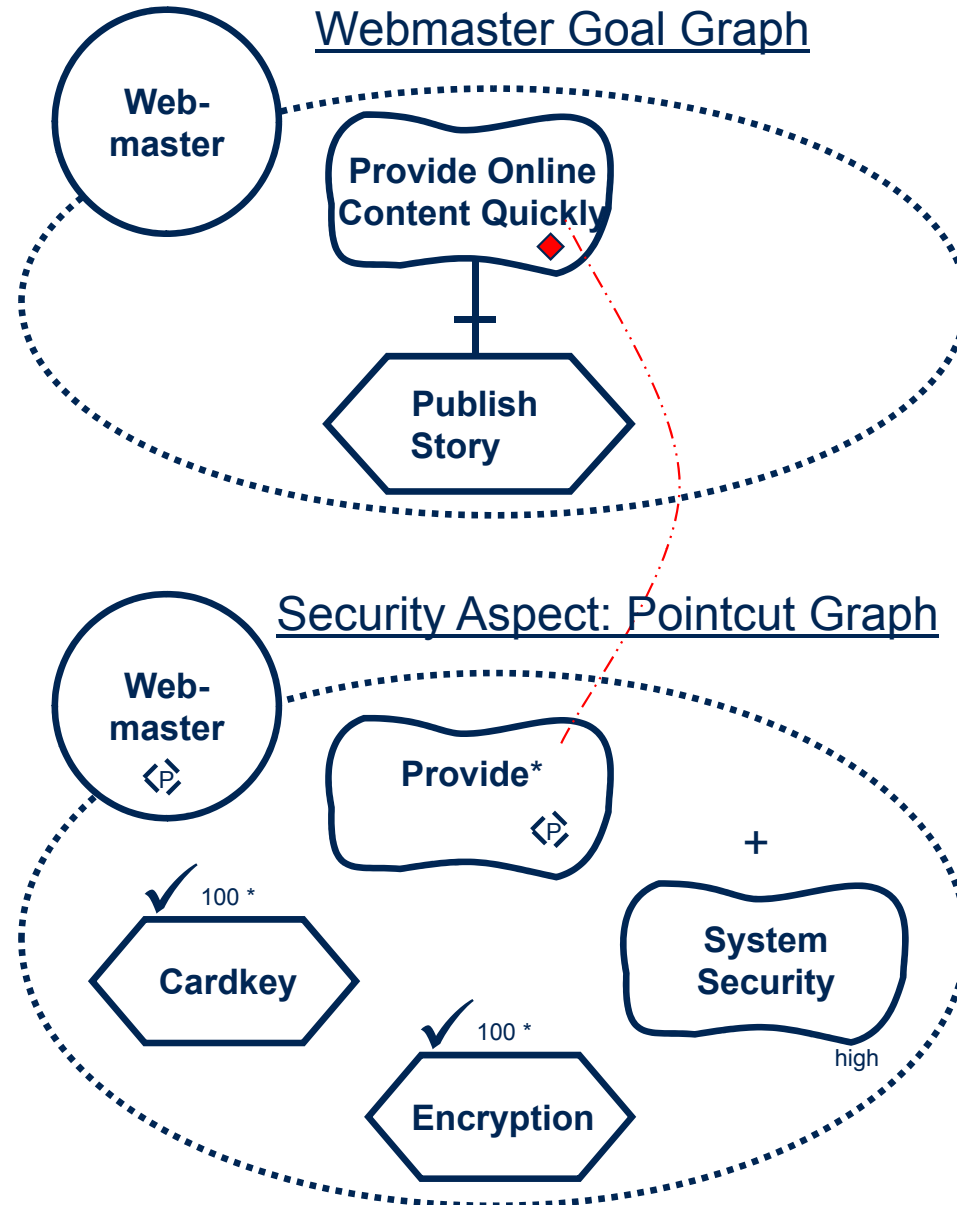
match 1 78 = 15

match 1 44 = 7

match 1 29 = 5

Note: For illustration purposes, IDs for definitions (def) and references (ref) are shown next to actors and intentional elements.

# Composition of AoGRL: Example (1)



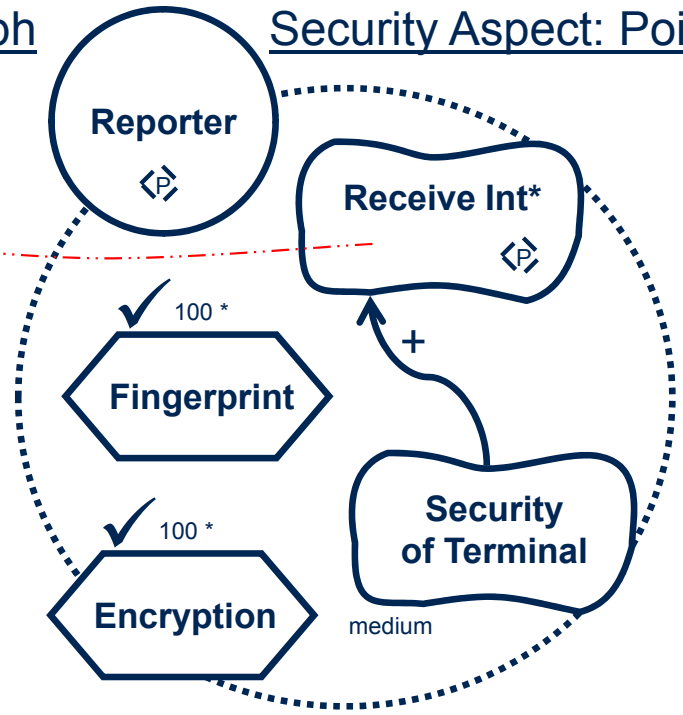


# Composition of AoGRL: Example (2)

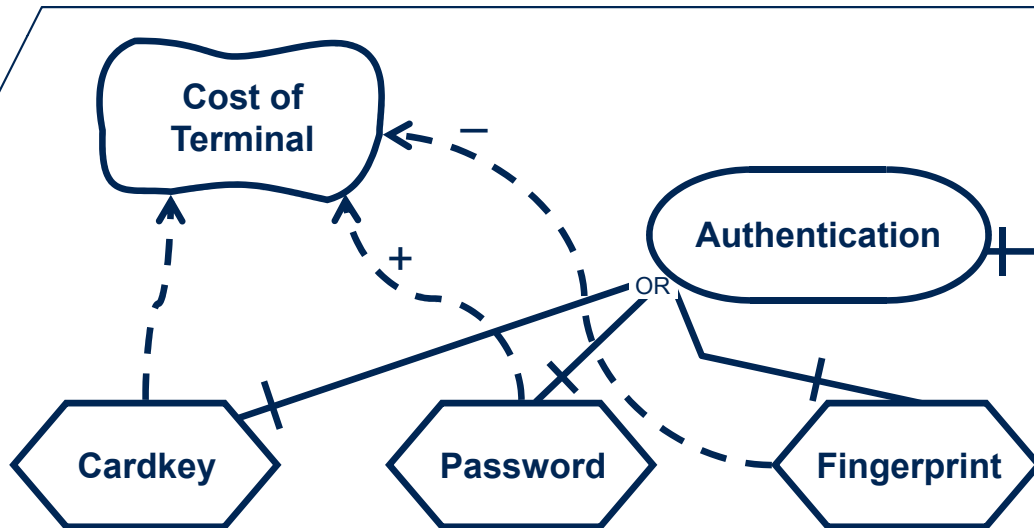
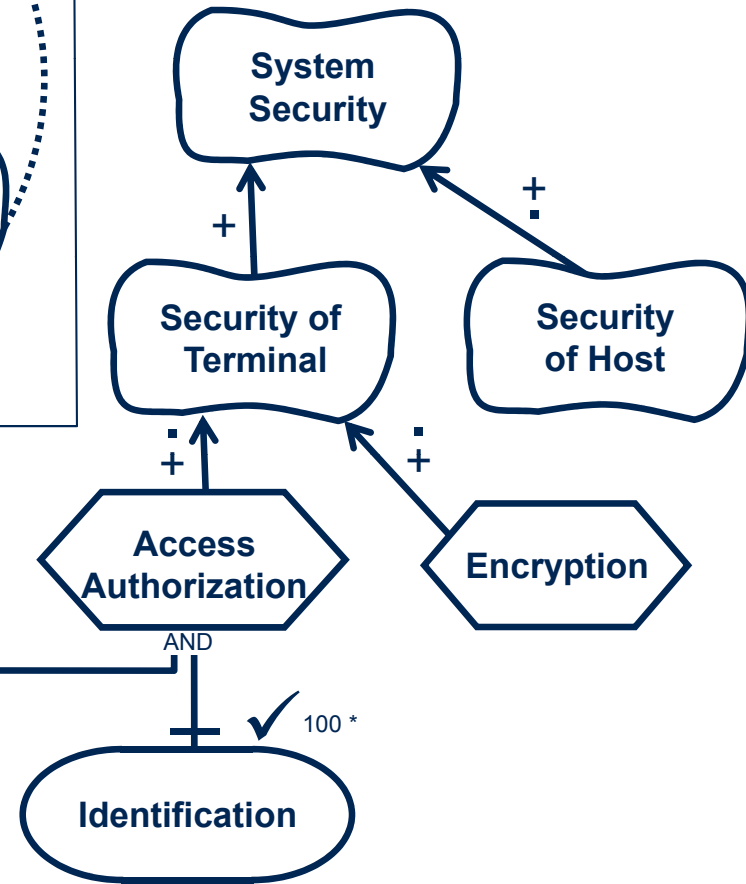
Reporter Goal Graph



Security Aspect: Pointcut Graph

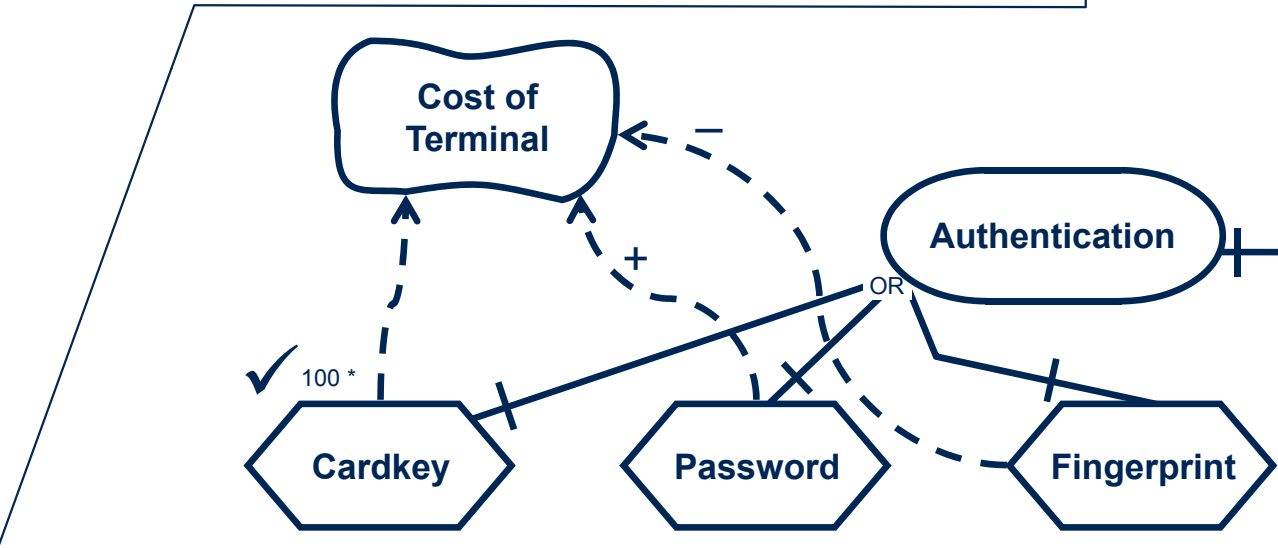
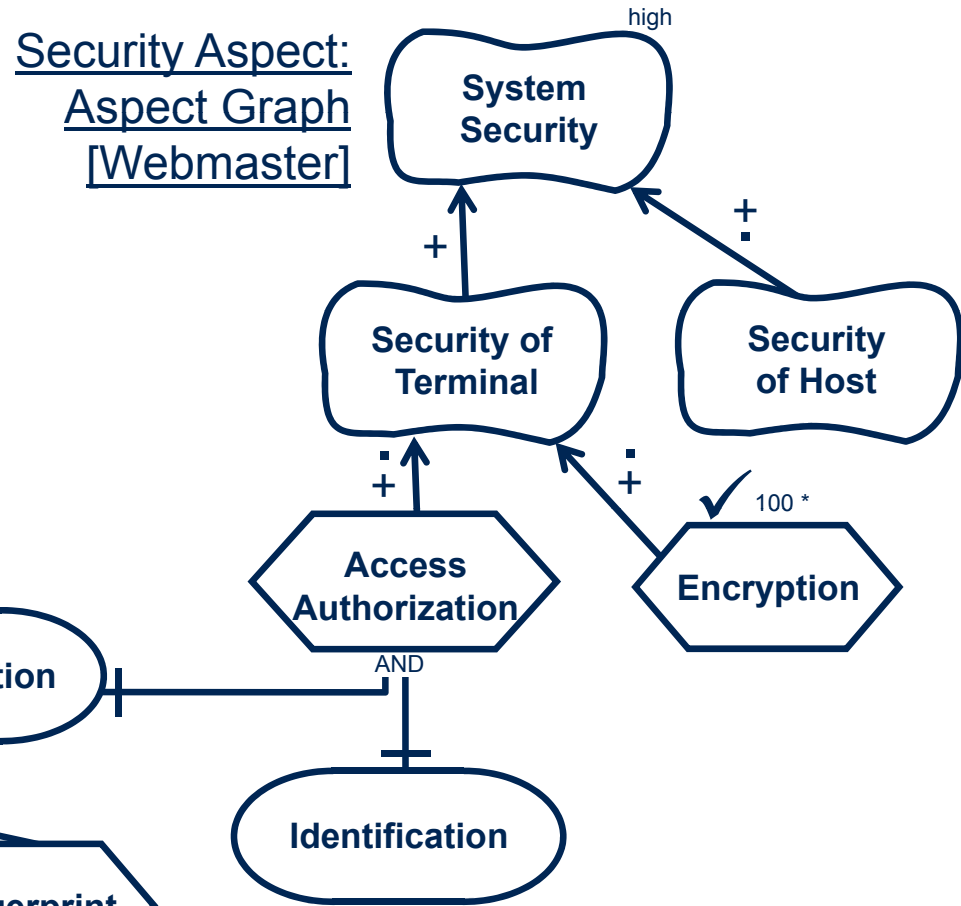
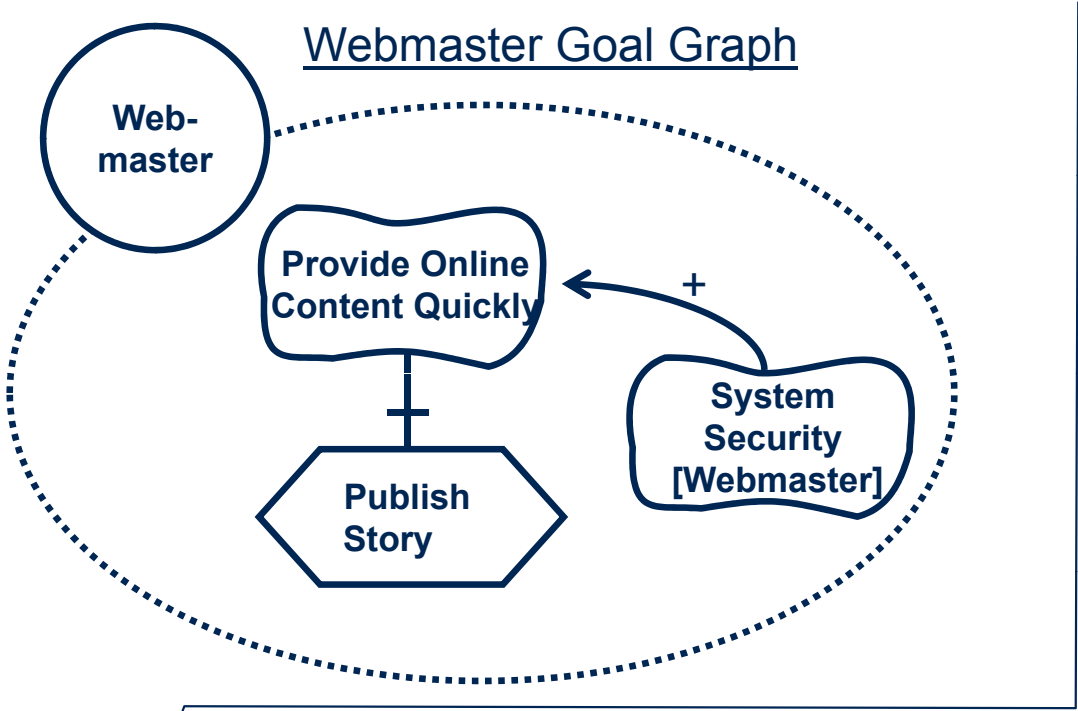


Security Aspect: Aspect Graph



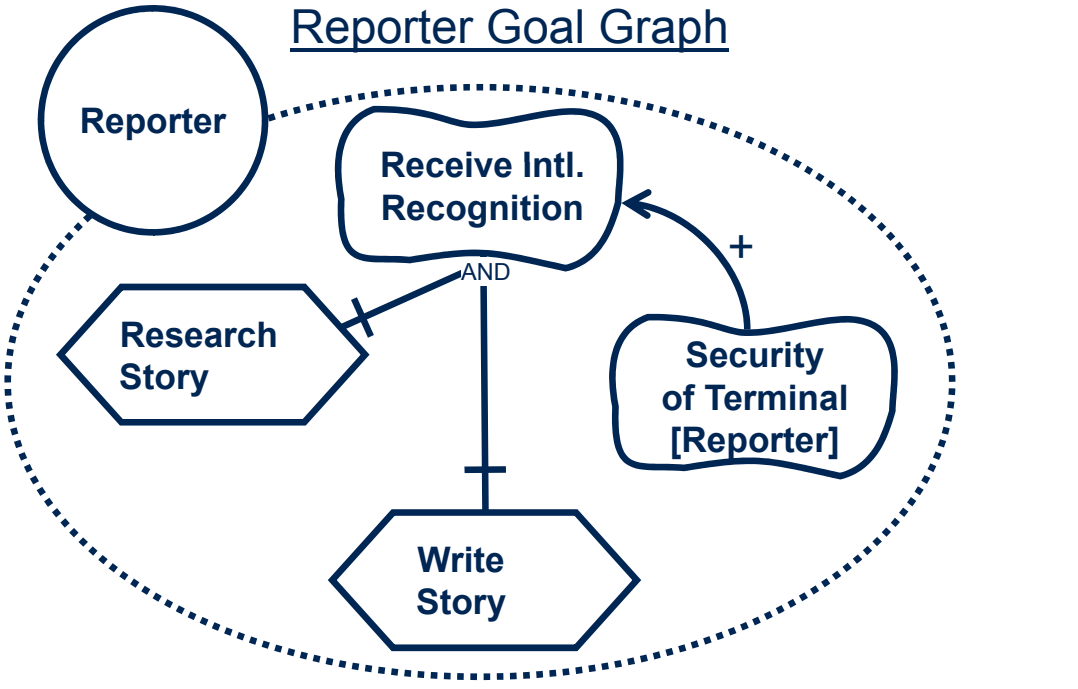


# Composition of AoGRL: Example (3)

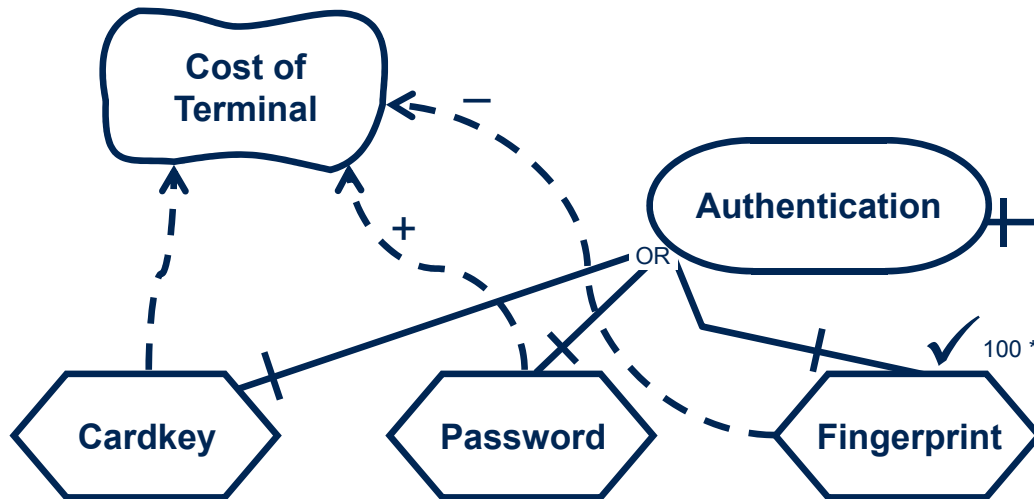
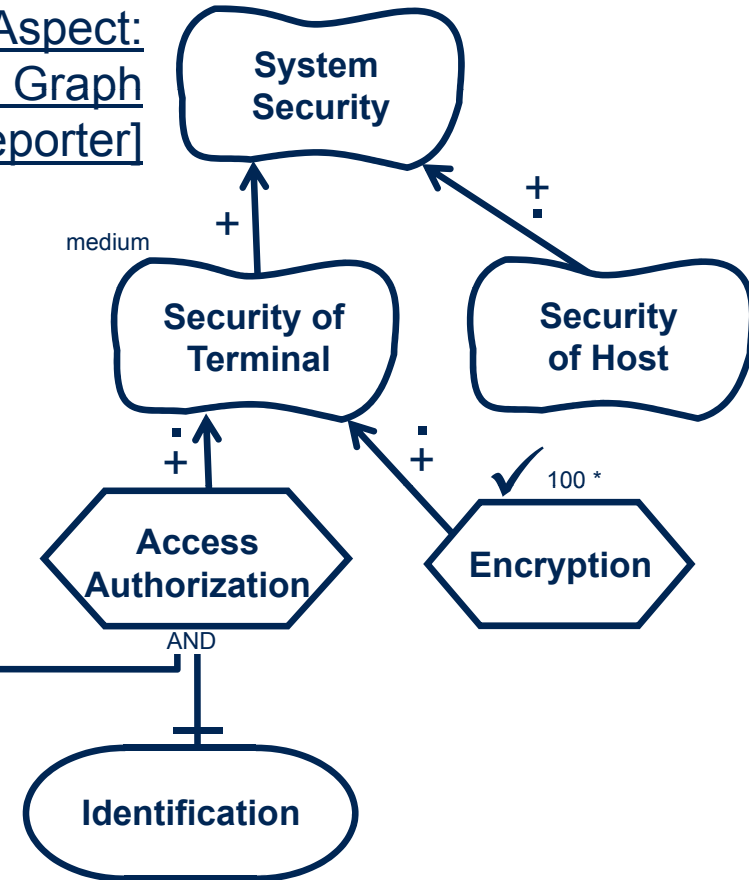


# Composition of AoGRL: Example (4)

Reporter Goal Graph



Security Aspect:  
Aspect Graph  
[Reporter]





# Aspect-oriented Analysis and Validation with AoURN



# GRL: Strategies and Evaluation Mechanism

- GRL allows a particular configuration of intentional elements to be defined in a **strategy** (i.e. one possible solution)
  - Captures the initial, user-defined satisfaction levels for these elements separately from the GRL graphs
  - Strategies can be compared with each other for **trade-off analyses**
- In order to analyze the goal model and compare solutions with each other, jUCMNav's customizable **evaluation mechanism** executes the strategies
  - Propagating levels to the other elements and to actors shows **impact** of proposed solution on high level goals for each stakeholder
  - Propagation starts at user-defined satisfaction levels of intentional elements (usually bottom-up)
  - Takes into consideration
    - Initial satisfaction levels of intentional elements
    - Links and contribution types
    - Importance defined for intentional elements

# GRL: Qualitative or Quantitative Approach

- Qualitative Approach
  - Contribution types: from Make to Break
  - Importance: High, Medium, Low, or None
  - Qualitative satisfaction levels
- Quantitative Approach
  - Contribution types: [-100, 100]
  - Importance: [0, 100]
  - Quantitative satisfaction levels: [-100, 100]
- Hybrid Approach is also possible
  - Qualitative contribution types
  - Quantitative importance
  - Quantitative satisfaction levels

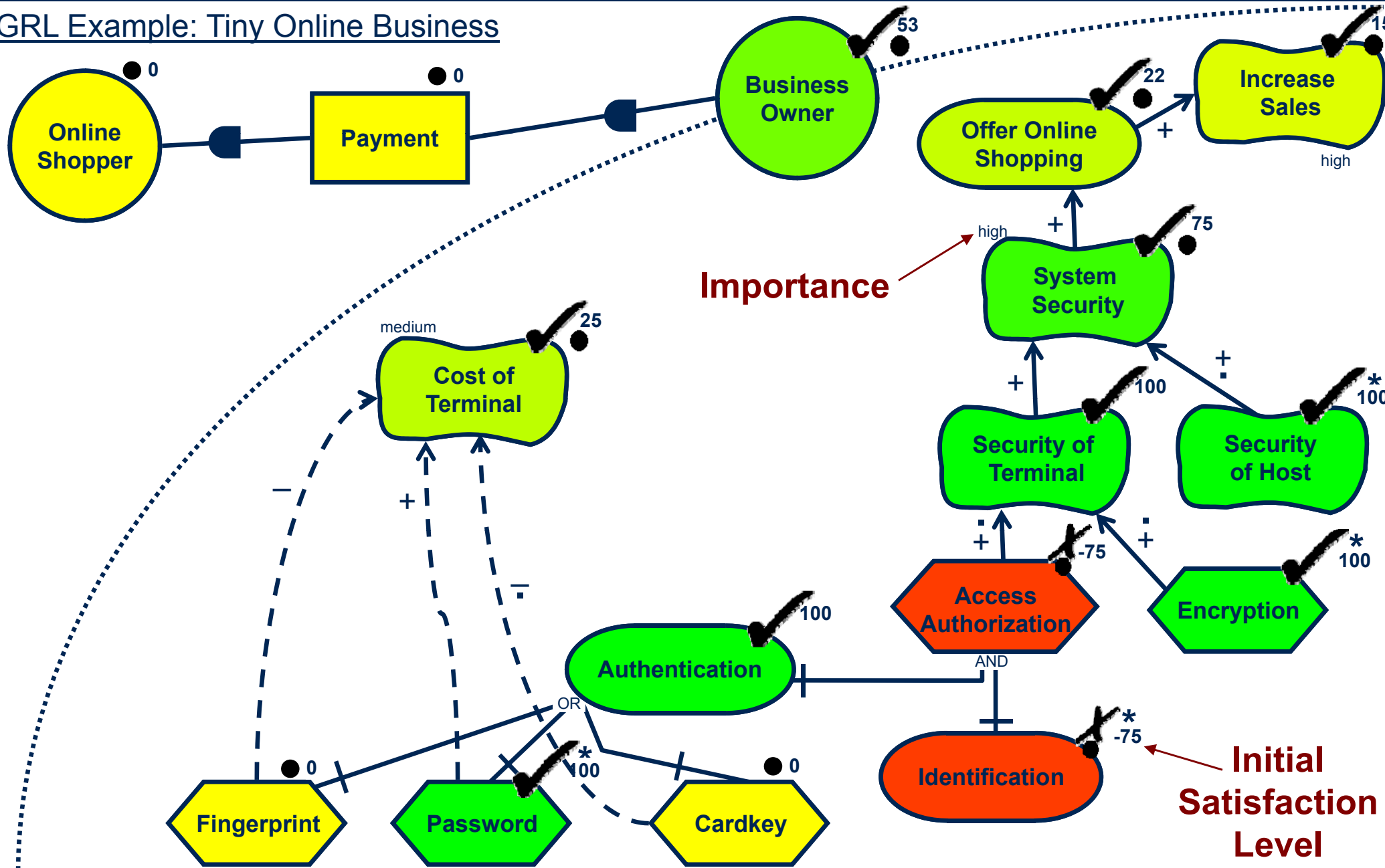
GRL Satisfaction Levels:  
(qualitative)





# GRL: Strategy Execution

## GRL Example: Tiny Online Business



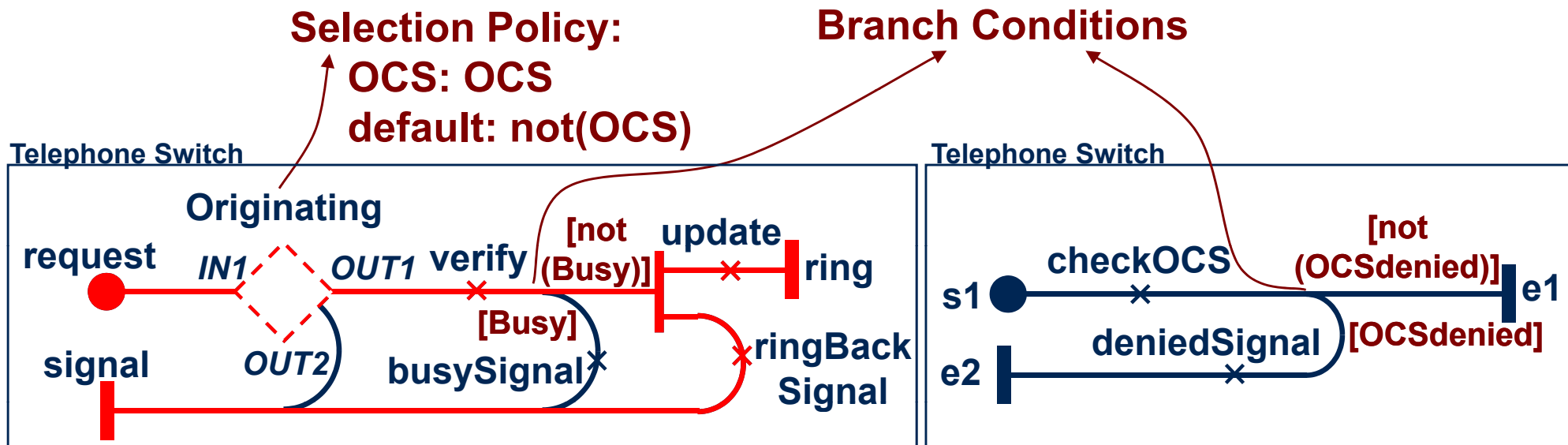


# Use Case Maps: Traversal Mechanism

- UCM **scenarios** describe one path through the UCM model (only one alternative at any choice point is taken)
  - Set of initial values for the variables used in conditions and responsibilities
  - Start points triggered, end points reached
  - Possibly pre/post conditions
- jUCMNav's **traversal mechanism** executes the UCM model given UCM scenario description(s) (i.e., test suite for UCM model)
  - **Intuitive** interpretation aligned with UCM semantics except for dynamic stubs which are deemed to contain an XOR for the selection of a single plug-in map
- Two options
  - Deterministic (only one alternative at any choice point can be enabled)
  - Non-deterministic (randomly choose an alternative from all enabled ones)
- Boolean, Integer, and Enumeration variables are evaluated and can be changed by responsibilities during the traversal of the UCM model
  - These variables are used in expressions for any alternative of a choice point

# Use Case Maps: Scenario Execution (1)

## UCM Example: Tiny Telephone System



a) Basic Call map

b) OCS plug-in map

- Scenario Definition “Simple Basic Call”

- Start point: req
- OCS = false
- Busy = false
- End points: ring, sig

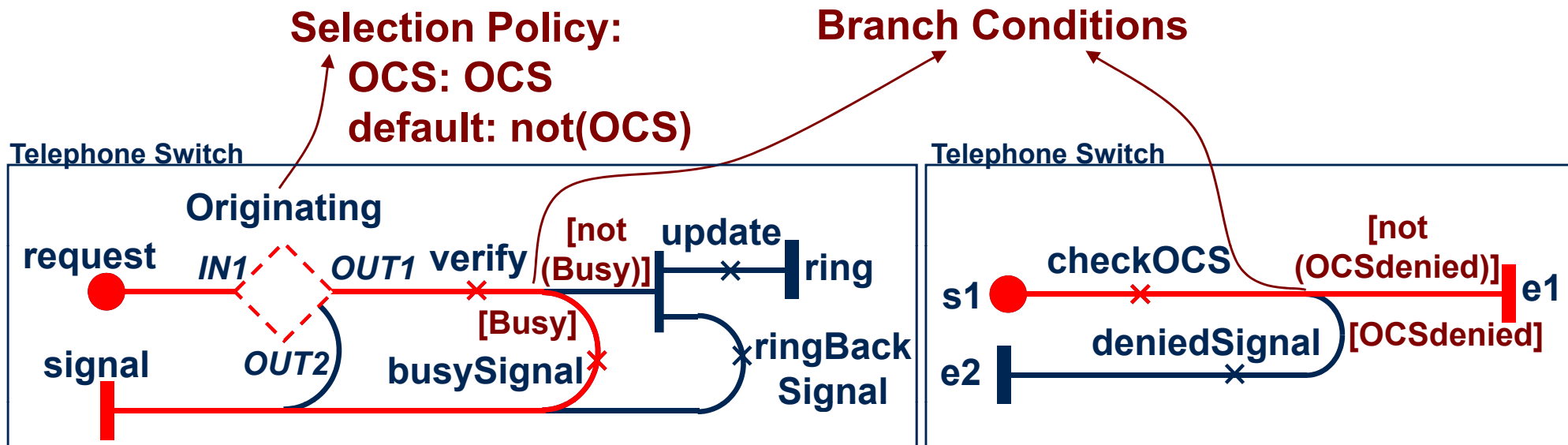


c) default plug-in map



# Use Case Maps: Scenario Execution (2)

## UCM Example: Tiny Telephone System



a) Basic Call map

b) OCS plug-in map

- Scenario Definition “Busy Call + OCS”

- Start point: req
- OCS = true
- OCSdenied = false
- Busy = true
- End point: sig



c) default plug-in map

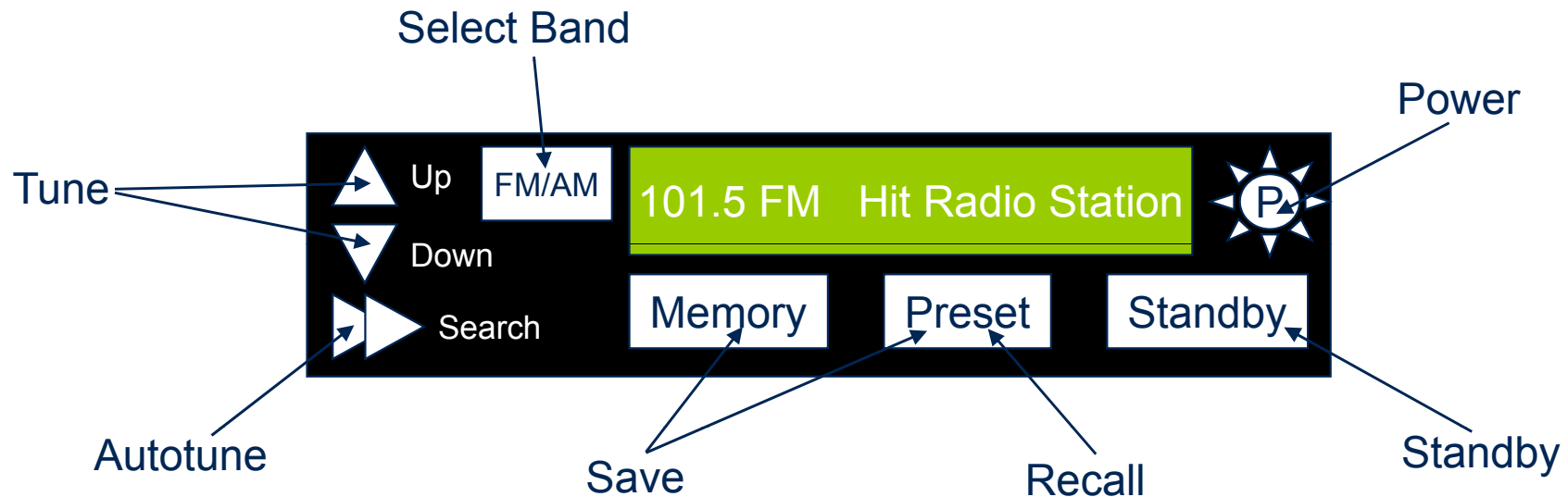


# Overview of Approach for AoUCM

- Step 1: Model each feature as a concern (aspect)
- Step 2: Create scenario definitions (= validation model)
  - Complete branch coverage for each feature
  - Global variables, formalization of conditions for choice points, pseudo-code for responsibilities
- Step 3: Model FI resolutions with aspects
  - Model as part of existing feature concerns
- Step 4: Create scenario definitions for FI resolutions
  - Reuse scenario definitions of individual features as much as possible
  - Requires **composition rules** for aspect-oriented scenario definitions
  
- A similar approach with similar results may be established for the evaluation of AoGRL models

# Features of Radio Device

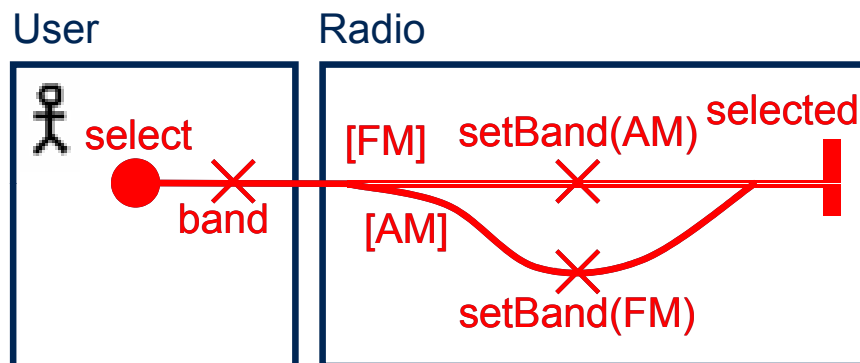
- 10 Features
  - Select Band, Tune, Autotune, Save, Recall, Traffic News, Power On / Off, Standby On / Off



- Update Display
- Remember Settings
  - Radio device remembers last band and frequency settings when turned off, defaults to these settings when turned on

# Example of Features: Select Band

- Step 1: Model feature



- Step 2: Create scenario definitions

- Global enumeration variable:  $\text{Band} = \{ \text{AM} , \text{FM} \}$
- Formalize conditions:  $[\text{FM}] \dots \text{Band} == \text{FM}$ ,  $[\text{AM}] \dots \text{Band} == \text{AM}$

Select Band	Scenario One	Scenario Two
Included Scenarios	n/a	n/a
Start Points	select	select
Initialization	Band = FM	Band = AM
End Points	selected	Selected
Postcondition	Band == AM	Band == FM

- Code for responsibilities changes the value of variable Band
  - $\text{setBand(AM)} \dots \text{Band} = \text{AM}$ ;  $\text{setBand(FM)} \dots \text{Band} = \text{FM}$



# UCM Scenario Definitions

- Scenario definitions may be included in other scenario definitions
  - Union of start points, end points, and pre/postconditions
  - Start points ordered by scenario include order
  - Initializations: applied in scenario include order before scenario starts – later ones may override earlier ones

Scenario Definition	Scenario One	Scenario Two	Combined Scenario
Included Scenarios	n/a	n/a	One, Two
Start Points	start1	start2	start1, start2
Initialization	var1 = X, var2 = Y	var1 = Z	var1 = Z, var2 = Y
End Points	end1	end2	end1, end2
Postcondition	var2 = A	var1 = B	var1 = B, var2 = A

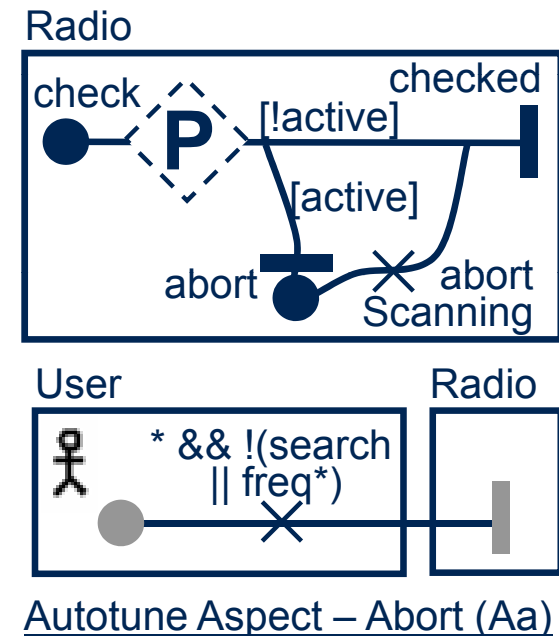
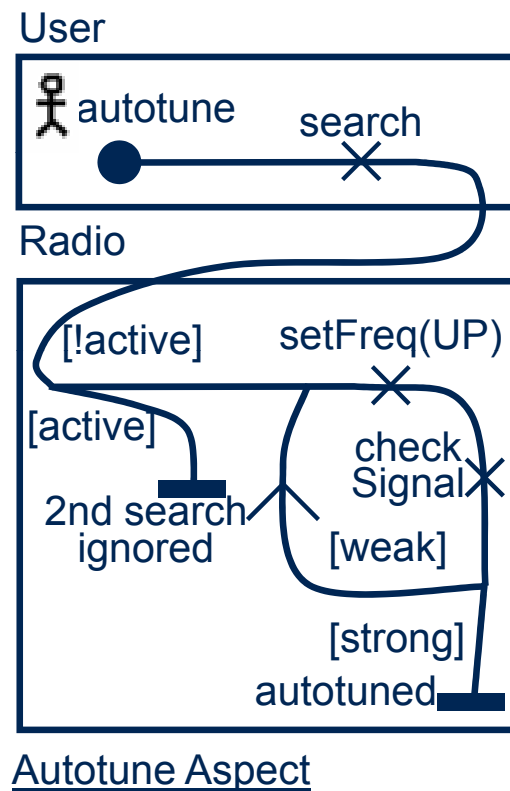
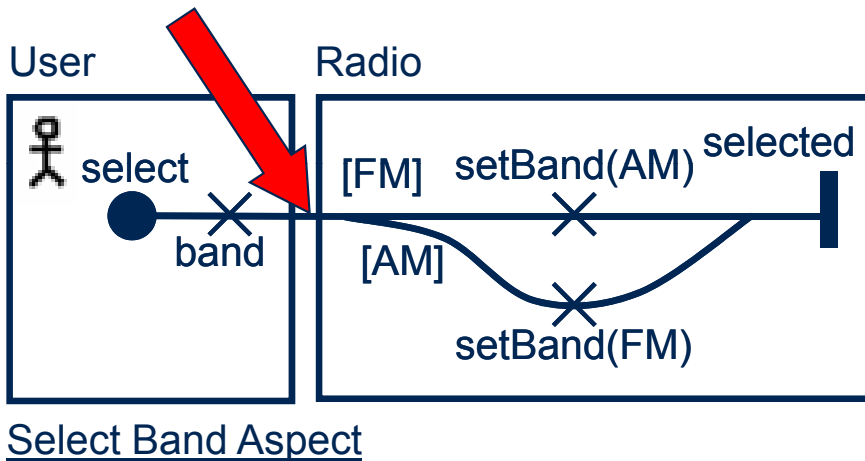
- Elements may be added to combined scenario definitions
- Next start point of scenario definition is only considered if the traversal of the previous start point is finished or stuck

# AoUCM Scenario Definitions

- Requires extensions to the current scenario definitions
- Control variables for loops (counters)
  - **NEW** Traversal mechanism exposes element hit count (`_hitCount`)
- Changes to scenario definition
  - **NEW** Elements may be deleted from scenario definitions
- Interrupting a scenario
  - **NEW** Allow interruption of a scenario before/after a path element if a condition evaluates to true
- There are more extensions...

# Example of Feature Interaction: Abort Autotune (1)

- Step 3: Model FI resolution
  - Autotune must be aborted when Select Band, Save, Recall, Power Off, or Standby Off are activated
  - Modeled as part of the Autotune feature concern





# Example of Feature Interaction: Abort Autotune (2)

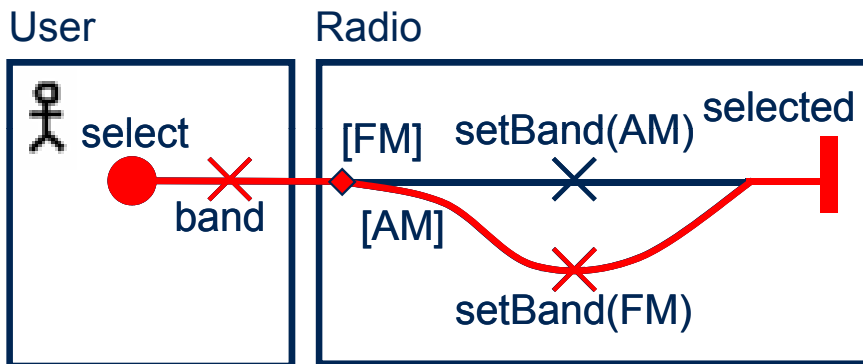
- Step 4: Create scenario definitions for FI resolution
  - Interaction Scenario Definition:
    - include** Autotune, Select Band
    - interrupt** after OR-join in Autotune if `_hitCount == 50`
    - delete** OR-join; **add** abort

Scenario Definition	Autotune	Select Band	Interaction
Included Scenarios	n/a	n/a	Autotune, Select Band
Start Points	autotune	select	autotune, select
Initialization	Freq = 32, StrongFreq = -1	Band = AM	Freq = 32, StrongFreq = -1 Band = AM
End Points	OR-join	selected	<del>OR-join</del> , abort, selected
Postcondition	Freq != StrongFreq	Band == FM	Freq != StrongFreq, Band = FM

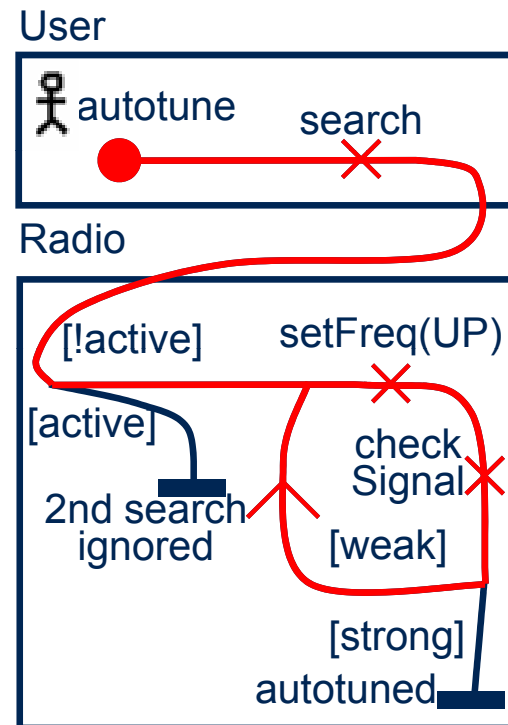
# Example of Feature Interaction: Abort Autotune (3)

Scenario Definition	Interaction
Included Scenarios	Autotune, Select Band
Start Points	autotune, select
Initialization	Freq = 32, StrongFreq = -1, Band = AM
End Points	<del>OR-join</del> , abort, selected
Postcondition	Freq != StrongFreq, Band = FM

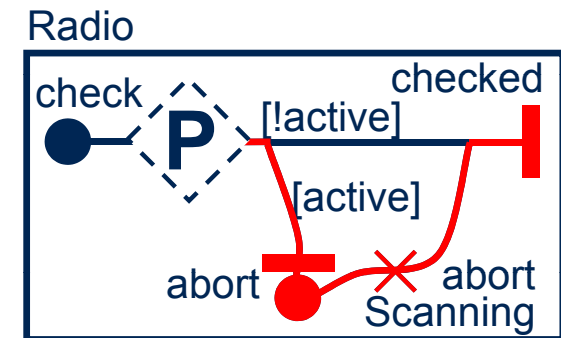
**interrupt** after OR-join in Autotune if `_hitCount == 50`



Select Band Aspect



Autotune Aspect



Autotune Aspect – Abort (Aa)



# Aspect-oriented Analysis and Validation

- Integrated aspect-oriented specification and validation scenario models at the requirements stage
- Incremental development of features
- **Composition rules** for aspect-oriented validation models
  - Include, add, delete, interrupt, redundant
- Reduced complexity of AoUCMs
- Improved scalability
- Feature concerns and FI resolution concerns are properly encapsulated
- Specification model is kept strictly separate from the validation model (thus further improving separation of concerns)
- It may be possible to
  - Infer composition rules for the validation model from the aspect-oriented composition rules of the specification model
  - Provide more internal information of the path traversal mechanism



# AoURN Tool: jUCMNav – Juice Up Your Modeling!

- GRL
- Strategies
- Evaluation
  
- UCMs
- Scenario Definition
- Scenario Execution
- Test Suite
  
- Support for AoURN
- Support for BPM
- MSC Generation
- Export to DOORS / CSM



**Pronounced: juicy – em – nav**

<http://jucmnav.softwareengineering.ca>

# AoURN Tool: jUCMNav

- Free (EPL), open-source plug-in for Eclipse
  - Based on Eclipse Modeling Framework (EMF) and the Graphical Editing Framework (GEF)
- Supports most of GRL and UCM notation elements
- Extensible architecture for customized GRL and UCM analysis techniques
- 4 GRL evaluation algorithms, with color highlight
- 1 UCM path traversal mechanism, with export to flat UCMs and MSCs
  - Integrated MSC viewer
- Other features
  - Export performance models to CSM
  - Integration with DOORS for requirements management
  - Extensions for Business Process Modeling, Monitoring, and Adaptation
  - Extensions for aspect-oriented modeling with AoURN
  - Verification of user-defined (OCL) rules
  - Report generation (PDF, HTML)
  - Export of diagrams to various bitmap formats

# GRL Editor with Strategy Evaluation

jUCMNav - EclipseTest33/Documents/CreditCardGateway.jucm - Eclipse Platform

File Edit Navigate Search Project Run jUCMNav Window Help

View all elements

Toolbar Perspective jUCMNav

Navigator Elements

Documents

- CS15112-2007
- CS15112-2008
- JasonThesis
- LabDOORS
- LabFI
- ORNEC
- SEG3601-2006
- SEG3601-2007
- UCM AO Via Verde SPL.jucm
- Benoit.csm
- UCM Benoit.jucm
- ComptableElectronique.jucm
- Copy of paper-based process-OH-Thesis-DC
- UCM CreditCardGateway.jucm

Navigator view

Outline

- GRL High Level Goals (2649)
  - CardHolder (2754)
  - Customer (2701)
  - Merchant (2932)
  - Payment Gateway
  - Buy Merchant Products (2844)
  - Cost of Products (3014)
  - Maximize Profit (2934)
  - Merchant Web Site Usability (3189)
  - Minimize Cost of Web Site (3018)
  - Minimize Implementation Cost (3031)
  - Minimize Merchant Cost (2944)
  - Minimize Service Cost (3025)
  - Need Products (2850)
  - Secure Payment (2836)
  - Sell Products (2940)
  - Trust (2838)
  - GRL Security Model (3036)
  - GRL Architectural Model (5378)
  - GRL Cost (6259)
  - DEF UICM Definitions

Outline view

Editor

Editor

Palette

- Select
- Links
  - Decomposition
  - Contribution
  - Dependency
  - Belief Link
- Components
- Actor
- Elements
  - Softgoal
  - Goal
  - Task
  - Resource
  - Belief
- KPI Model
- Indicator
- Dimension
- KPI Model Link

Palette

Third Party Payme... 3D Payment Proces... Bank Processing Security Dependen... High Level Goals Security Model Architectural Model Cost >>4

Scenarios and Stra List of Key Perform

- UCM Scenarios
  - GRL Evaluation Strategies
    - 3D Payment Processing (6663)
      - GRL 3D With Encryption/Certificate (6666)
      - GRL 3D Without Encryption/Certificate (6665)
    - Gateway to Customer Payment Processing (6664)
    - Standard Payment
    - Third Party Payme
  - Enumerations
  - Variables

Scenario and Strategies view

Property

Property	Value
Info	
description	
id	3030
name	Minimize Implementation Cost
Parent	Payment Gateway (3012)
Metadata	
Metadata	[click to edit]
Miscellaneous	
criticality	None
decompositionType	And

Properties view

Info



# UCM Editor with Scenario Traversal

jUCMNav Execution - EclipseTest33/Documents/WirelessIN.jucm - Eclipse Platform

File Edit Navigate Search Project jUCMNav Run Window Help

View all elements

Scenarios and Strategies Navigator

UCM \*WirelessIN.jucm

**UCM Scenarios**

- UCMarchAlternatives (5)
  - UCM ServiceAndSDFinSCP\_NotOK (406)
    - Included scenarios
    - Start points
      - RootScenario: StartConnection (100)
    - Initializations
      - authorized=false
      - deployment=IN\_SCP
    - Preconditions
    - End points
      - RootScenario: Done (102)
    - Postconditions
  - UCM ServiceAndSDFinSCP\_OK (6)
  - UCM ServiceInMSC\_NotOK (404)
  - UCM ServiceInMSC\_OK (314)
  - UCM ServiceInMSC\_SDFinSN\_NotOK (405)
  - UCM ServiceInMSC\_SDFinSN\_OK (407)

**MS**

StartConnection

Reject

**MSC**

CCF

Authorization

INI

OUT1

[OK]

[NotOk]

LogReject

**HLR**

LRFh

ChkLoc

Done

**Palette**

- Select
- PathTool
- Components
  - Team
  - Object
  - Process
  - Agent
  - Actor
  - Other
- Paths
  - Responsibility
  - Stub
  - Dynamic Stub
  - Pointcut Stub
  - Timer
  - Waiting Place
  - Direction Arrow
  - Or Fork
  - And Fork
  - Or Join
  - And Join

Properties Outline Elements

Property	Value
<b>Info</b>	
description	
id	406
name	ServiceAndSDFinSCP_NotOK
<b>Metadata</b>	
Metadata	[click to edit]
<b>Scenario / Strategy</b>	
group	UCMarchAlternatives (5)

GRLmodel RootScenario ServiceInMSC ServiceAndSDFinSCP ServiceInMSC\_SDFinSN

Problems

1 error, 0 warnings, 0 infos

Description	Resource	Path	Location
Scenario should have reached end point: ucm.map.impl.EndPo	WirelessIN.jucm	EclipseTest33/Documents	Done

# Integrated MSC Viewer for Exported UCM Scenarios

jUCMNav - demo/jUCMNavDemo\_v5.jucmscenarios - Eclipse SDK

File Edit Navigate Search Project CSM Viewer Run MSC Viewer Window Help

demo jUCMNav

Navigator

demo

UCM jUCMNavDemo\_v5.jucm

Environment Researcher Hospital REB

RequestData

Ini

Init

Wait

Wait

EndPain

EndPain

TrustUser

Star

WaitREB

WaitREBdone

WaitREBdone

NewRequest

TryAgain

TryAgain

RequestData

Init

TryAgain

Rejected

Accepted

Palette

- Select
- PathTool
- Components
  - Team
  - Object
  - Process
  - Agent
  - Actor
  - Other
- Paths
  - Responsibility
  - Stub
  - Dynamic Stub
  - Pointcut Stub
  - Timer
  - Waiting Place
  - Direction Arrow
  - Or Fork
  - And Fork
  - Or Join
  - And Join

Environment Researcher Hospital REB

RequestData

Ini

Init

Wait

Wait

EndPain

EndPain

TrustUser

Star

WaitREB

WaitREBdone

WaitREBdone

NewRequest

TryAgain

TryAgain

RequestData

Init

TryAgain

Rejected

Accepted

ProtectPrivacy MainProcess Review TrustUser Agreement

Scenarios and List of Key Per Properties Problems Key Performance Indicators

Property	Value
Info	
description	
id	1008
name	REBnotReady
Metadata	

UCM Scenarios

- MyScenarios (6)
  - AcceptAgreement (7)
  - AcceptTrust (1005)
  - REBnotReady (1008)
  - RejectAgreement (1266)

# Aspect-oriented URN (1)

jUCMNav - demo/voting station.jucm - Eclipse SDK

File Edit Navigate Search Project CSM\_Viewer Run jUCMNav Window Help

Navigator

- .project
- UCM jUCMNavDemo\_v5.jucm
- UCM jUCMNavDemo\_v5.jucmscenarios
- UCM voting station.jucm

voting station.jucm

Poll Official Voting Machine Backend Server

report selectReport reported saveResults

presentOptions

Report Result Authentication Authentication PC Remote Service >>10

Poll Official

reported

Voting Machine

presentOptions

IN1 OUT1

Report Result Authentication Authentication PC >>12

Outline Element

- Authentication (155)
  - UCM Authentication (153)
    - requiresAuthentication
    - UCM Authentication PC
- Caching (1562)
  - UCM Caching (1289)
    - requiresCaching (133)
      - UCM Caching PC (1290)
- Cast Vote (156)
  - UCM Cast Vote (1665)
- Data Replication (1841)
  - UCM Data Replication (1566)
    - requiresData (2368)
- Encryption (2616)
  - UCM Encryption (2613)

voting station.jucm

Poll Official Voting Machine <<confidential>> Authentication Server

authenticate insertSmartCard enterPIN end takeSmartCard

displayLogin

ejectSmartCard [fail] [success]

requiresAuthentication

authenticate

Authentication Authentication PC Remote Service Remote Service PC Caching Caching PC Data Replication Data Replication PC Cast Vote Encryption >>5



# Aspect-oriented URN (2)

**Customer**

- usePoints
- fillmembership
- processOrder
- redeemMoviePoints
- payForMovie

**Online Video Store**

- sendMovie
- creditMoviePoints
- pointsUsed

**Context Menu:**

- Undo
- Redo Apply Concern
- Cut
- Copy
- Paste
- Select All
- Delete
- Zoom In (Ctrl+=)
- Zoom Out (Ctrl+-)
- Path Operations
- Insert Component
- Add UCM
- Add GRL graph
- Duplicate
- Hyperlink
- Manage Concerns
- Apply Concern
- Import
- Export
- Report
- Edit Metadata

**Customer**

- buy
- selectMovie
- payForMovie
- bought

**Online Video Store**

- processOrder
- sendMovie

**Path:** in to pointcut stub sendMovie ----- out from pointsUsed[139]

# Summary, References, and Appendix (Notation Overview)

# Summary (1)

- The User Requirements Notation (URN) is an ITU-T standard
- Modeling with the Goal-oriented Requirement Language (GRL)
  - Focuses on answering “**why**” questions
  - Intentions, functional / non-functional requirements, rationales
- Modeling with Use Case Maps (UCM)
  - Focuses on answering “**what**” questions
  - Scenarios, services, architectures
- While modeling with URN as a whole, goals are **operationalized** into tasks, and tasks are **elaborated** in (mapped to) UCM path elements and scenarios
  - Moving towards answering “**how**” questions
  - Can guide the selection of an architecture or scenarios
- Enables the elicitation/specification of systems, standards, and products, their analysis from various angles, and transformations



## Summary (2)



- The Aspect-oriented User Requirements Notation (AoURN) combines **goal-modeling**, **scenario-modeling**, and **aspect-oriented modeling** at the requirements level in one framework
- Graphical and familiar
- Better encapsulation of concerns in URN models (goals and scenarios)
- URN models are more easily maintained and reused
- A standardized way of modeling non-functional requirements and use cases
- Enhanced matching mechanism based on semantics
- Tool Support
  - Some support for AoURN has already been added to the jUCMNav tool
  - More to come...

# References (1)

-  URN Virtual Library (>290 entries), <http://www.UseCaseMaps.org/pub/>
-  URN tool: jUCMNav, University of Ottawa, <http://jucmnav.softwareengineering.ca/jucmnav/>
-  ITU-T, Recommendation Z.151 (11/08): User Requirements Notation (URN) - Language Definition, Geneva, Switzerland, approved November 2008.
-  Amyot, D., and Mussbacher, G.: “Development of Telecommunications Standards and Services with the User Requirements Notation”. *Joint ITU-T and SDL Forum Society Workshop on "ITU System Design Languages"*, Geneva, Switzerland (September 2008)
-  Mussbacher, G.: “Aspect-oriented User Requirements Notation”. PhD thesis, SITE, University of Ottawa, Canada, 2010.
-  Mosser, S., Mussbacher, G., Blay-Fornarino, M., and Amyot, D.: “From Aspect-oriented Requirements Models to Aspect-oriented Business Process Design Models – An Iterative and Concern-Driven Approach for Software Engineering”. *10th Intl. Conference on Aspect-Oriented Software Development (AOSD 2011)*, Porto de Galinhas, Brazil (March 2011)
-  Mussbacher, G., Amyot, D., Araújo, J., and Moreira, A.: “Requirements Modeling with the Aspect-oriented User Requirements Notation (AoURN): A Case Study”. Katz, S., Mezini, M., and Kienzle, J. (Editors), *Transactions on Aspect-Oriented Software Development (TAOSD) VII*, Springer, LNCS 6210, pp. 23–68 (2010)



# References (2)

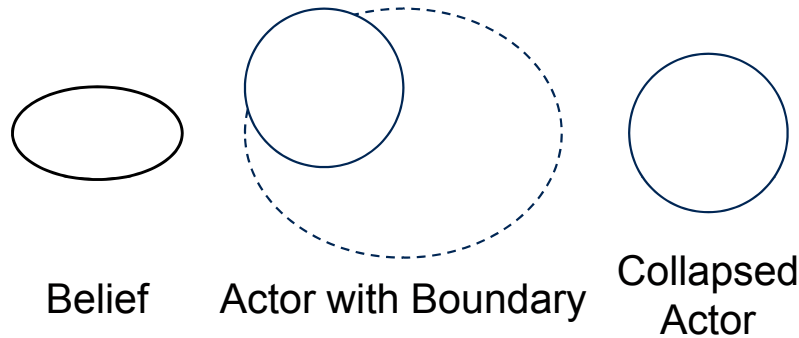
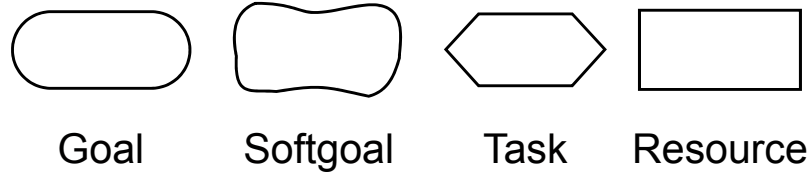
-  Pourshahid, A., Mussbacher, G., Amyot, D., and Weiss, M.: “Toward an Aspect-Oriented Framework for Business Process Improvement”. *International Journal of Electronic Business (IJEB)*, Inderscience Publishers, 8(3), pp. 233–259 (2010)
-  Mussbacher, G., Whittle, J., and Amyot, D.: “Modeling and Detecting Semantic-Based Interactions in Aspect-Oriented Scenarios”. *Requirements Engineering Journal (REJ)*, Springer, 15(2), pp. 197–214 (2010)
-  Mussbacher, G., Amyot, D., and Whittle, J.: “Refactoring-Safe Modeling of Aspect-Oriented Scenarios”. *ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS 2009)*, Denver, USA, Springer, LNCS 5795, pp. 286–300 (October 2009)
-  Mussbacher, G. and Amyot, D. (2009) Extending the User Requirements Notation with Aspect-oriented Concepts. 14th SDL Forum (SDL 2009), Bochum, Germany, Springer, LNCS 5719, pp. 115–132 (September 2009)
-  Mussbacher, G. and Amyot, D.: “The User Requirements Notation (URN) and Aspects”. Tutorial at *17th IEEE International Requirements Engineering Conference (RE'09)*, Atlanta, Georgia, USA (September 2009)



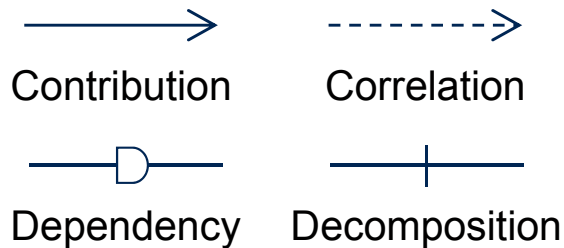
# References (3)

-  Mussbacher, G., Whittle, J., and Amyot, D.: “Semantic-Based Interaction Detection in Aspect-Oriented Scenarios”. *17th IEEE International Requirements Engineering Conference (RE'09)*, Atlanta, USA, IEEE CS, pp. 203–212 (September 2009)
-  Mussbacher, G., Amyot, D., Weigert, T., and Cottenier, T.: “Feature Interactions in Aspect-Oriented Scenario Models”. 10th International Conference on Feature Interactions (ICFI 2009), Lisbon, Portugal (June 11-12, 2009)
-  Mussbacher, G., Araújo, J., Moreira, A., and Whittle, J.: “Aspect-Oriented Requirements Engineering With Scenarios”. Tutorial at 7th International Conference on Aspect-Oriented Software Development, Brussels, Belgium (March 31 – April 4, 2008)
-  Mussbacher, G., Amyot, D., Whittle, J., and Weiss, M.: “Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM)”. *10th International Workshop on Early Aspects (EA 2007)*, Vancouver, Canada (March 13, 2007). Moreira, A. and Grundy, J. (Editors), *Early Aspects: Current Challenges and Future Directions*, Springer, LNCS 4765, pp 19-38 (December 2007)
-  Mussbacher, G., Amyot, D., and Weiss M.: “Visualizing Early Aspects with Use Case Maps”. Rashid, A. and Aksit, M. (Editors), *Transactions on Aspect-Oriented Software Development III*, Springer, LNCS 4620, pp 105-143 (November 2007)

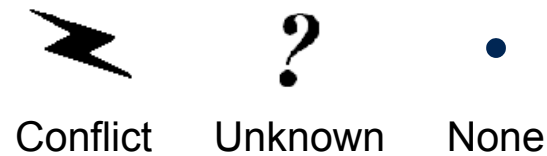
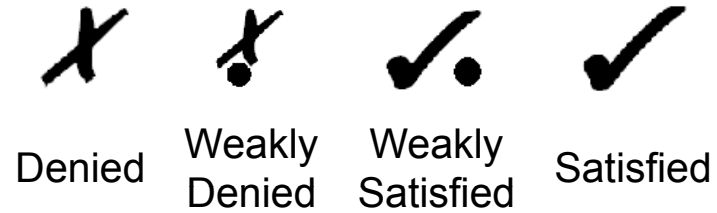
# Appendix: Notation Overview – GRL



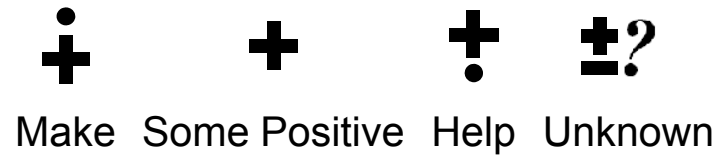
**(a) GRL Elements**



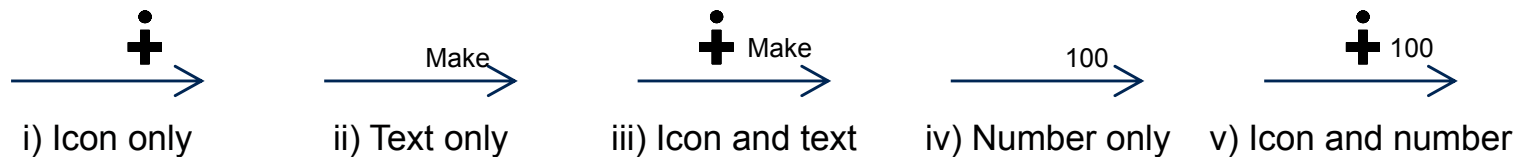
**(b) GRL Links**



**(c) GRL Satisfaction Levels**



**(d) GRL Contributions Types**



**(e) Representations of Qualitative and Quantitative Contributions**



# Appendix: Notation Overview – UCM (Behavior)



Path with Start Point with Precondition CS and End Point with Postcondition CE



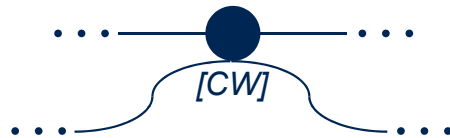
Responsibility



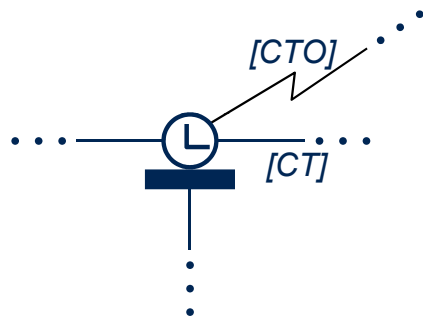
Empty Point



Direction Arrow



Waiting Place with Condition and Asynchronous Trigger



Timer with Timeout Path, Conditions, and Synchronous Release



Static Stub with In-Path ID and Out-Path ID

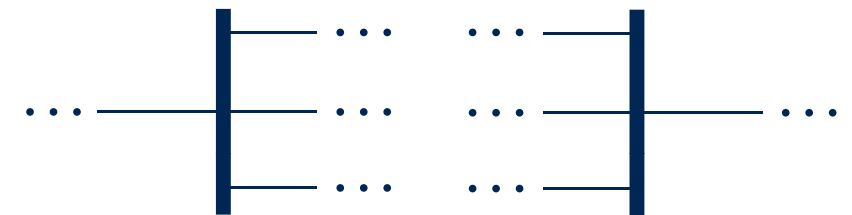


Dynamic Stub with In-Path ID and Out-Path ID



Or-Fork with Conditions

Or-Join



And-Fork

And-Join



Synchronizing Stub with In-Path ID, Out-Path ID, and Synchronization Threshold



Blocking Stub with In-Path ID, Out-Path ID, Synchronization Threshold, and Replication Indicator



# Appendix: Notation Overview – UCM (Structure)

Components:



Team



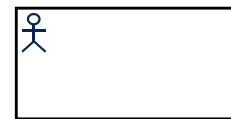
Process



Object



Agent



Actor



Protected Component

*parent:*



Context-dependent  
Component

# Appendix: Notation Overview – AoURN

Concern Interaction Graph:

models conflicts and dependencies between concerns

Aspect Graph:

models aspectual properties

Pointcut Graph:

models pointcut expression and composition rule

Aspect Map:

models aspectual properties and composition rule

Pointcut Map:

models pointcut expression



Aspect Marker



Pointcut Marker



Pointcut Deletion Marker

<<anytype>>

Anytype Pointcut Element

(a) AoURN Diagrams

(b) AoGRL Elements



Aspect Marker

Tunnel Entrance  
Aspect Marker

Tunnel Exit  
Aspect Marker

Conditional  
Aspect Marker



Local Start /  
End Points

Pointcut Stub

Replacement  
Pointcut Stub

Start / End of  
Pointcut Expression

Anything  
Pointcut Element

(C) AoUCM Elements