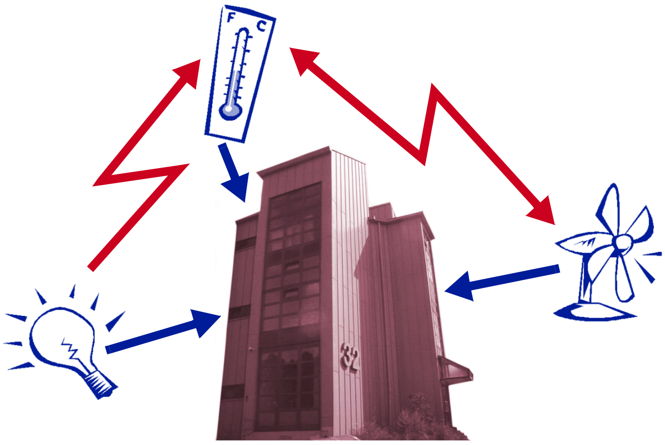
Feature Interaction Detection in Building Control Systems by Means of a Formal Product Model



Andreas Metzger, Christian Webel
Department of Computer Science
University of Kaiserslautern, Germany

Contents

Motivation

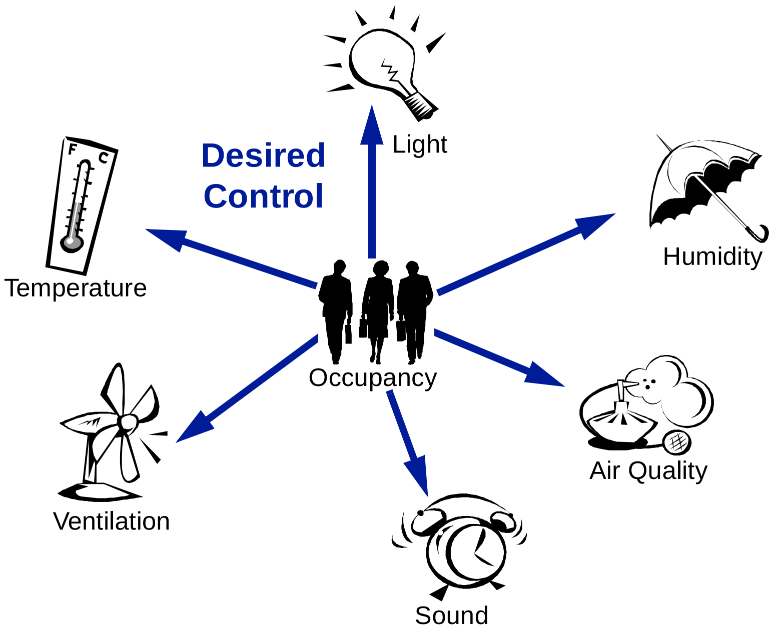
The Product Model Approach

Feature Interaction Detection

Conclusion

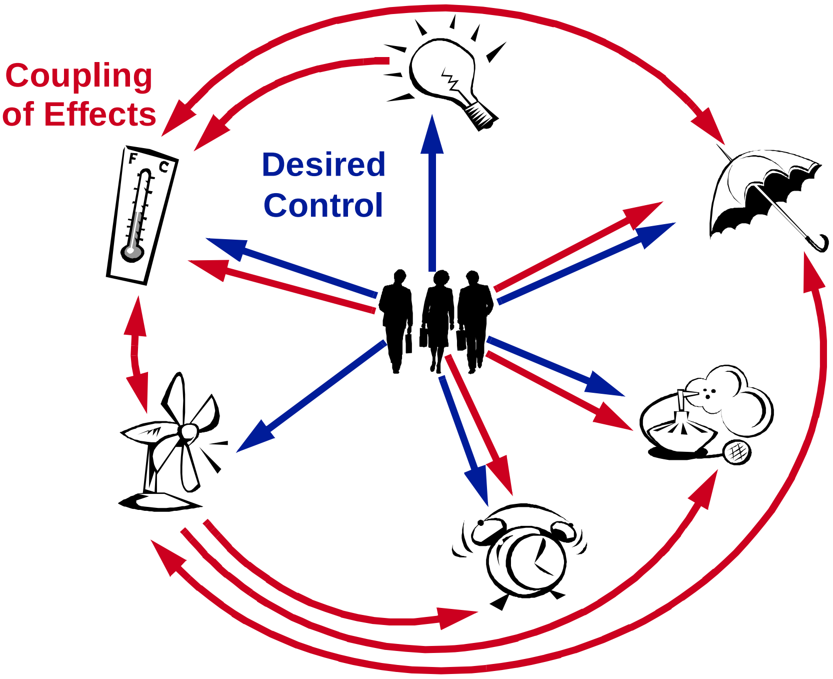
Motivation

Complexity of Building Control Systems



Motivation

Complexity of Building Control Systems



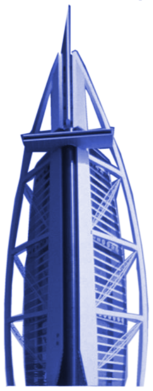
→ **Integrated** Building Control System

Motivation

Complexity of Building Control Systems

300 Requirements
1000 Data Points (Objects)

• • •



8000 Requirements
26000 Data Points (Objects)

Problems

- Extension
  - → Introduction of **Undesirable** Relationships
- Reuse
  - → Elimination of **Required** Interrelations

→ Solution: **Automatic** Detection of Interactions

— The Product Model Approach —

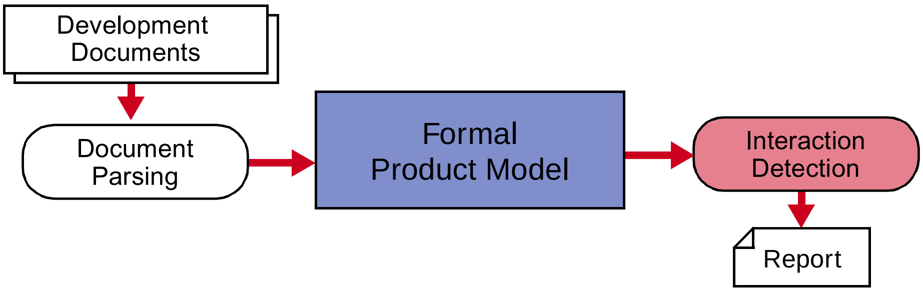
The Product Model Approach

Terms

Product Model
Meta-Model of Development Artefacts and Relations

Formal Product Model
Concise Semantics of Entities
Access to Entities (“Repository”)

“Tool Chain”



Development Documents

Document Parsing

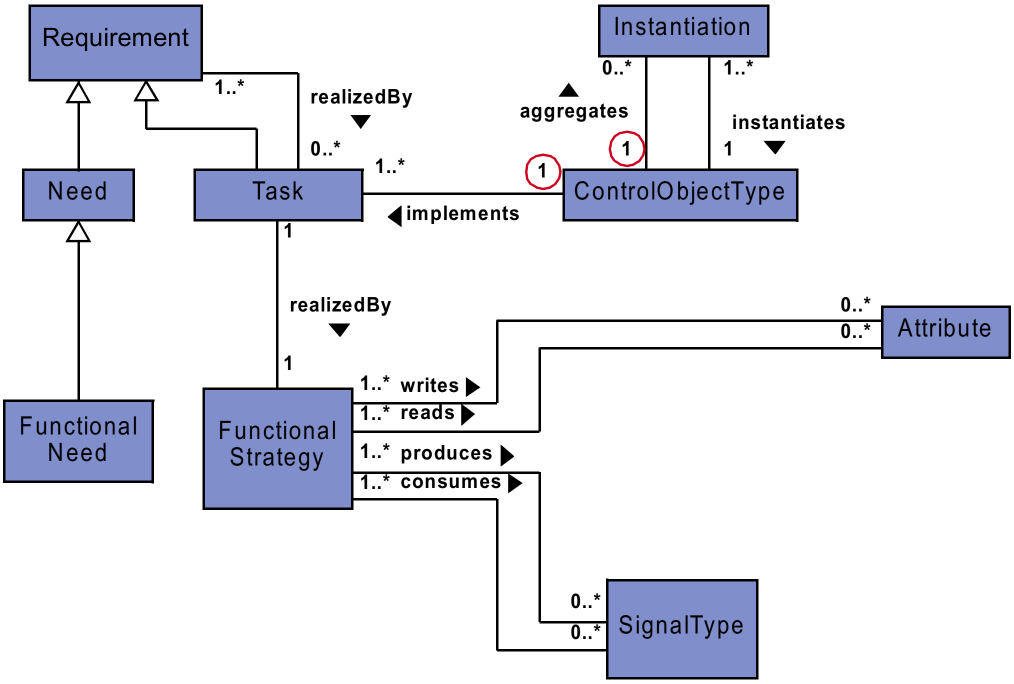
Formal Product Model

Interaction Detection

Report

The Product Model Approach

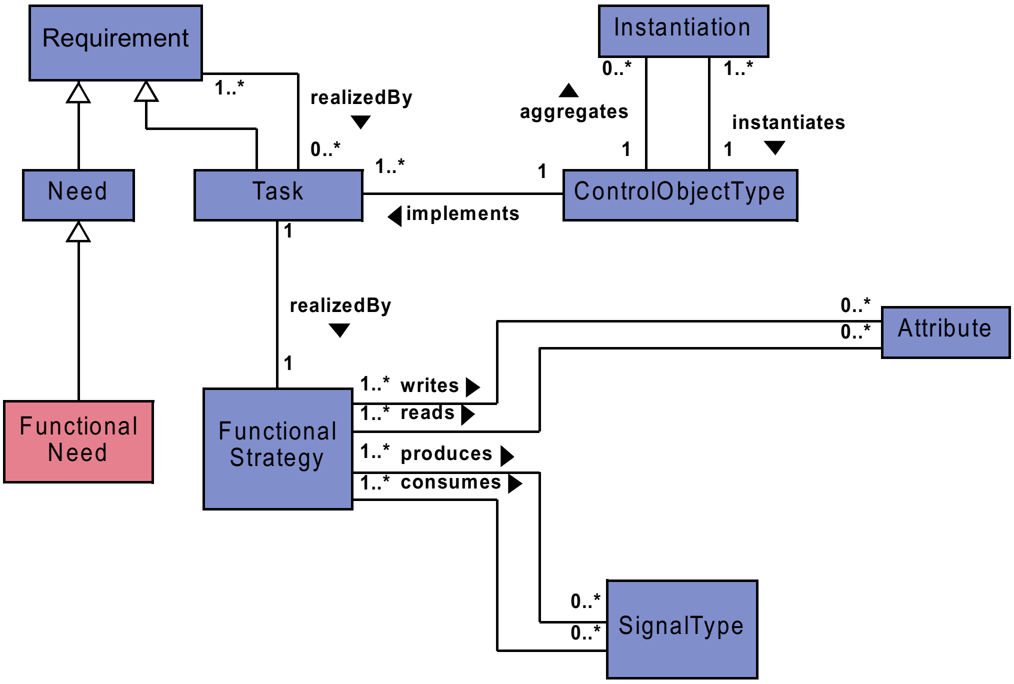
Artefact Types and Relations



— Feature Interaction Detection —

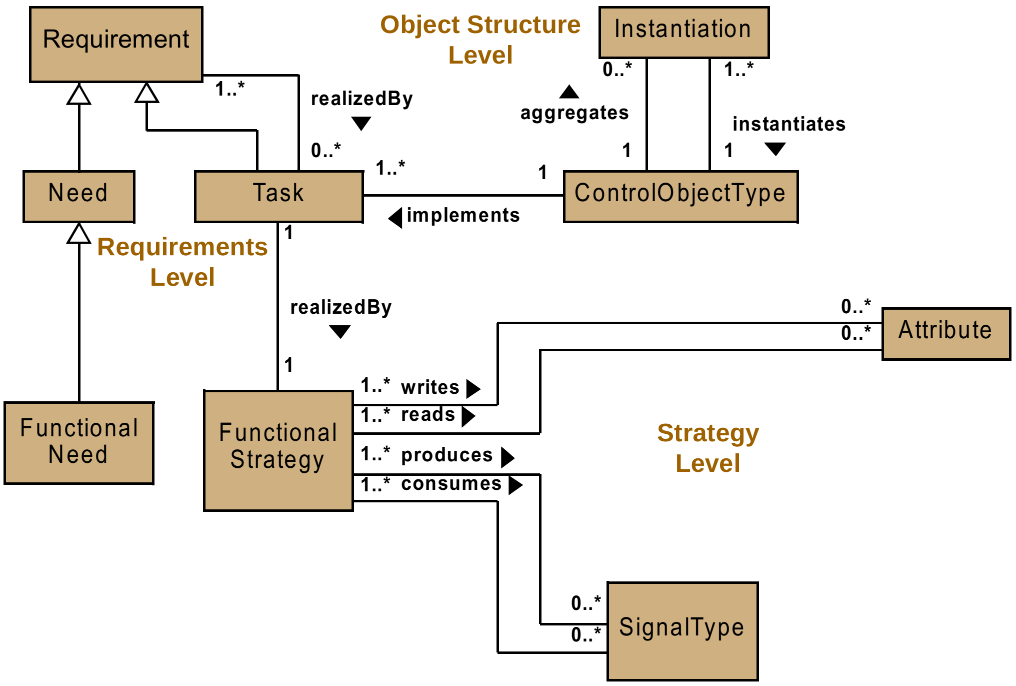
Feature Interaction Detection

Scheme



Feature Interaction Detection

Scheme



Requirement

Need

Task

Functional Need

Functional Strategy

Instantiation

ControlObjectType

Attribute

SignalType

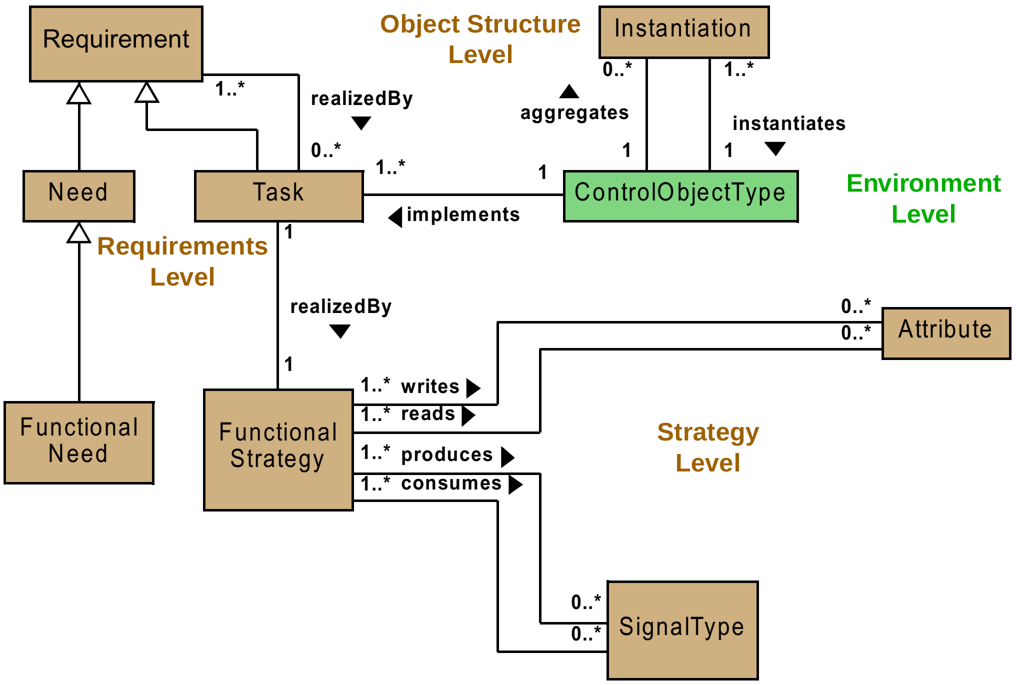
Object Structure Level

Requirements Level

Strategy Level

Feature Interaction Detection

Scheme



Requirement

Need

Task

Functional Need

Functional Strategy

Instantiation

ControlObjectType

Attribute

SignalType

Object Structure Level

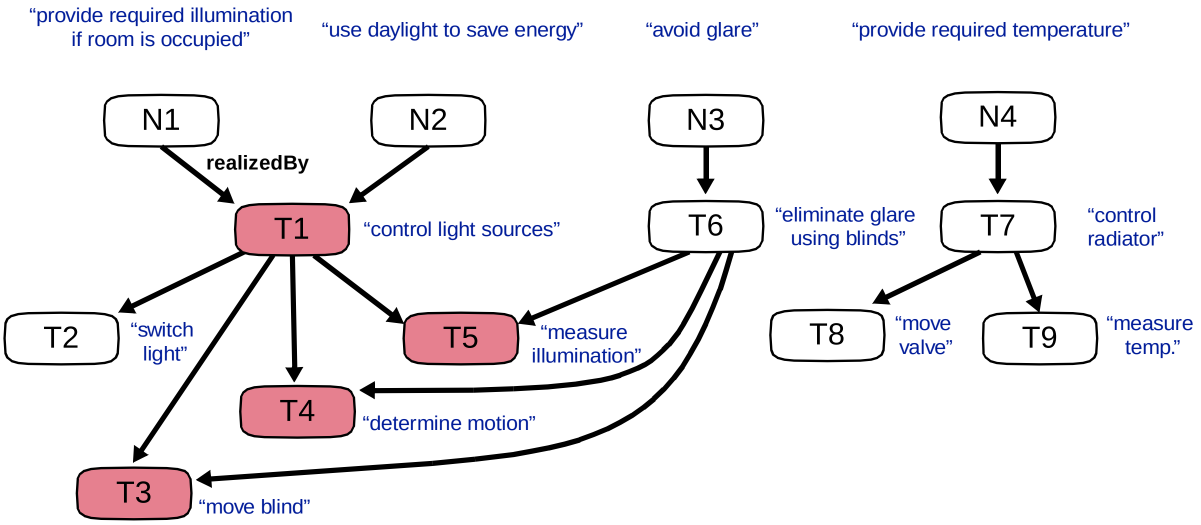
Requirements Level

Environment Level

Strategy Level

Feature Interaction Detection

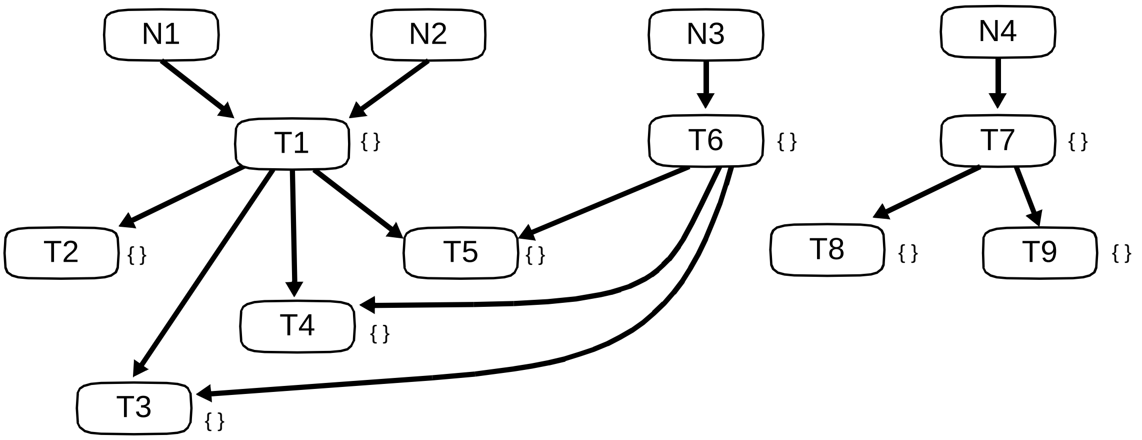
Detection at Requirements Level: Concept



Andreas Metzger – Feature Interaction Workshop – Ottawa, Canada – June, 2003

Feature Interaction Detection

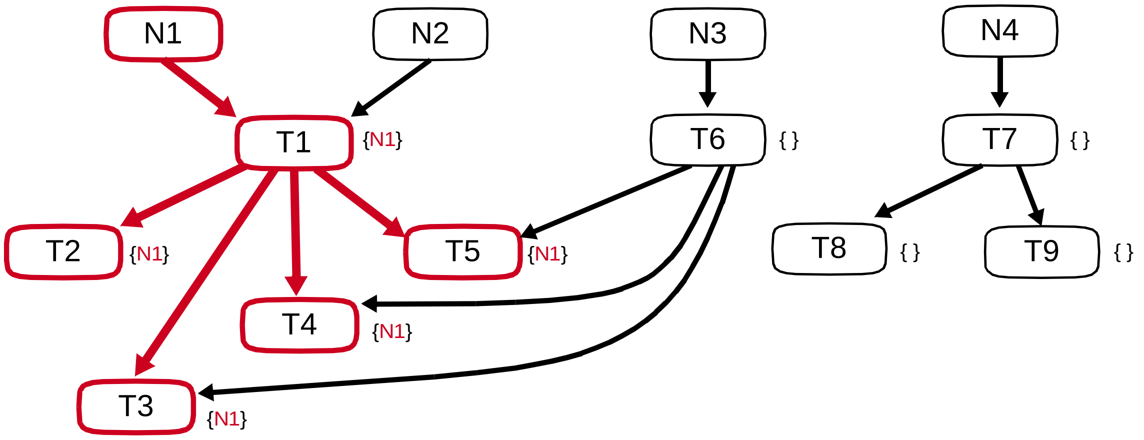
Detection at Requirements Level: Realization



Andreas Metzger – Feature Interaction Workshop – Ottawa, Canada – June, 2003

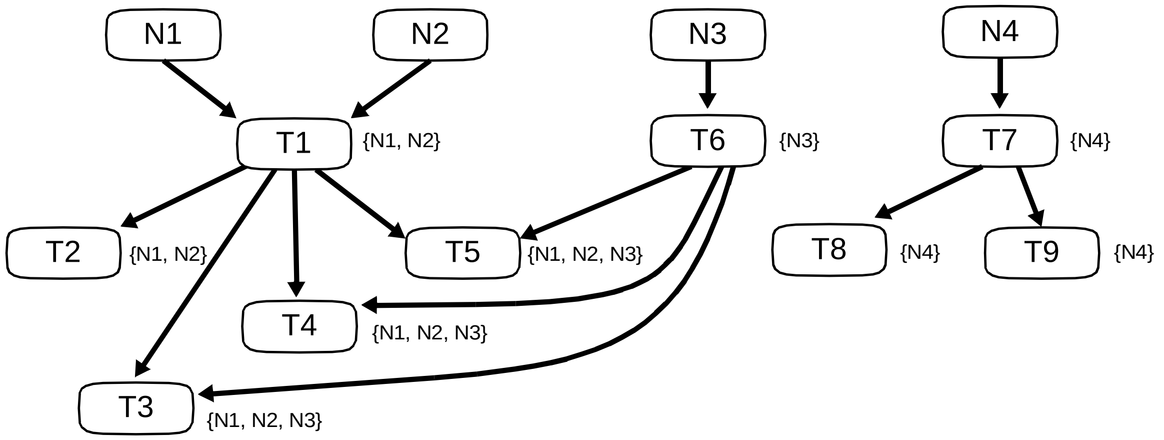
Feature Interaction Detection

Detection at Requirements Level: Realization

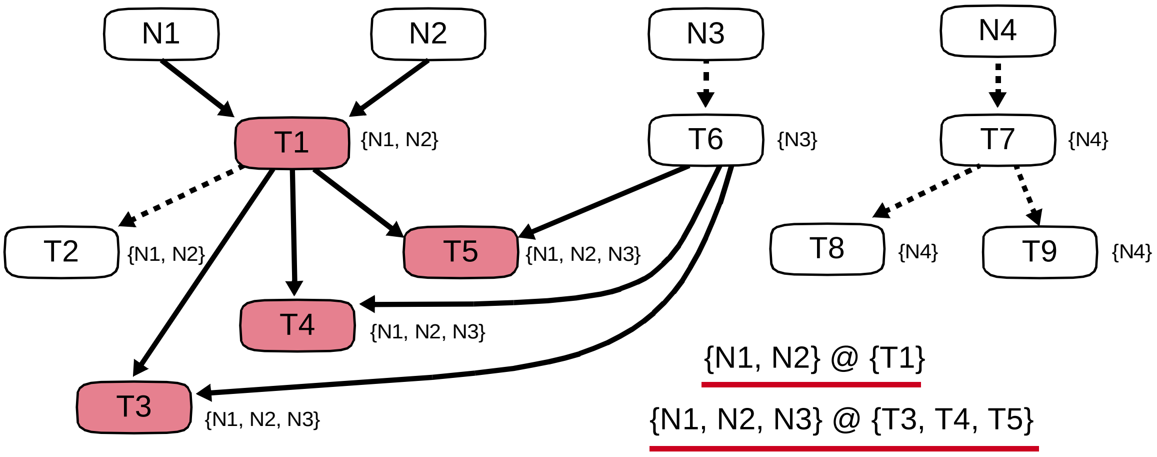


Feature Interaction Detection

Detection at Requirements Level: Realization



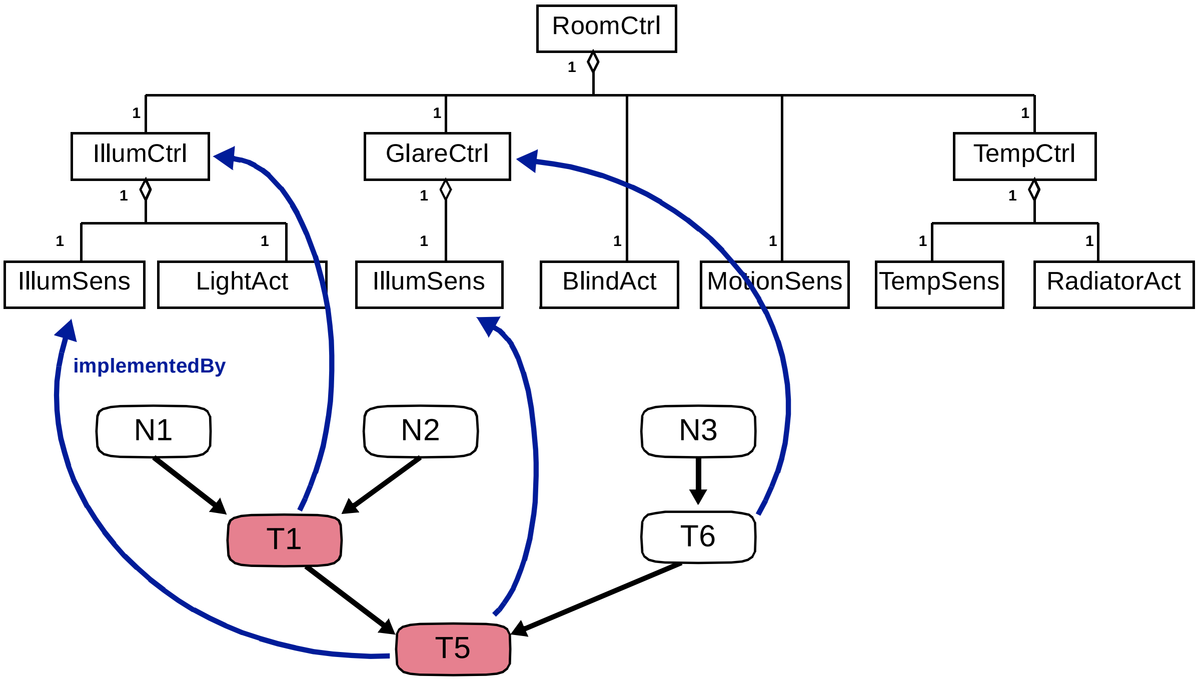
Detection at Requirements Level: Realization



{N1, N2} @ {T1}

{N1, N2, N3} @ {T3, T4, T5}

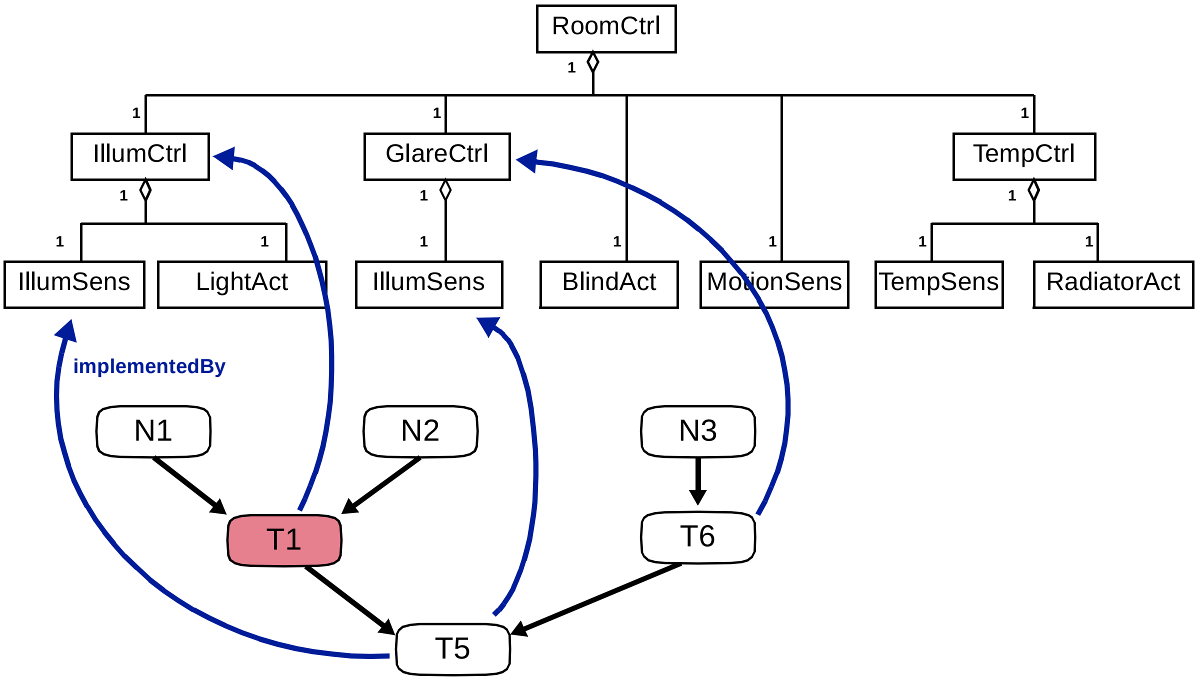
Detection at Object Structure Level: Concept



{N1, N2, N3} @ {T3, T4, T5}

Feature Interaction Detection

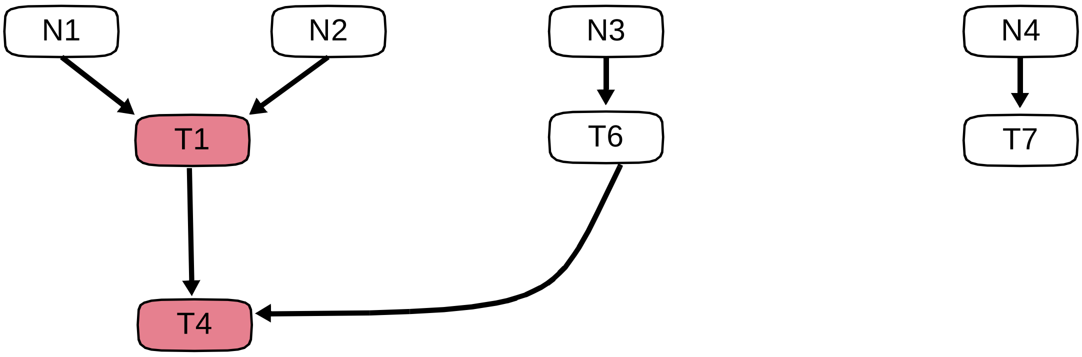
Detection at Object Structure Level: Concept



{N1, N2, N3} @ {T3, T4, ~~T5~~}

Feature Interaction Detection

Detection at Strategy Level: Concept

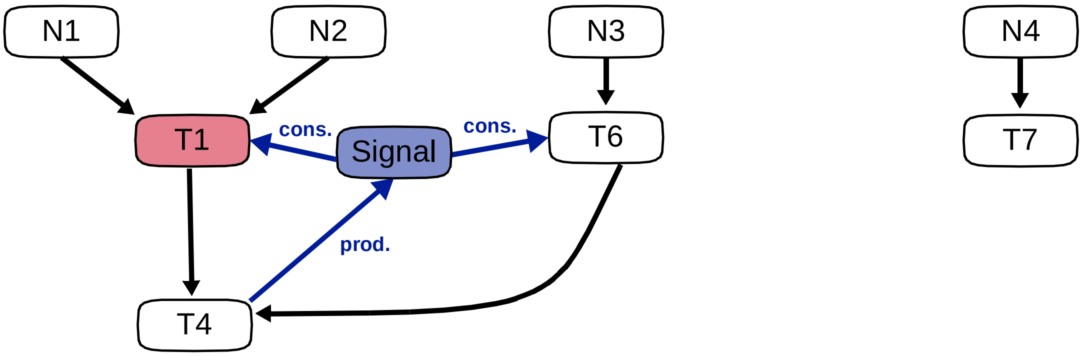


{N1, N2, N3} @ {T3, T4, ~~T5~~}

Signals/Attributes → Coupling of Strategies → Coupling of Tasks

Feature Interaction Detection

Detection at Strategy Level: Concept

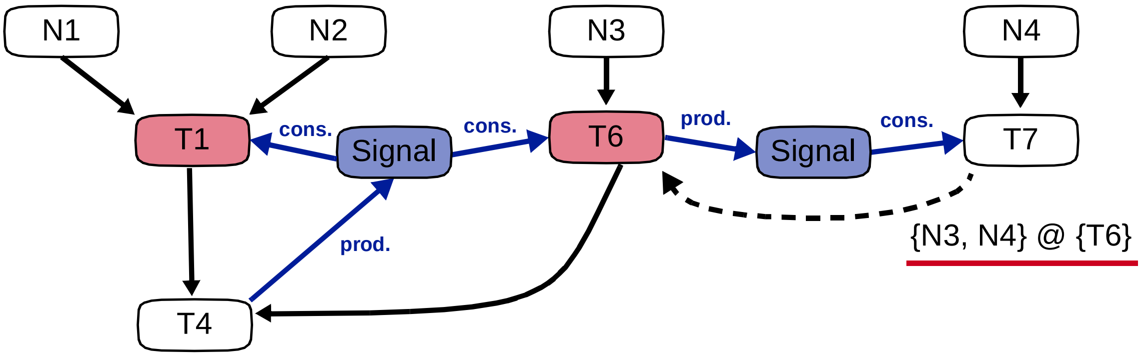


{N1, N2, N3} @ {T3, ~~T4~~, ~~T5~~}

Signals/Attributes → Coupling of Strategies → Coupling of Tasks

Feature Interaction Detection

Detection at Strategy Level: Concept



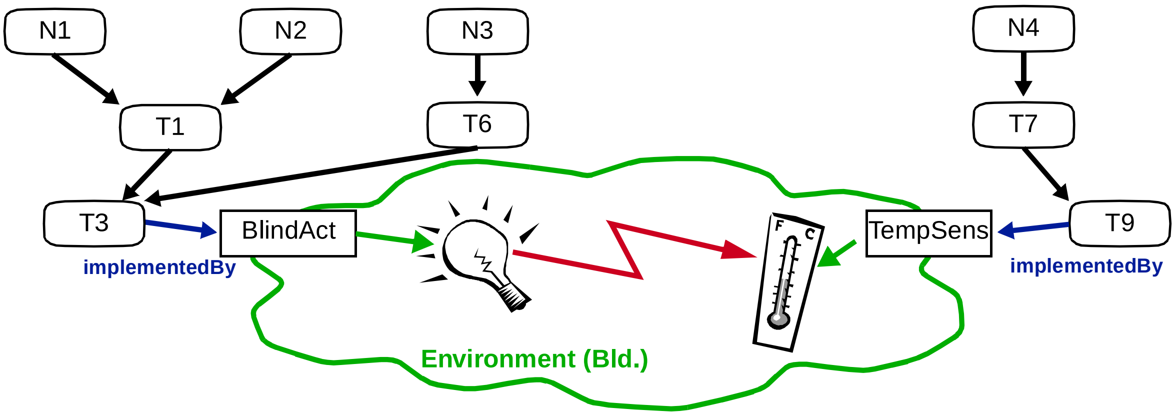
{N3, N4} @ {T6}

{N1, N2, N3} @ {T3, ~~T4~~, ~~T5~~}

Signals/Attributes → Coupling of Strategies → Coupling of Tasks

Feature Interaction Detection

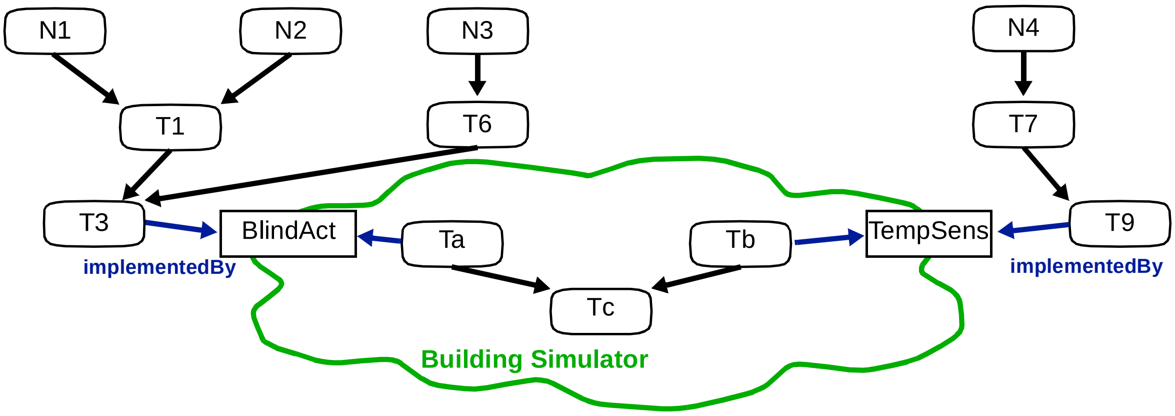
Detection at Environment Level: Concept



{N1, N2, N3, N4} @ Env

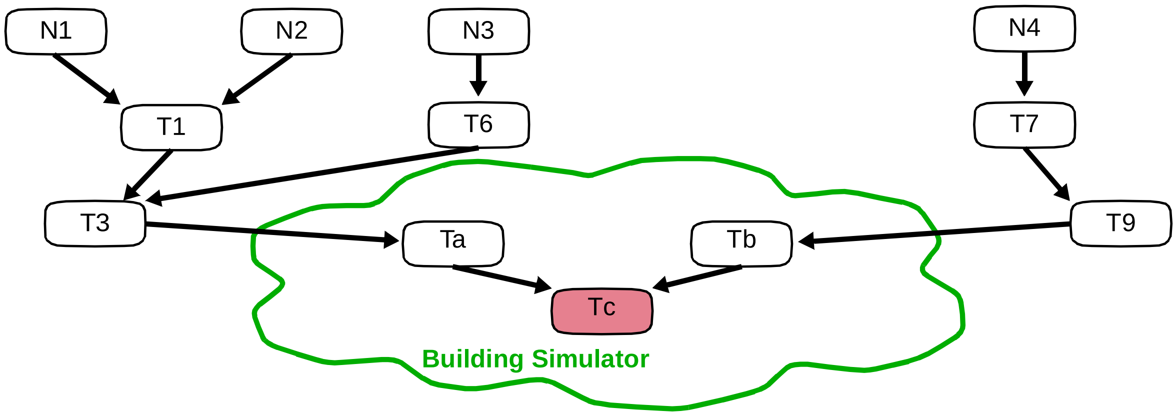
Feature Interaction Detection

Detection at Environment Level: Realization



Feature Interaction Detection

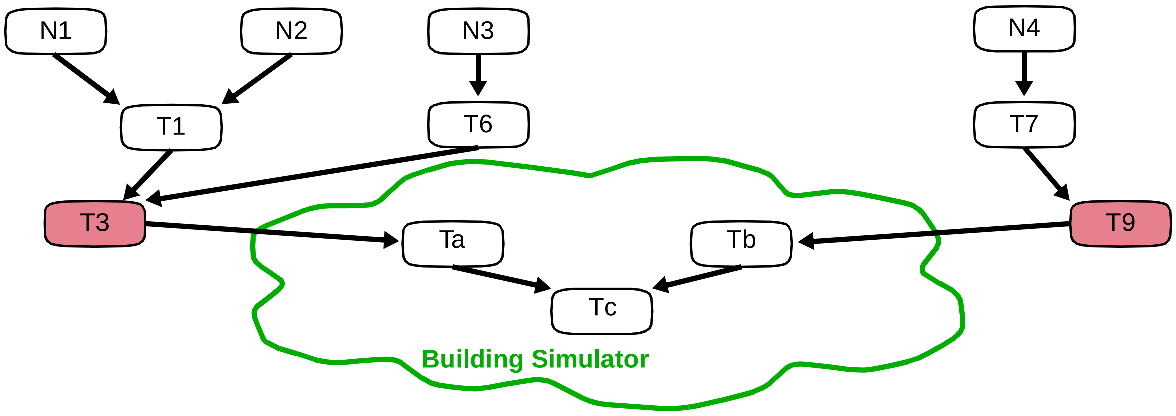
Detection at Environment Level: Realization



{N1, N2, N3, N4} @ {Tc}

Feature Interaction Detection

Detection at Environment Level: Realization



{N1, N2, N3, N4} @ {T3, T9}

— Conclusion —

Conclusion

General Applicability

Level of Information	***Domain Dependency of Approach***	***Method Dependency of Approach***
Require-ments	none	weak (traceability relation)
Object Structure	strong (strict aggregation, static structure)	weak (aggregation rel., traceability rel.)
Strategies	weak (reactive systems: signals)	strong (traceability relation)
Environment	strong (reactive system)	strong (environment simulator)

Efficiency

Feature Interaction Detection Tool: ~1900 Lines of Code (Java)

Conclusion

- Efficient Approach for FI Detection in Building Control Systems
  - Systematic Mapping of Concepts to Code
  - Abstraction from Development Documents (Product Model)
- Application During System Development
  - Detection of Interactions after Extension/Reuse Activity
  - Metric (“Complexity of System”) → Quality Control

Perspectives

- Extension of Application Domain
  - Reactive Systems with Static Structure; e.g., Automotive Control
- Refinement of Detection Concepts
  - Static Analysis of Behavior (Refinement of Product Model)

— Additional Slides —

Results

Case Studies

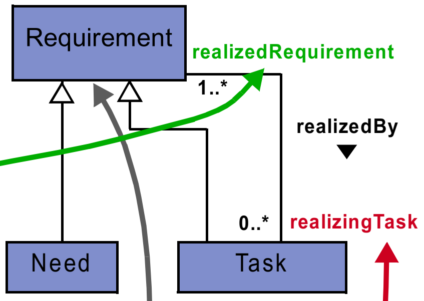
Complexity	*Floor32*	*Floor32X*	Δ
Requirements (Needs, Tasks)	285 (52, 233)	316 (64, 252)	31 (12, 19)
Control Object Types	37	40	3

Feature Interactions @ Points of Interaction	*Floor32*	*Floor32X*	Δ
at Object Structure Level	32 @ 47	38 @ 53	6 @ 6
at Environment Level	38 @ 63	44 @ 69	6 @ 6
Δ	6 @ 6	6 @ 16	

Feature Interaction Detection

Detection at Requirements Level: Implementation

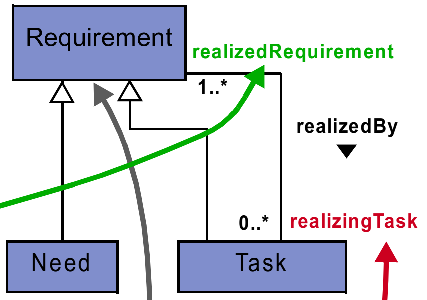
```
class Main {

 detectInteraction() {

  forall(need ∈ Needs) {
   need.mark(need);
  }

  forall(task ∈ Tasks) {

   if(|task.needSet| > 1 AND
      |task.realizedRequirement| > 1)

         /* True Point of Interaction
            identified! */
   }

  }

 }

}
```



```
class Requirement {

 Set needSet;

 mark(Need need) {
  needSet = needSet ∪ need;

  forall(task ∈ realizingTasks) {
     task.mark(need);
  }

 }

}
```

Feature Interaction Detection

Detection at Requirements Level: Implementation

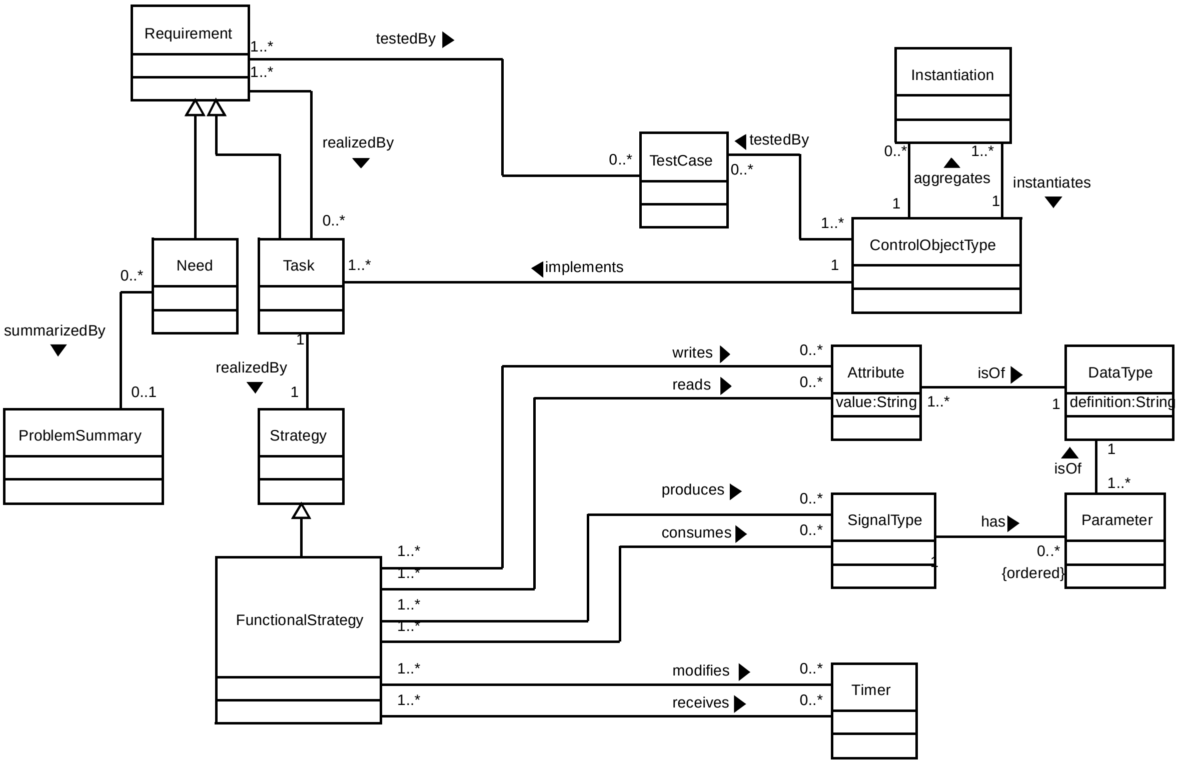


```
public class Main {

 void detectInteraction() {
  Iterator allNeeds = /* ... */;
  Need need;
  while(allNeeds.hasNext()) {
   need = (Need)allNeeds.next();
   need.mark(need);
  }

  Iterator allTasks = /* ... */;
  Task task;
  while(allTasks.hasNext()) {
   task = (Task)allTasks.next();
   if(task.getNeedSet().size() > 1 &&
      task.getRealizedRequirement().
       size()) > 1) {
         /* True Point of Interaction
            identified! */
   }
  }
 }
}
```

```
public class Requirement {
 Set needSet;

 void mark(Need need) {
  needSet.add(need);
  Iterator it = getRealizingTask();
  Task task;
  while(it.hasNext()) {
   task = (Task)it.next();
   task.mark(name);
  }
 }
}
```

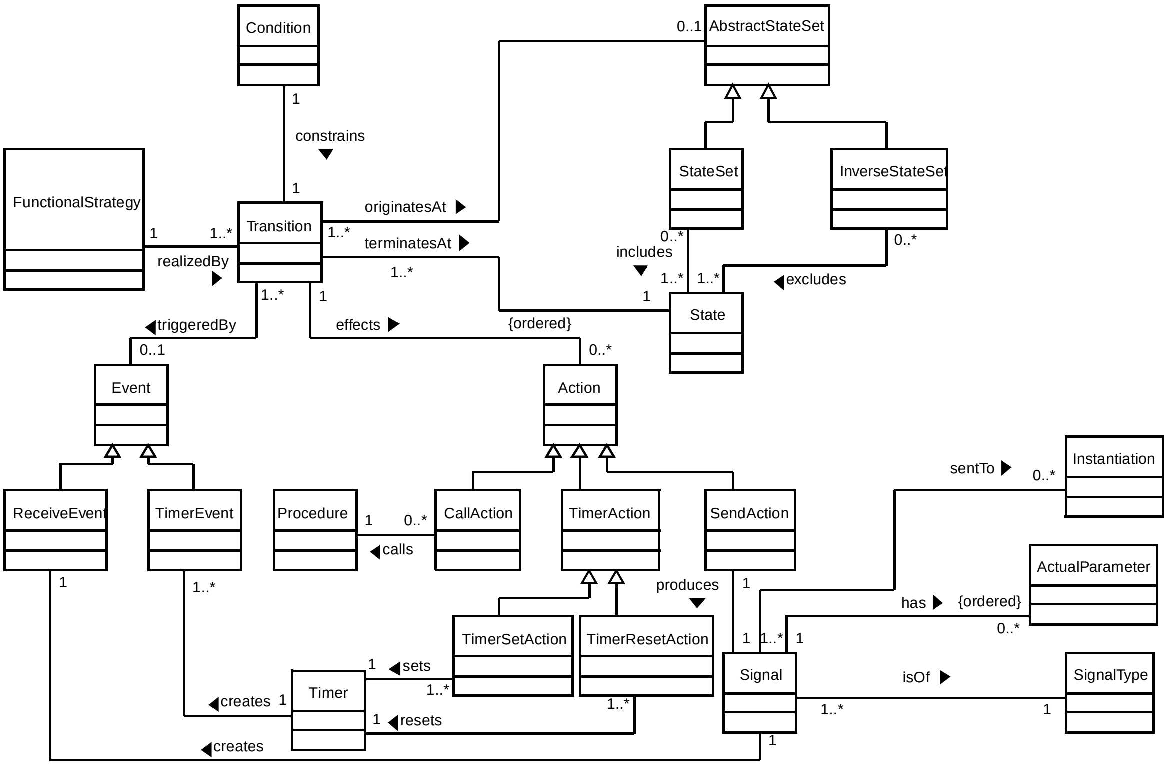
The Product Model Approach

Extended Artefact Types and Relations (Part 1)



The Product Model Approach

Extended Artefact Types and Relations (Part 2)


Condition
0..1
AbstractStateSet
1
constrains
1
FunctionalStrategy
Transition
originatesAt
StateSet
InverseStateSet
1
1..*
1..*
terminatesAt
0..*
includes
0..*
realizedBy
1..*
1..*
1..*
excludes
1..*
1
1
State
triggeredBy
effects
{ordered}
0..1
0..*
Event
Action
sentTo
0..*
Instantiation
ReceiveEvent
TimerEvent
Procedure
1
0..*
CallAction
TimerAction
SendAction
calls
ActualParameter
1
1..*
produces
1
has
{ordered}
TimerSetAction
TimerResetAction
1
1..*
1
0..*
1
sets
Signal
1..*
isOf
SignalType
creates
1
Timer
1..*
1..*
1
1
resets
1
creates