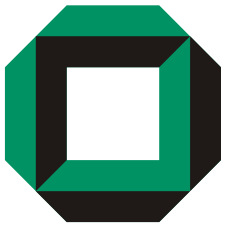


**Universität Karlsruhe (TH)**
Forschungsuniversität • gegründet 1825

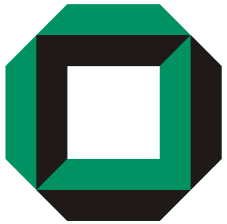

# Software Components and Software Architecture

## Software Design on its Road to an Engineering Discipline

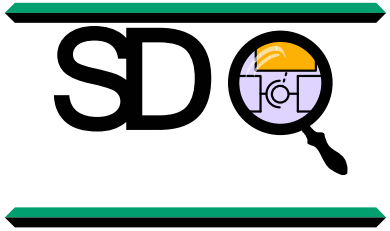
**Prof. Dr. Ralf Reussner**

**Vortragender: Steffen Becker**

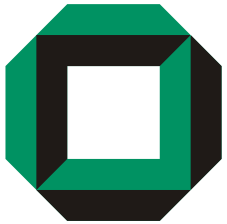
**Abteilungsleiter Software Engineering, FZI**



# Overview

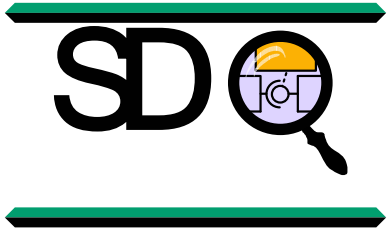



- Is Software Engineering an Engineering Discipline?
- Role of Software Components
- Palladio Component Model
  - Parametric Contracts
  - Prediction of Quality Properties
- Example
- Conclusions



# Elements of an Engineering Discipline

[Shaw&Garlan95]




## Craft

- Customer and Developer often the same person
- Talent ad Experience instead of Understanding

## Industrial Manu-facturing

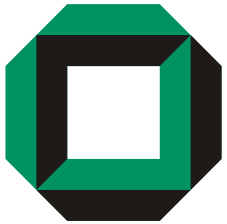
- Division of Labor
- Education of Specialists
- Use of third party tools

## Engineering

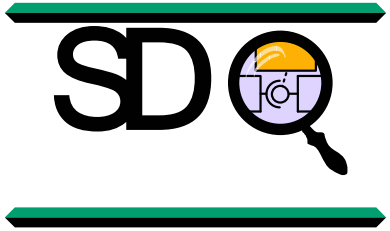
- Goal-driven optimisation of
  - Products
  - Processes

  requires
- Understanding of the effects of design decisions and changes

→ Theories on products and processes



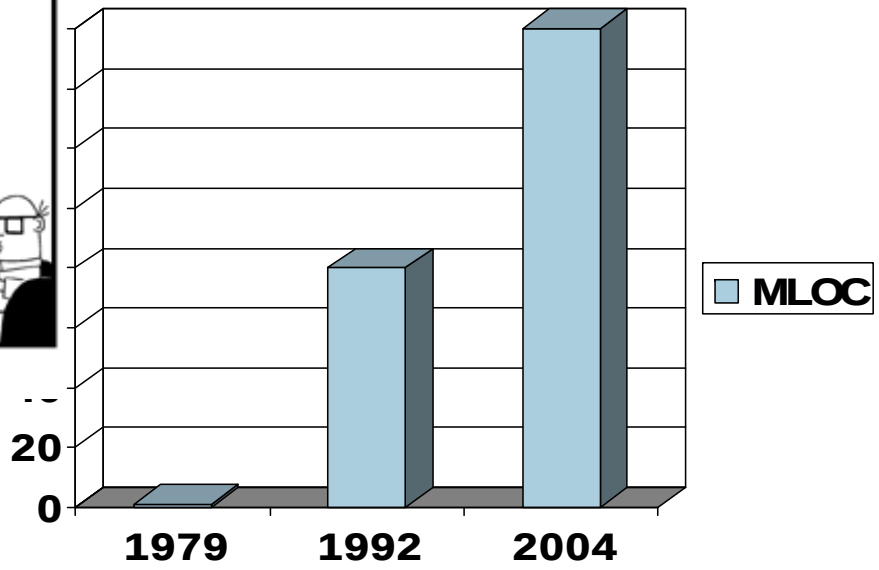
# No Progress in SE?




- The same problems since 1968 (first Software Engineering Conference)
- „the problem of achieving sufficient reliability in the data systems...“
- „the difficulties of meeting schedules and specifications on large software projects“
- „the highly controversial question of whether software should be priced separately from hardware“

# Where stands „Software Engineering“ as an Engineering Discipline?

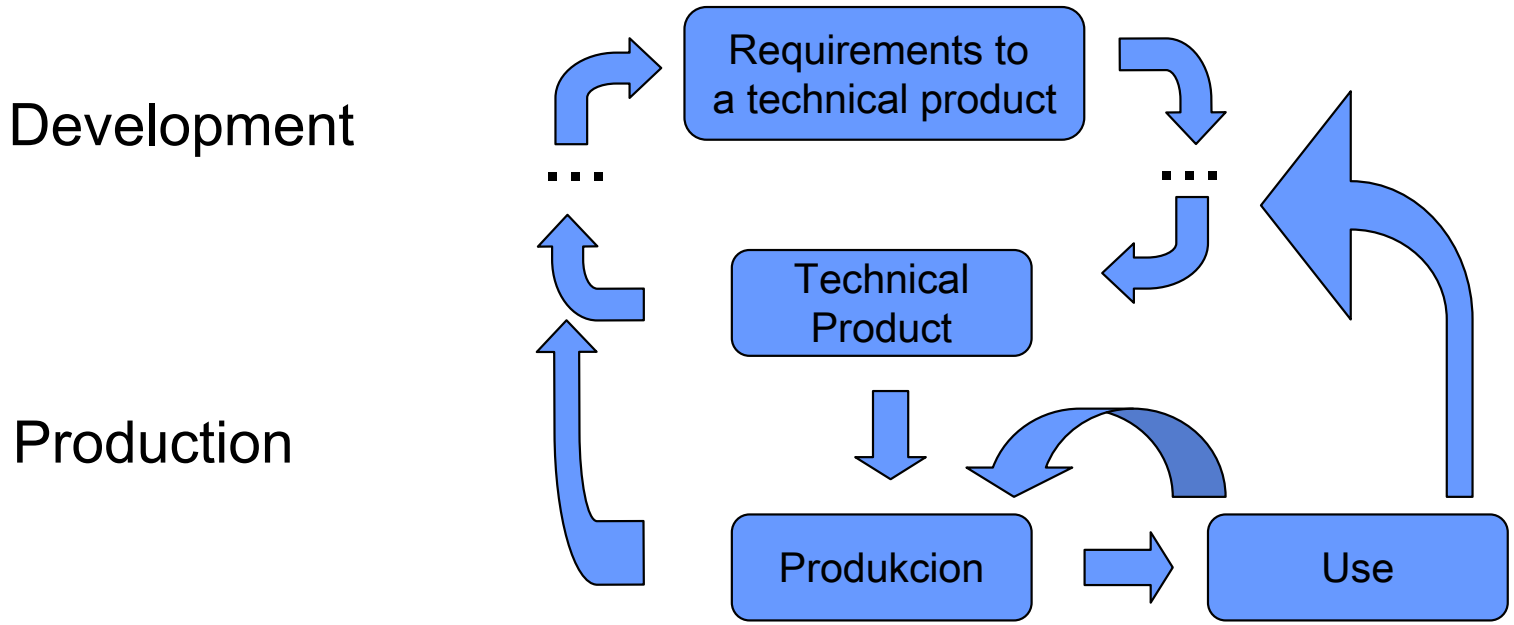

© Scott Adams, Inc./Dist. by UFS, Inc.



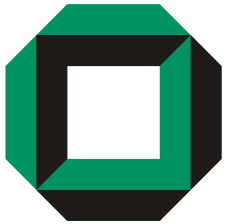

Size of what is considered as “large” software systems

- Progress: the same problems since decades, but for considerably larger and complex systems
- „Planing crisis“ instead of a „Software crisis“ [Glass00]:
  - Budgets and schedules are rarely done by the developer, much more by managers, sales persons and customers

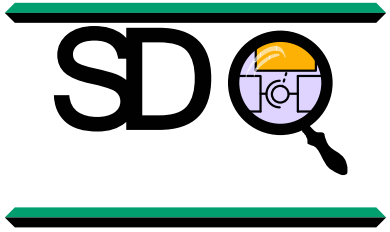
# Development and Production




- Technical Production: well understood, planable, repeatable
- Problems of Software Engineering are problems in development, not production



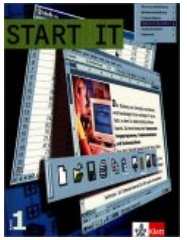
# Software Engineering: industrial manufacturing

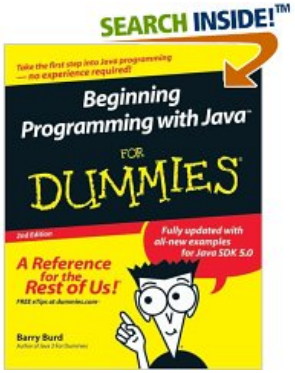



- division of labor
  - Roles
  - Tools (e.g., Versioning)
- Use of specialised tools
- (Spezialised Education)
- Design patterns as a vocabular on proven solutions to recurring problems

# Problems

- Lack of Understanding and Professionalism
  - „New Motors in three month.“
  - „Sky scrapers in 5 days.“
  - Why do not we find books like:
    - „Heart Transplantations for Dummies“
    - „Nuclear Weapons in 21 days“
    - „Flying the Airbus: Easy Access!“
  - Sky scrapers as large garden houses
- Counter productive avoidance of up front costs
- Real problem of integrating and using legacy systems





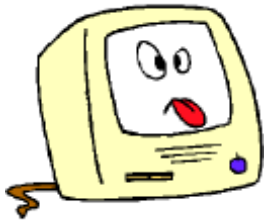
# Treatment of Quality Properties today

1. Specification

2. Ignoring

3. Testing

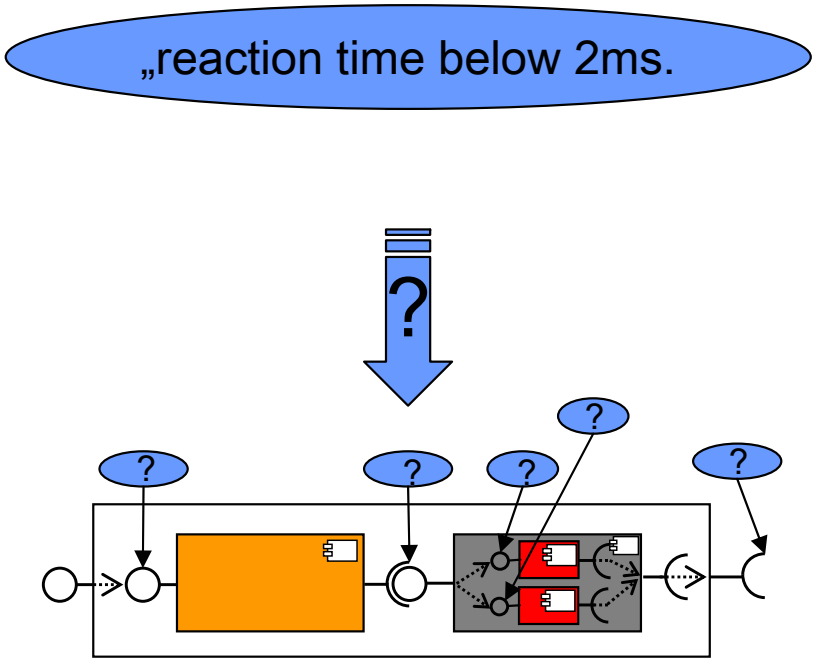
4. Re-Implementing / Re-Designing / Re-Negotiating



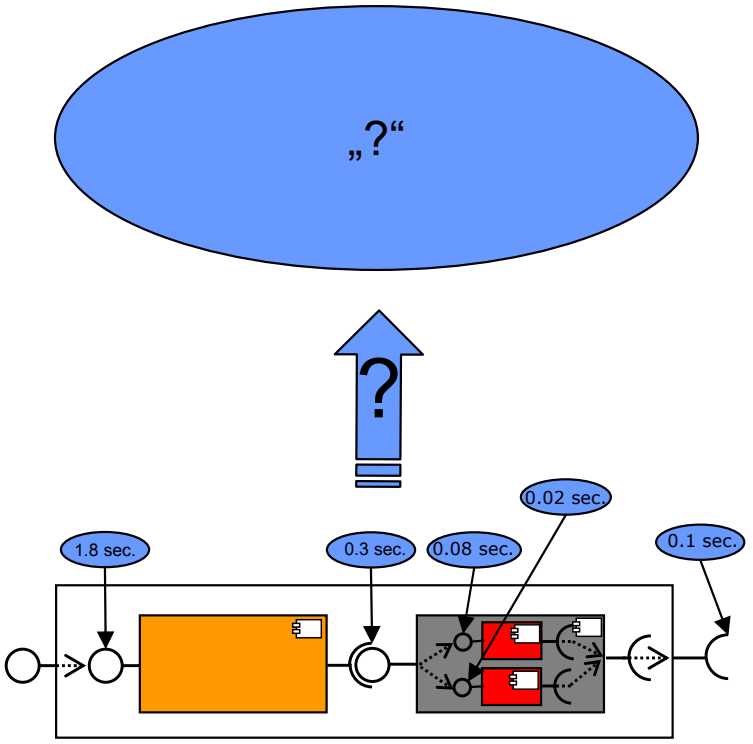
# Missing Properties of an Engineering Discipline

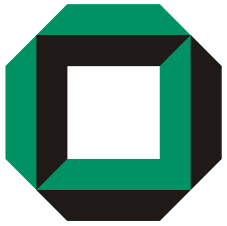
- Systematic Treatment of Quality Attributes

Decomposition of global System-Requirements

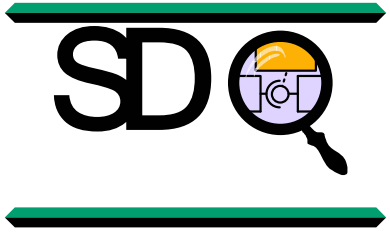



Prediction of global System-Properties



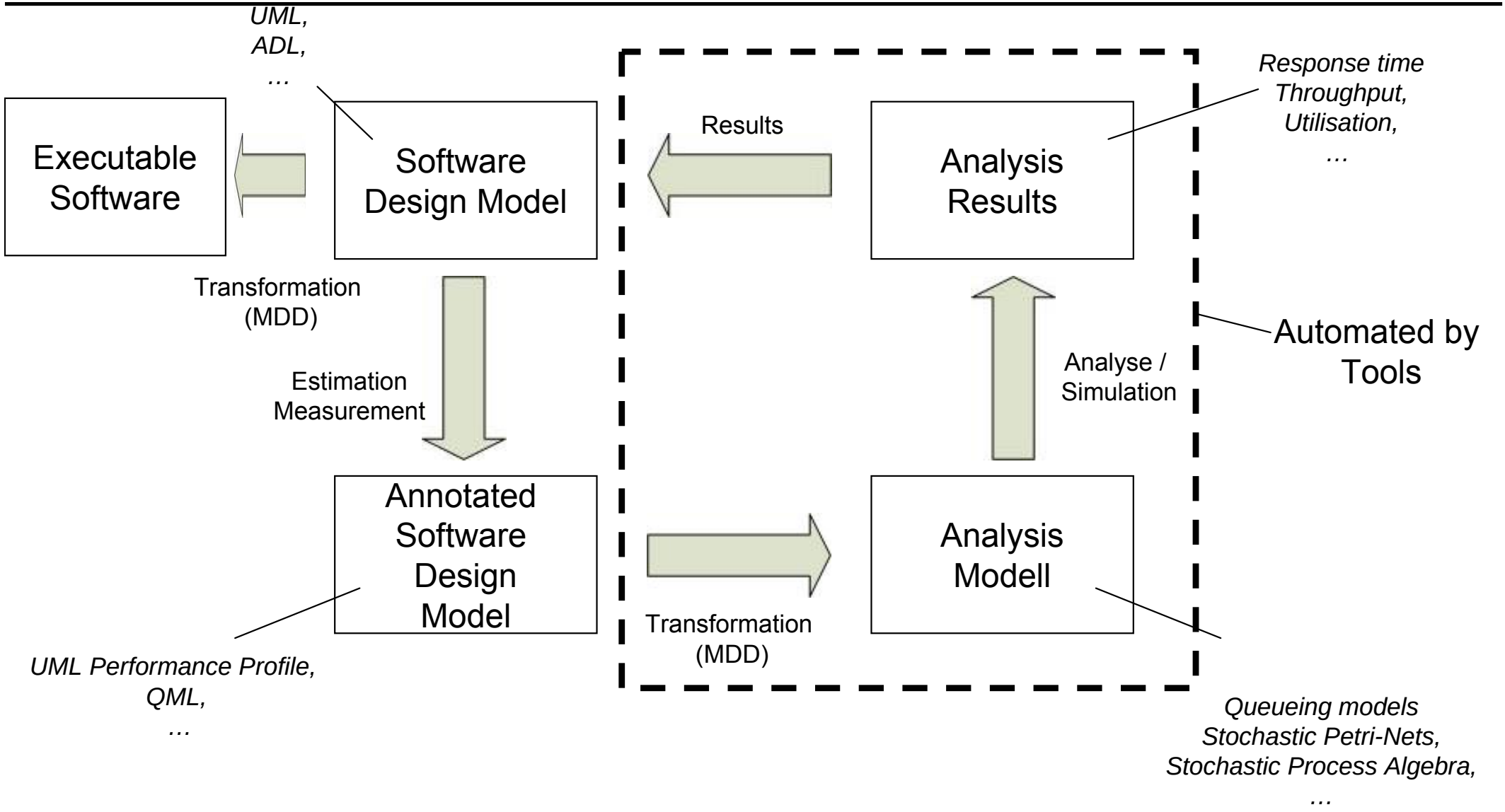


# Role of Components in an Engineering Discipline



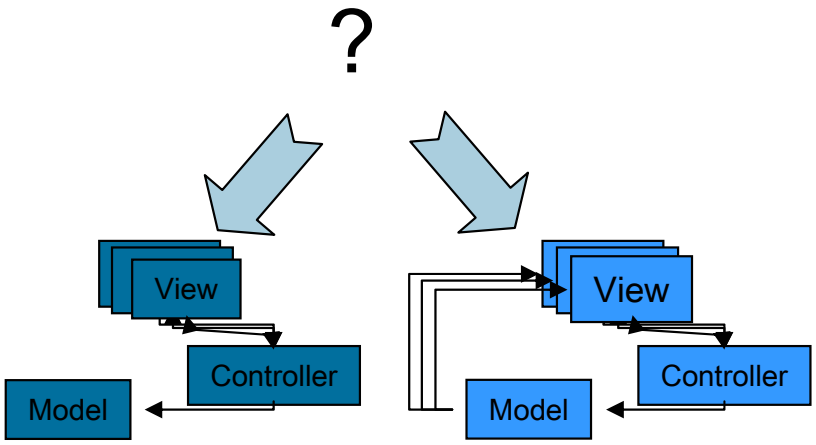

- All engineering disciplines have components.
- Components lower the degrees of freedom during development and, hence, increase the predictability of quality attributes.
- The re-use of components blurrs the boundaries between development of new software, evolution of software and integration of software (which reflects just the reality).
- Re-use of components / composition of systems is isomorphic to re-use / composition of prediction models

# Model-based Prediction of Quality



# Scenarios for Model-based Quality Prediction

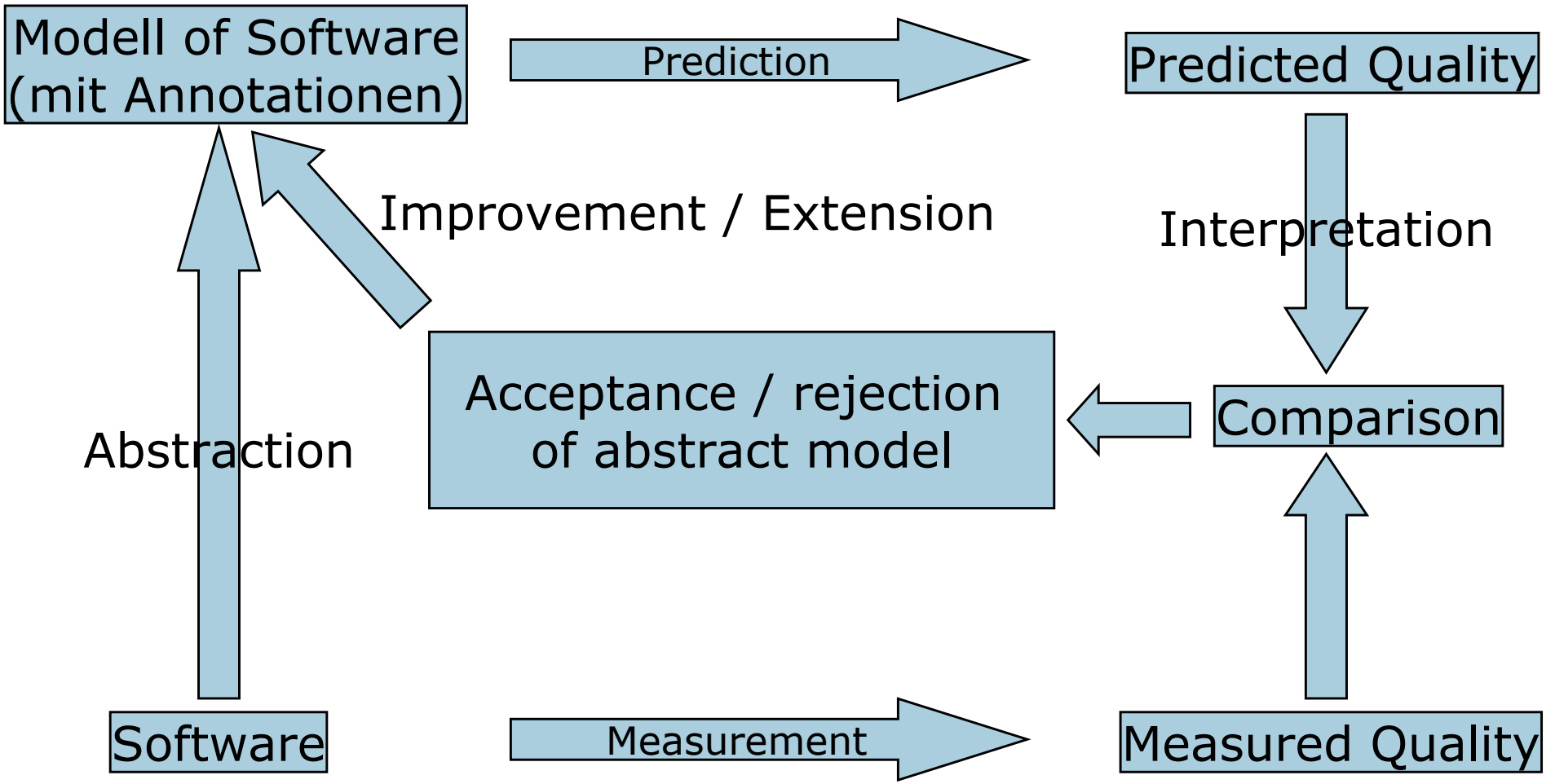
- Evaluation of Design Alternatives




- Dimeniosing of Ressources (“sizing”)

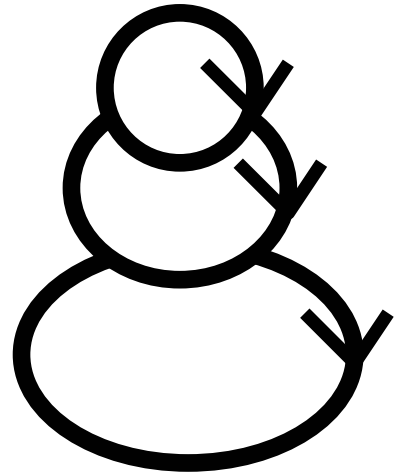
- Change of Usage Profile

# Scientific Approach

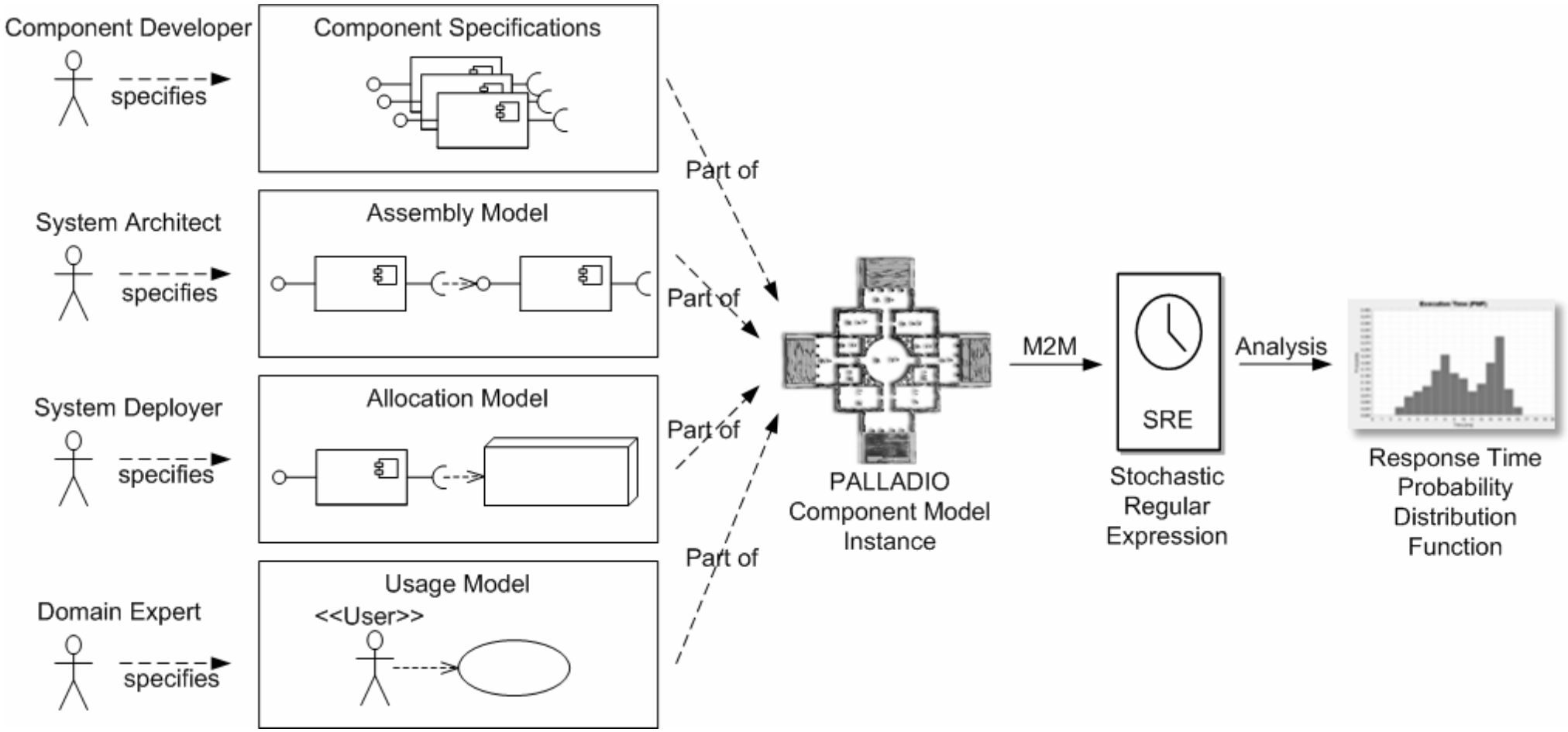


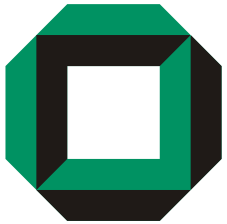
# Further Validation and Deployment

- Type 1: Validation of Prediction Model
- Type 2: Validation of Applicability
  - Case Studies and Controlled Experiemts with Students
- Typ 3: Validation of Benefits
  - in comparison to different methods
  - Limitations of the Approach
  - Required prerequisites
  - FZI
  - Industrial Partners

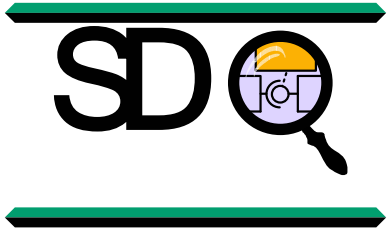


# Palladio Component Model





# What is a component?



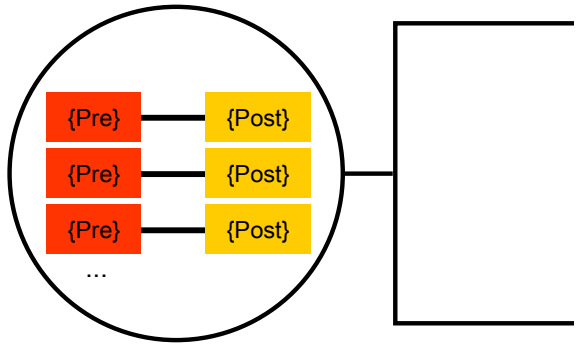

- "A component is a contractually specified building unit of software which can be readily composed or deployed."
  - "readily composed or deployed":
    - without having to understand the interna as a human
    - these are the two main things to be done with components
  - not necessarily "black-box": Information on interna can be available to tools.
- "Components are for composition, much beyond is unclear…" (Clemens Szyperski)

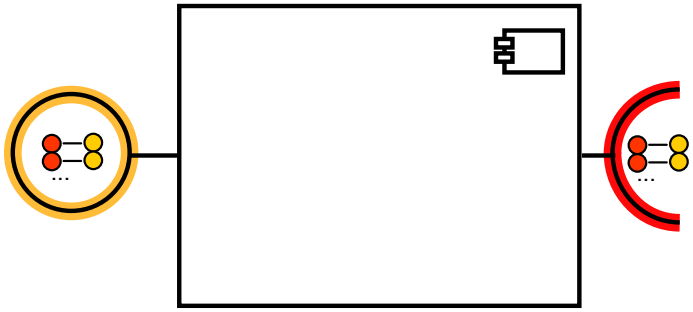
# Contractual use of Components

- Design-by-contract (B. Meyer, 1992)
  - " The service supplier guarantees the post-condition, if the client guarantees the precondition of the service."
- Contractual Use of Components

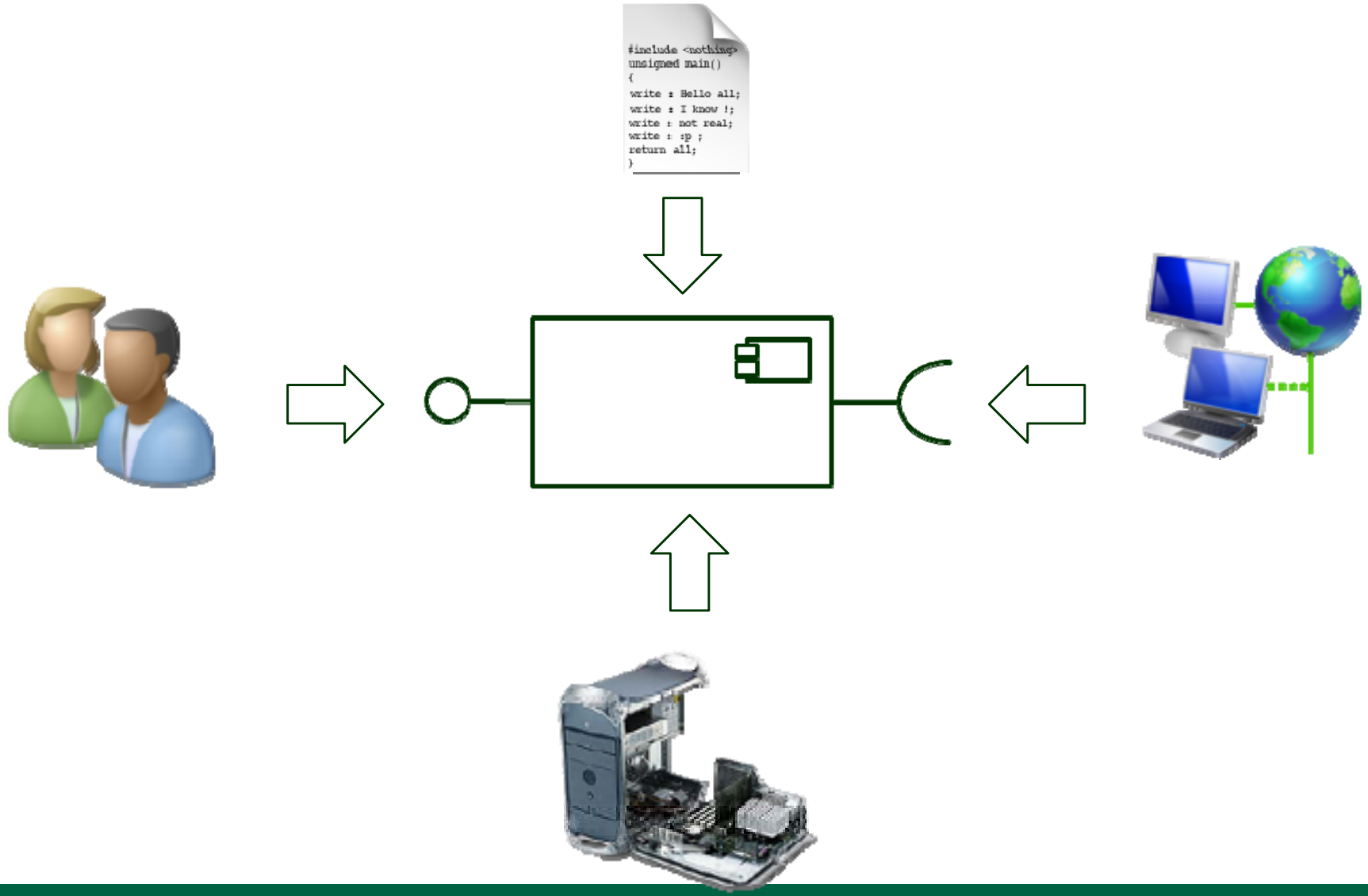
  (at system-(re-) configuration time)

  "The component offers the provided services (as specified in the provides interfaces), if the environment guarantees the required services (as specified in the requires interfaces)."



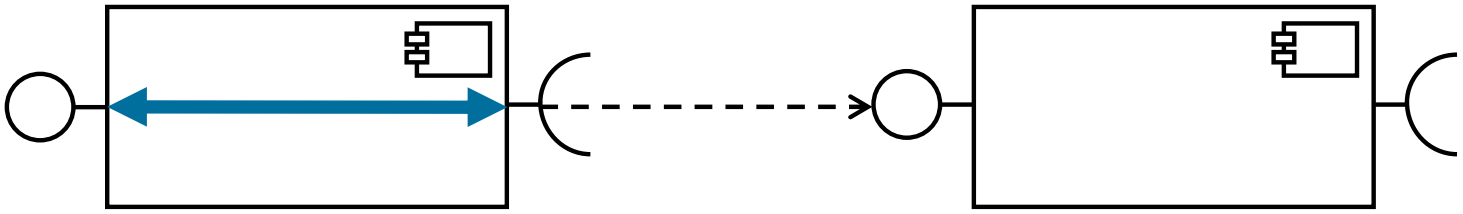



# Factors on Component Quality



# Parametric Contracts for Components

- Computation of environment-dependent provides-interface
- Computation of use-dependent requires-interface (‘wp-calculus’)
- Fine-grain quality description / adaptation of large-grain components



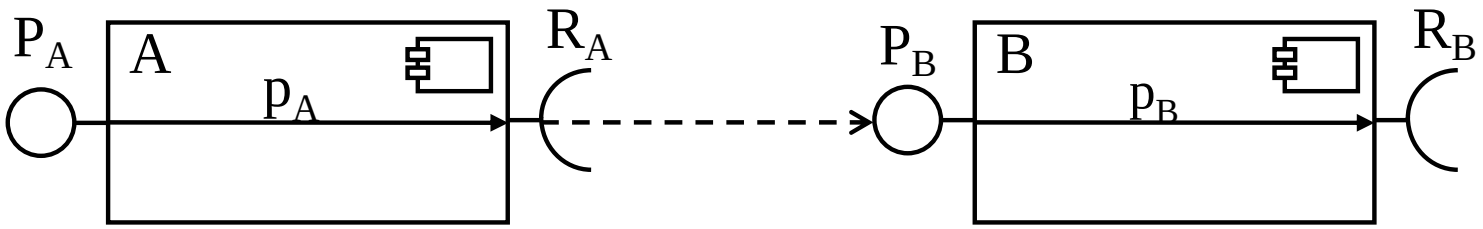
# Computation of Parametric Contracts

- Computation of context-dependent provides-interface:

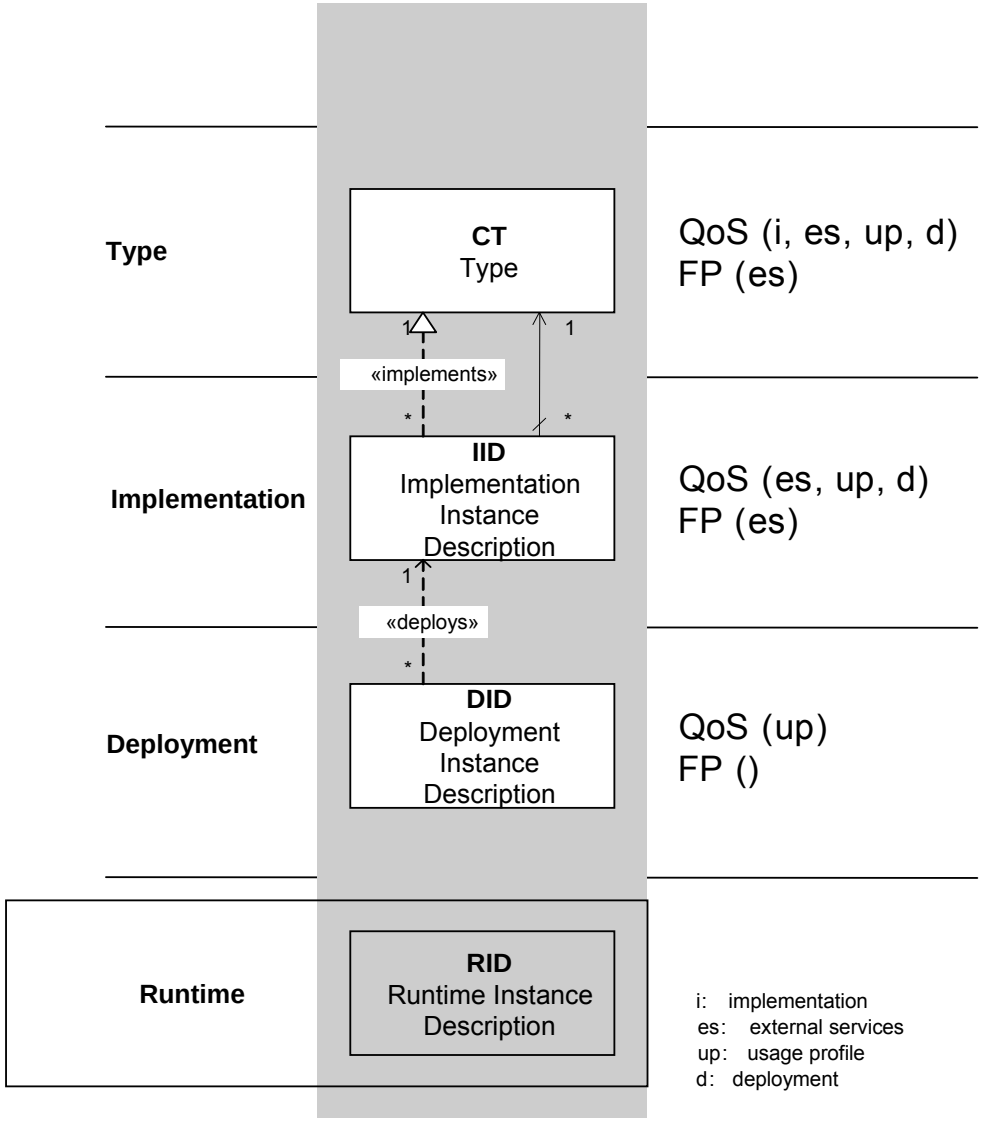
1. $(R_A = p_A(P_A))$
2. $R_{A'} := R_A \cap P_B$
3. $P_{A'} = p_A^{-1}(R_{A'})$

- Computation of context-dependent requires-interface:

1. $P_{B'} := R_A \cap P_B$
2. $R_{B'} = p_B(P_{B'})$

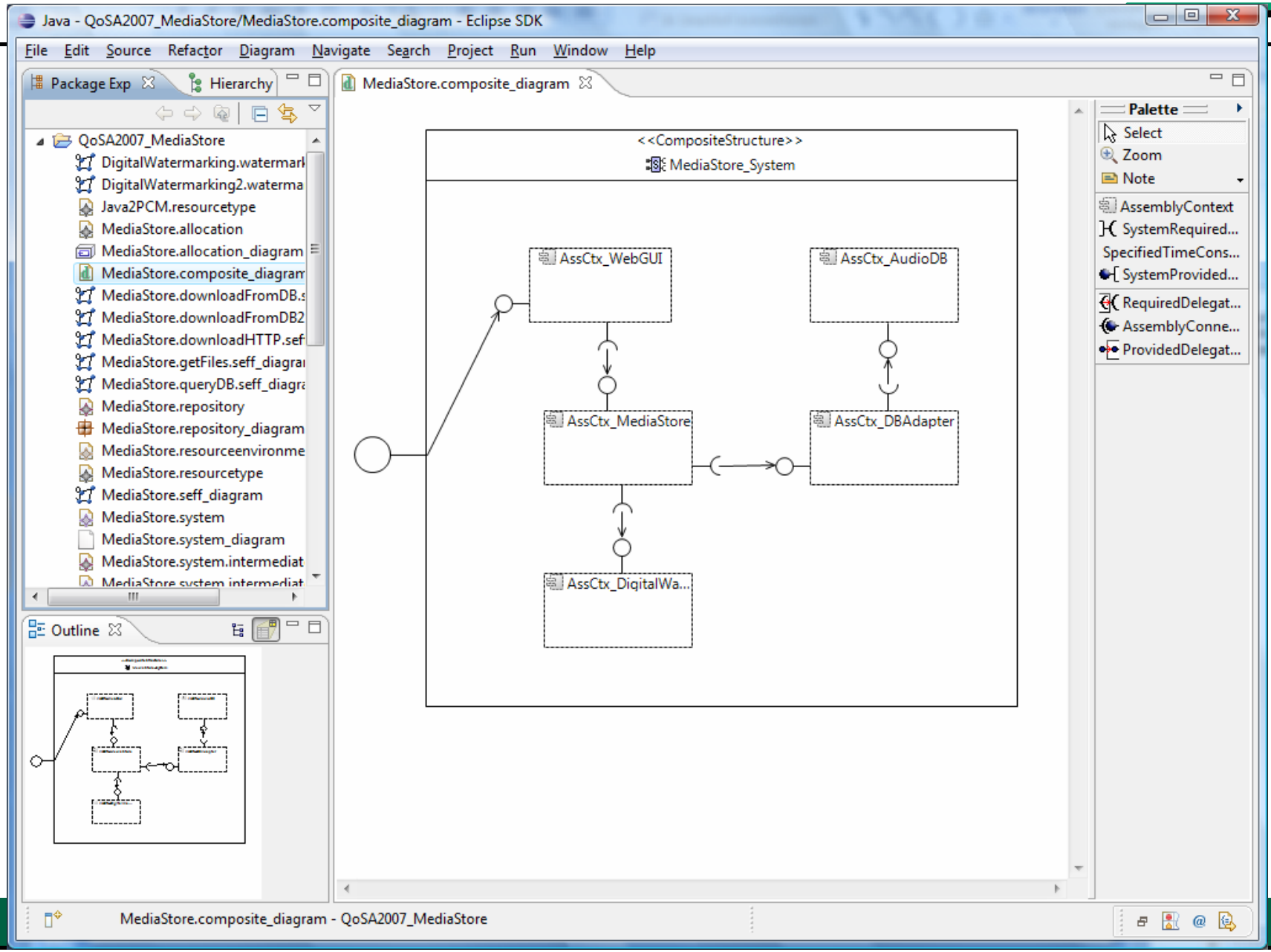


# Different Abstraction of Components

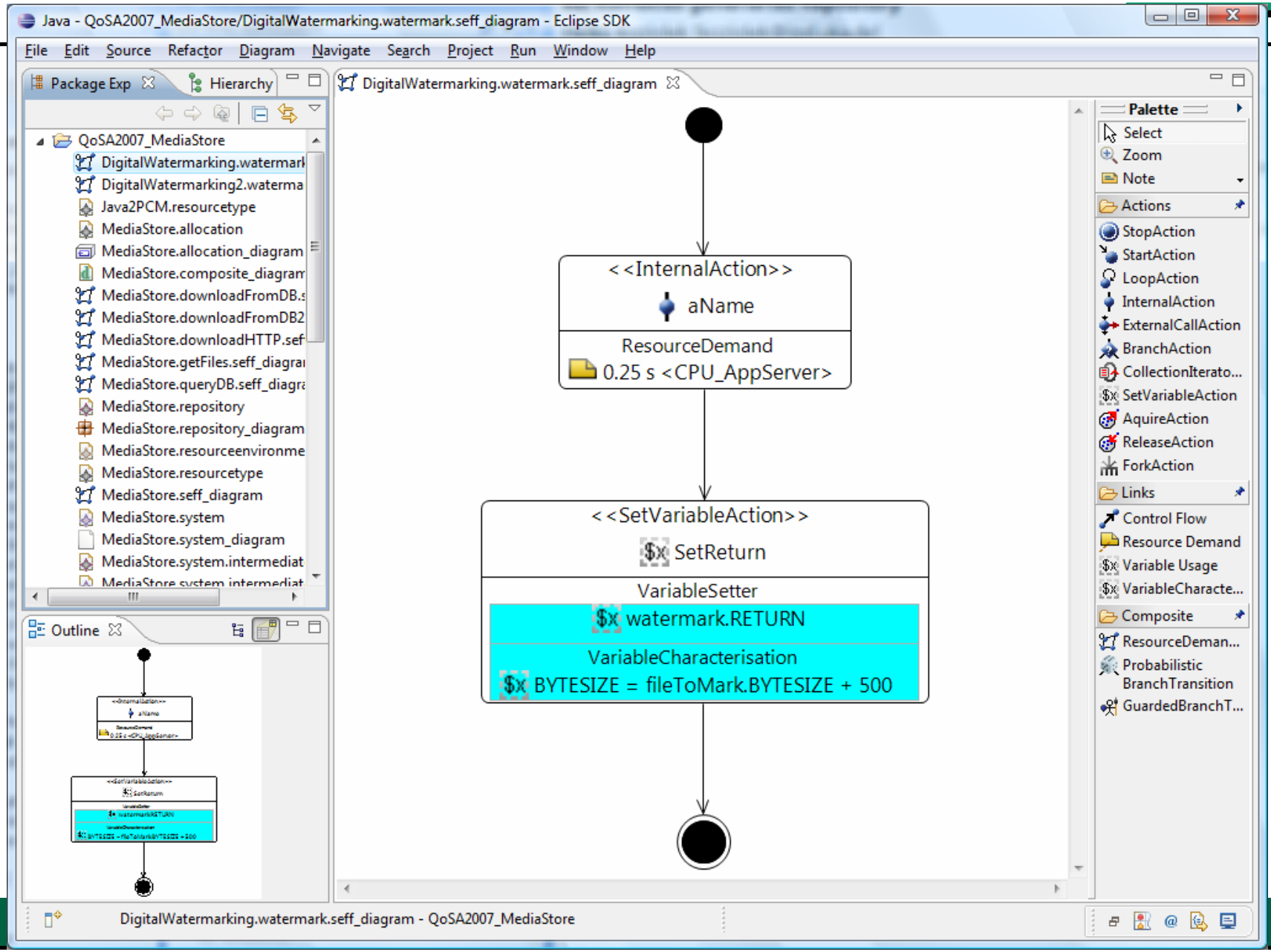



Not considered within the Palladio ComponentModel

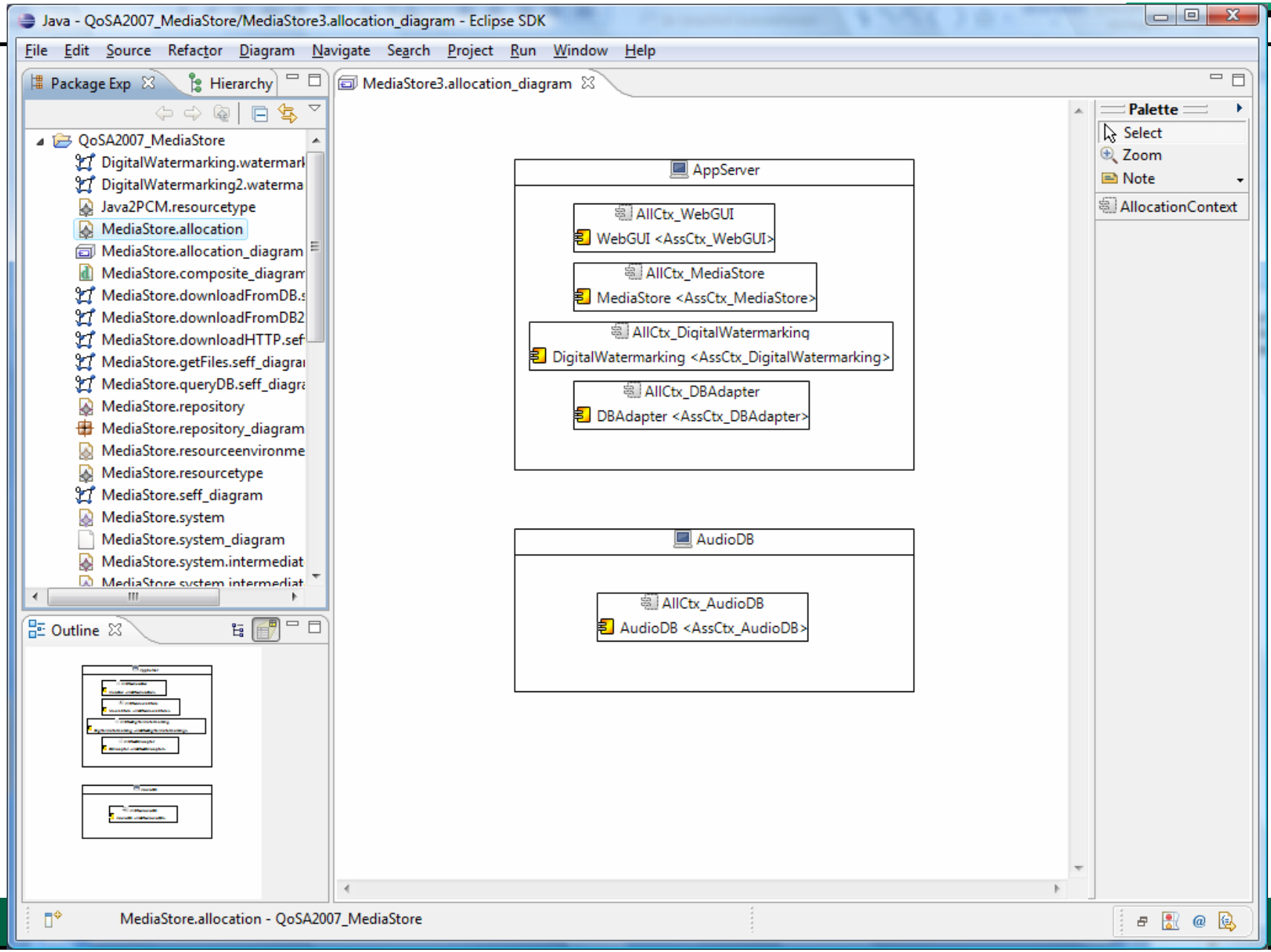
# Assembly Model



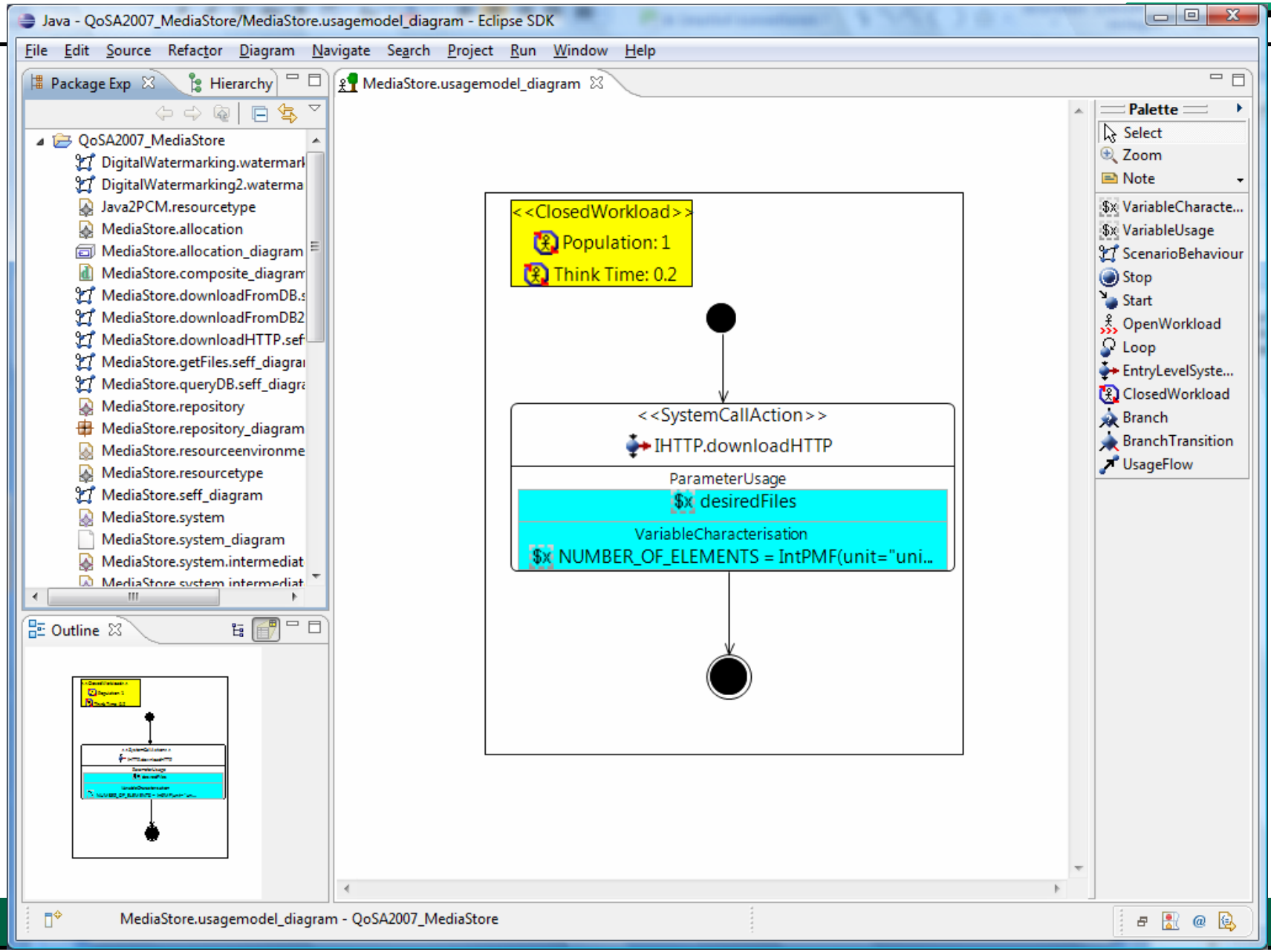
# Component Behaviour Specification



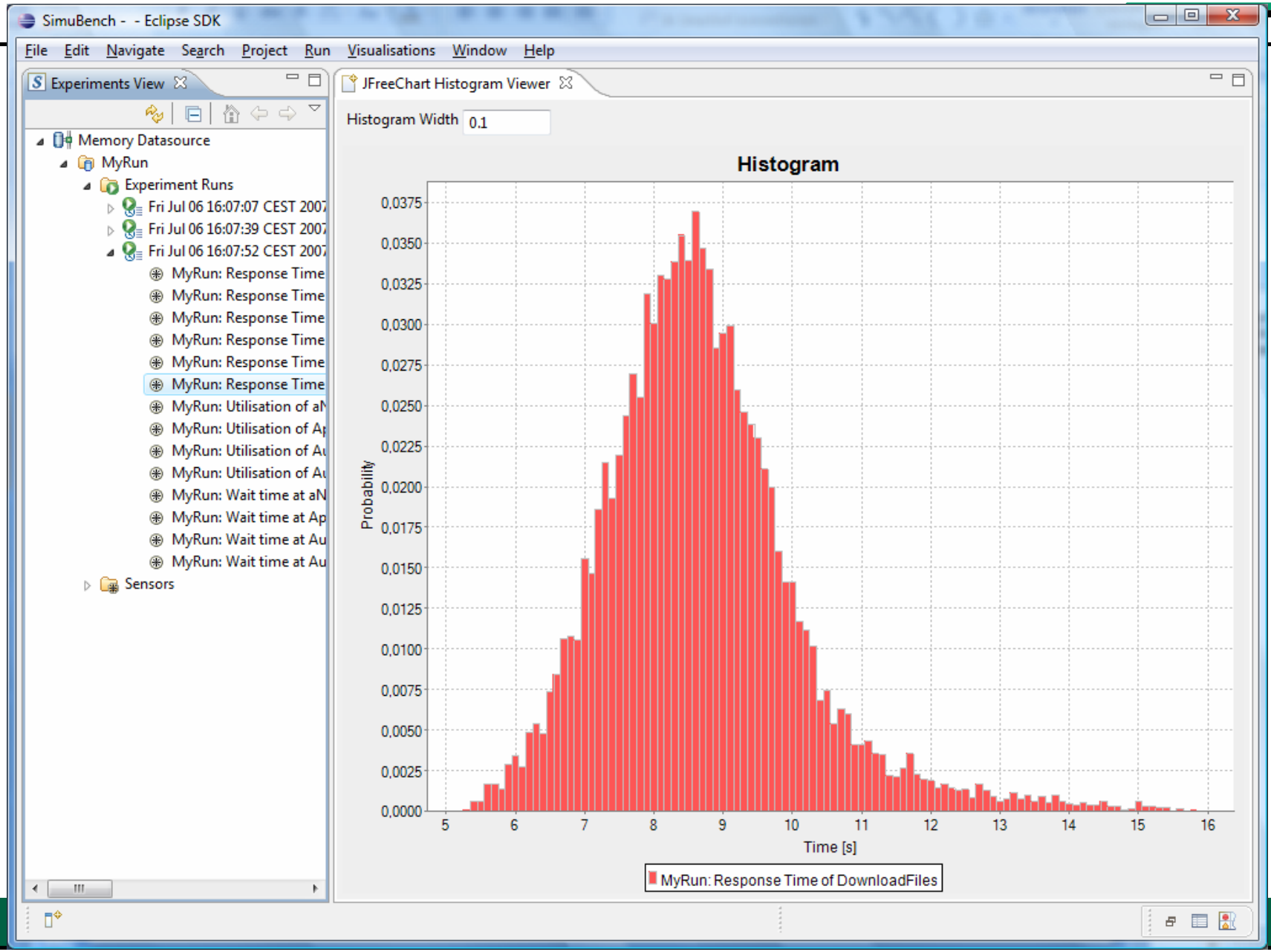
# Allocation Model



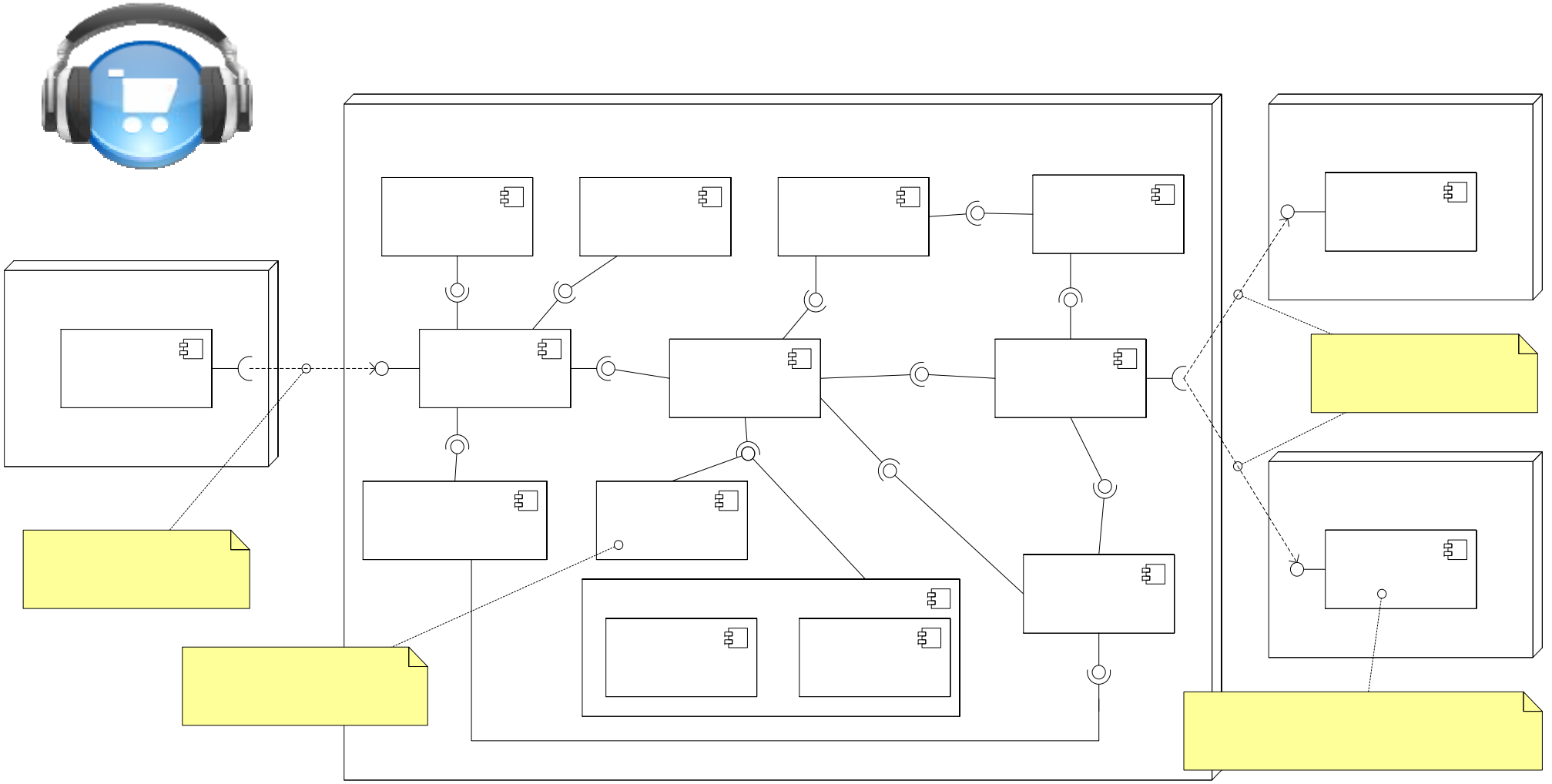
# Usage Model


Java - QoSA2007_MediaStore/MediaStore.usagemodel_diagram - Eclipse SDK
File Edit Source Refactor Diagram Navigate Search Project Run Window Help
Package Exp
Hierarchy
QoSA2007_MediaStore
DigitalWatermarking.watermark
DigitalWatermarking2.waterma
Java2PCM.resourcetype
MediaStore.allocation
MediaStore.allocation_diagram
MediaStore.composite_diagram
MediaStore.downloadFromDB.s
MediaStore.downloadFromDB2
MediaStore.downloadHTTP.sef
MediaStore.getFiles.seff_diagra
MediaStore.queryDB.seff_diagra
MediaStore.repository
MediaStore.repository_diagram
MediaStore.resourceenvironme
MediaStore.resourcetype
MediaStore.seff_diagram
MediaStore.system
MediaStore.system_diagram
MediaStore.system.intermediat
Outline
MediaStore.usagemodel_diagram
<<ClosedWorkload>>
Population: 1
Think Time: 0.2
<<SystemCallAction>>
IHTTP.downloadHTTP
ParameterUsage
$x desiredFiles
VariableCharacterisation
$x NUMBER_OF_ELEMENTS = IntPMF(unit="uni...
Palette
Select
Zoom
Note
VariableCharacte...
VariableUsage
ScenarioBehaviour
Stop
Start
OpenWorkload
Loop
EntryLevelSyste...
ClosedWorkload
Branch
BranchTransition
UsageFlow
MediaStore.usagemodel_diagram - QoSA2007_MediaStore

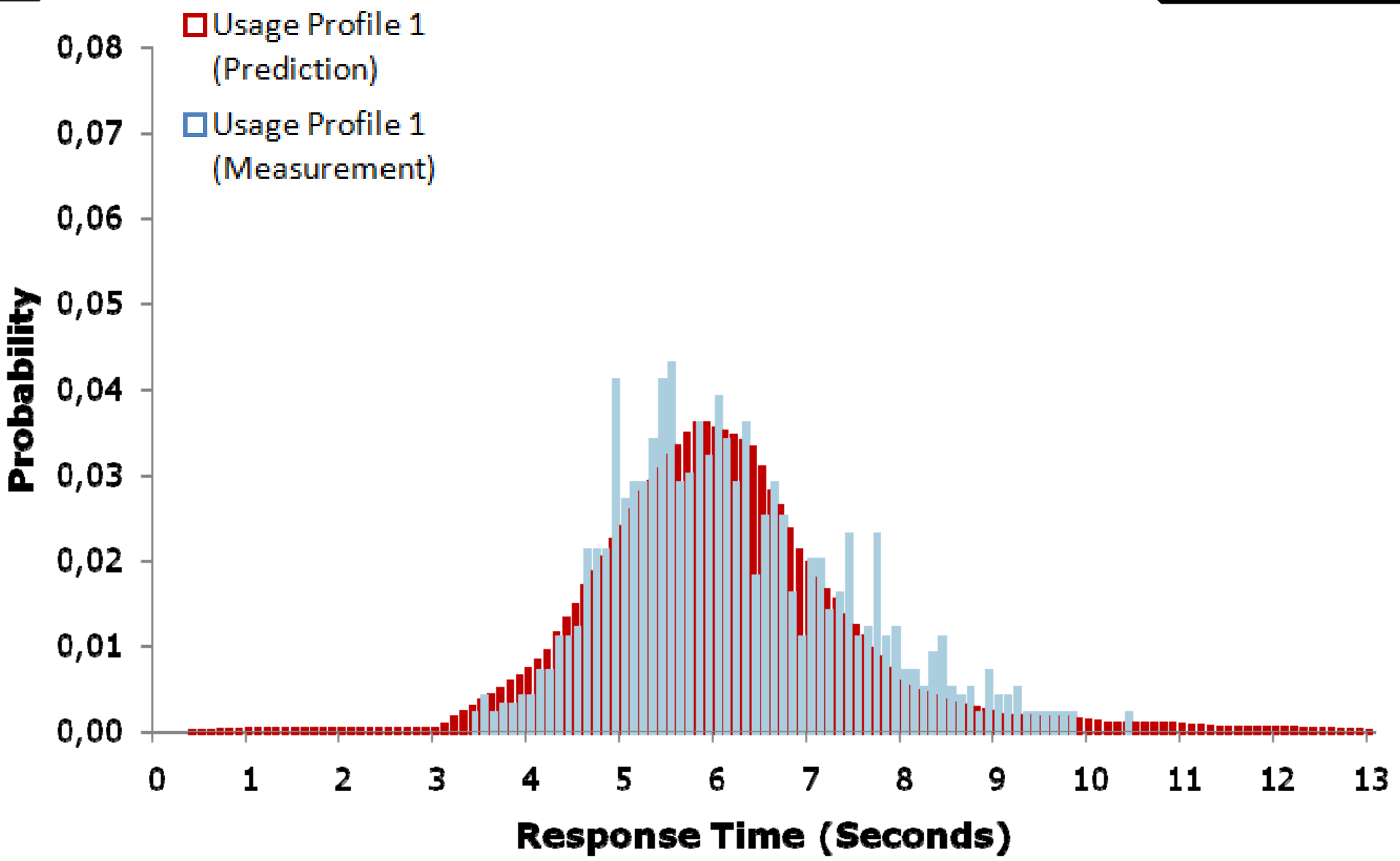
# Result: Distribution of Response Time

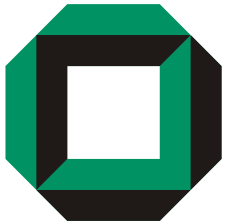


# MediaStore - Architecture

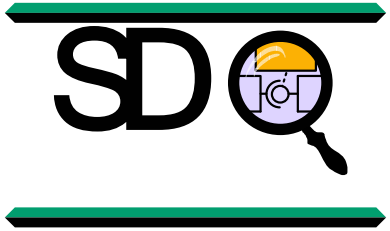


# Results


Usage Profile 1 (Prediction)
Usage Profile 1 (Measurement)
Probability
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0,00
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Response Time (Seconds)



# Conclusions



- Prediction and Understanding of the Consequences of Design Decsions is THE central characteristic of an engineering discipline.
- Components and MDD lower the degrees of freedom in implementation
- Creativity is on design-model level
- Quality-driven Design requires prediction models
  - automatically generated from design models
- Definition of design and prediction models follows the scientific process of the natural sciences.
  - No proofs possible, but empirical validations necessary

Software Engineering becomes "architecture-centric".
Code-centration is as meaningful as
"**melting-tin-centration**" of an Electrical Engineer

Klaus Krogmann:
Modell-Reconstruction

Johannes Stammel:
Performanz-Maintain-
ability-Trade-off

Steffen Becker:
Model transformations
and meta modelling

Jens Happe:
Parallelism and analysis model

Michael Kuperberg:
Influence of deployment

Henning Groenda:
Architecture evaluation

Chris Rathfelder:
Architecture evaluation

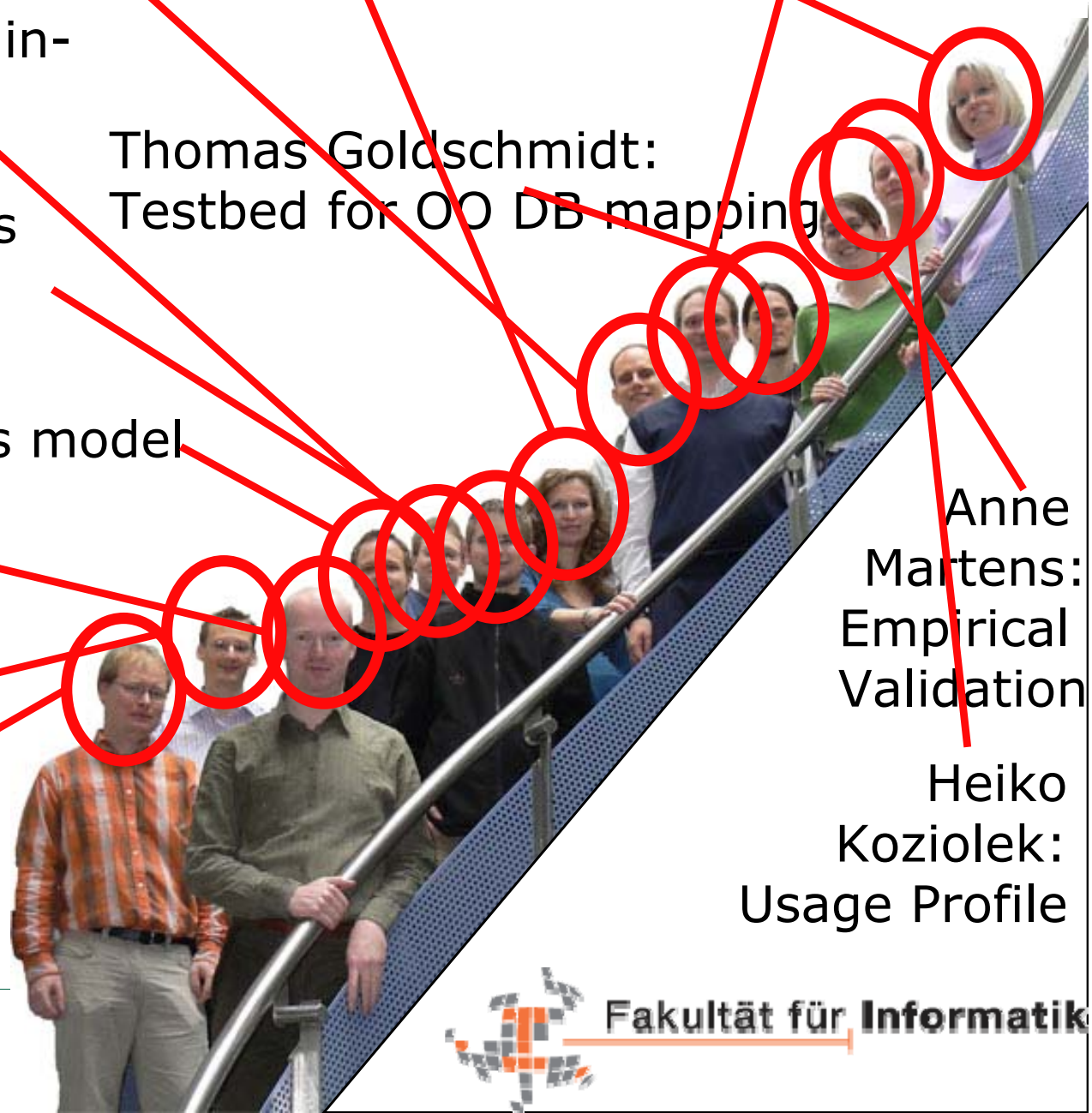
Elena Kienhöfer /
Elke Sauer:
Sekretaries

A Visitor

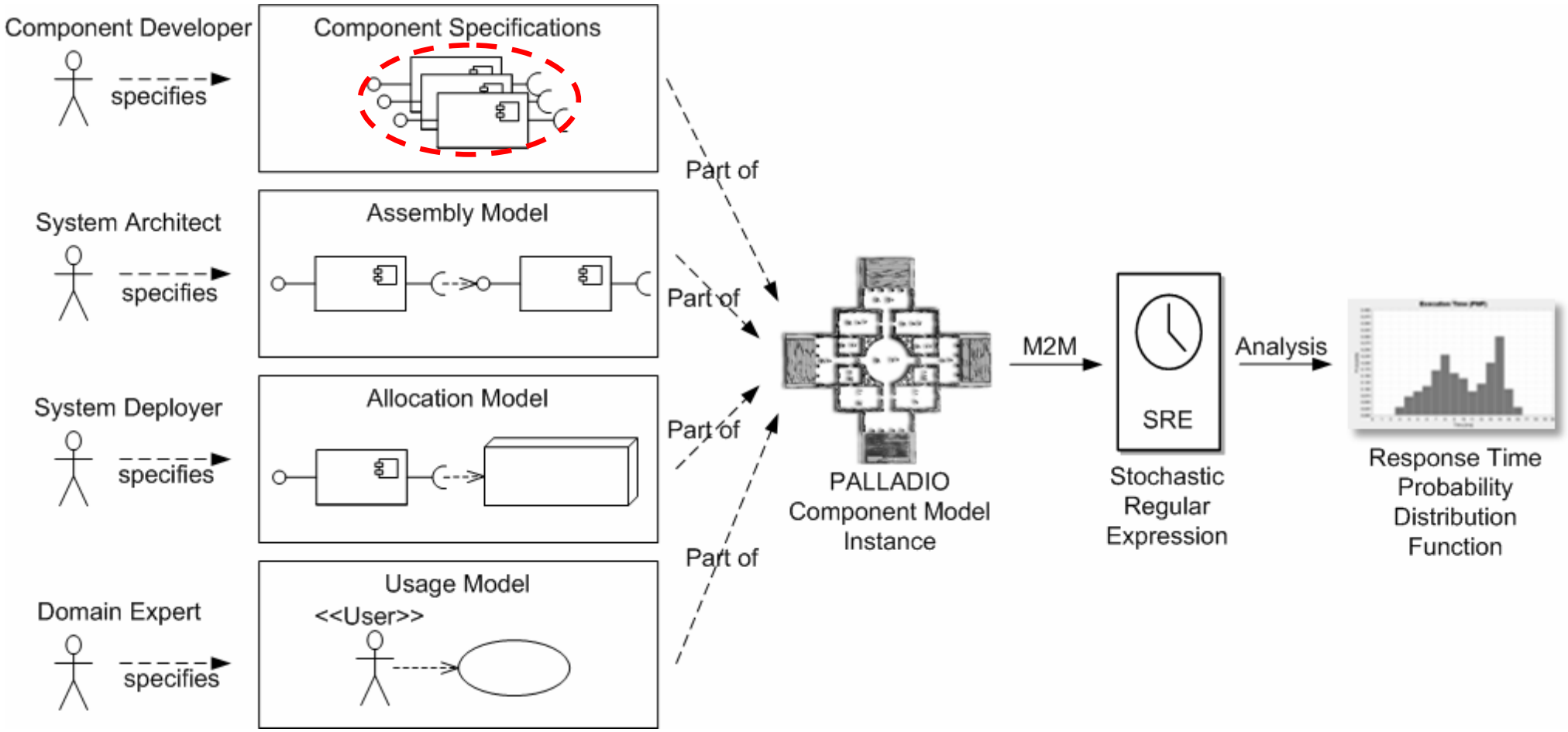
Thomas Goldschmidt:
Testbed for OO DB mapping

Anne
Martens:
Empirical
Validation

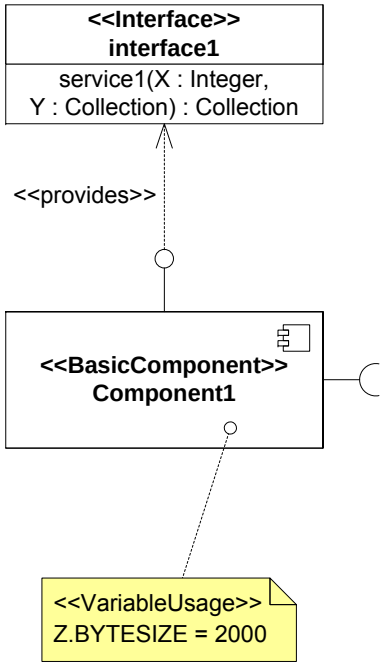
Heiko
Koziolek:
Usage Profile



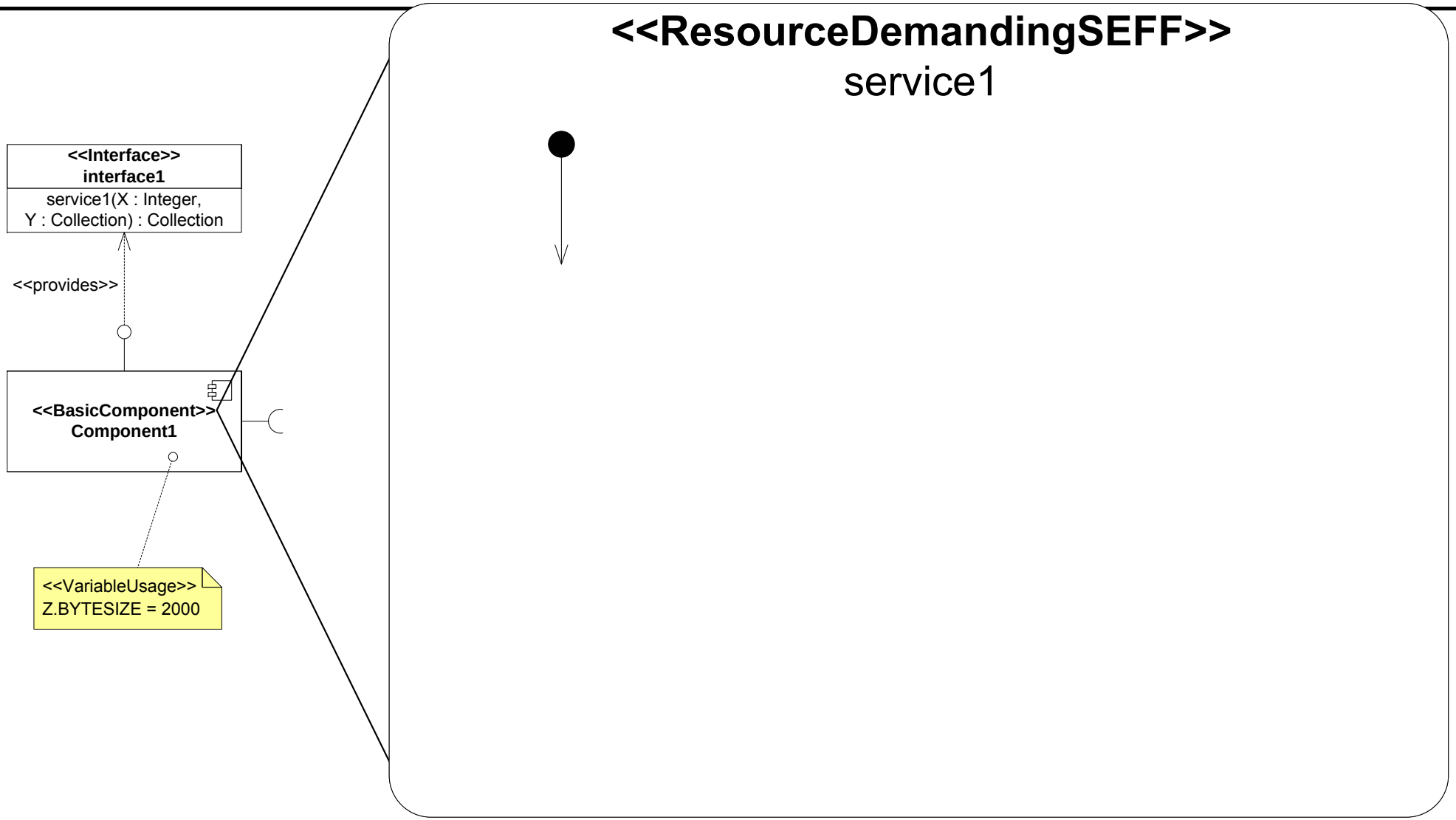
# Palladio Component Model



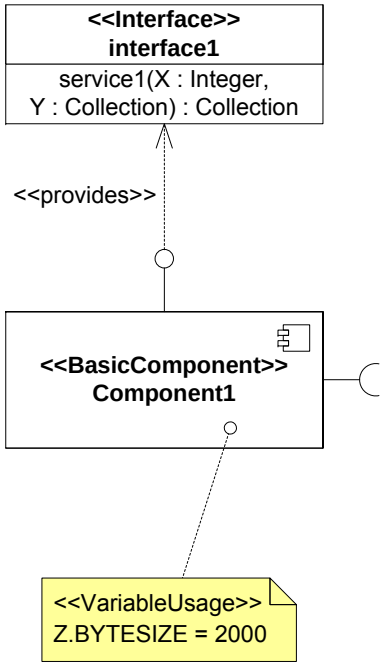
# Service Effect Specification

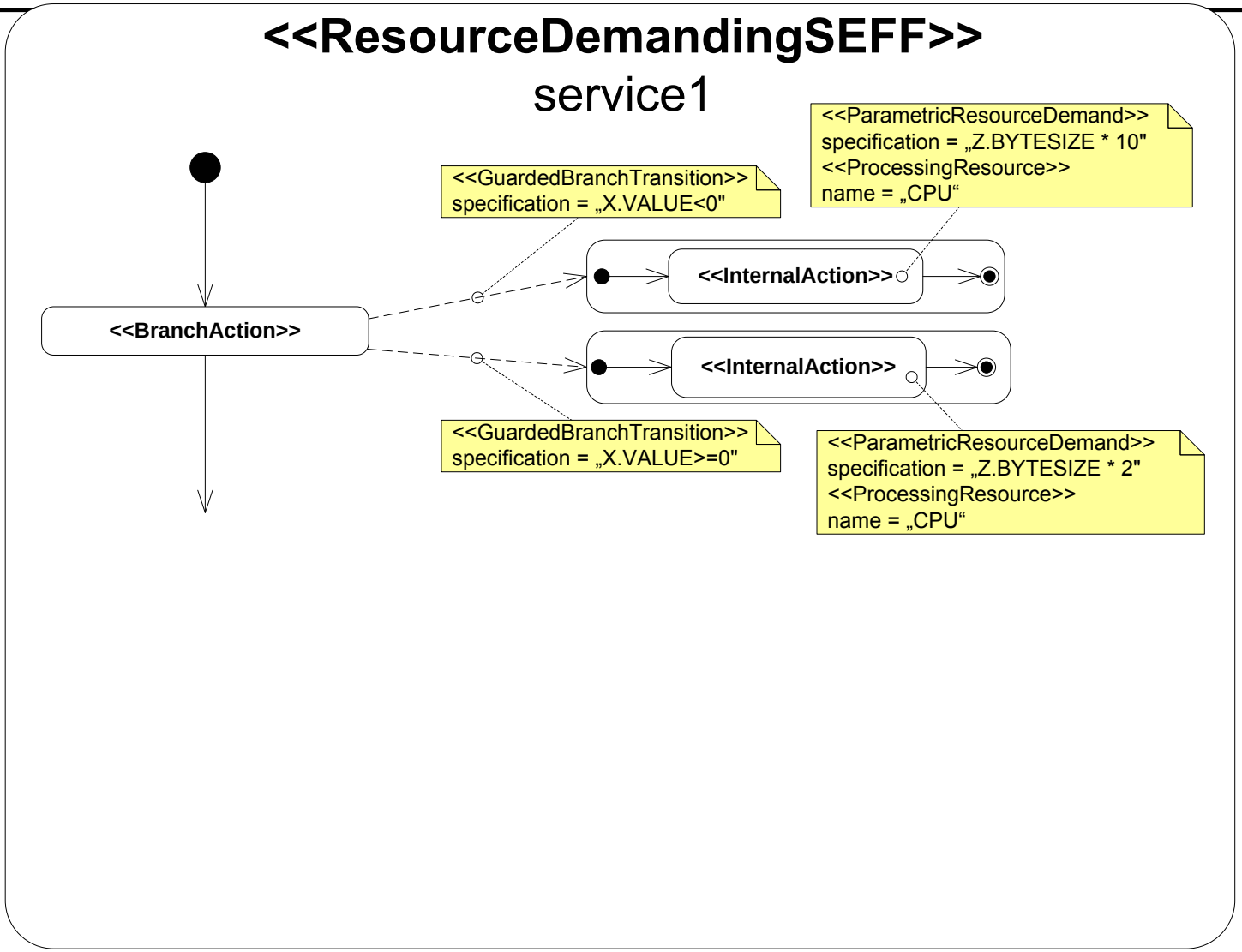


# Service Effect Specification

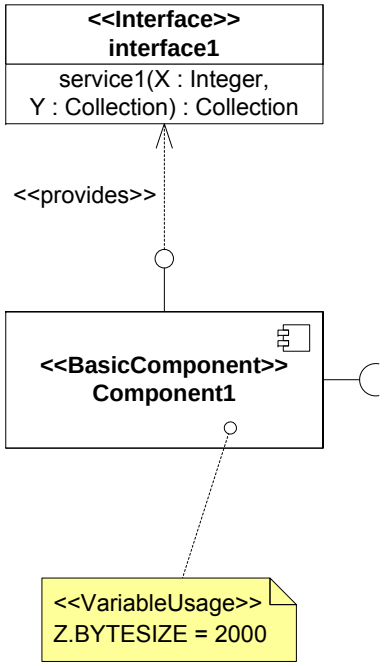


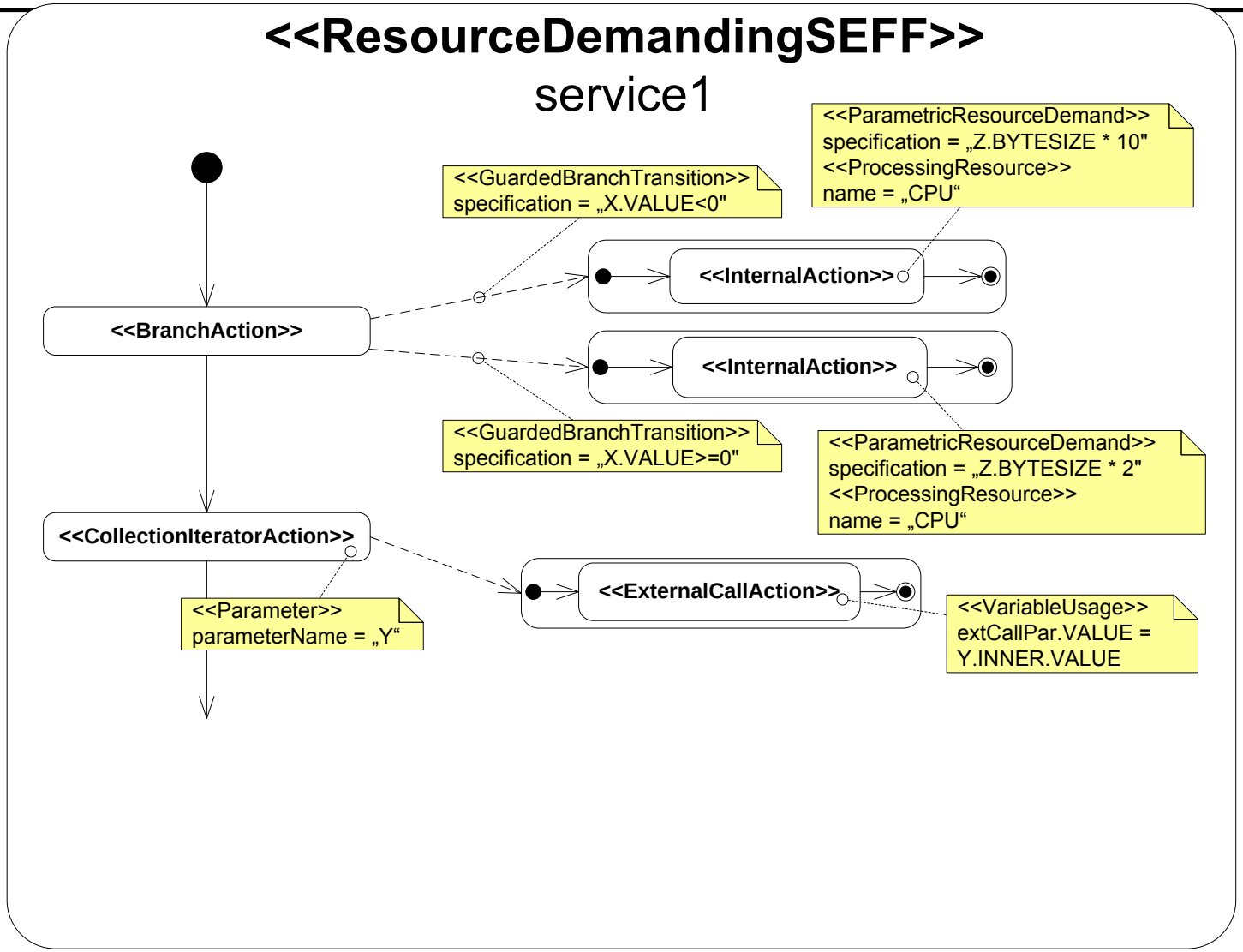
# Service Effect Specification



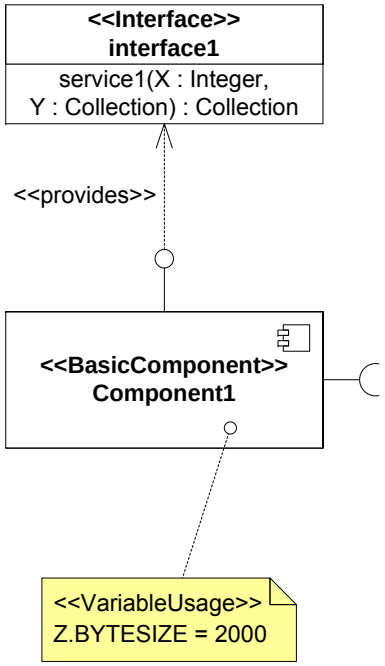



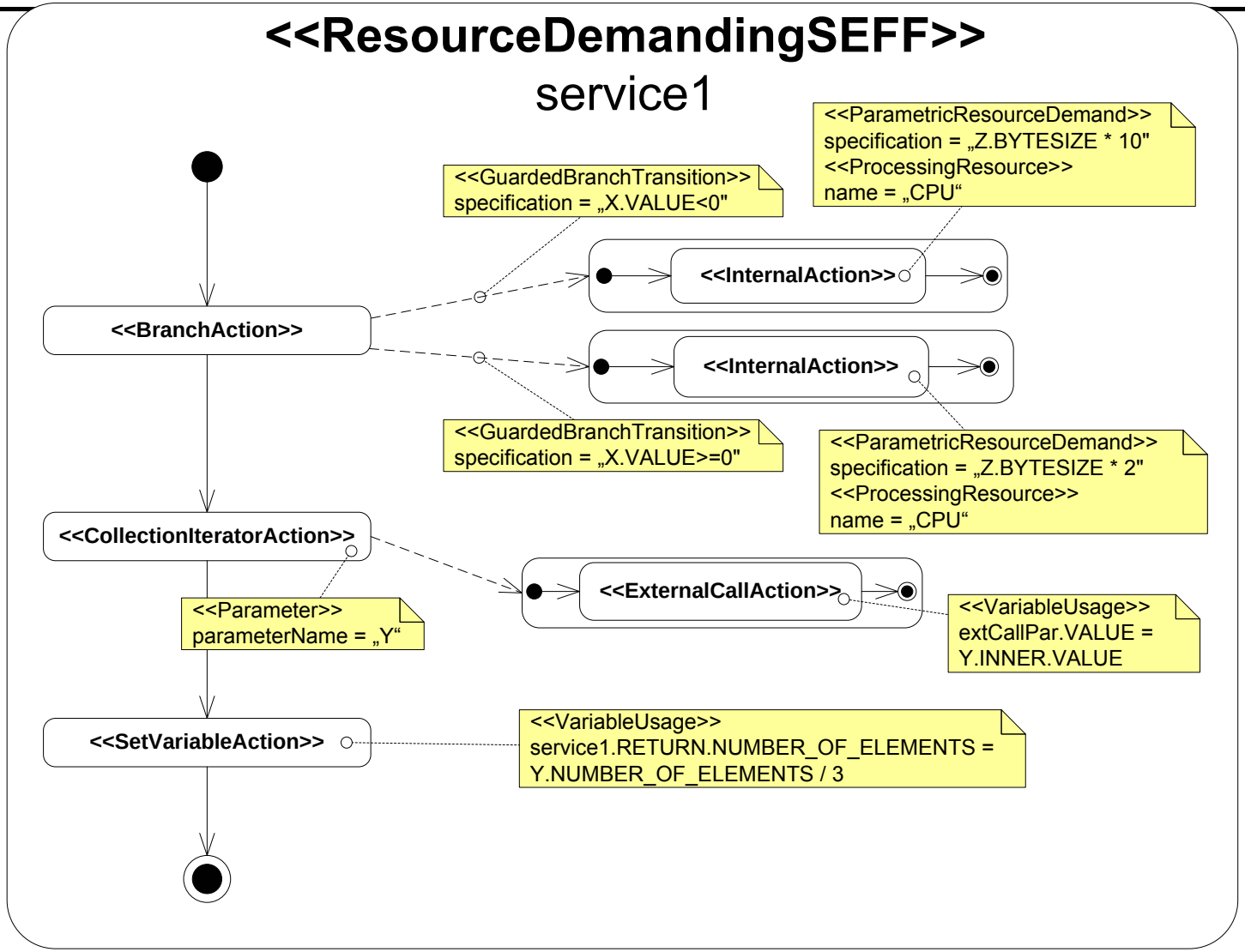
# Service Effect Specification



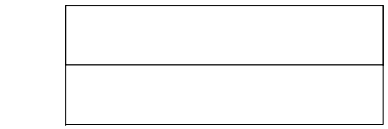



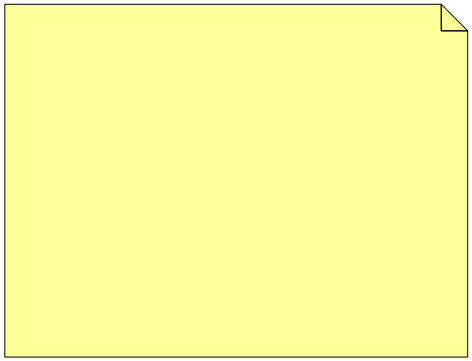
# Service Effect Specification

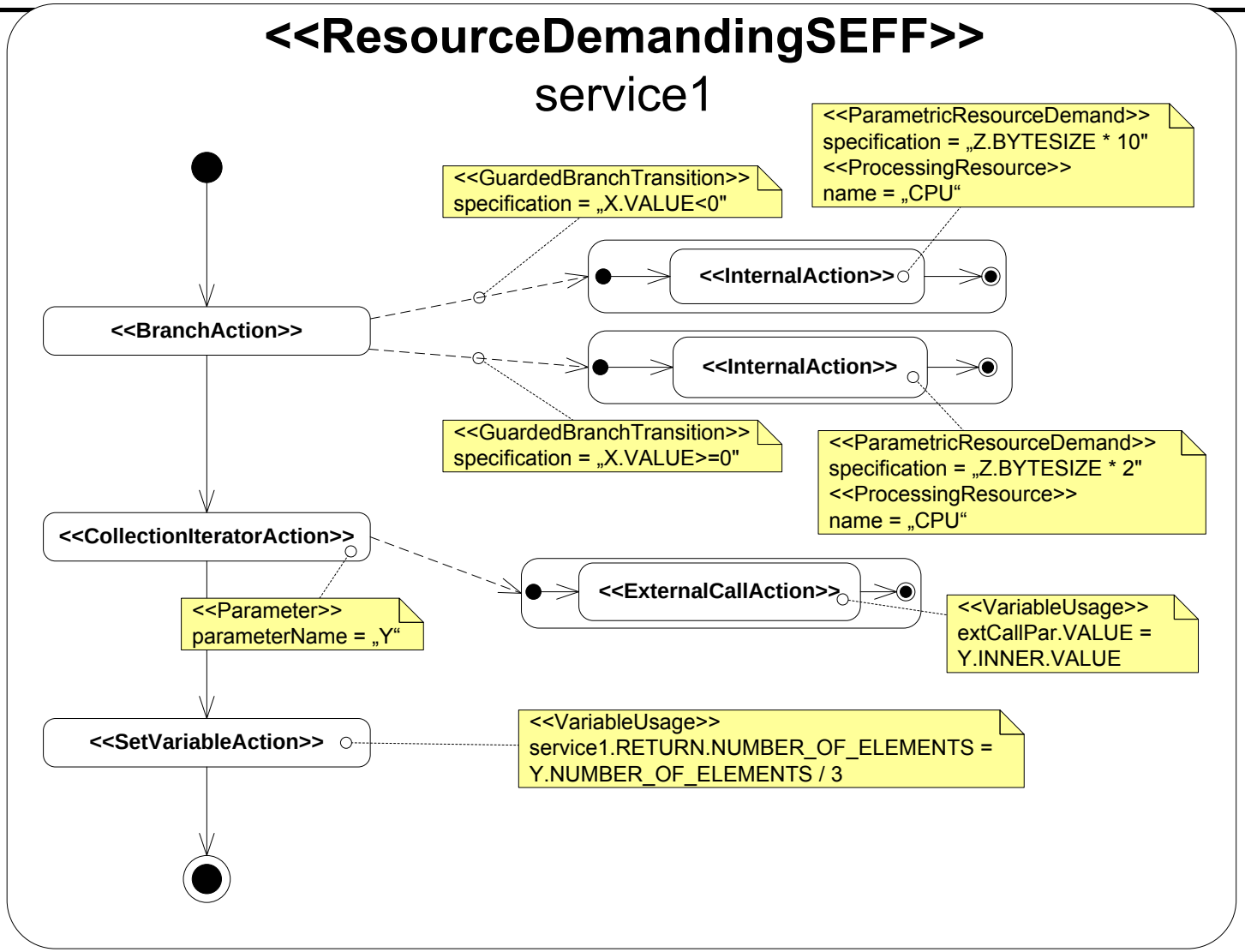


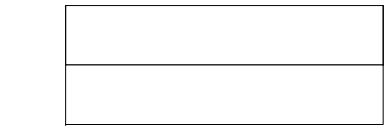
# Service Effect Specification

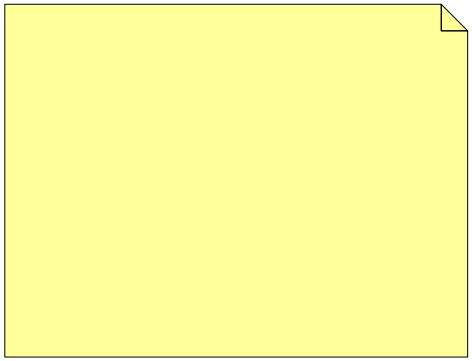


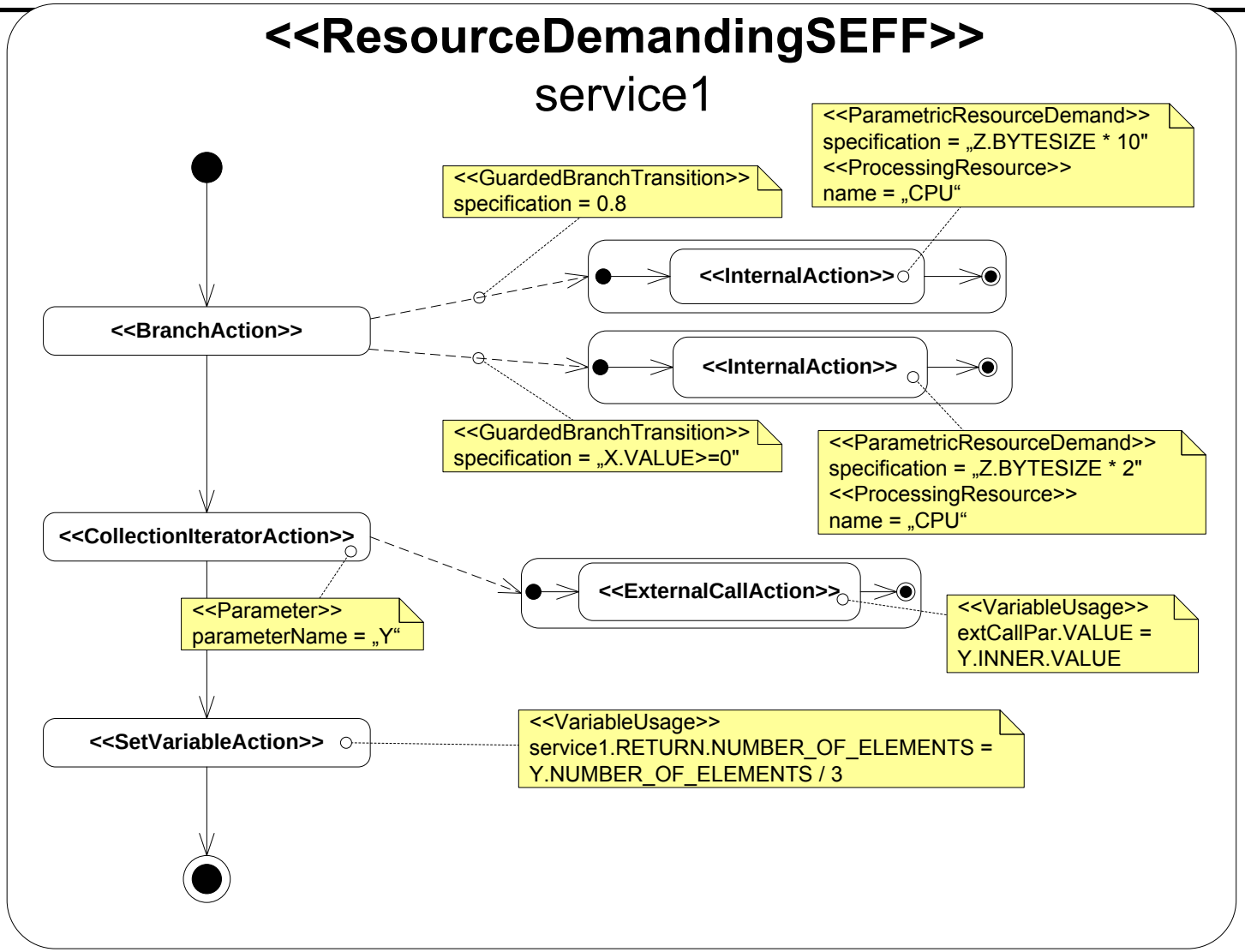




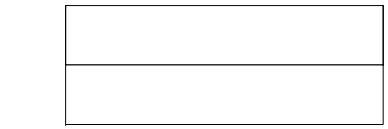
# Service Effect Specification

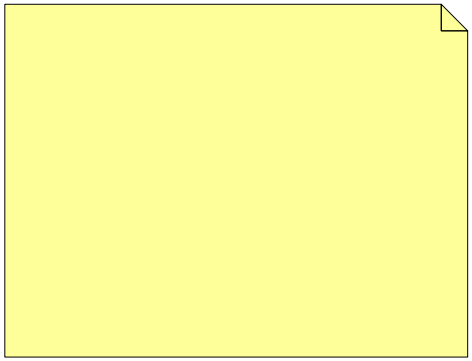


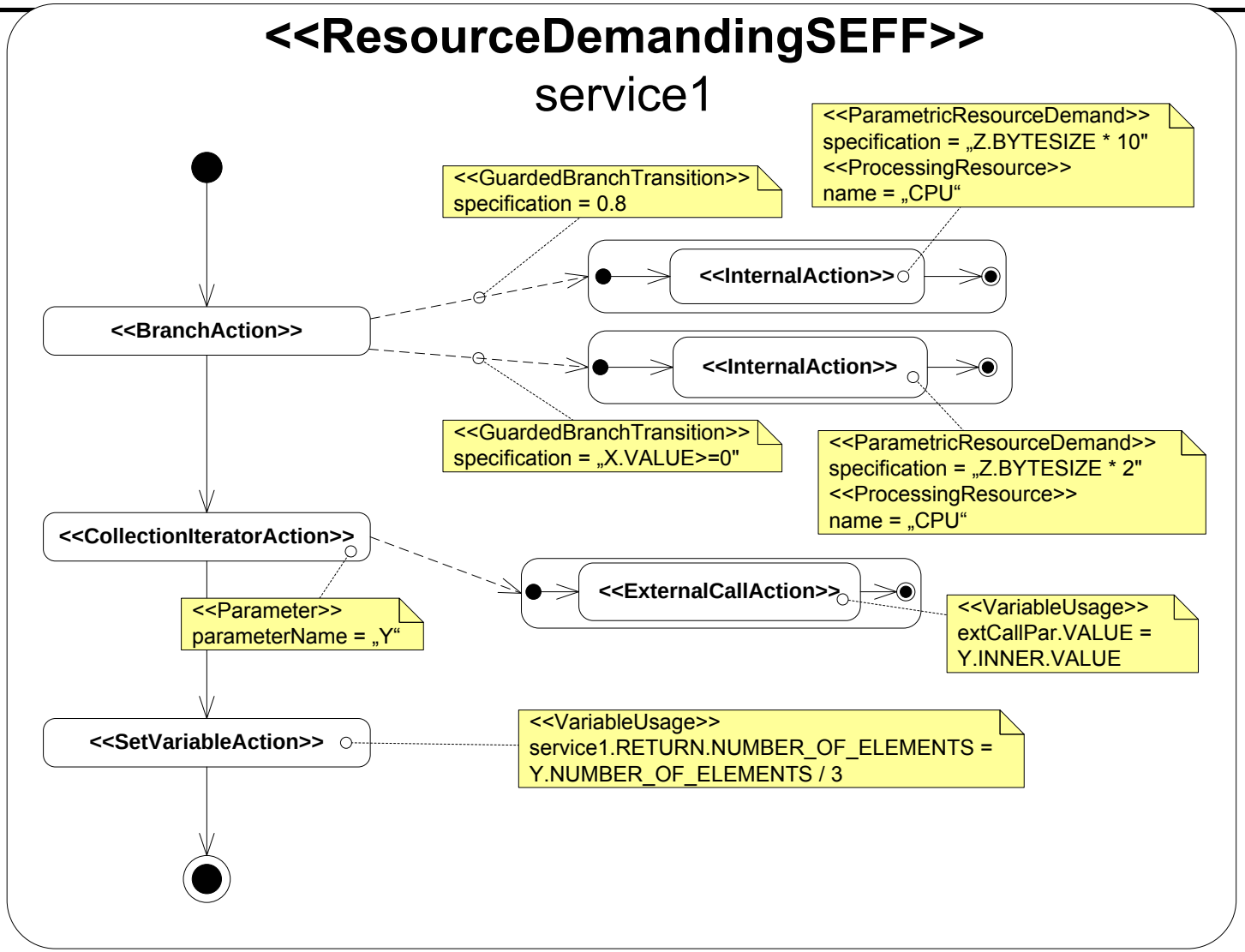




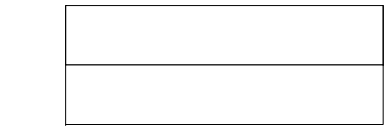
# Service Effect Specification

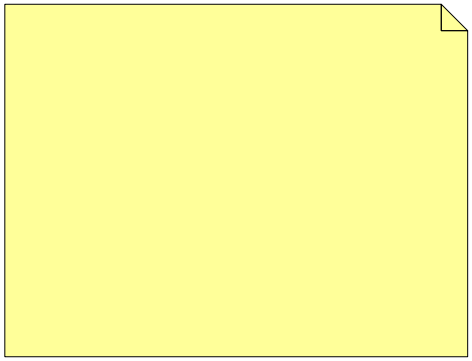


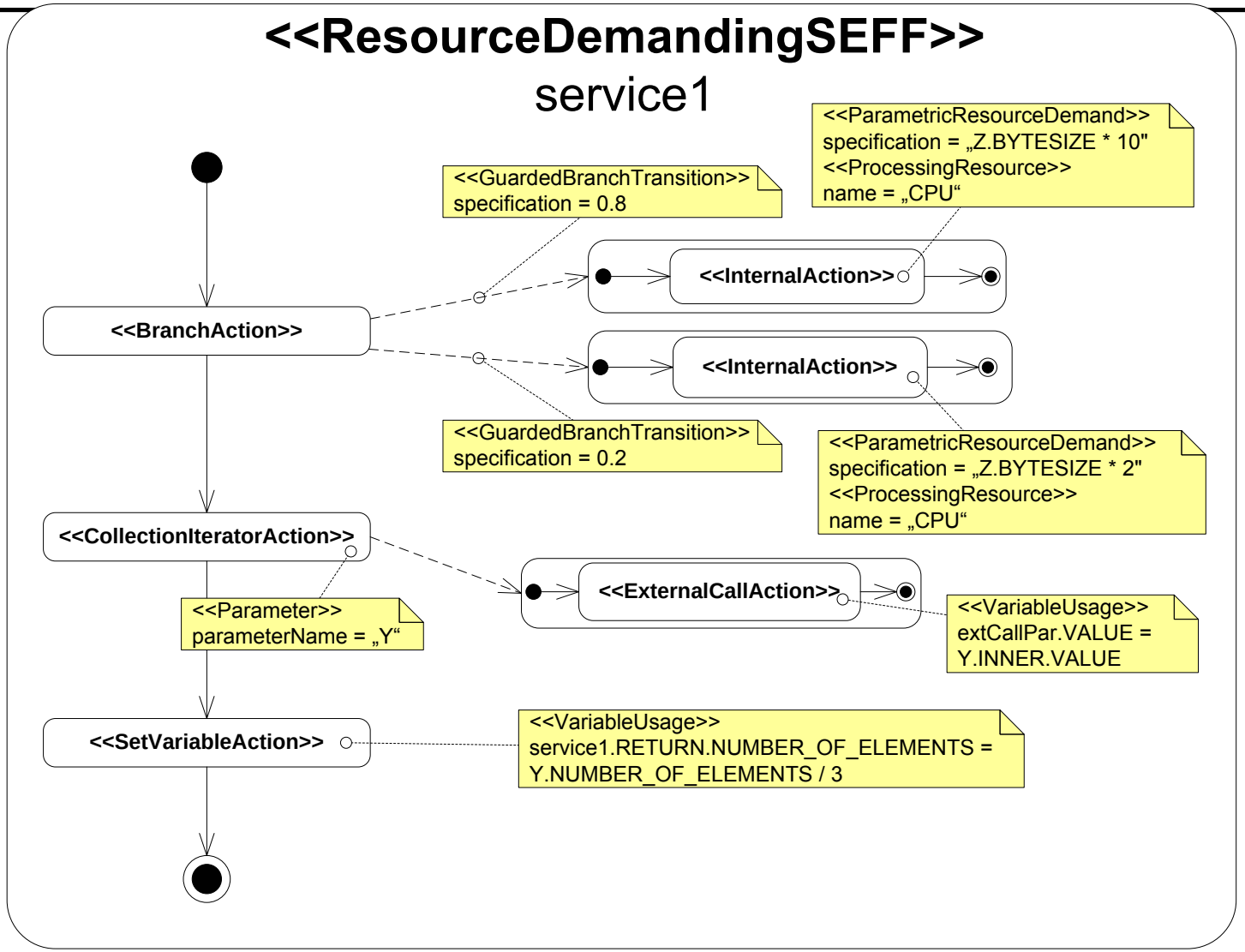




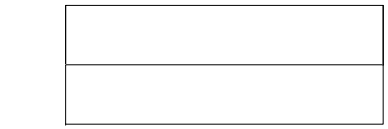
# Service Effect Specification

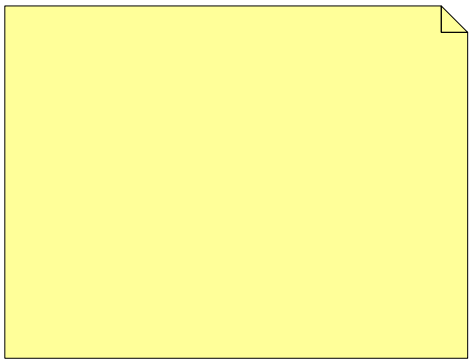


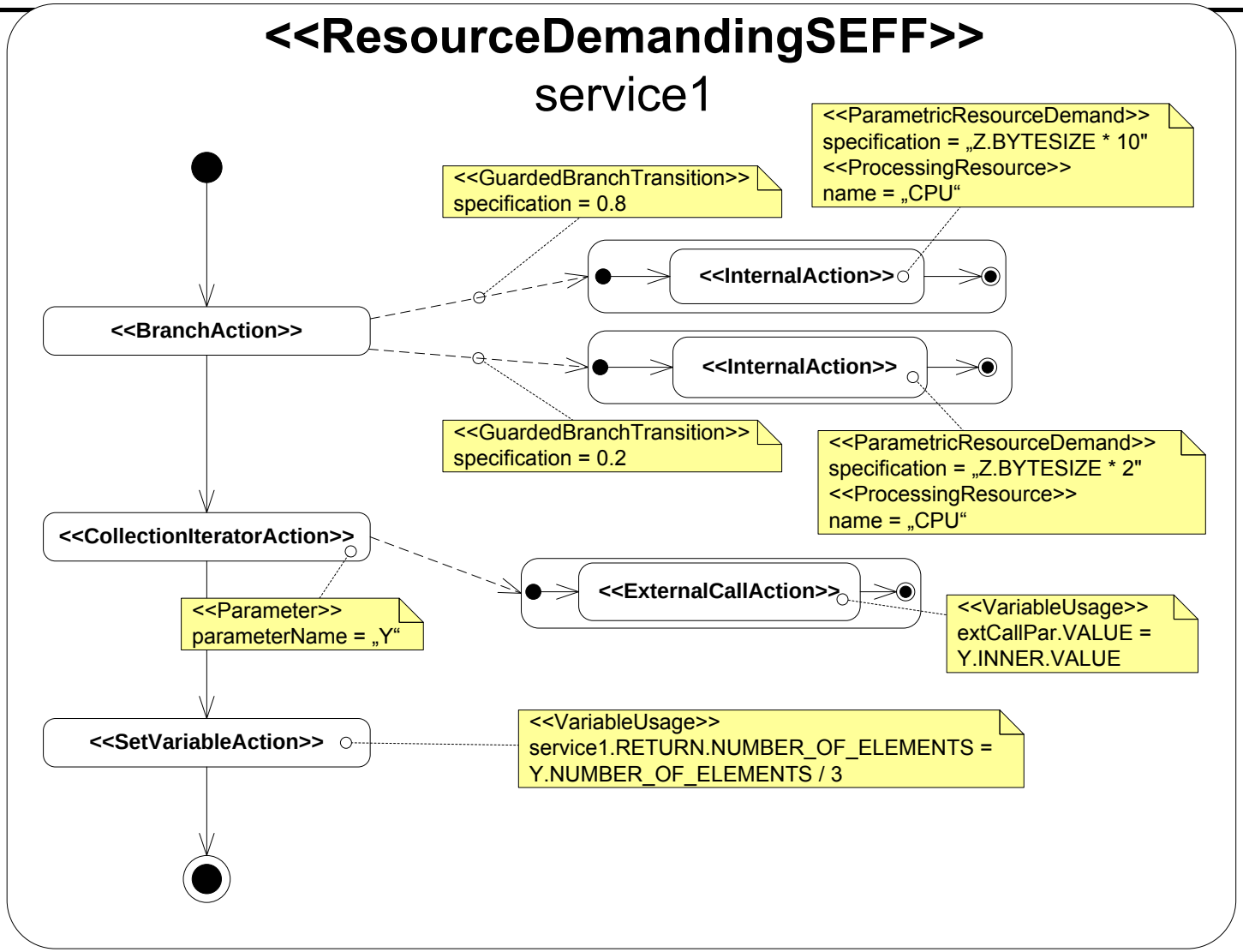




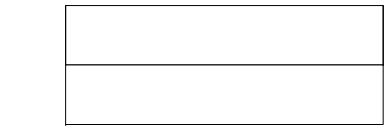
# Service Effect Specification

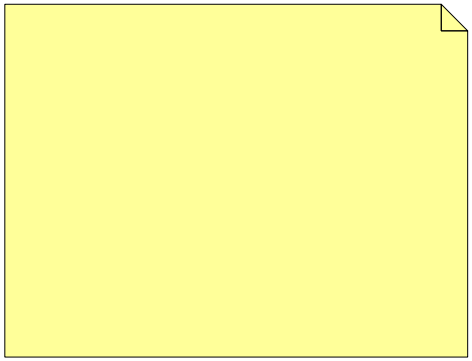


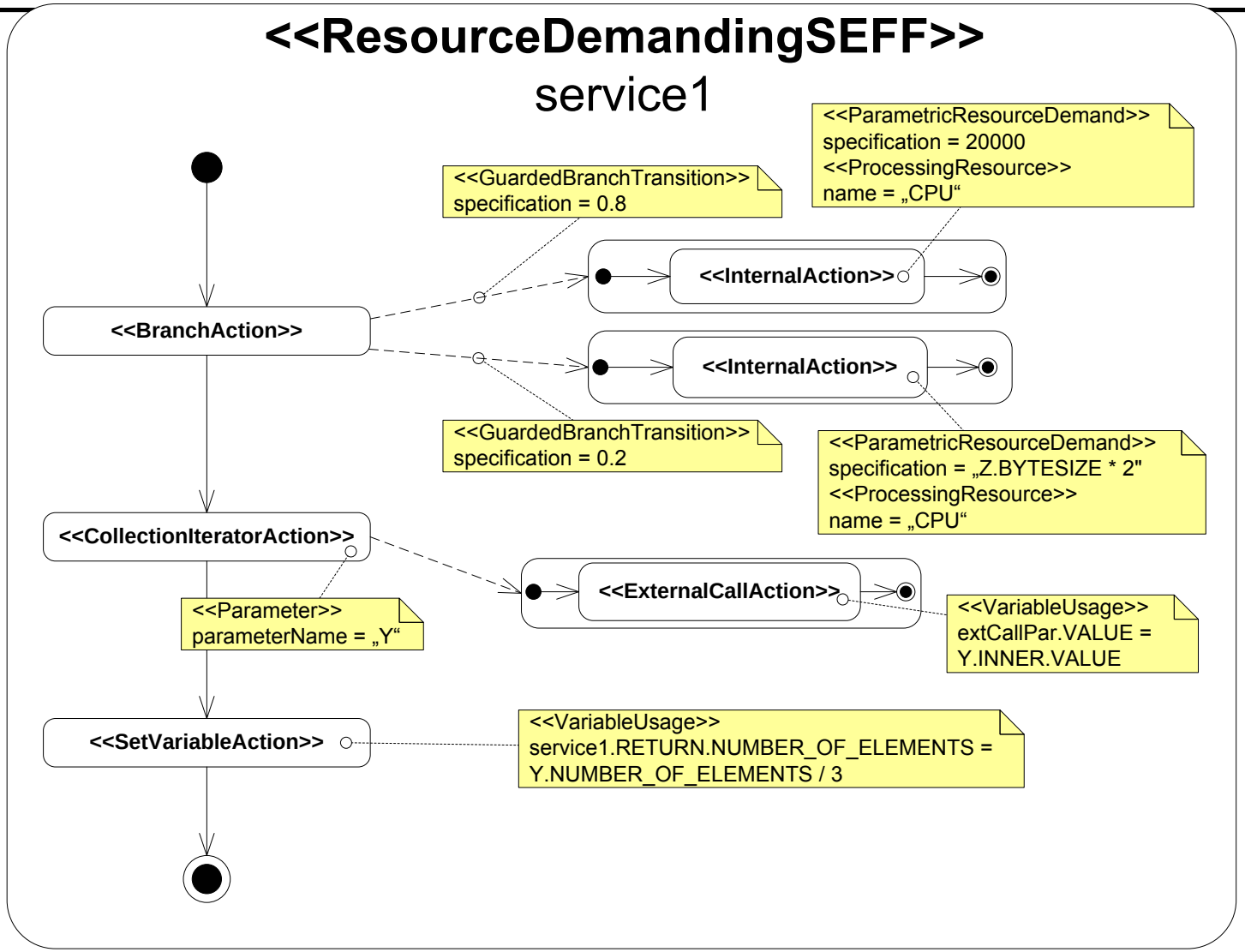




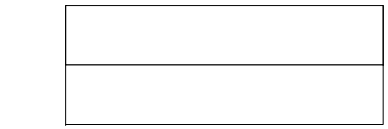
# Service Effect Specification

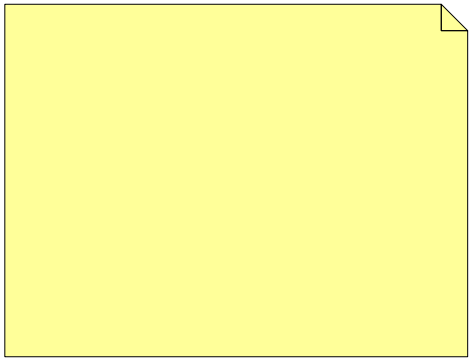


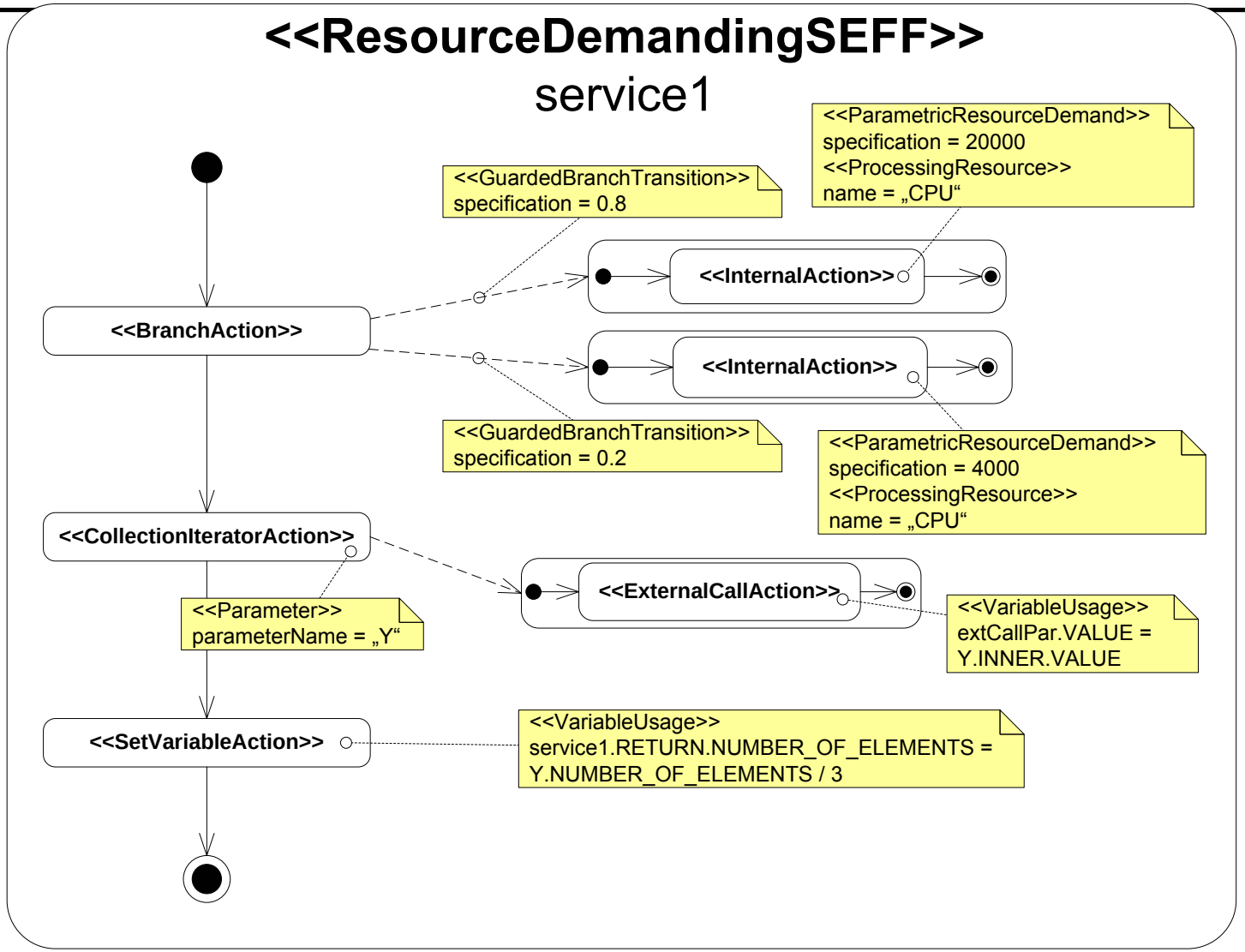




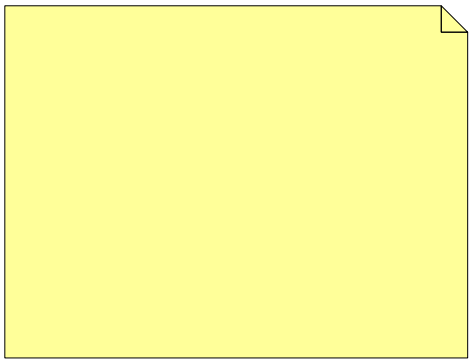
# Service Effect Specification

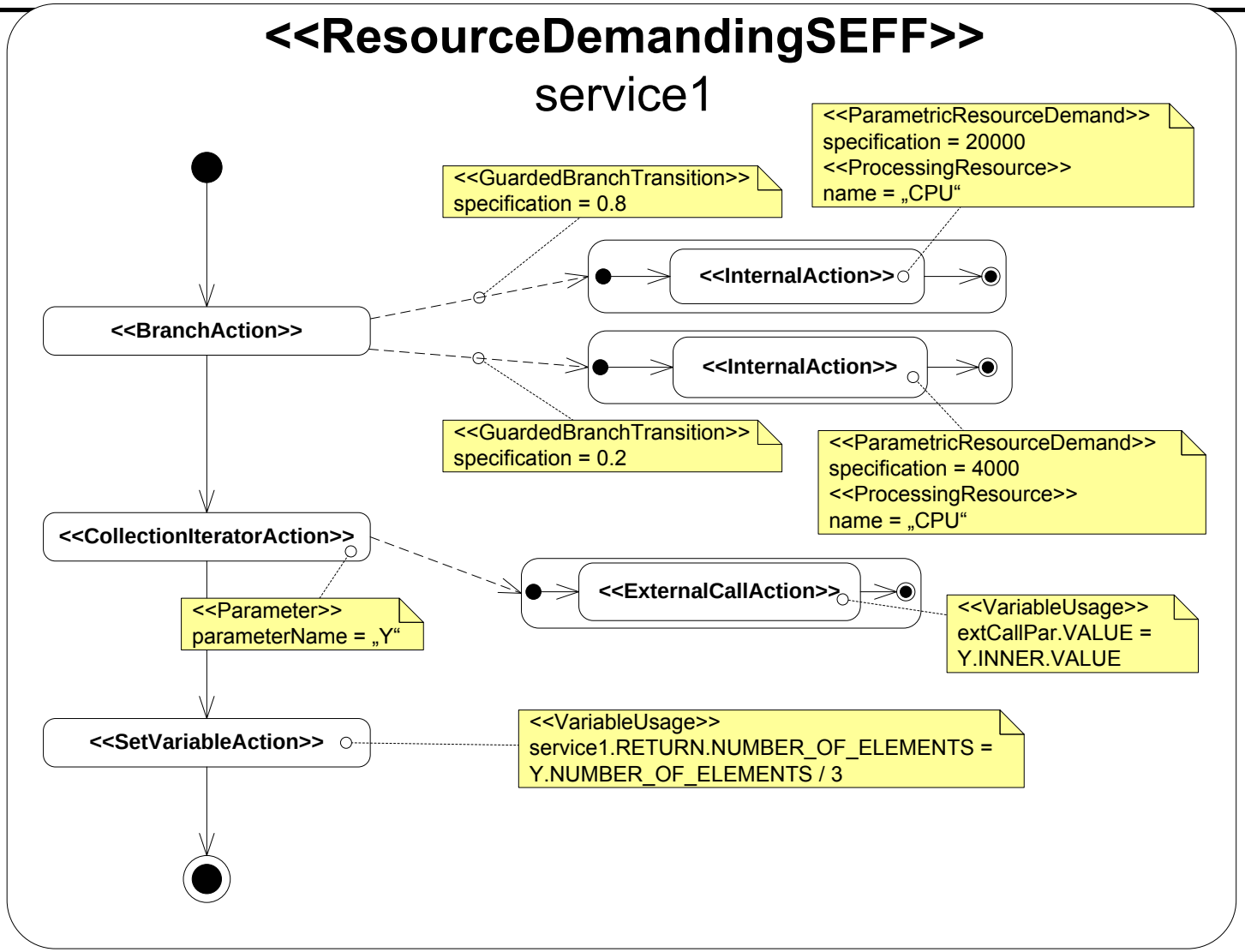




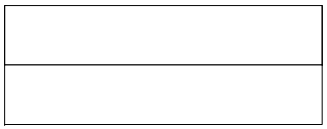


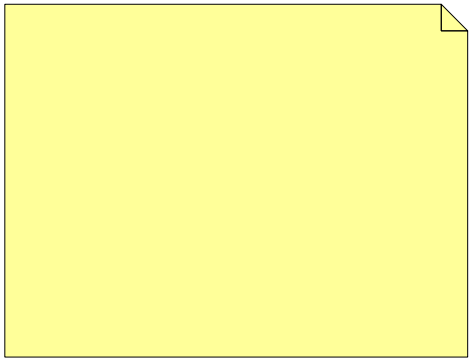
# Service Effect Specification

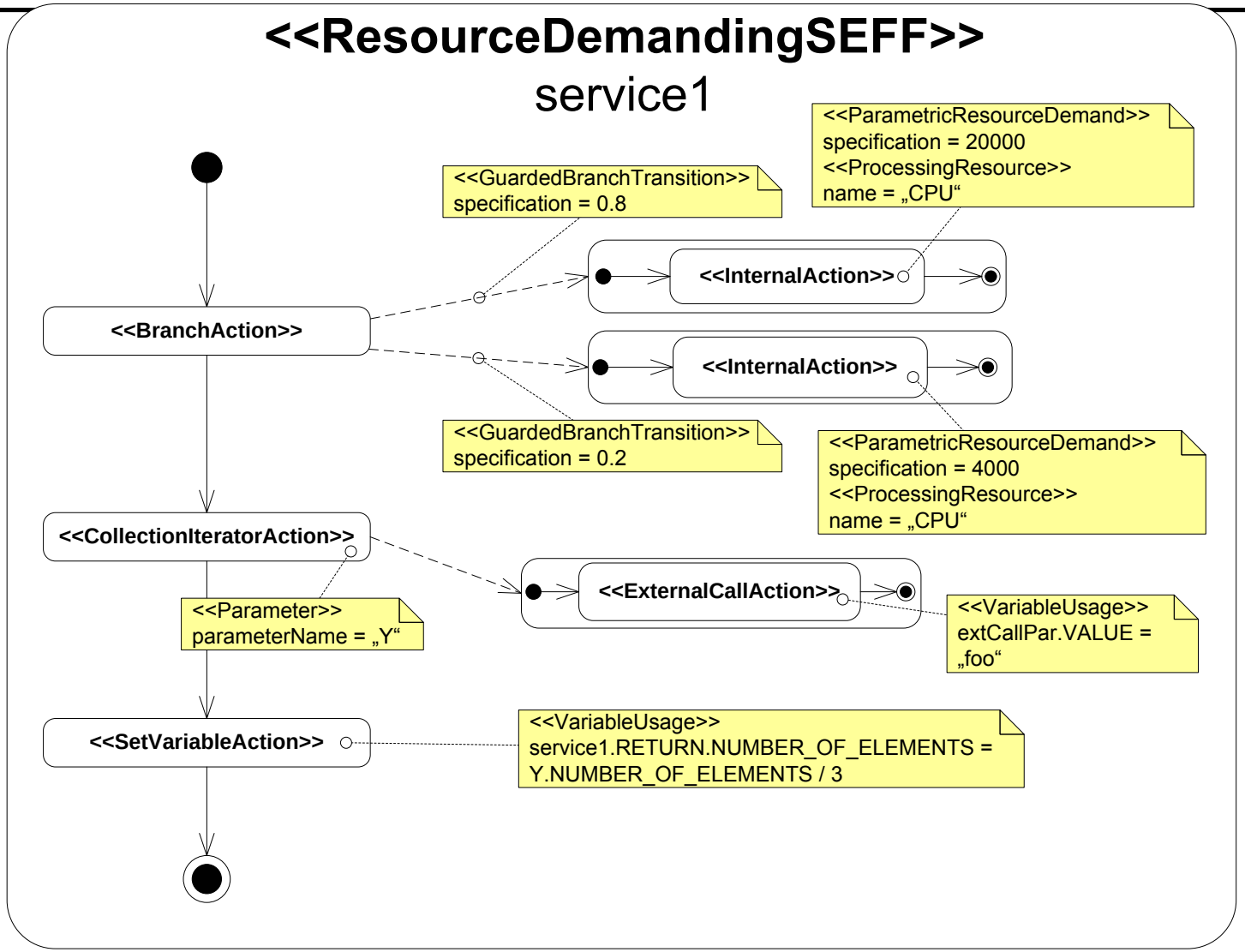




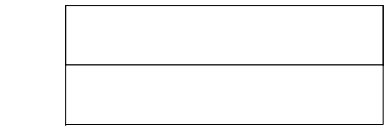
# Service Effect Specification

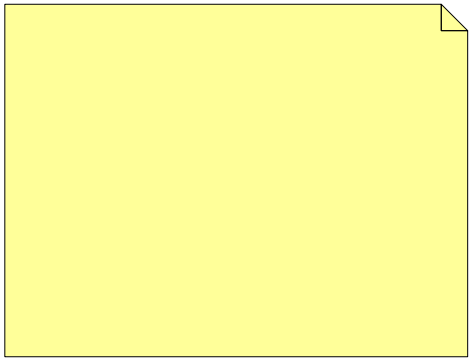


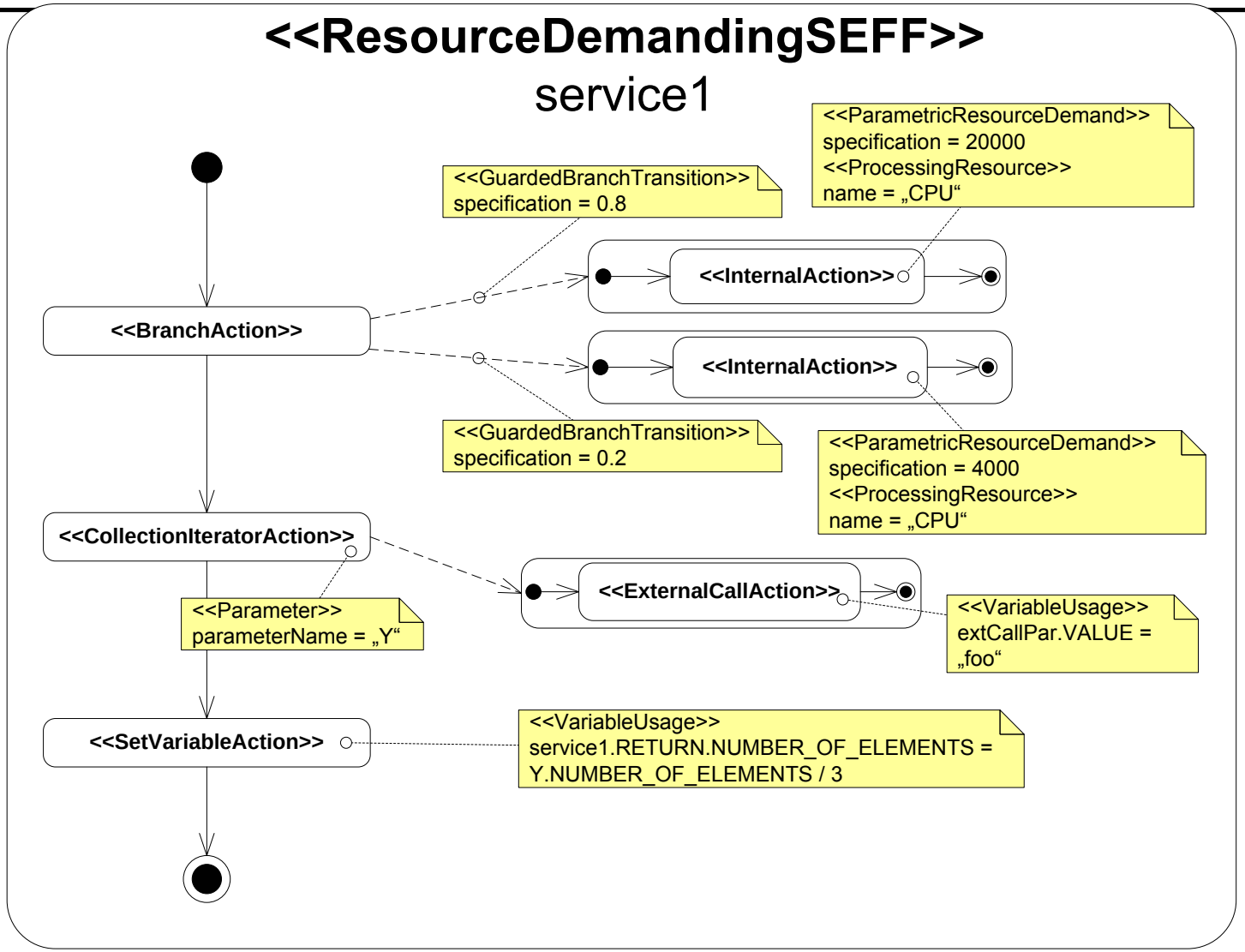




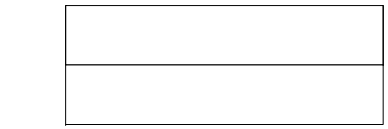
# Service Effect Specification

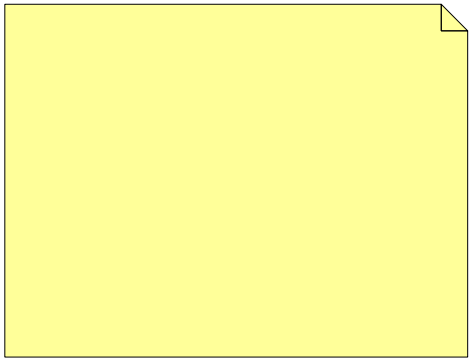


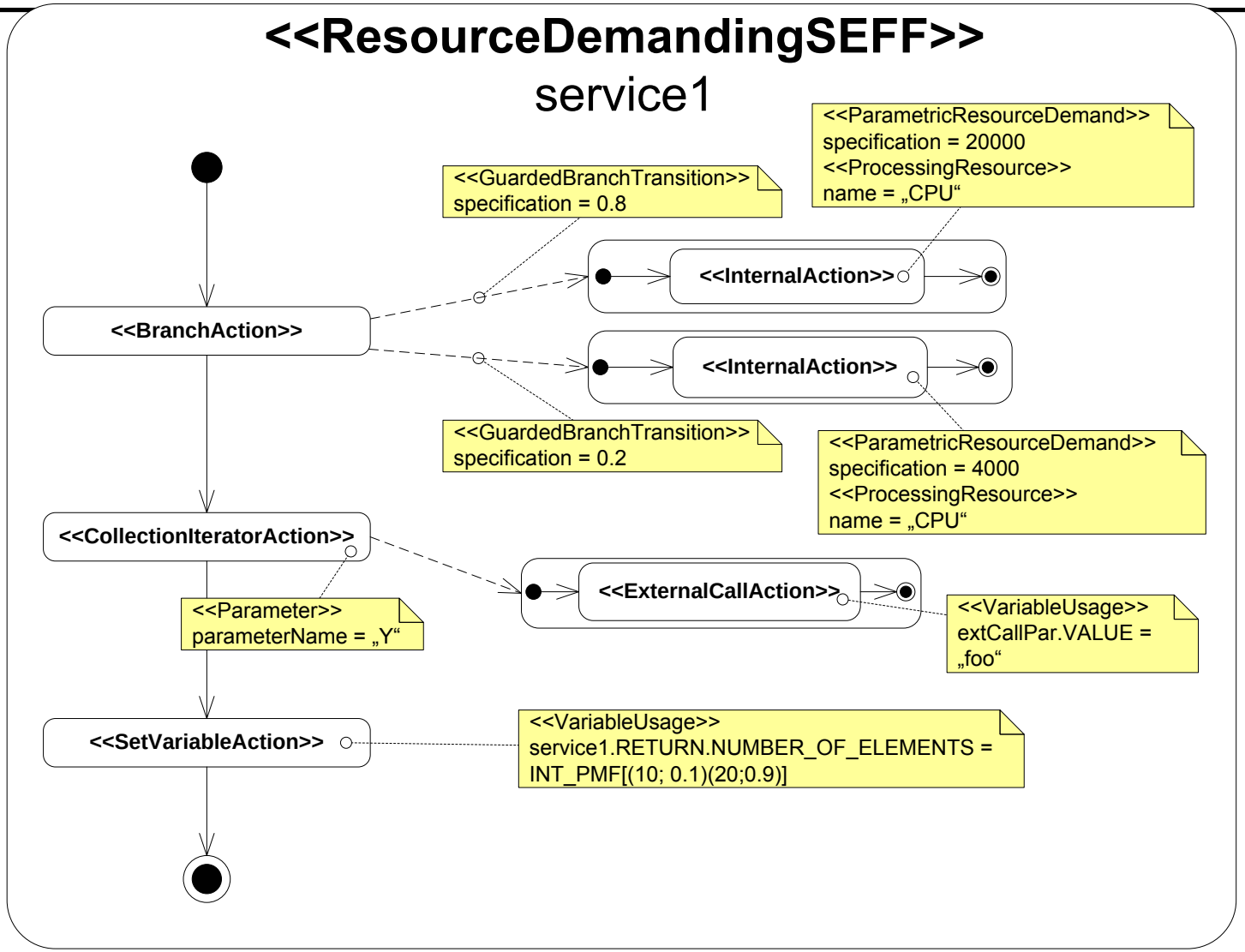




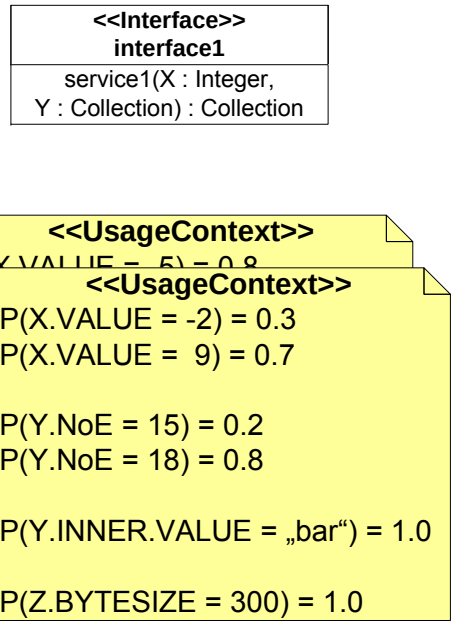
# Service Effect Specification

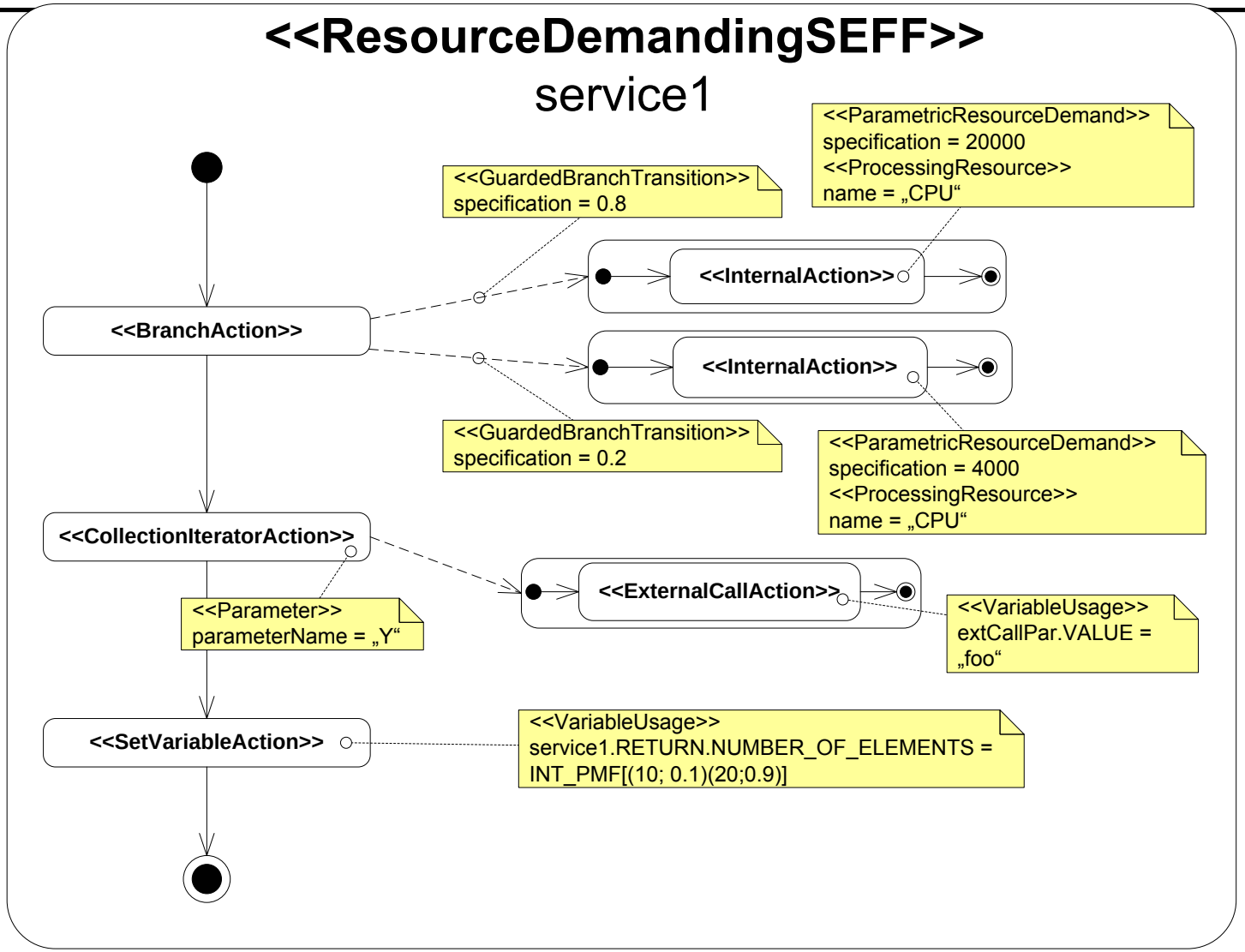




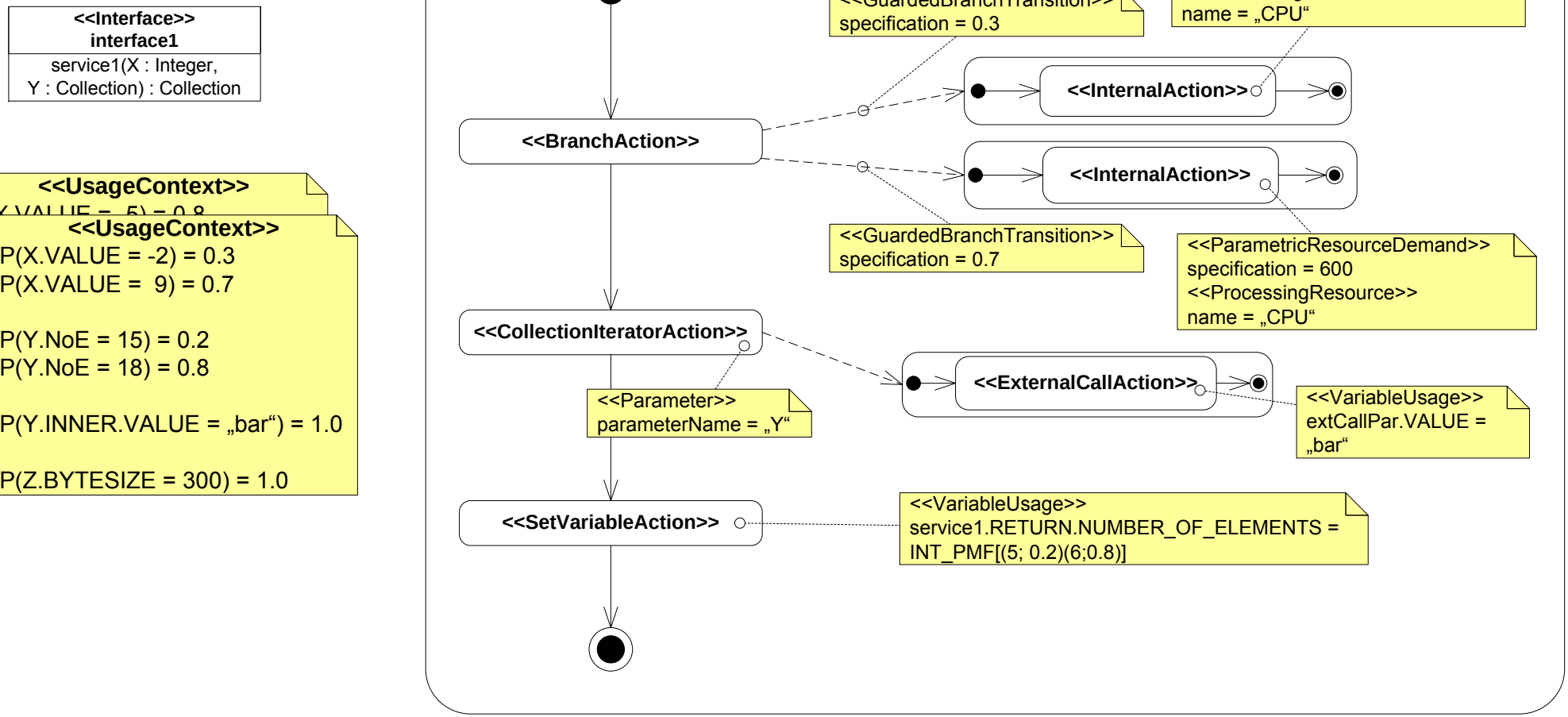


# Service Effect Specification

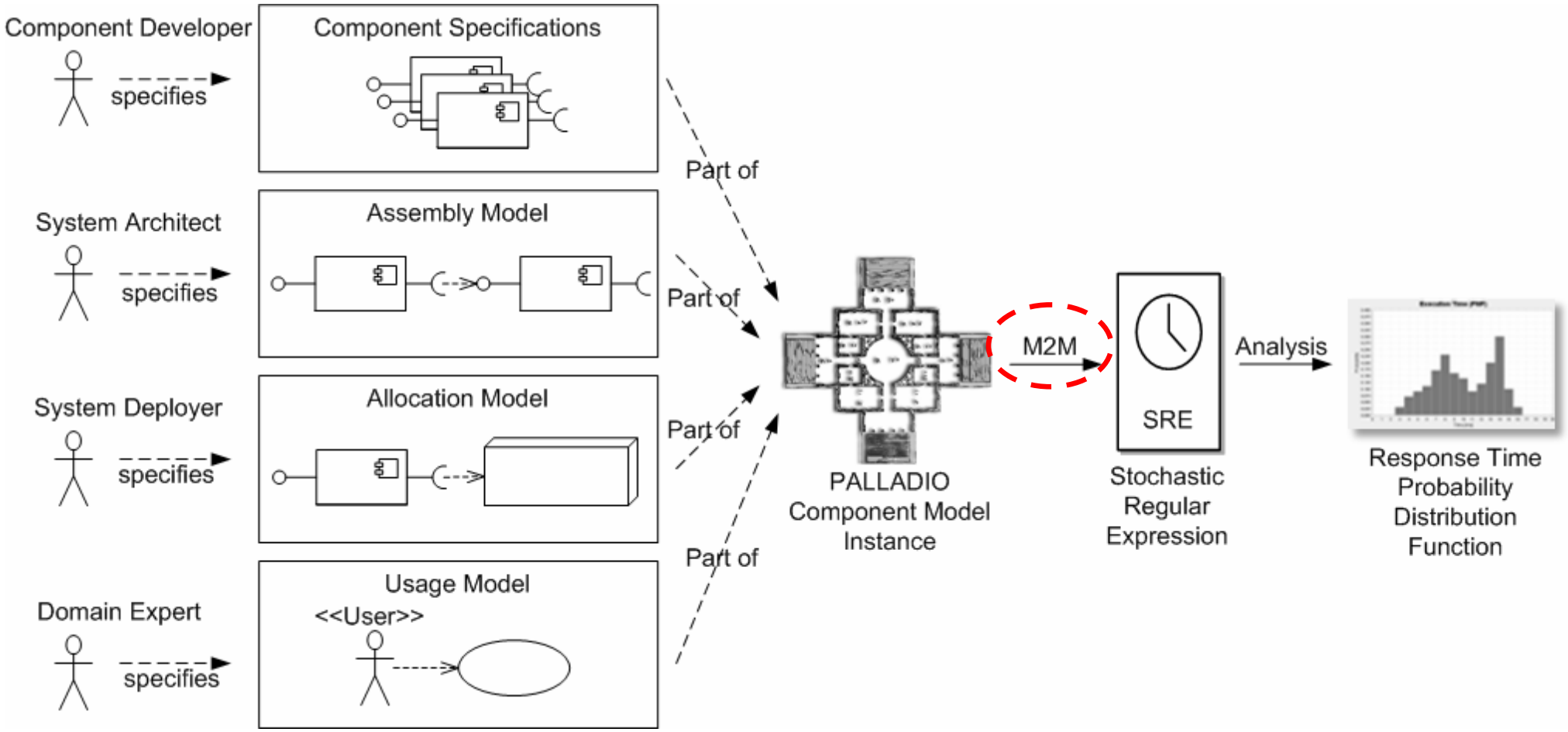


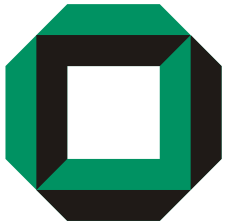



# Service Effect Specification

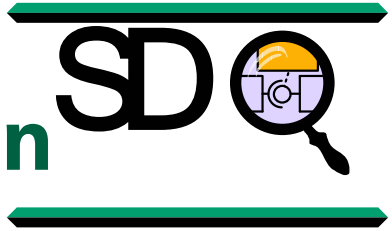


# Palladio Component Model





# Stochastic Regular Expression



Sequence

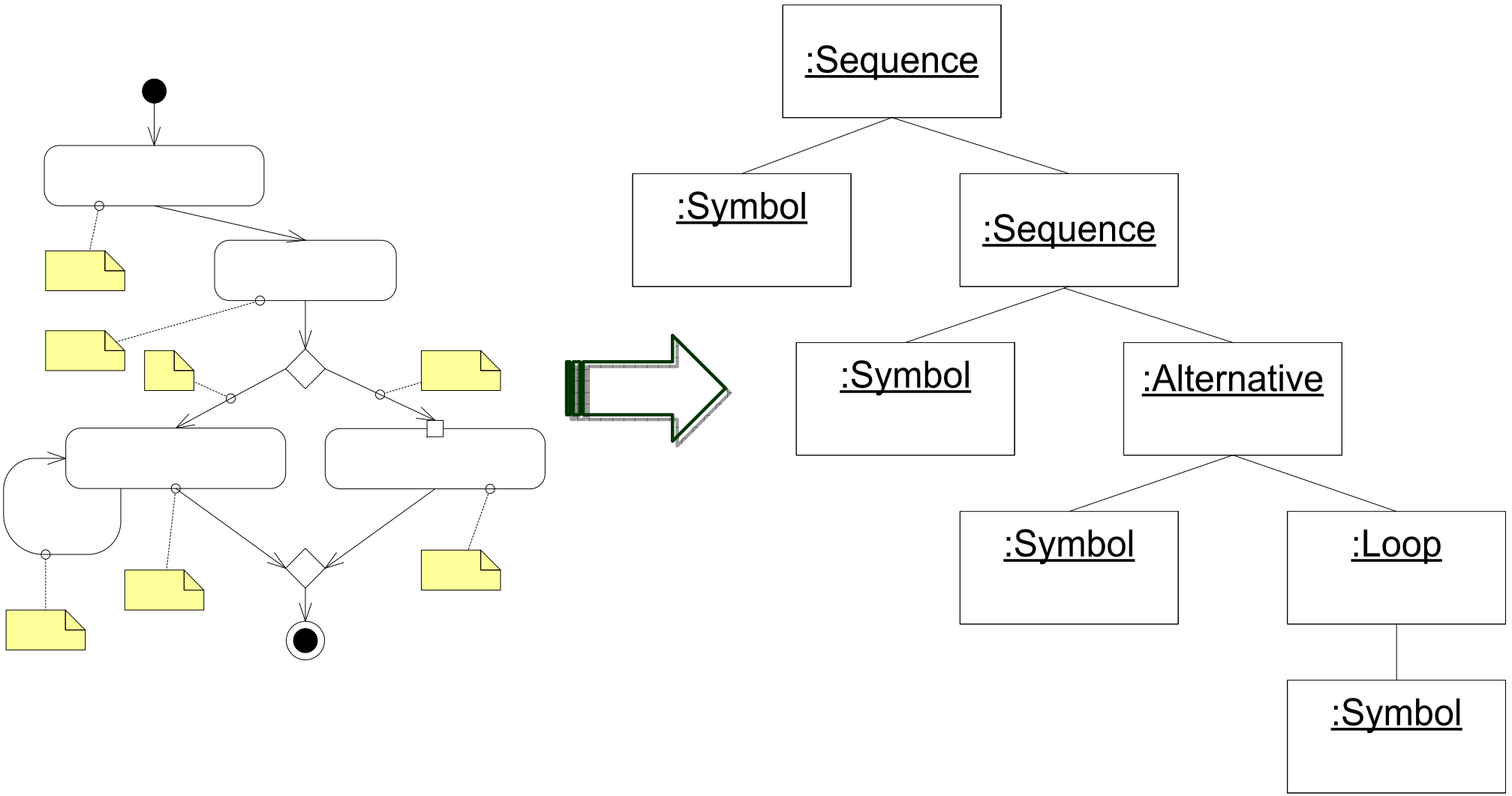
Loop Iterations (l)

$$P := a \mid P \cdot Q \mid P +_{\pi} Q \mid P^{*(l)}$$

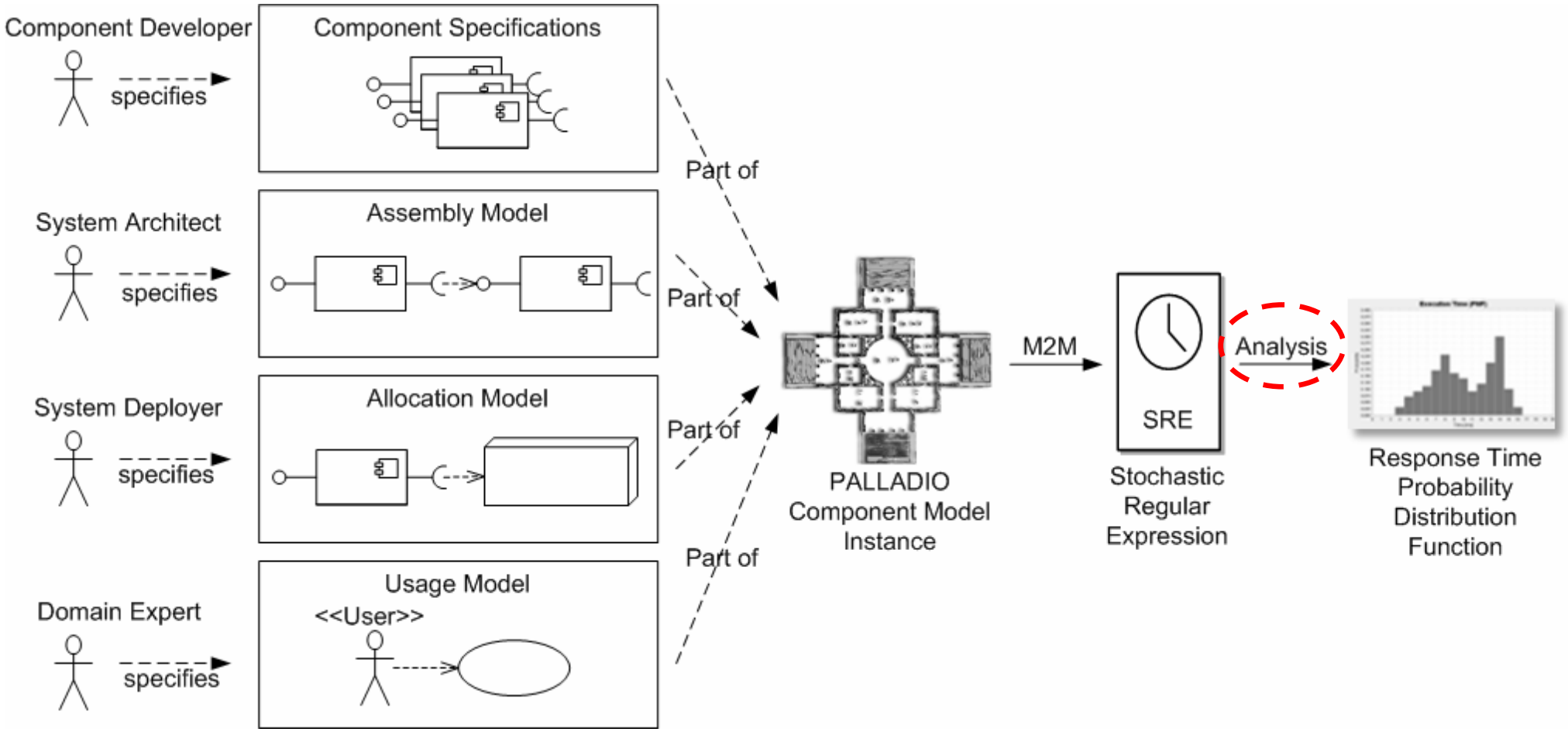
Symbol

Alternative Probability $\pi$, 1-$\pi$

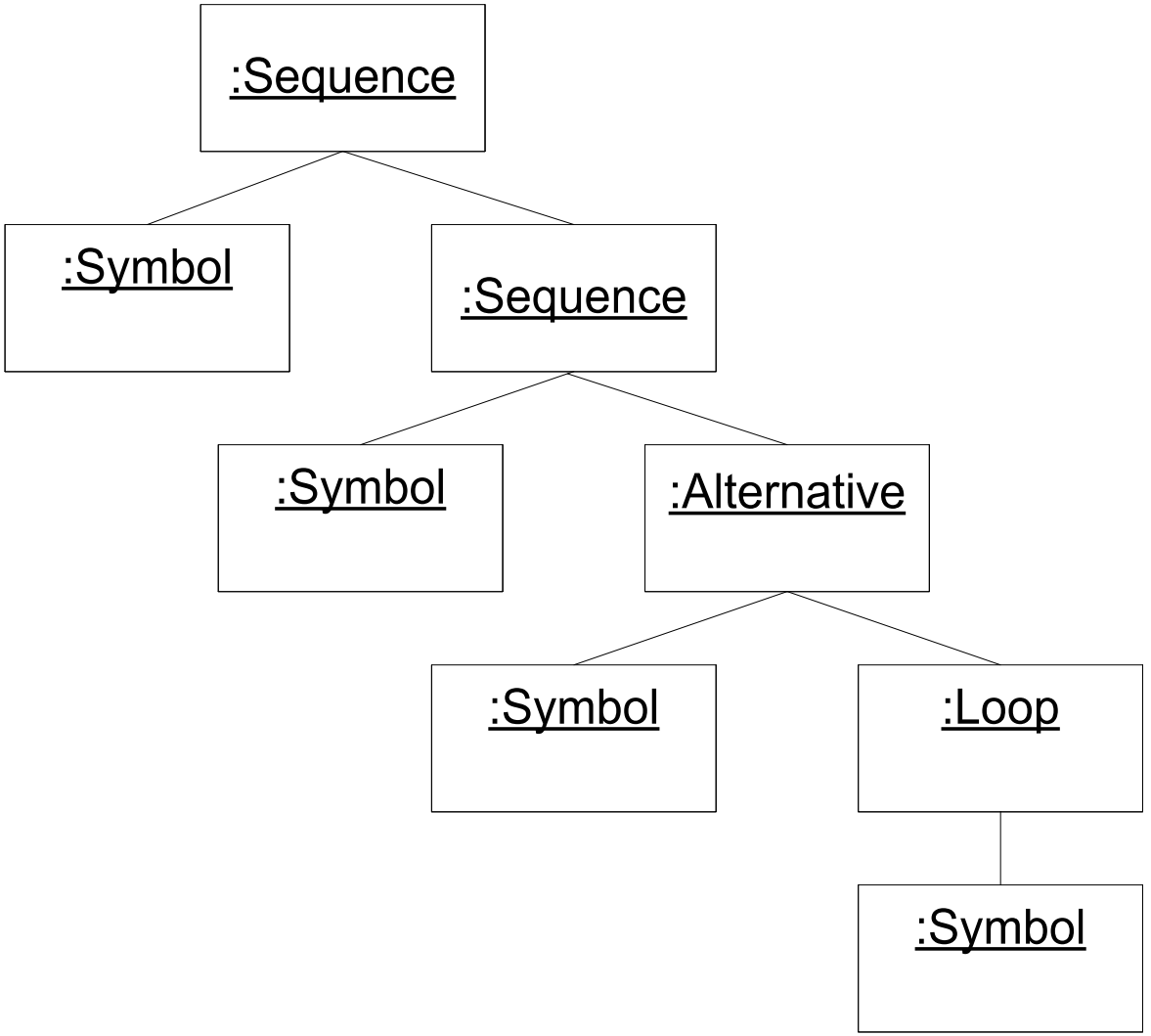
# Transformation



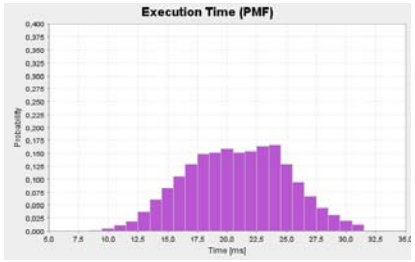
# Palladio Component Model



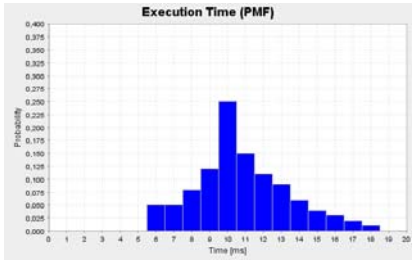
# Model Solution

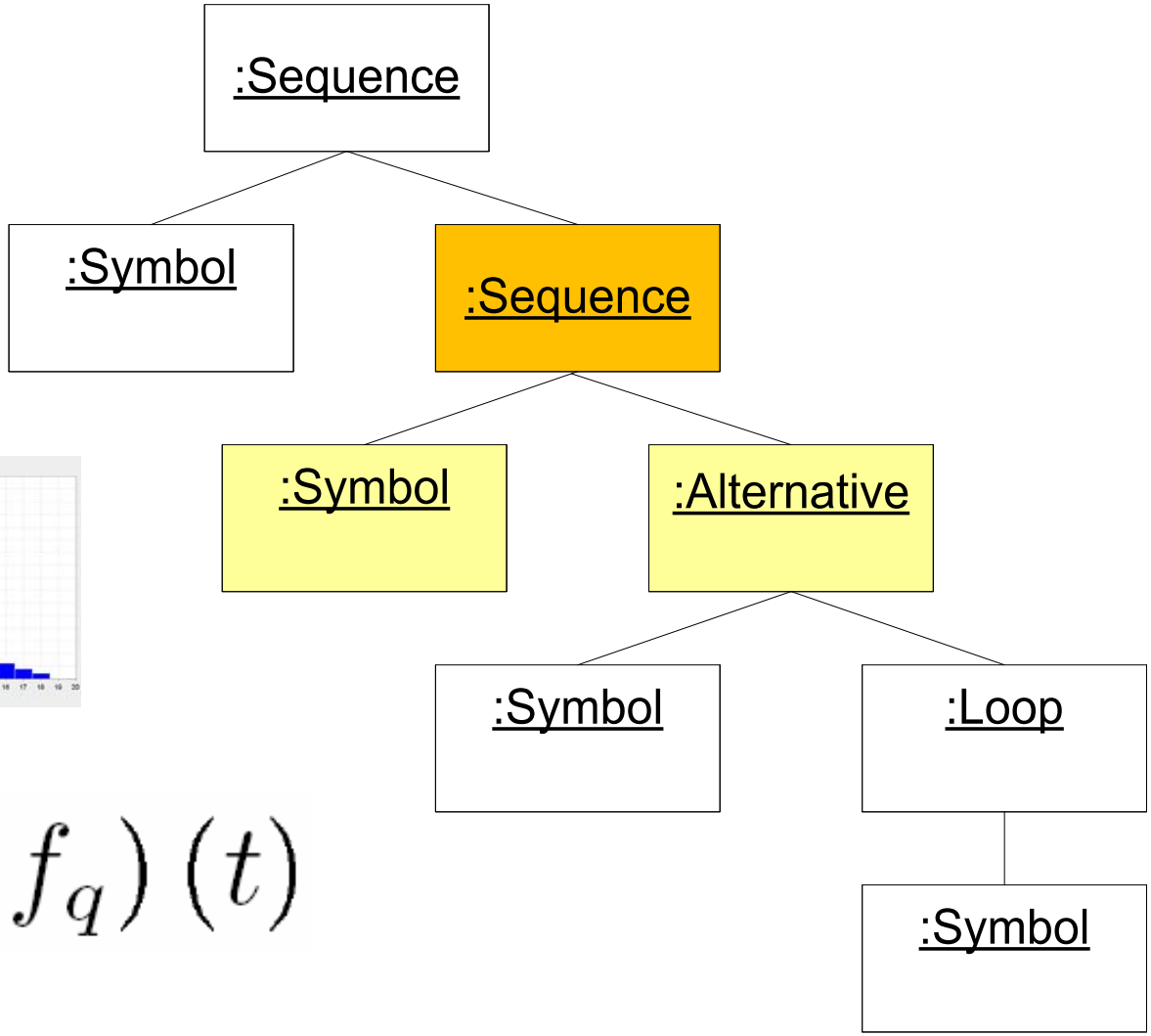


# Model Solution





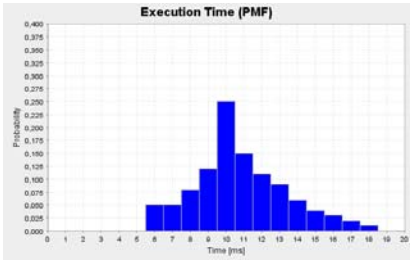


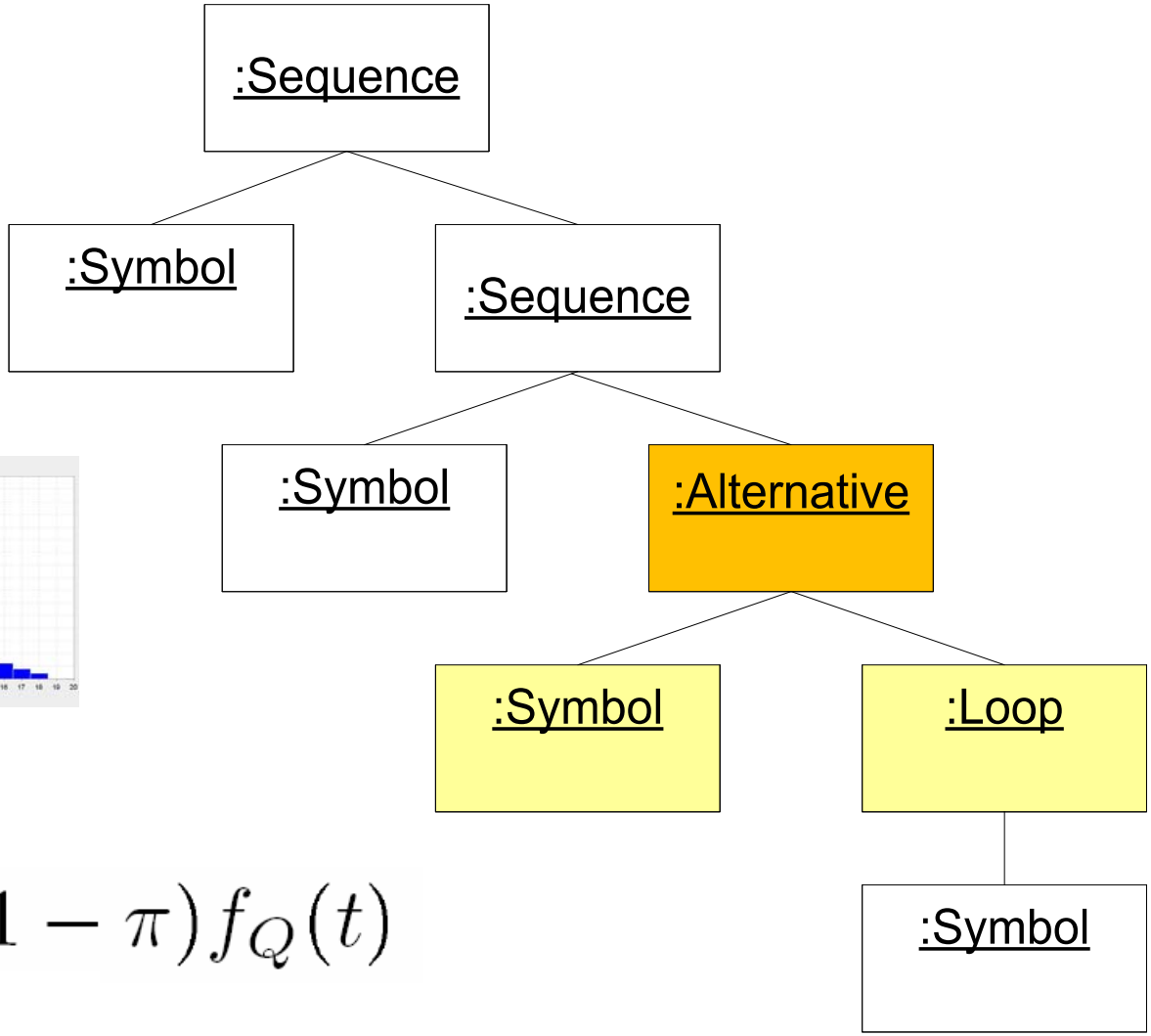




$$f_{P \cdot Q}(t) = (f_p \circledast f_q)(t)$$

# Model Solution

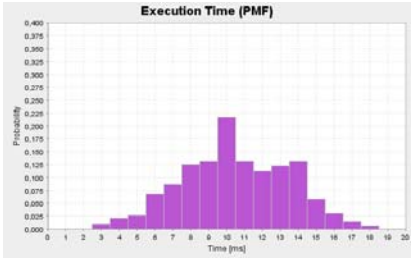


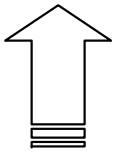


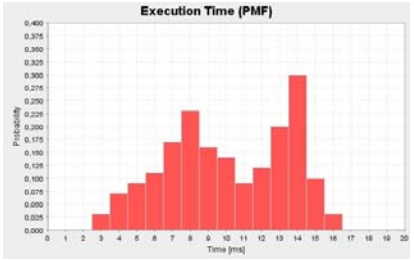



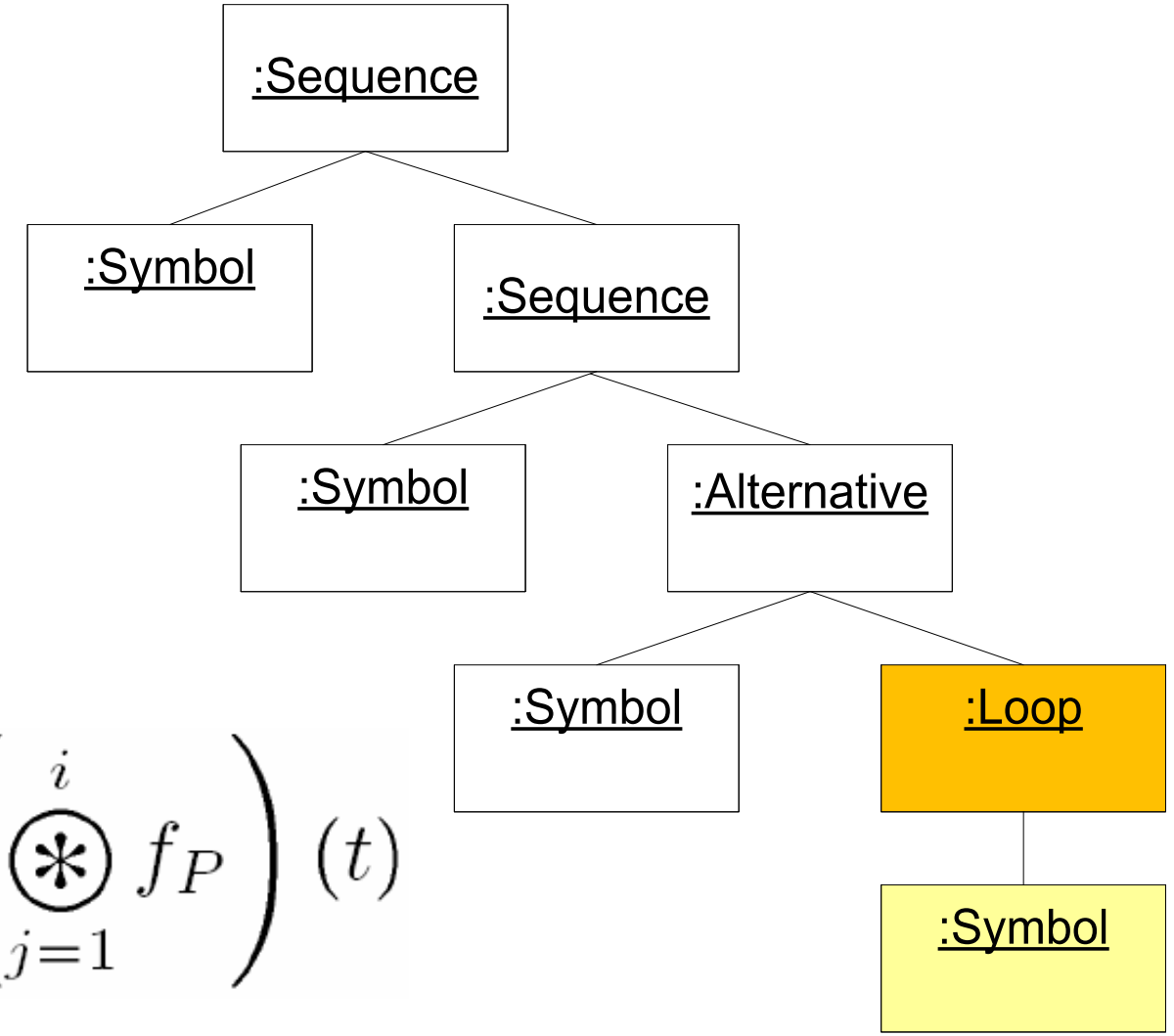

$$f_{P+_\pi Q}(t) = \pi f_P(t) + (1-\pi) f_Q(t)$$

# Model Solution



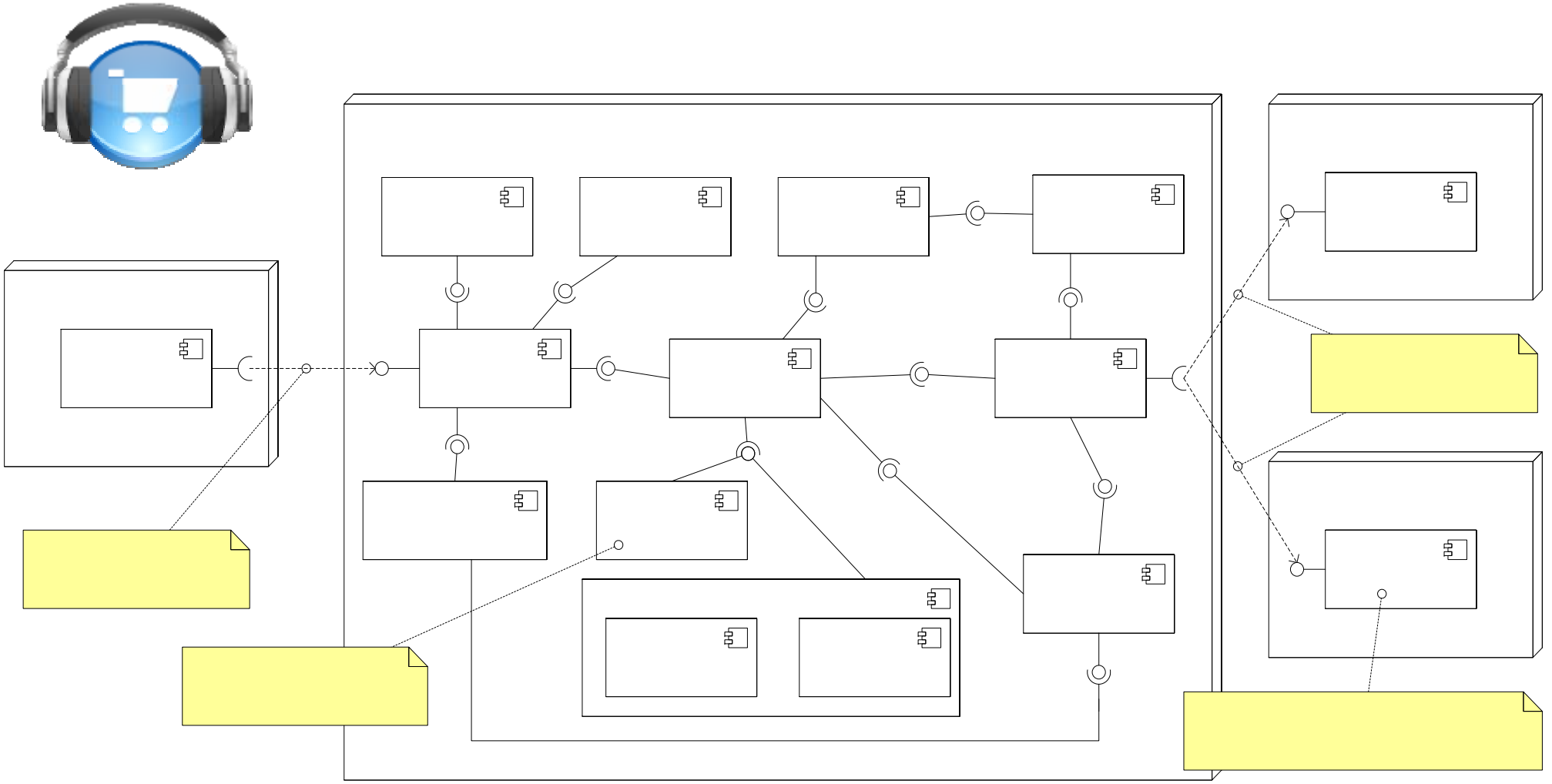





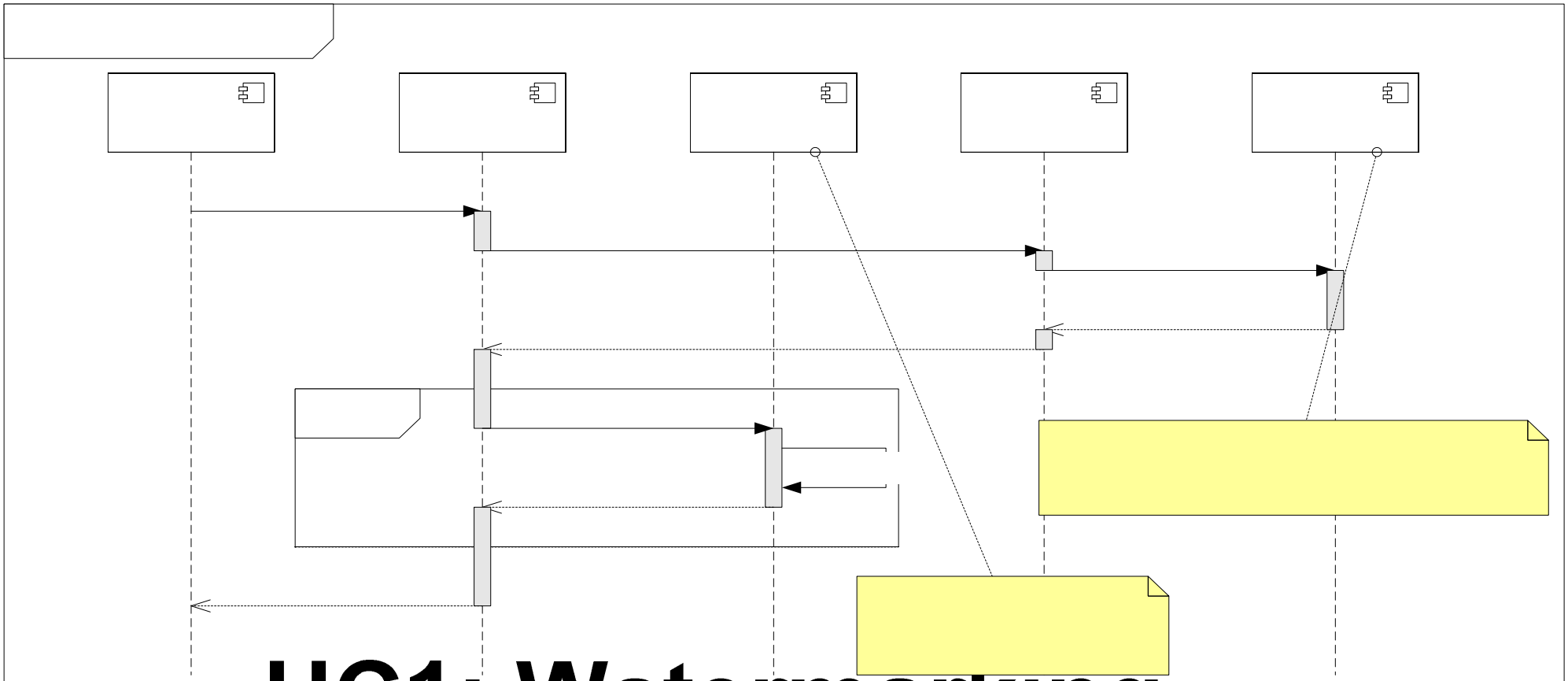




$$f_{P*(l)}(t) = \sum_{i=0}^{N} p_l(i) \left( \circledast_{j=1}^{i} f_P \right)(t)$$

# MediaStore - Architecture



# Download - Use Case



UC1: Watermarking



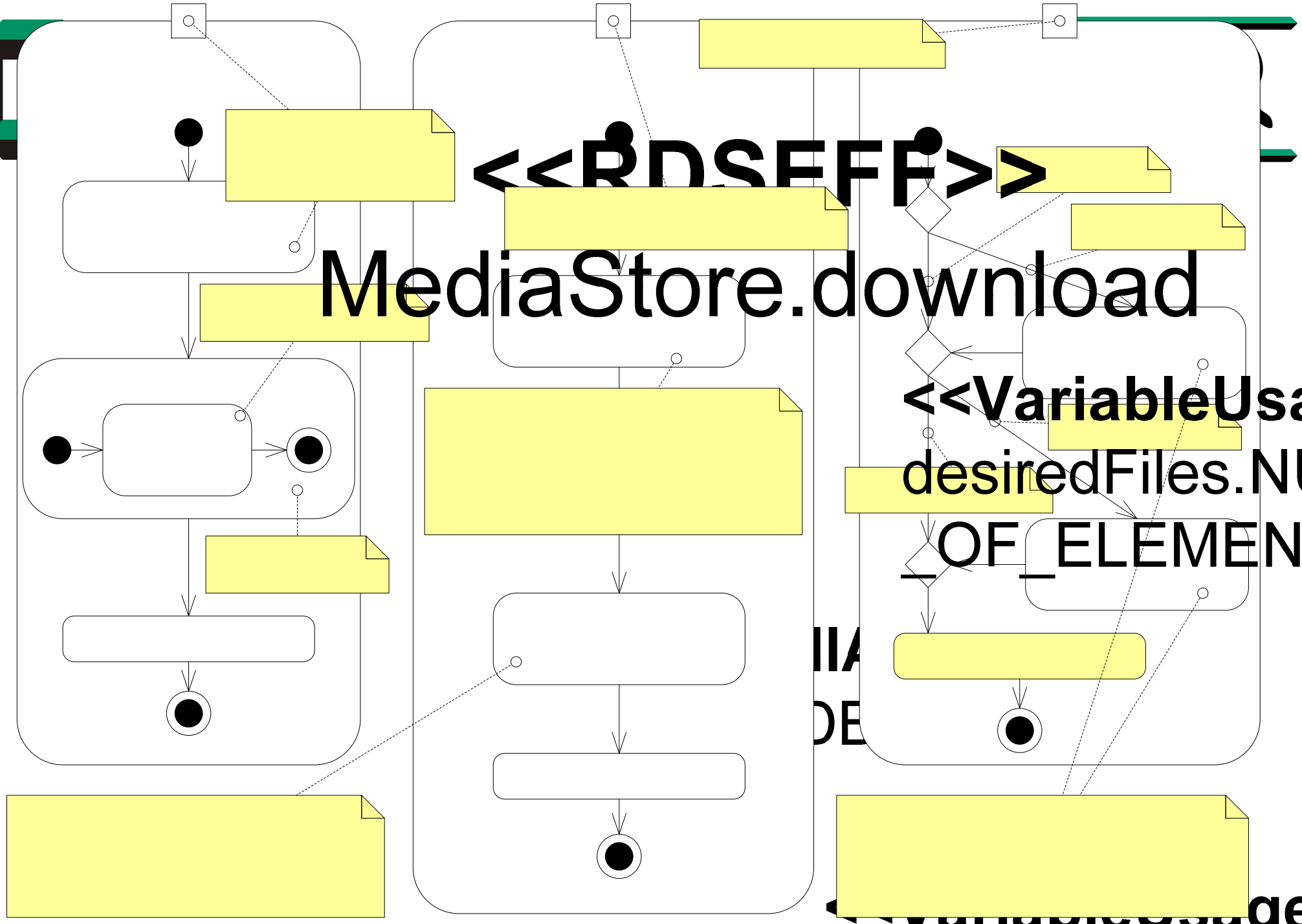
WebGUI

Mee

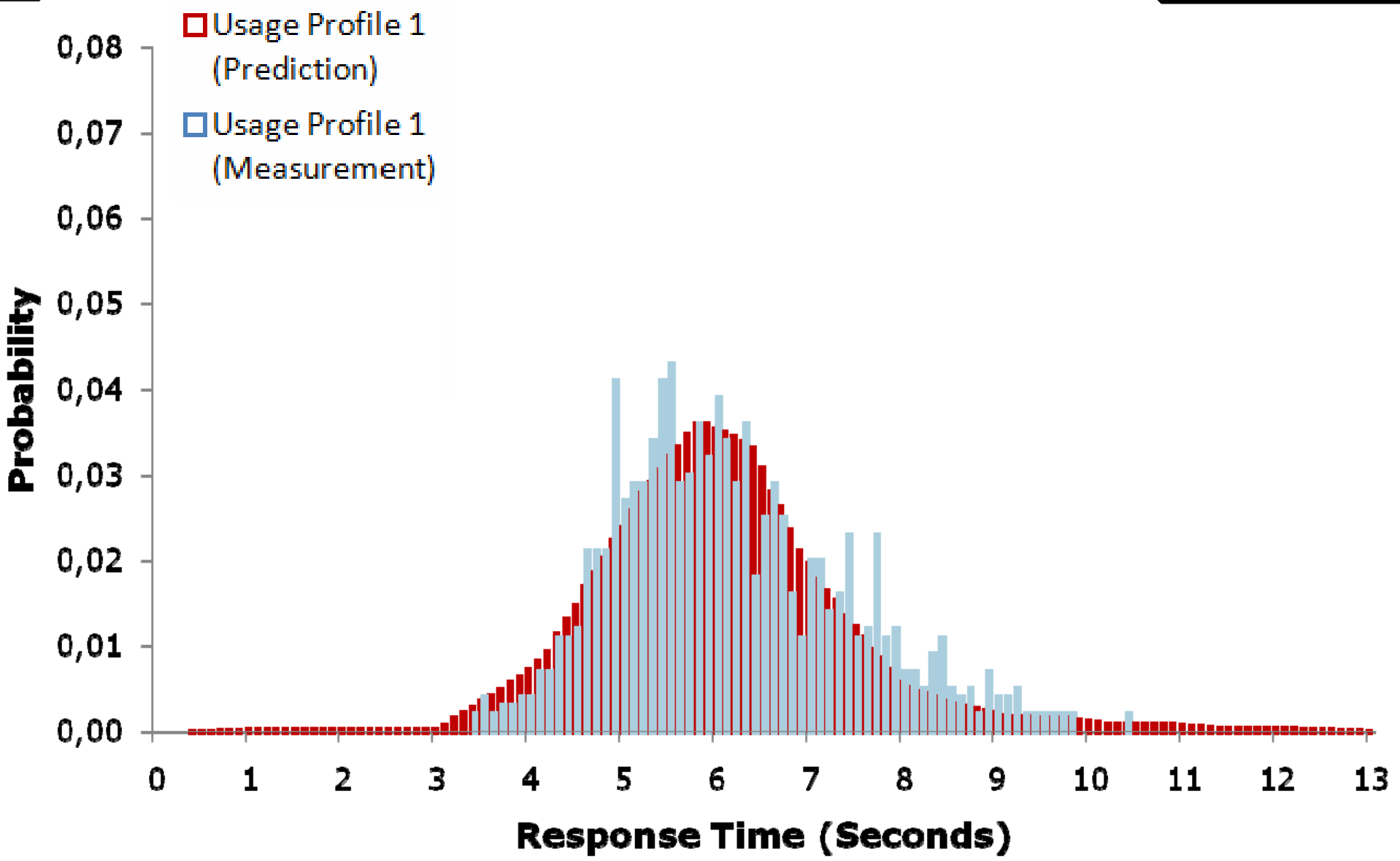
# Case Study: Usage Profiles

| Parameter Characterisation | Usage Profile 1 (AudioFiles) | Usage Profile 2 (VideoFiles) |
|---|---|---|
| DesiredFiles.NUMBER_OF_ELEMENTS | 10-14 | 1 |
| StoredFiles.NUMBER_OF_ELEMENTS | 250000 | 10000 |
| StoredFiles.INNER.BYTESIZE | 1-15 MB (audio) | 95-105MB (video) |
| ProbIncludeID.VALUE | 1.0 | 1.0 |
| ProbIncludeLyrics.VALUE | 0.0 | 1.0 |

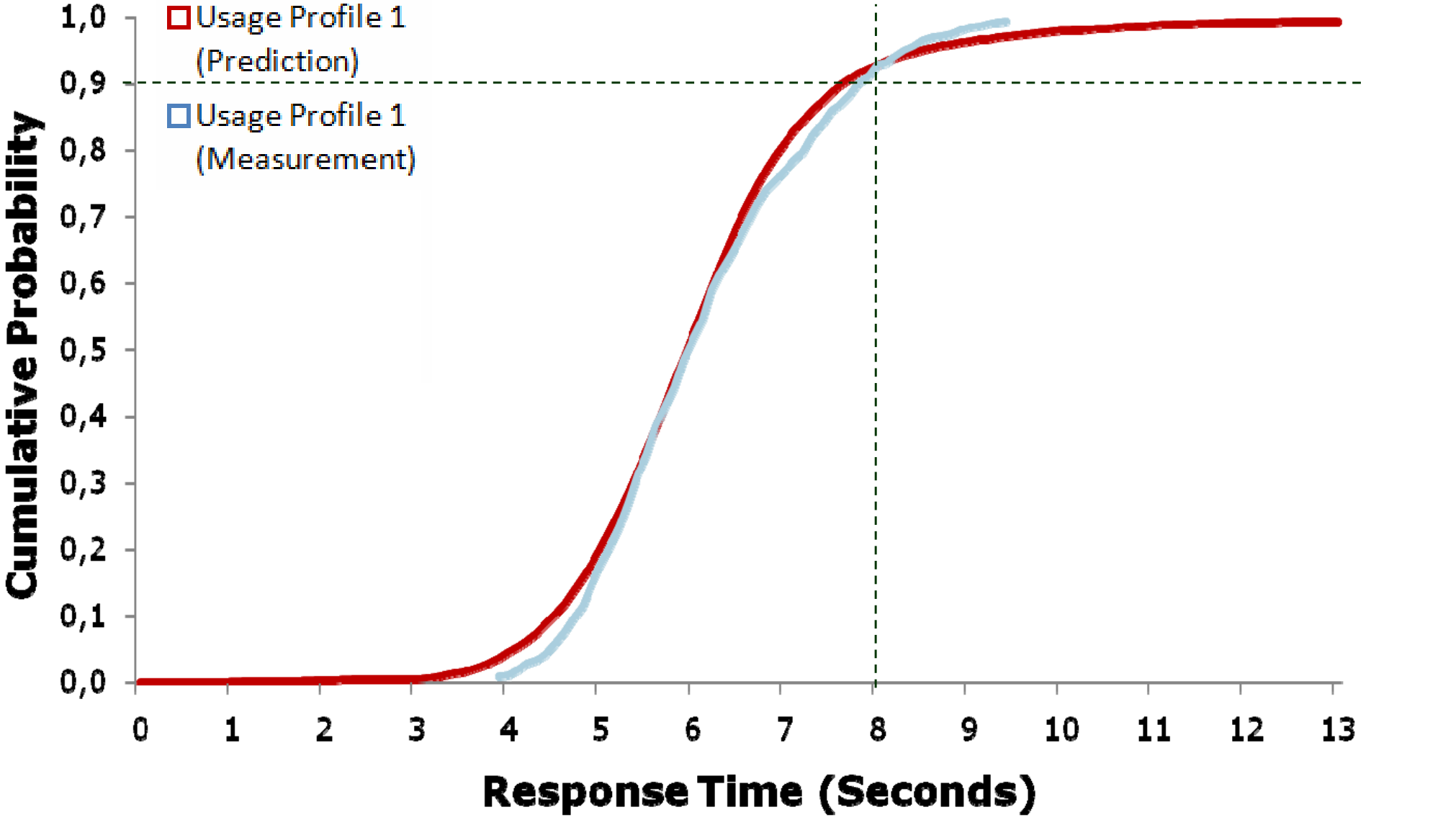
**<<UsageModel>>**
MediaStoreUsage


<<RDSEFF>>
MediaStore.download
<<VariableUsag
desiredFiles.NUM
_OF_ELEMENTS
IIA
DE
<<VariableUsage>
fileToMark.BYTESI

# Results


Usage Profile 1 (Prediction)
Usage Profile 1 (Measurement)
Probability
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0,00
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Response Time (Seconds)

# Results


Usage Profile 1
(Prediction)
Usage Profile 1
(Measurement)
Cumulative Probability
1,0
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0,0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Response Time (Seconds)