Model-Driven Non-Functional Analysis

Vittorio Cortellessa
Dipartimento di Informatica
Università dell'Aquila

Progetto PRIN PaCo (Performability-aware Computing)
Kickoff Meeting, Bertinoro, 23-24 ottobre 2008

SUMMARY

- Non-functional analysis and software development
- Model-driven performance analysis
- Model-driven reliability analysis
- Model-driven tradeoff analysis
- Tool support
- Research perspectives

Non-functional analysis of software

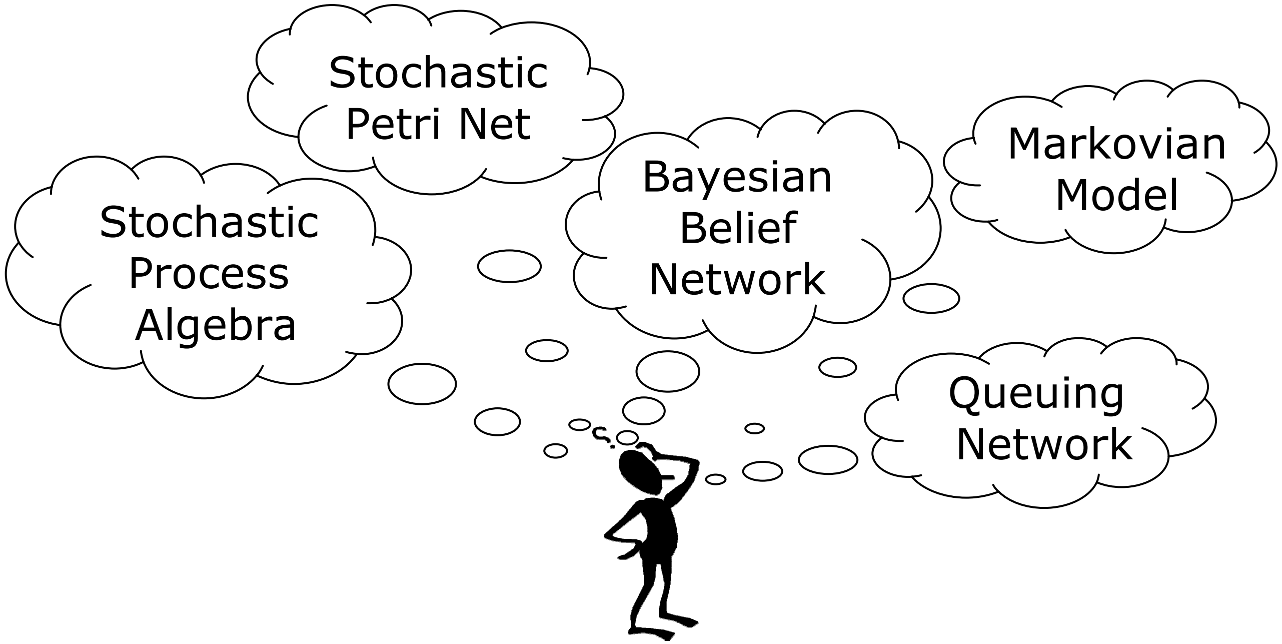
Software modeling domain (vocabulary) is intrinsically distant from non-functional analysis one.

Software modeling

Non-functional analysis

Reluctance to embed non-functional modeling and analysis in the software development process.

Non-functional analysis of software



"Human made" (ad-hoc) non-functional models : difficult to build and prone to errors

Non-functional analysis of software

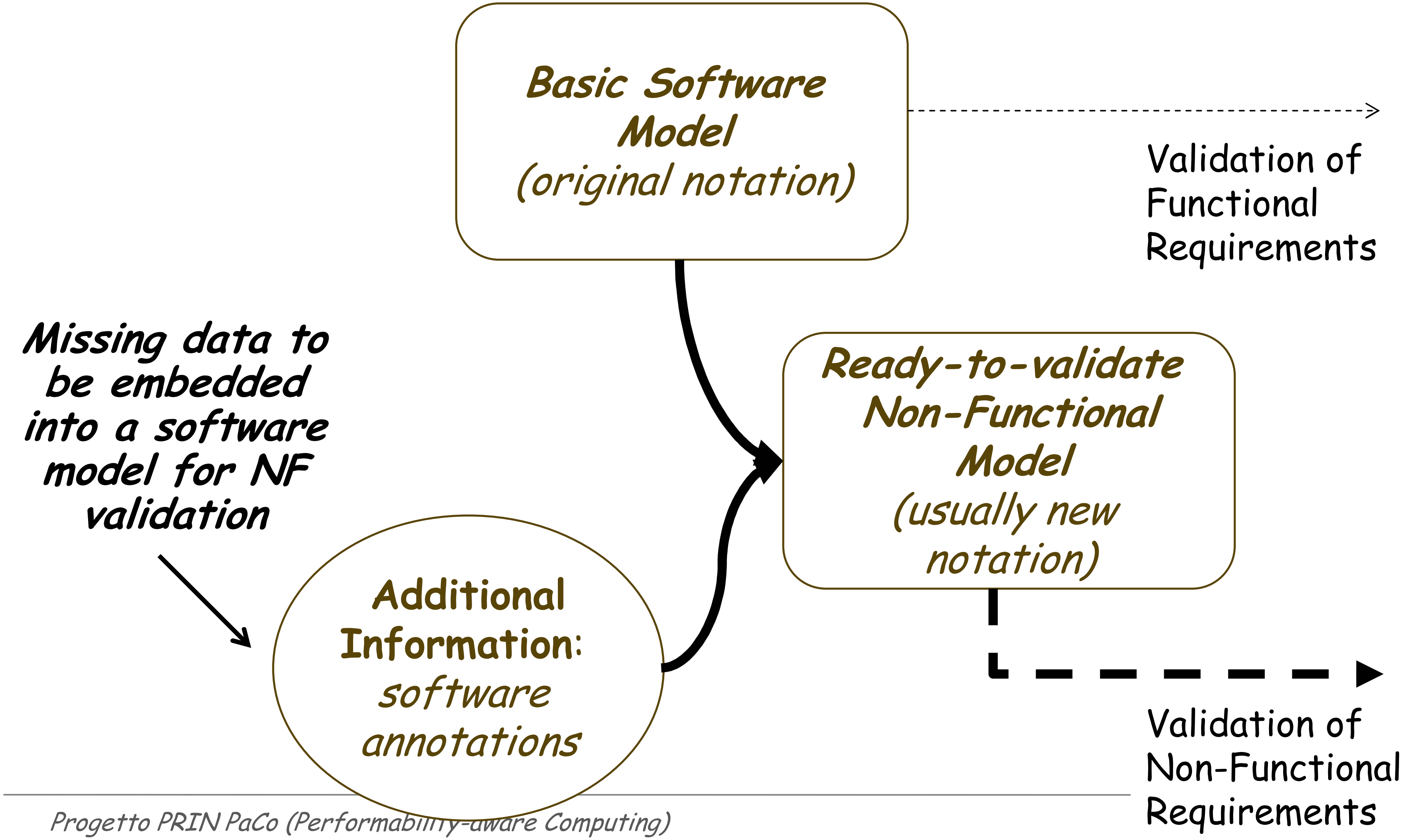
A new vision in non-functional analysis (1998-today)

Introducing **automation** to **generate non-functional models** from software models

⇓

Big effort devoted to design **model transformations** having **performance/reliability/... models** as targets

Non-functional analysis of software: a general schema



Model annotations

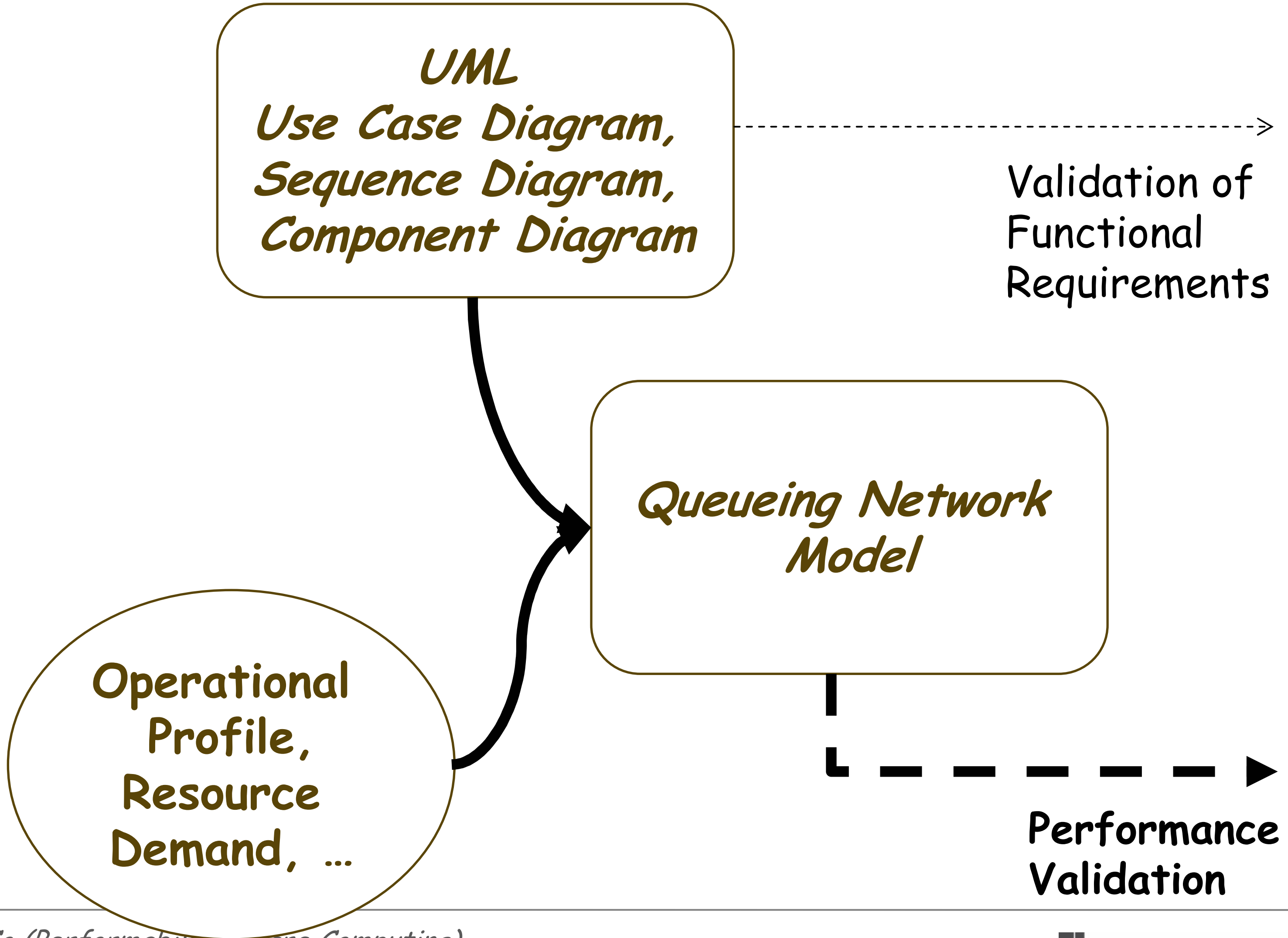
In **UML** the **MARTE Specification** shall play the role of "container" of concepts related to non-functional attribues

OMG, A UML Profile for MARTE, ptc/07-08-04

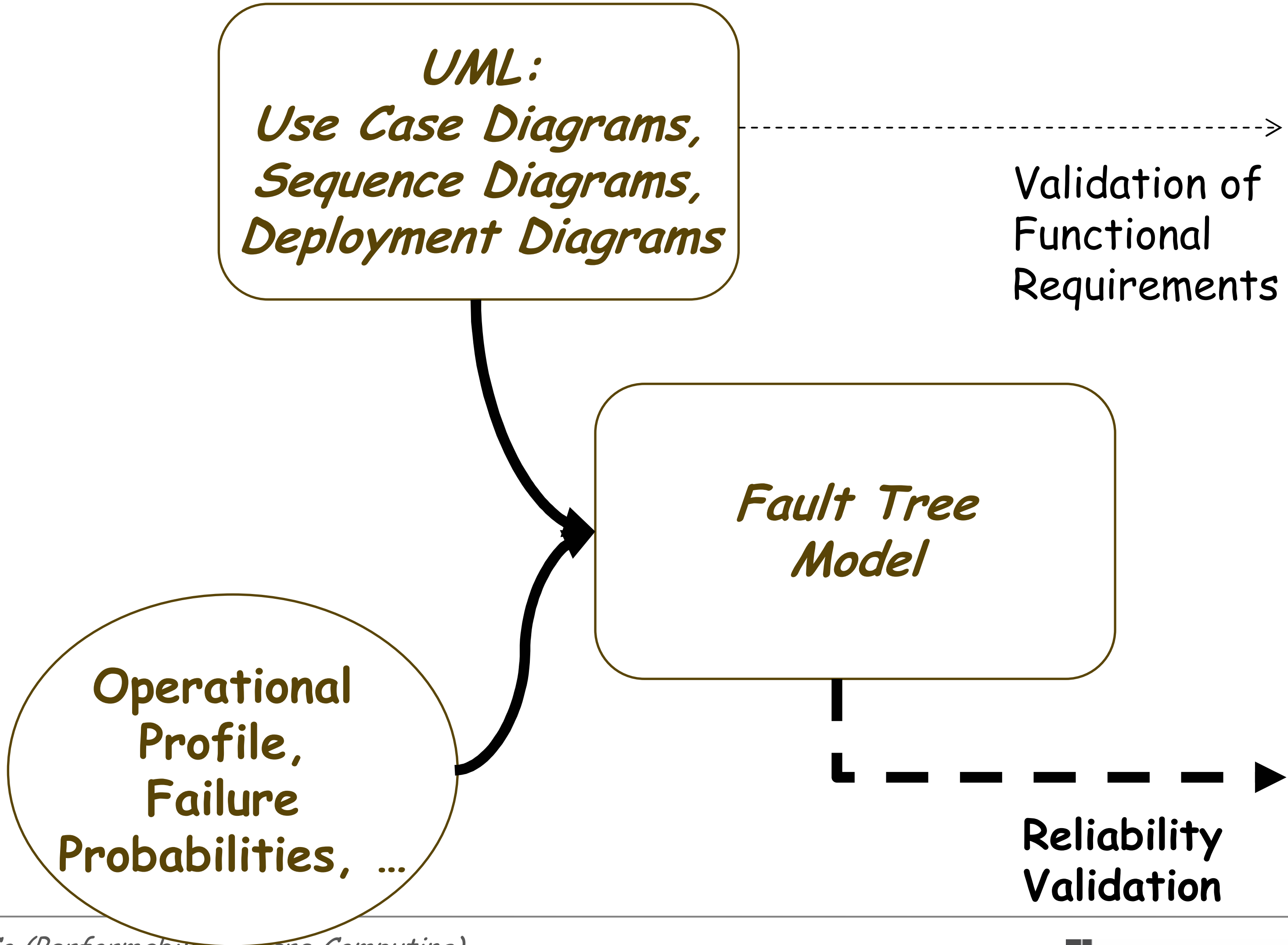
All **non-UML** notations that intends to embed non-functional concepts have to define **appropriate syntax and semantics** for those concepts

In general, different non-functional attributes share more than one concept, thus it may be reasonable to design "**core" specifications for common concepts** plus **specific concepts for each attribute**

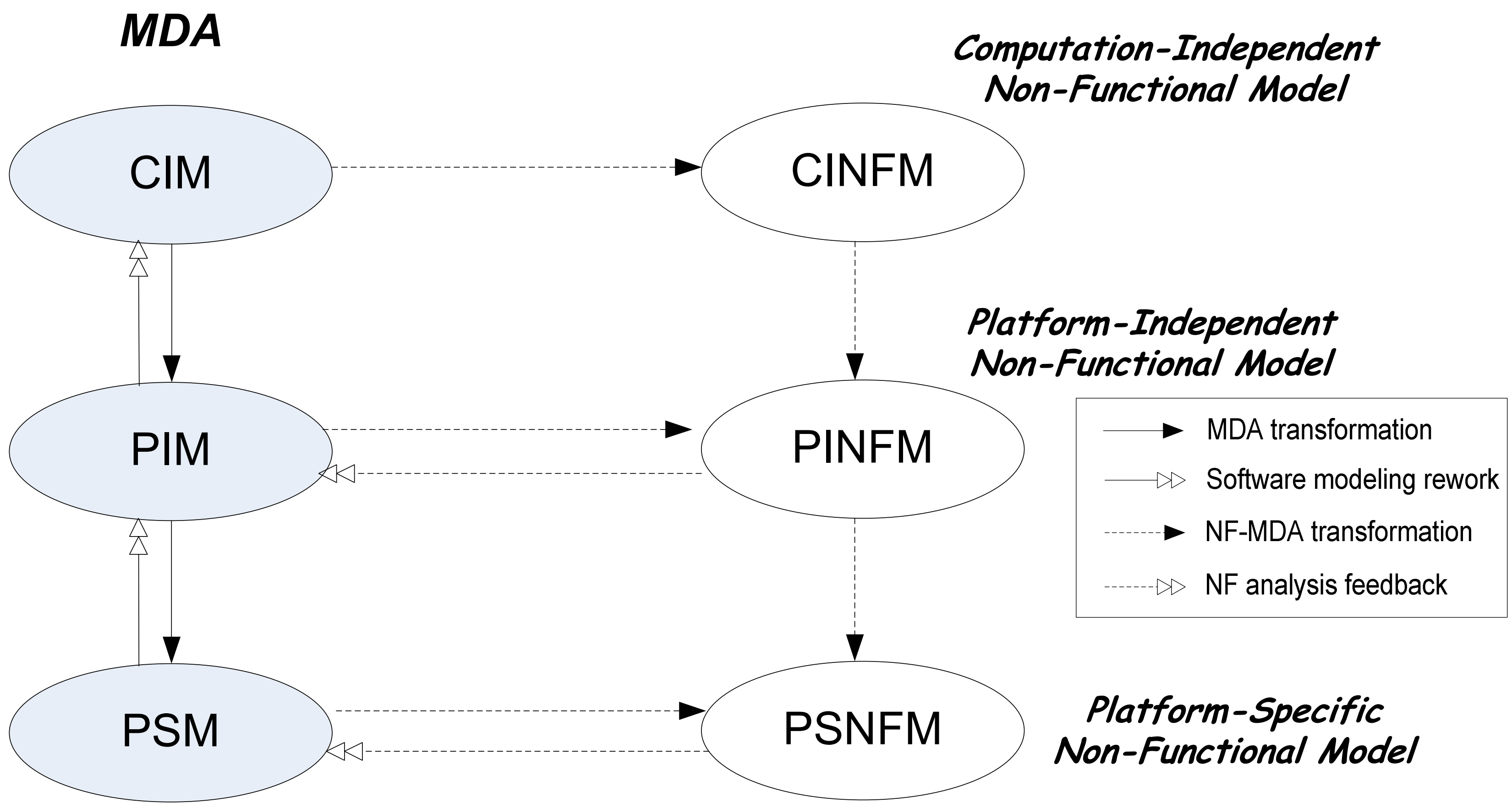
Non-functional analysis of software: an instance of the schema…



Non-functional analysis of software: … and yet another instance

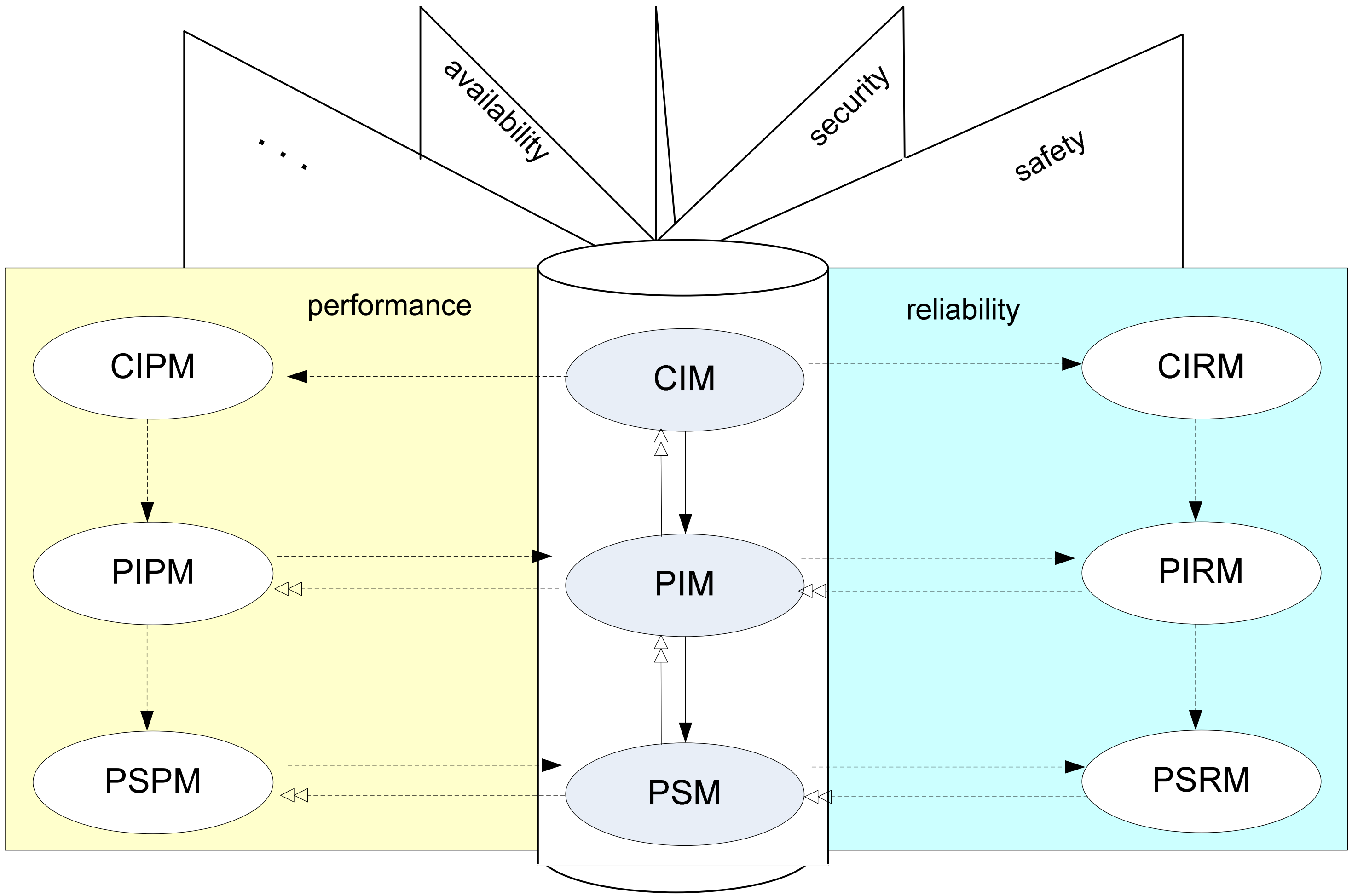


Non-functional models in Model-Driven Architecture



Horizontal and vertical transformations

Non-functional models in Model-Driven Architecture



Non-functional models in Model-Driven Architecture

- Systematic embedding of **non-functional modeling and analysis** in a model-driven **development process**

- **Automation** is the key **to allow** a real **integration** of these activities **within the process**

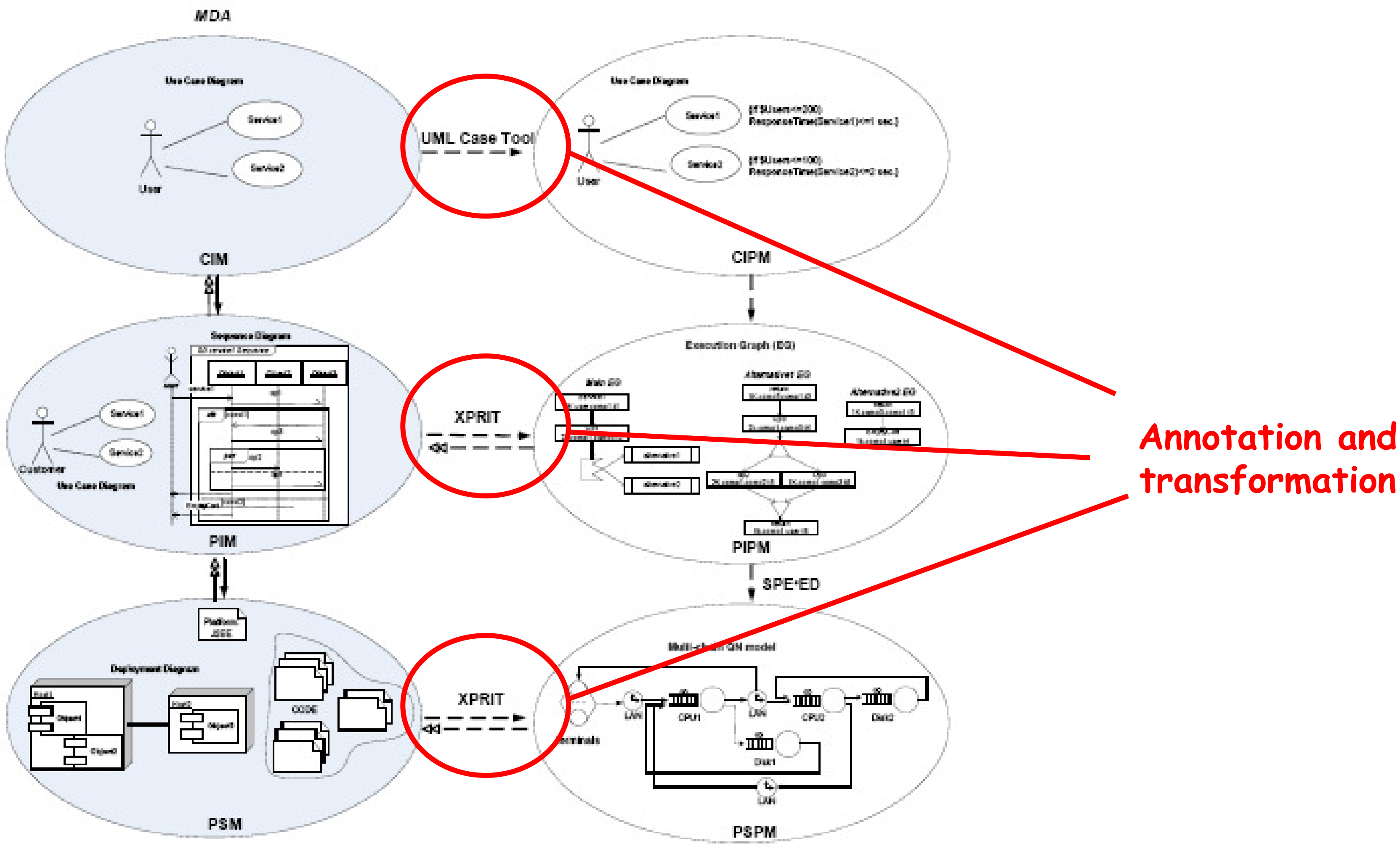
- The definition of **CINFM, PINFM, PSNFM** may not be trivial (or may be nonsense) for certain non-functional attributes

- The same consideration applies to the **additional transformations** introduced **in NFMDA**

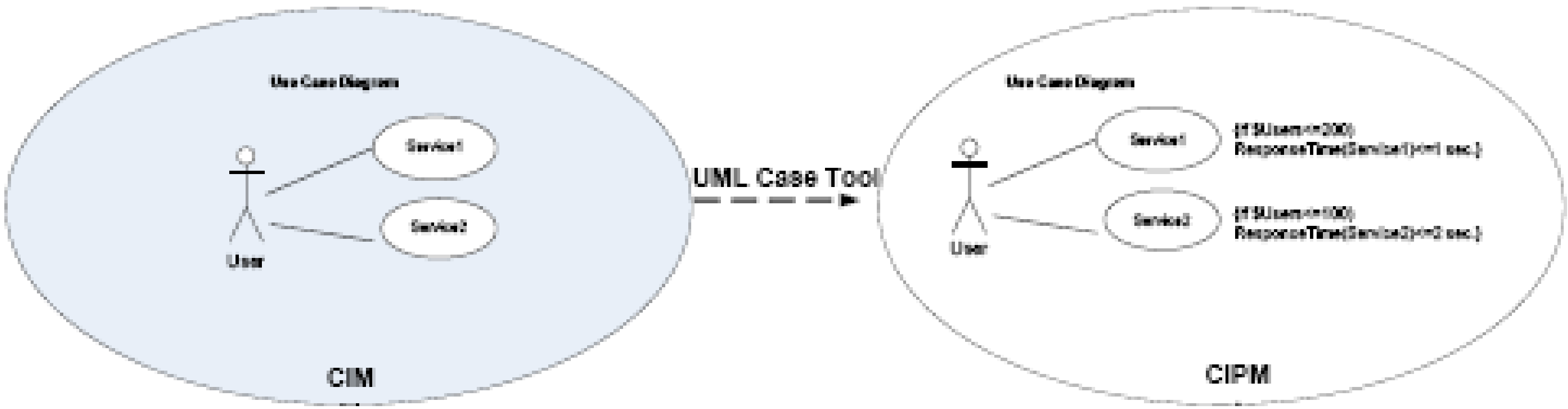
SUMMARY

- Non-functional analysis and software development
- Model-driven performance analysis
- Model-driven reliability analysis
- Model-driven tradeoff analysis
- Tool support
- Research perspectives

Software Performance MDA: XPRIT methodology

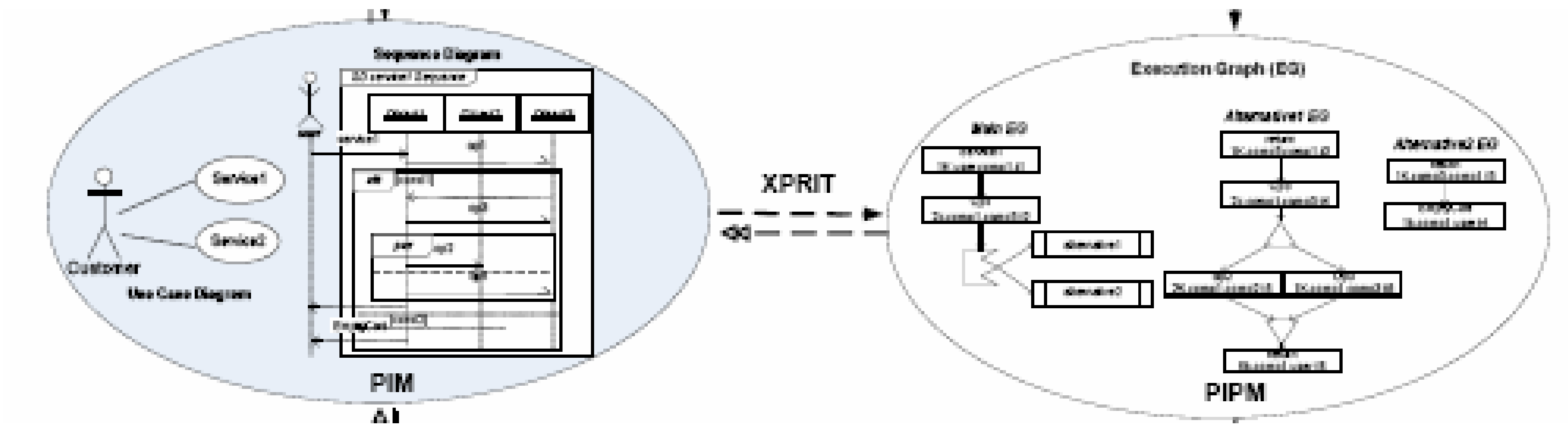


Software Performance MDA: CIM $\longrightarrow$ CIPM



Performance requirements are annotated on a Use Case Diagram

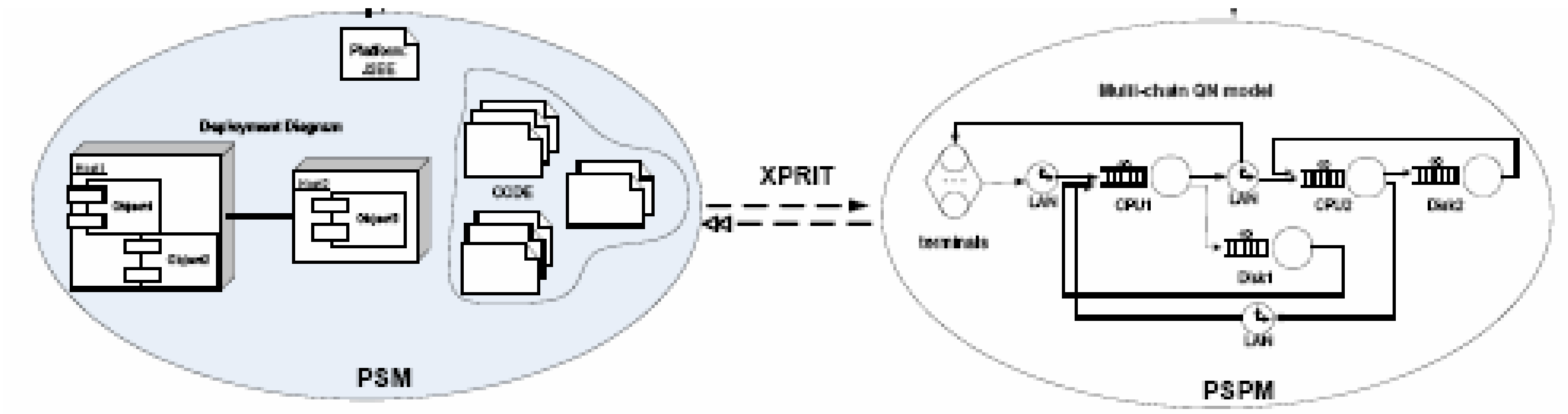
Software Performance MDA: PIM $\longrightarrow$ PIPM



Software dynamics is annotated with **operational profile** and **resource demands** and transformed in **Execution Graphs**

At a limited extent, **performance analysis** can be carried out **at Platform Independent level**

Software Performance MDA: PSM ⟶ PSPM



A Deployment Diagram is annotated with **device characteristics** and transformed into a **Queuing Network** representing the **computational platform**

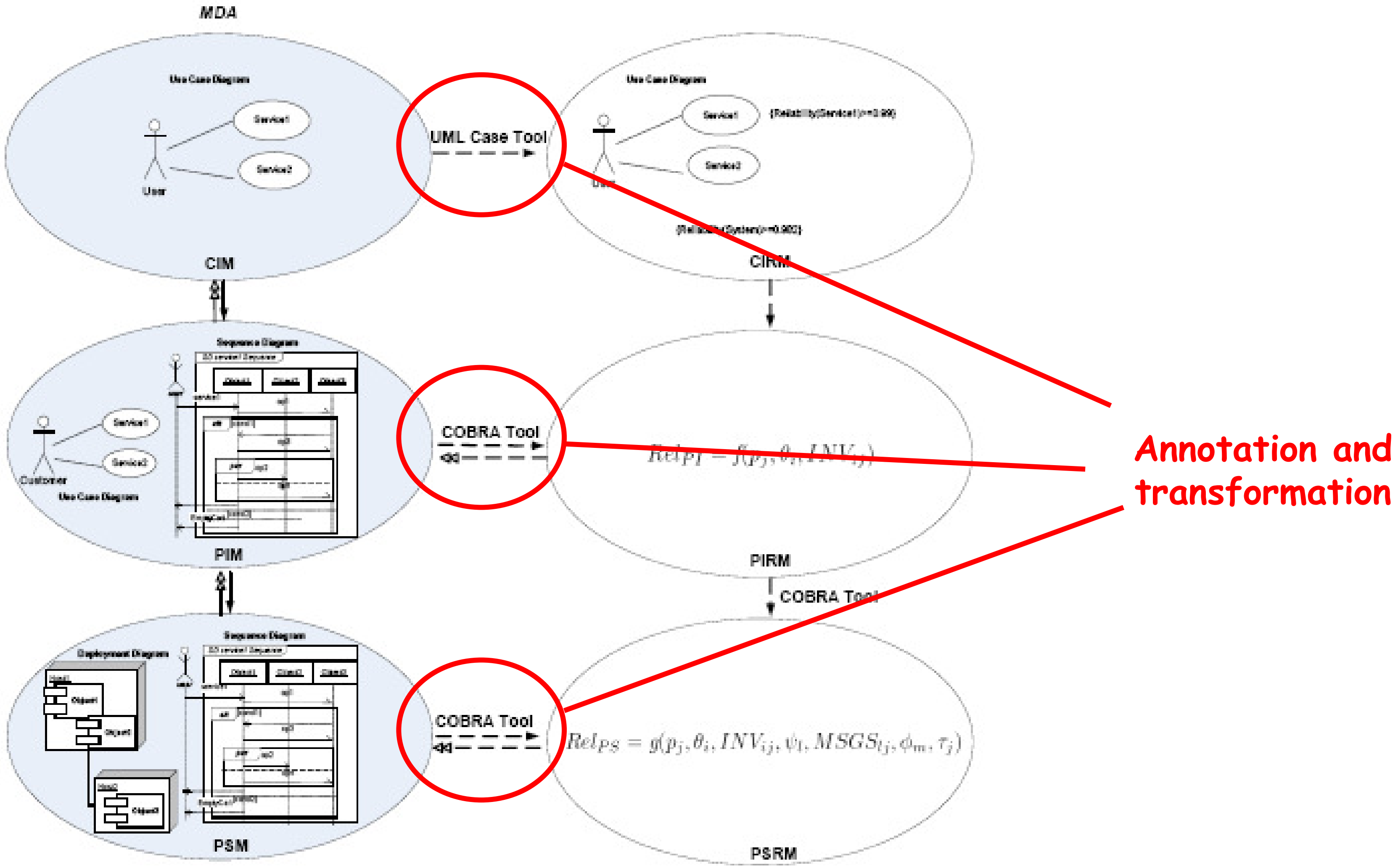
Workload is synthesized from the Execution Graphs (PIPM)

Performance analysis embedding **resource contention** is carried out at Platform Specific level

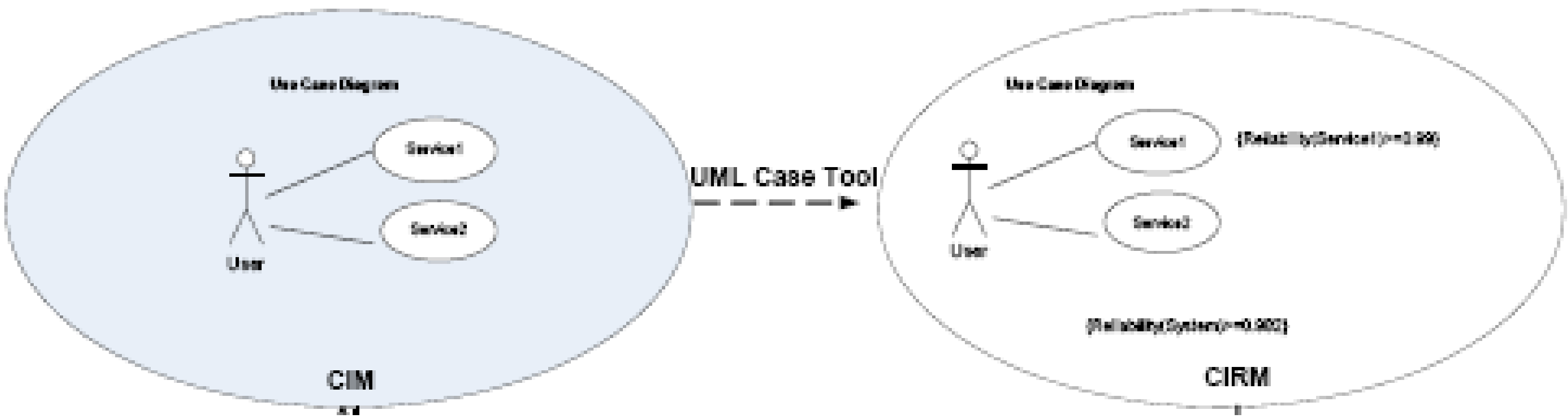
SUMMARY

- Non-functional analysis and software development
- Model-driven performance analysis
- Model-driven reliability analysis
- Model-driven tradeoff analysis
- Tool support
- Research perspectives

Software Reliability MDA: COBRA methodology

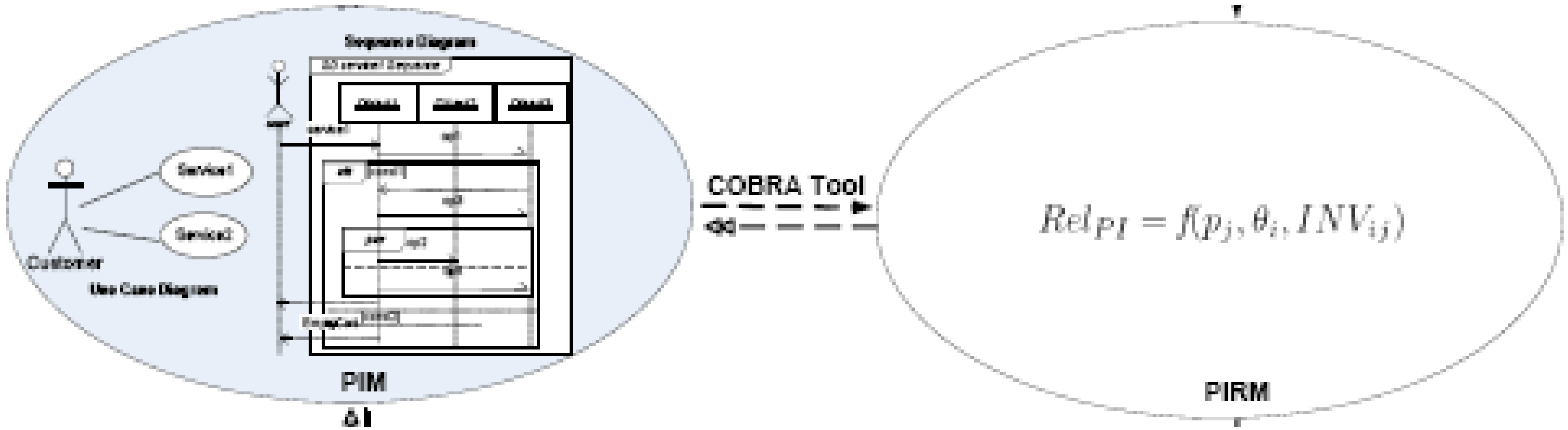


Software Reliability MDA: CIM $\longrightarrow$ CIRM



Reliability requirements are annotated on a Use Case Diagram

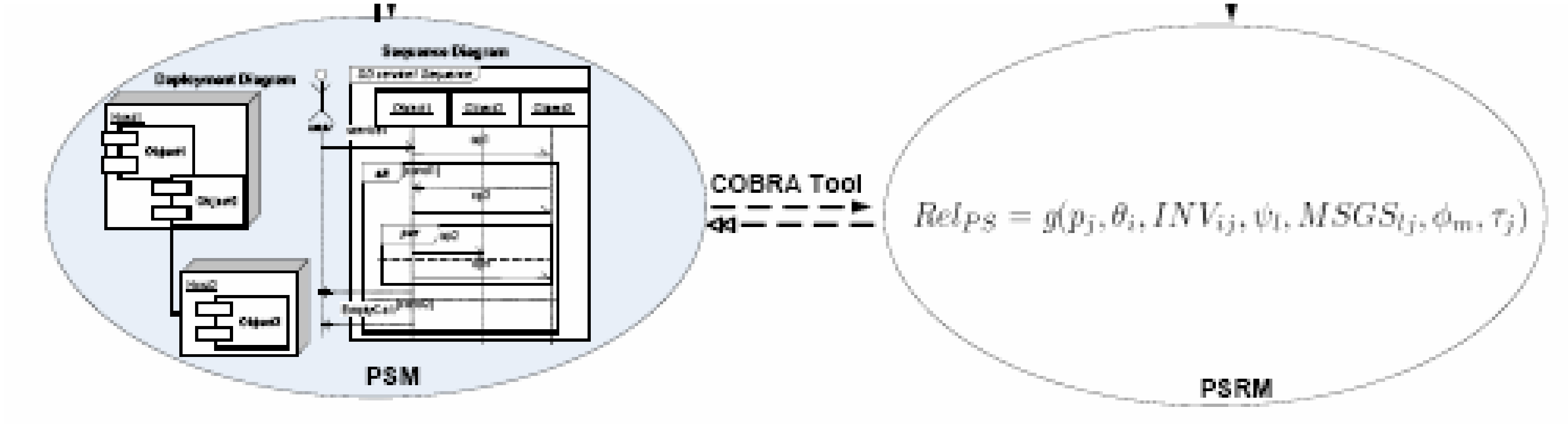
Software Reliability MDA: PIM $\longrightarrow$ PIRM



Software dynamics is annotated with **operational profile** and **probabilities of software component failures** and transformed into **a function (of random variables)**

Reliability analysis (only based on **software failures**) can be carried out at Platform Independent level

Software Reliability MDA: PSM $\longrightarrow$ PSRM



A Deployment Diagram is annotated with **hardware failure** (sites and connectors) **probabilities** and transformed in **another function (of random variables)**

Reliability analysis (based on **software and hardware failures**) can be carried out at Platform Specific level

SUMMARY

- Non-functional analysis and software development
- Model-driven performance analysis
- Model-driven reliability analysis
- Model-driven tradeoff analysis
- Tool support
- Research perspectives

Model-driven tradeoff analysis

1. Optimization models for cost/quality tradeoffs

2. A compositional approach to security/performance tradeoff

Model-driven tradeoff analysis

1. **Optimization models for cost/quality tradeoffs**

2. A compositional approach to security/performance tradeoff

Search-Based Software Engineering

It is inspired by the observation that many activities in software engineering can be **formulated as optimization problems**

OPERATIONAL RESEARCHERS

SOFTWARE ENGINEERS

__Exact__ **optimization techniques** of operation research like **linear** or dynamic **programming**

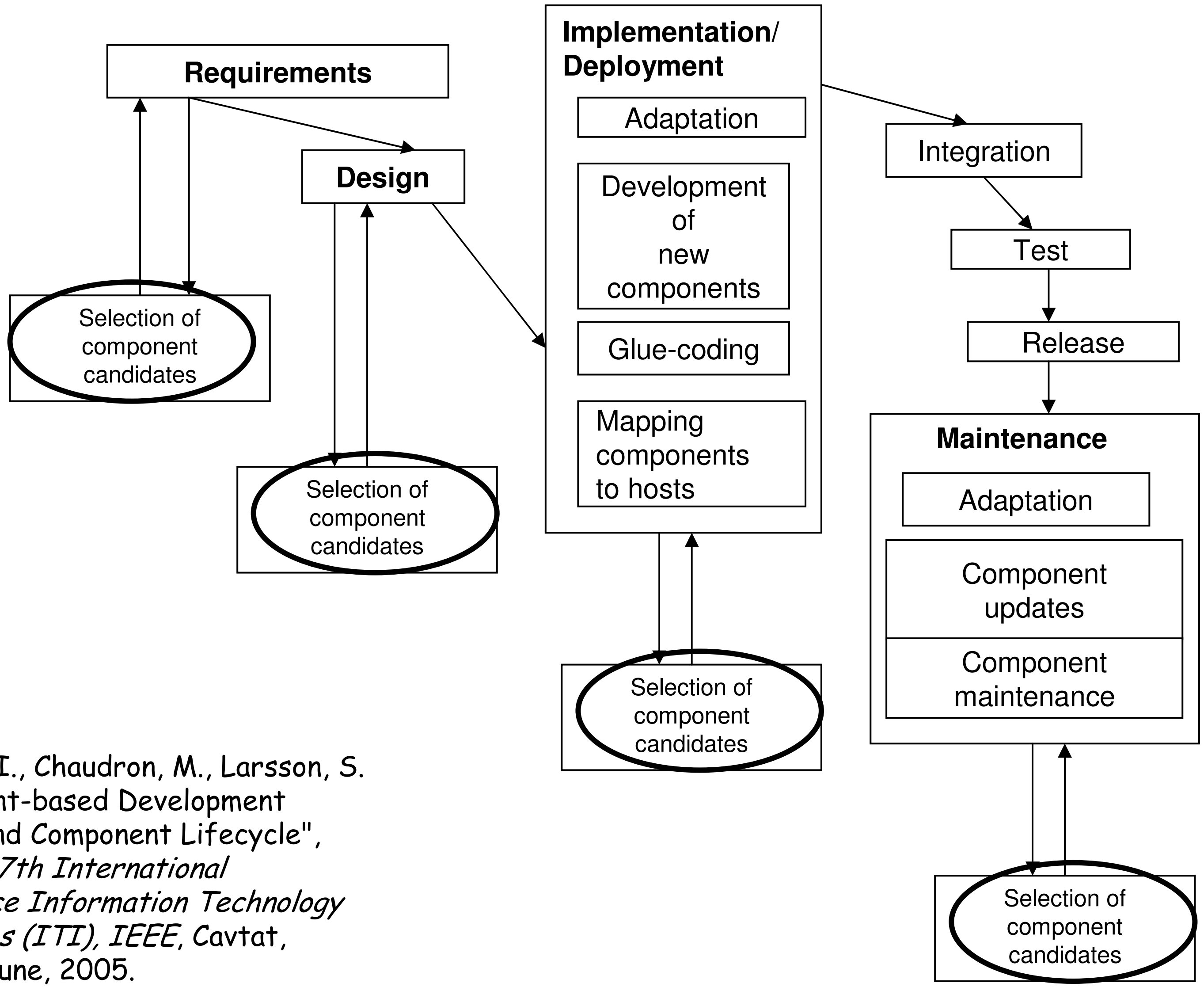
Metaheuristic search techniques, like genetic algorithms, simulated annealing and **tabu search**, to find __**near optimal**__ or good-enough **solutions**

The (class of) problem(s) faced

Studying the **tradeoffs between software cost and other non-functional** attributes (such as reliability and performance) in different phases of the software lifecycle

We have tackled this problem under the development paradigm of **Component-Based Software Engineering** in relationship to the **selection of components**

Selection of components and lifecycle



Crnkovic, I., Chaudron, M., Larsson, S. "Component-based Development Process and Component Lifecycle", *Proc. of 27th International Conference Information Technology Interfaces (ITI), IEEE*, Cavtat, Croatia, June, 2005.

Selection of components and lifecycle

- **Human-based search**
- Usually **based only on functional characteristics** of components
- **Non-functional characteristics** of components make the **quality** of a software product
- SBSE may help to **raise the focus of software engineers** from manual selection to model parameterization
- Efficient model solution techniques may allow to **quickly analyze multiple alternatives**

Selection of components and lifecycle

The problem

Selection of components driven by non-functional properties in Component-based Systems

REQUIREMENTS PHASE

Cost vs satisfaction of requirements

ARCHITECTURAL DESIGN PHASE

Cost vs reliability and delivery time

IMPLEMENTATION/DEPLOYMENT PHASE

Cost vs reliability and completion time

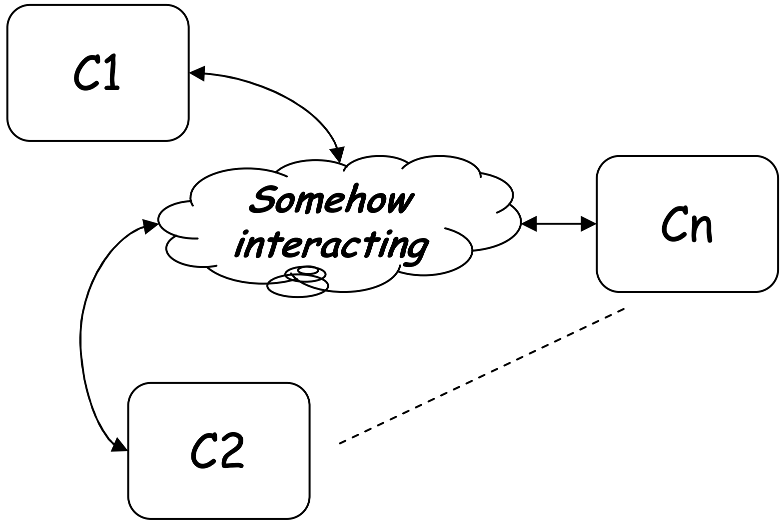
Architectural Design Phase

We have introduced an optimization model that determines **the instance to choose for each component** (either one of the available COTS products or an in-house developed one) in order to minimize the software costs under delivery time and reliability constraints.

In addition, the solution provides **the amount of testing** to be performed on each in-house component in order to achieve a certain reliability level.

Architectural Design Phase

We assume in this phase that the architecture of the software system has been designed



- All functionally equivalent
- They differ for cost and non-functional properties

For each component i :

$\overline{J_i}$ is a set of instances that can be in-house developed

J_i is a set of COTS instances that can be bought

VARIABLES

In general, a "build-or-buy" decisional strategy can be described as a set of 0/1 variables defined as follows ($\forall i = 1,\ldots,n$):

$$x_{ij} = \begin{cases} 1 & \text{if instance } C_{ij} \text{ is chosen } \quad (j \in J_i \;\; or \;\; j \in \bar{J}_i) \\ 0 & \text{otherwise} \end{cases}$$

The variables must fulfill the following constraints:

$$\sum_{j \in J_i \cup \bar{J}_i} x_{ij} = 1, \;\; \forall\, i = 1, \ldots, n$$

For each component i, exactly one instance is either bought as COTS or in-house developed.

COST OBJECTIVE FUNCTION

We express the Cost Objective Function as follows:

$$\sum_{i=1}^{n}\left(\sum_{j\in \bar{J}_i} \overline{c}_{ij}(t_{ij} + \tau_{ij} N_{ij}^{tot}) x_{ij} + \sum_{j\in J_i} c_{ij} x_{ij}\right)$$

Development cost of a in-house instance

For each instance *j* and component *i* let:

$\overline{c}_{ij}$ be the unitary development cost

t_{ij} be the estimated development time

τ_{ij} be the average time required to perform a test case

- Cost of an in-house instance : development cost plus testing cost
- N_{ij}^{tot} (additional variable) : number of tests on in-house instance j of i

Cost Objective Function

We express the Cost Objective Function as follows:

$$\sum_{i=1}^{n}\left(\sum_{j\in \overline{J}_i}\overline{c}_{ij}(t_{ij}+\tau_{ij}N_{ij}^{tot})x_{ij}+\sum_{j\in J_i}c_{ij}x_{ij}\right)$$

Cost of a COTS component

For each instance *j* and component *i* let:

c_{ij} be the buying cost

DELIVERY TIME CONSTRAINT

A maximum threshold T is given on the delivery time of the whole system.

The following expression represents the delivery time of the component i:

$$\sum_{j \in \bar{J}_i} (t_{ij} + \tau_{ij} N_{ij}^{tot}) x_{ij} + \sum_{j \in J_i} d_{ij} x_{ij}$$

Delivery time of an in-house instance.

For each instance j and component i let:

t_{ij} be the estimated development time
τ_{ij} be the average time required to perform a test case

DELIVERY TIME CONSTRAINT

A maximum threshold T is given on the delivery time of the whole system.

The following expression represents the delivery time of the component i:

$$\sum_{j \in \bar{J}_i} (t_{ij} + \tau_{ij} N_{ij}^{tot}) x_{ij} + \sum_{j \in J_i} d_{ij} x_{ij}$$

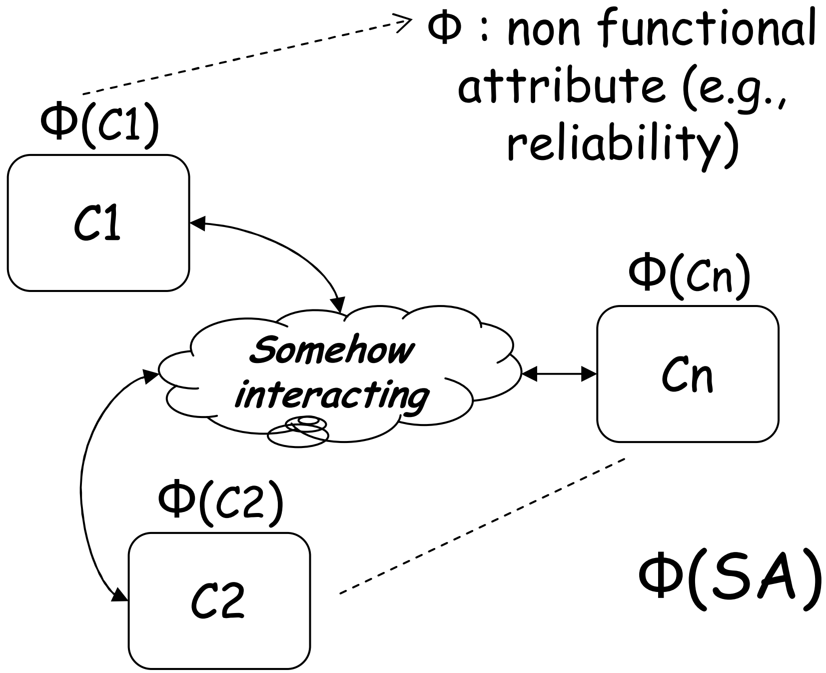
Delivery time of a COTS component

For each instance j and component i let:

d_{ij} be the delivery time

RELIABILITY CONSTRAINT

A minimum threshold *R* is given on the reliability of the whole system.



Software architecture SA

$$\Phi(SA) = \Phi(C1) \otimes \Phi(C2) \otimes \ldots \otimes \Phi(Cn)$$

It is not easy to express the relationships between non-functional characteristics of components and quality of the whole system

RELIABILITY CONSTRAINT

$$REL(SA) = \prod_{i=1}^{n} \Phi_i$$

Jung H.W. et al. "Selecting Optimal COTS Products Considering Cost and Failure Rate", *Proc. of ISSRE*, 1999.

$$\phi_i = e^{-fnum_i}$$

the average number of failures of a component instance

$$fnum_i = \sum_{j \in \bar{J}_i} \theta_{ij} s_i x_{ij} + \sum_{j \in J_i} \mu_{ij} s_i x_{ij}$$

The probability of failure on demand θ_{ij} of the in-house developed instance C_{ij}

Bertolino, A. and Strigini, L.: "On the use of testability measures for dependability assessment", *on Software Engineering, 22(2):97-108*, February, 1996.

$$\theta_{ij} = \frac{Testab_{ij} * p_{ij}(1 - Testab_{ij})^{N_{ij}^{suc}}}{(1 - p_{ij}) + p_{ij}(1 - Testab_{ij})^{N_{ij}^{suc}}}$$

the number of successful (i.e. failure free) tests performed on an in-house instance

OPTIMIZATION MODEL

$$\min \sum_{i=1}^{n} \left(\sum_{j \in \bar{J}_i} \bar{c}_{ij} (t_{ij} + \tau_{ij} N_{ij}^{tot}) x_{ij} + \sum_{j \in J_i} c_{ij} x_{ij} \right)$$

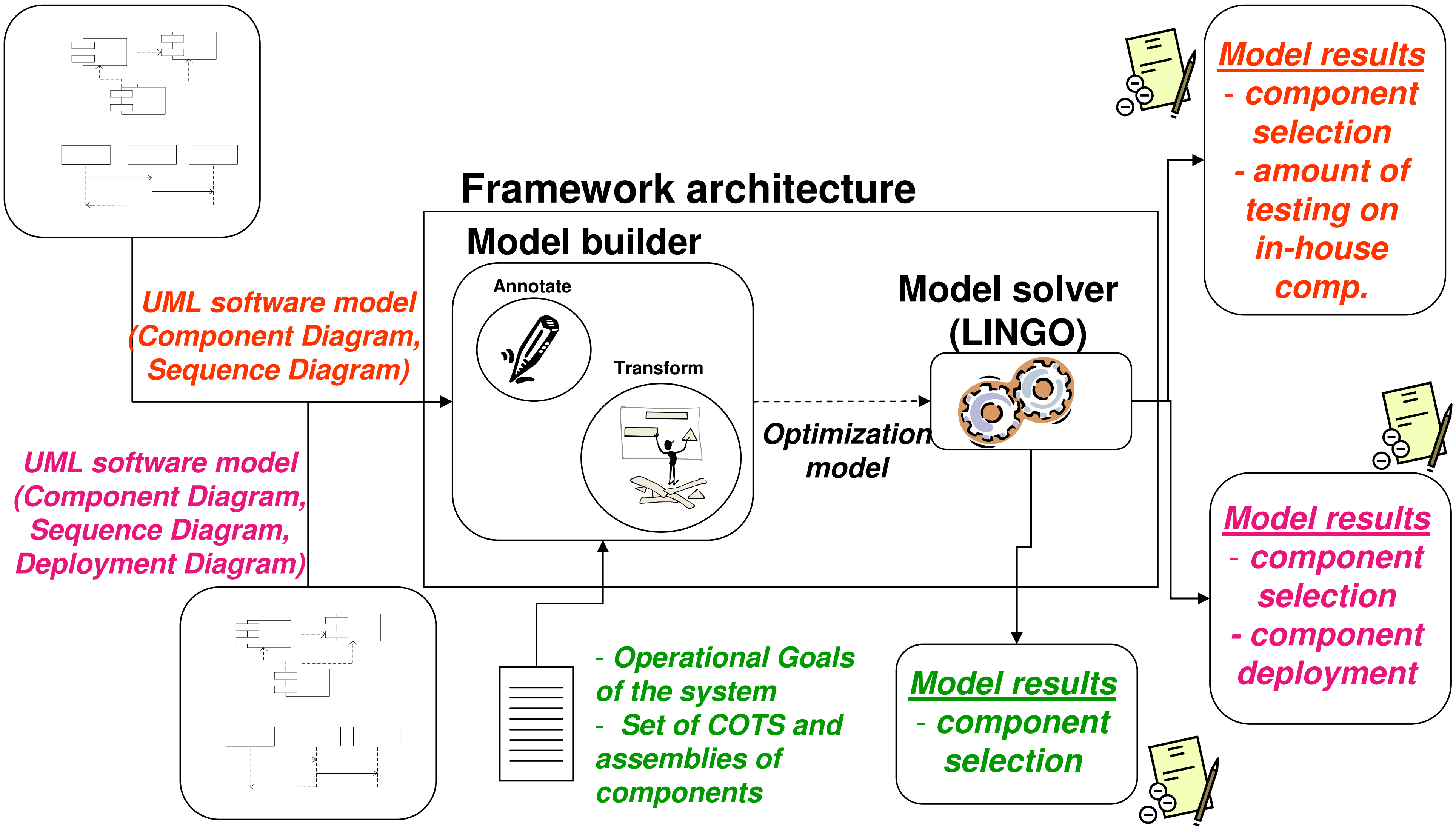
$$\max_{i=1,..,n} \left(\sum_{j \in \bar{J}_i} (t_{ij} + \tau_{ij} N_{ij}^{tot}) x_{ij} + \sum_{j \in J_i} d_{ij} x_{ij} \right) \le T$$

$$\prod_{i=1}^{n} e^{-\left(\sum_{j \in \bar{J}_i} \theta_{ij} s_i x_{ij} + \sum_{j \in J_i} \mu_{ij} s_i x_{ij}\right)} \ge R$$

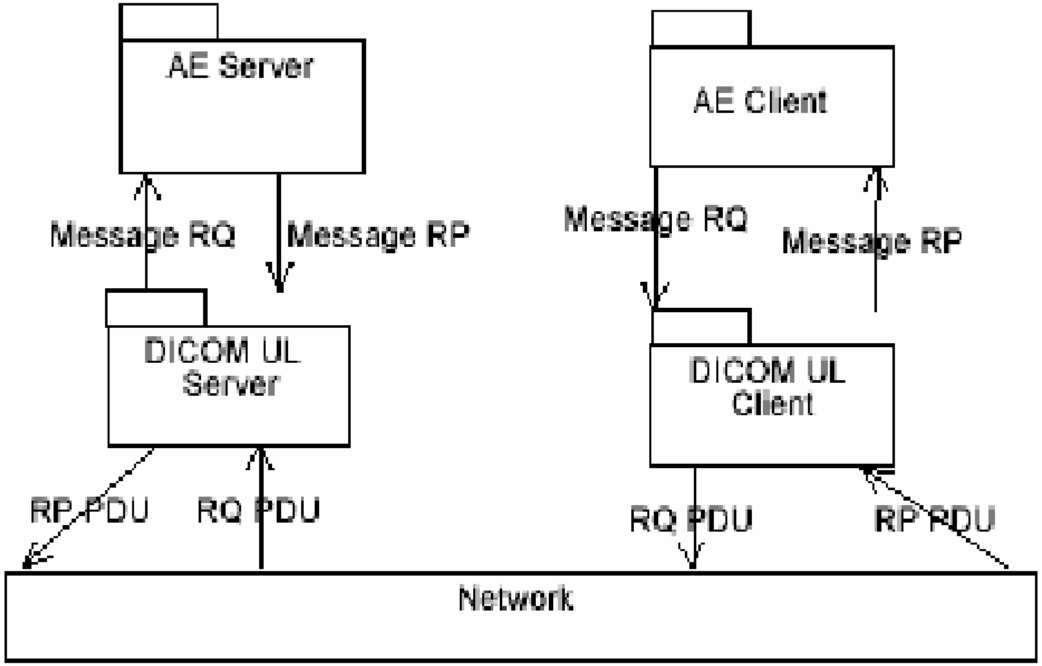
$$\sum_{j \in J_i \cup \bar{J}_i} x_{ij} = 1, \forall i = 1, \ldots, n$$

$$x_{ij} \in \{0,1\}, \forall i = 1, \ldots, n, \forall j \in J_i \cup \bar{J}_i$$

A framework for component selection

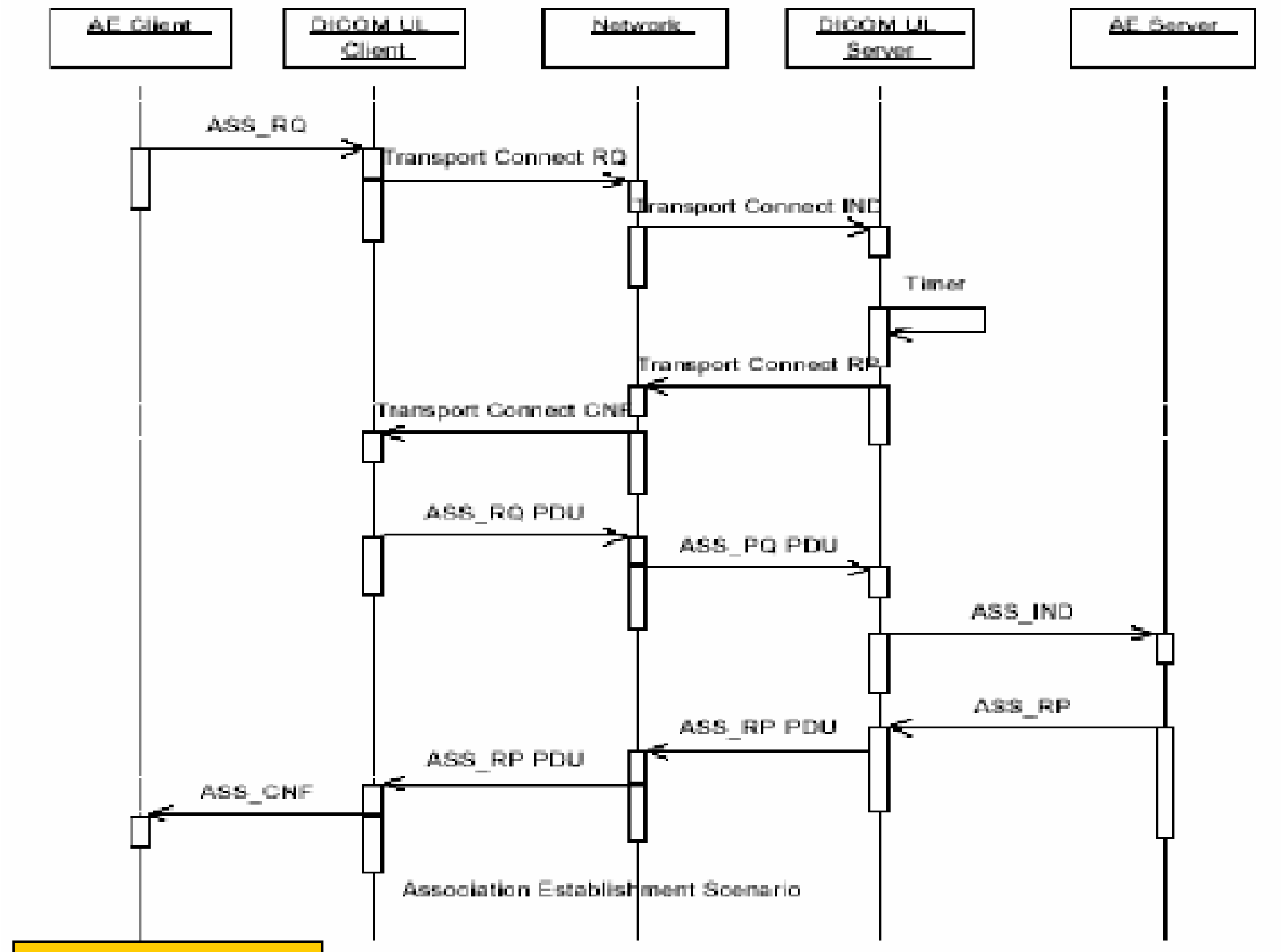


An example: a distributed medical system



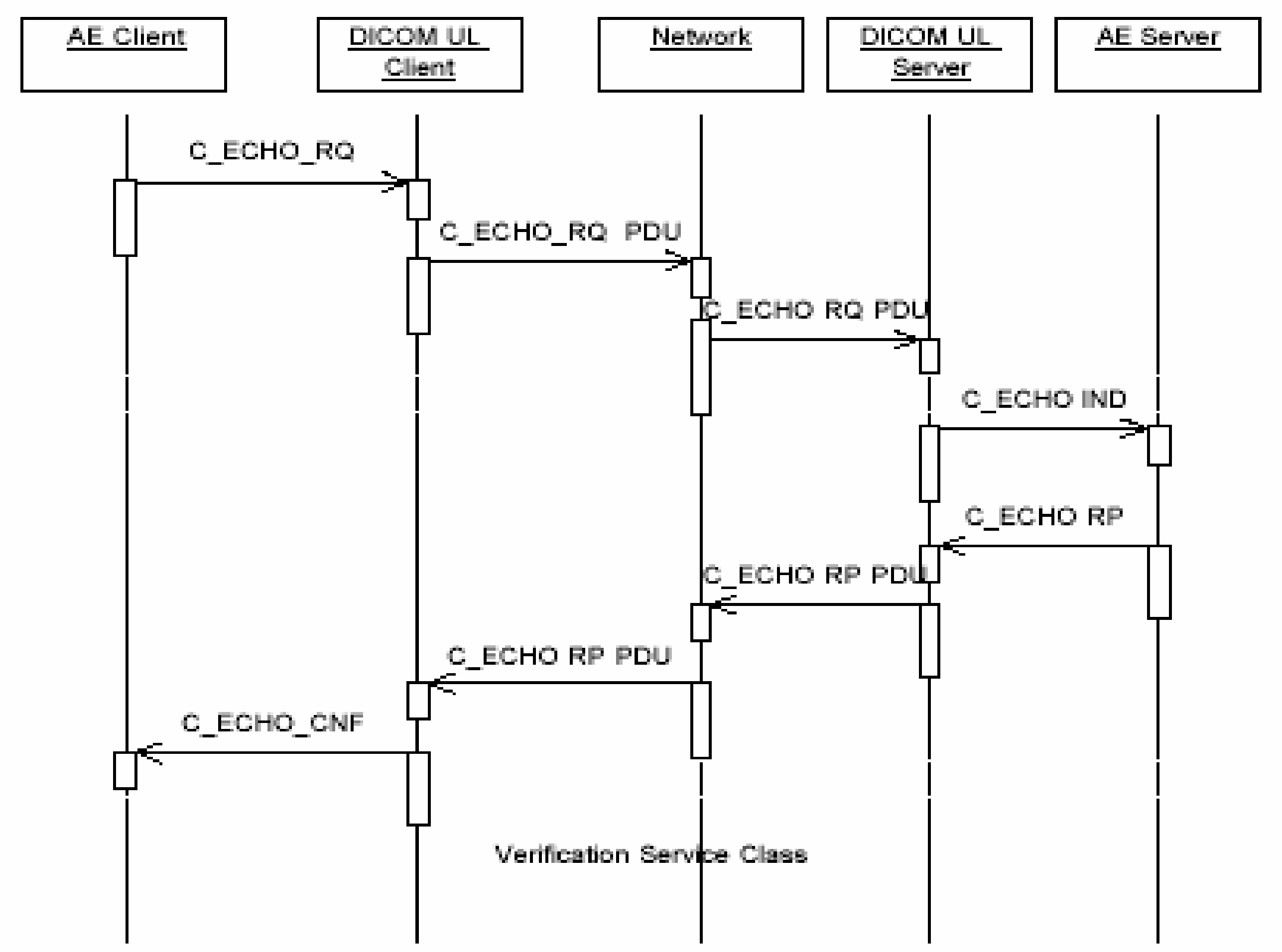
It is a client/server system, where the *AE Client* subsystem is connected via a network (*Network* subsystem) to the *AE Server* subsystem. The communication between the entities of the system is performed using Digital Imaging and Communication in Medicine (DICOM) standard, which is typically used, for example, for producing, processing and exchanging medical images.

An example: a distributed medical system



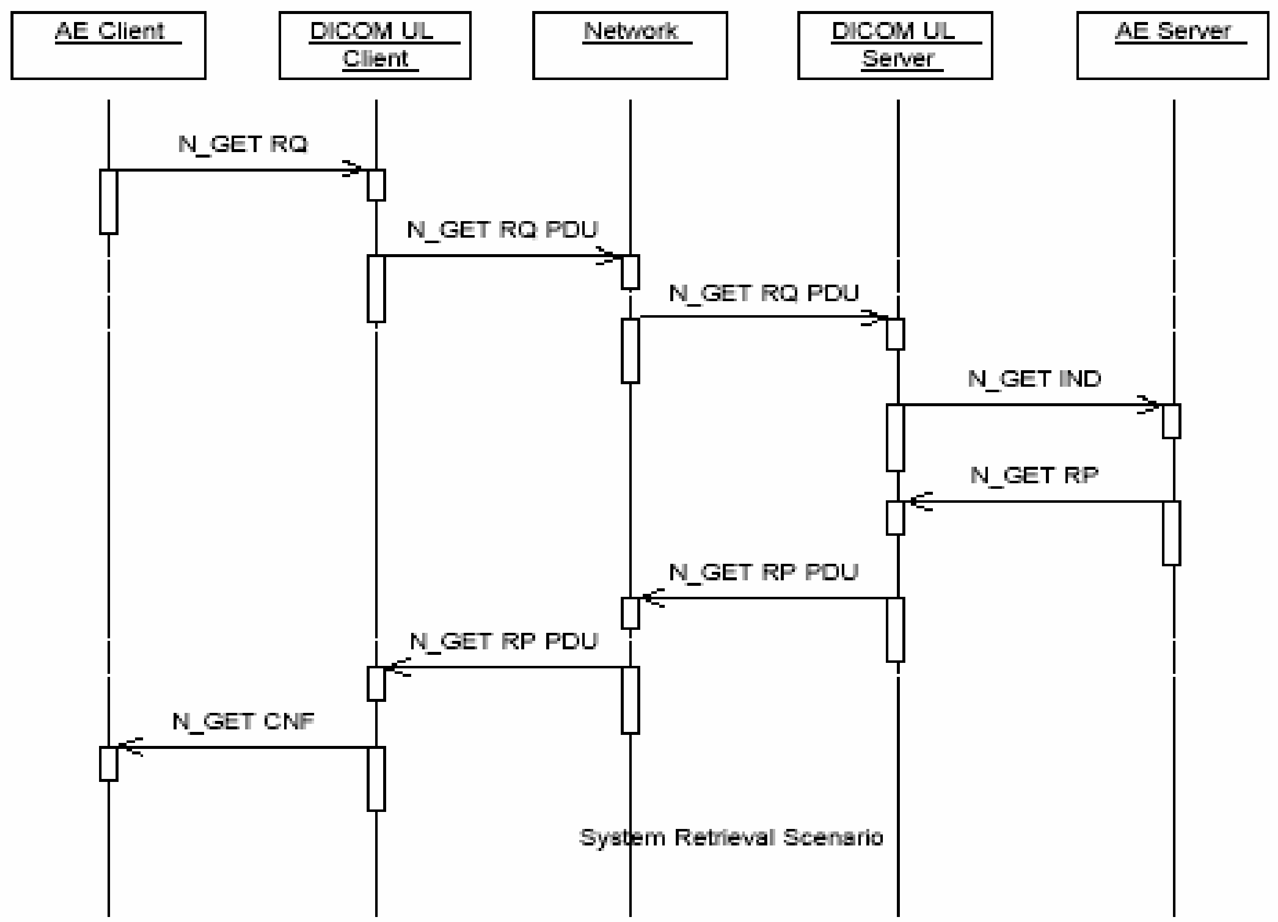
First Scenario

An example: a distributed medical system



Second Scenario

An example: a distributed medical system



Third Scenario

An example: a distributed medical system

	Component name	COTS alternatives	Cost c_{ij}	Average delivery time d_{ij}	Average no. of invocations s_i	Prob. of fail. on demand μ_{ij}
C_1	*AE Client*	C_{11}	14	3	$1.\bar{9}$	0.001
		C_{12}	6	3		0.11
C_2	*DICOM UL Client*	C_{21}	6	4	$2.\bar{3}$	0.009
		C_{22}	12	3		0.001
		C_{23}	14	3		0.0001
C_3	*Network*	C_{31}	12	2	$2.\bar{6}$	0.005
		C_{32}	14	4		0.0003
		C_{33}	15	7		0.0001
C_4	*DICOM UL Server*	C_{41}	5	4	$2.\bar{6}$	0.006
		C_{42}	10	3		0.0002
C_5	*AE Server*	C_{51}	5	3	$0.\bar{9}$	0.004
		C_{52}	10	5		0.0003
		C_{53}	11	5		0.000001
		C_{54}	11	7		0.0001

Parameters for COTS instances

Parameters for in-house developed components

	Component name	Development Time t_{i0}	Testing Time τ_{i0}	Unitary development cost $\bar{c}_{i0}$	Faulty Probability p_{i0}	Testability $Testab_{i0}$
C_1	*AE Client*					
C_2	*DICOM UL Client*	6	0.007	1	0.8	0.006
C_3	*Network*	6	0.007	1	0.8	0.009
C_4	*DICOM UL Server*	3	0.007	1	0.3	0.006
C_5	*AE Server*	4	0.007	1	0.5	0.009

Model Solution

We have solved the optimization model for multiple values of bounds *T* and *R*.
The former spans from 4 to 30 whereas the latter from 0.89 to 0.99.

Model Solution

	R=0.89	R=0.97	R=0.99
T=4	$[C_{11}, C_{21}, C_{31}, (C_{40},0), (C_{50},0)]$, SysRel=0.95539 , Cost=39	$[C_{11}, C_{21}, C_{32}, (C_{40},128), C_{51}]$ SysRel=0.970000, Cost=42.896	$[C_{11}, C_{23}, C_{32}, (C_{40},139), (C_{50},0)]$ SysRel=0.990000, Cost=49.973
T=6	$[C_{11}, (C_{20},0), (C_{30},0), (C_{40},0), (C_{50},0)]$ SysRel=0.95915 , Cost=33	$[C_{11}, C_{22}, (C_{30},0), (C_{40},44), (C_{50},80)]$ SysRel= 0.970008 , Cost=39.868	$[C_{11}, C_{22}, C_{32}, (C_{40},112), (C_{50},137)]$ SysRel=0.99, Cost=48.743
T=12	$[C_{11}, (C_{20},0), (C_{30},0), (C_{40},0), (C_{50},0)]$ SysRel=0.95915, Cost=33	$[C_{11}, (C_{20},0), (C_{30},233), (C_{40},0), (C_{50},0)]$ SysRel=0.970002 , Cost=34.631	$[C_{11}, (C_{20},467), (C_{30},532), (C_{40},129), (C_{50},150)]$ SysRel=0.99, Cost=41.946
T=30	$[C_{11}, (C_{20},0), (C_{30},0), (C_{40},0), (C_{50},0)]$ SysRel=0.95915, Cost=33	$[C_{11}, (C_{20},0), (C_{30},233), (C_{40},0), (C_{50},0)]$ SysRel=0.970002, Cost=34.631	$[C_{11}, (C_{20},488), (C_{30},459), (C_{40},148), (C_{50},164)]$ SysRel=0.990002, Cost=41.813

Optimal solution for all previous cases

(i.e. selection of components and number of tests)

Model-driven tradeoff analysis

1. Optimization models for cost/quality tradeoffs

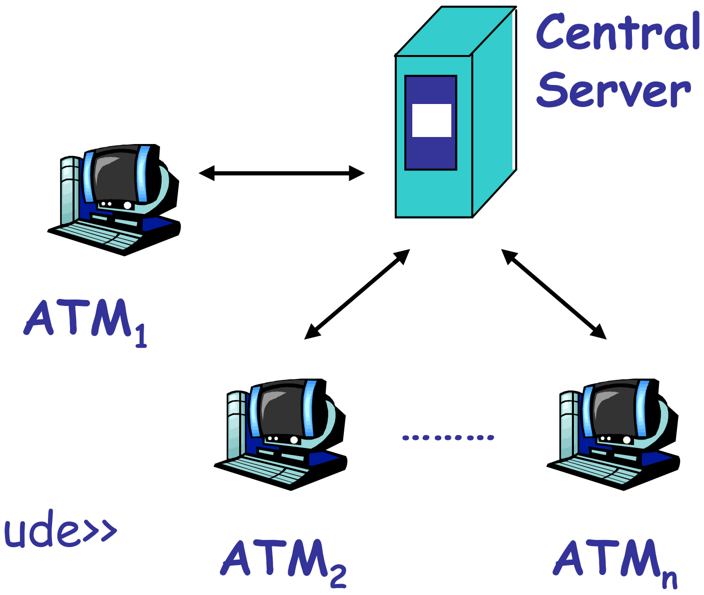
2. **A compositional approach to security/performance tradeoff**

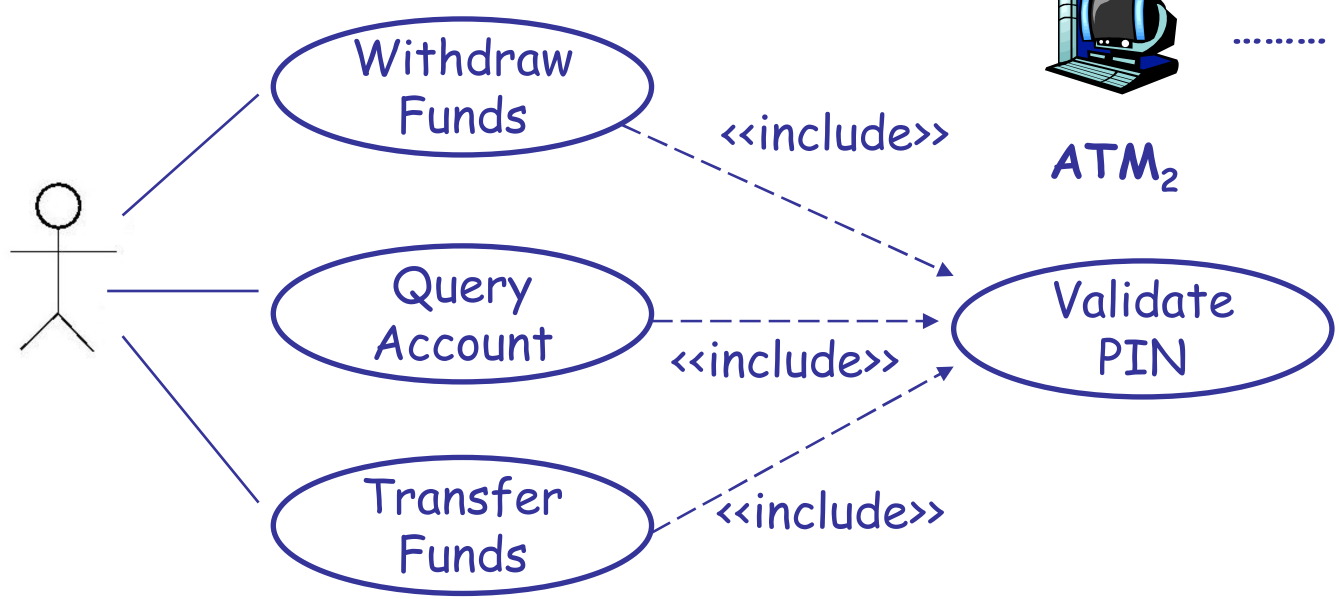
A compositional approach to security/performance tradeoff

Do we have time for it now?

Experimentation:
Application Model

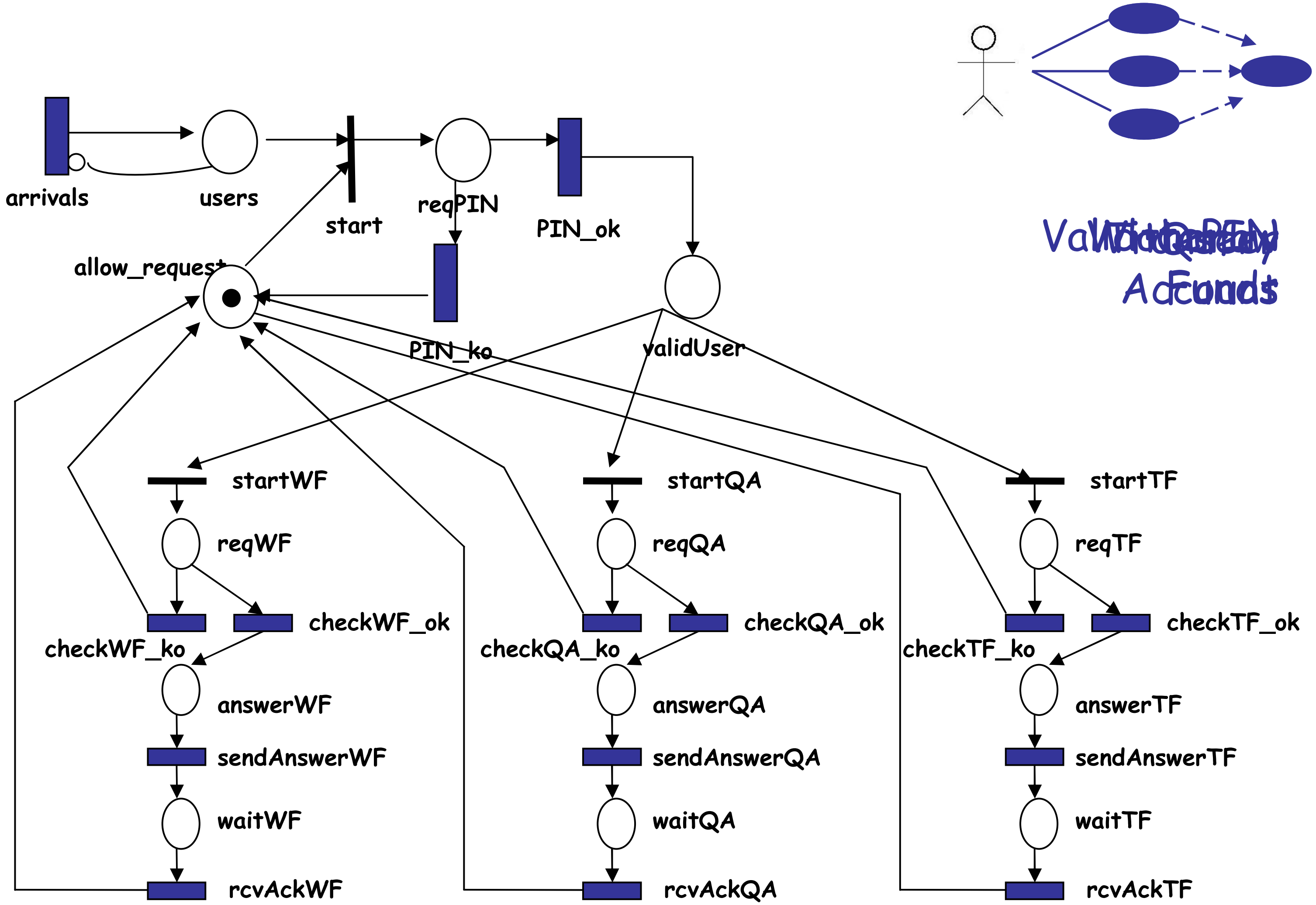
- Banking System





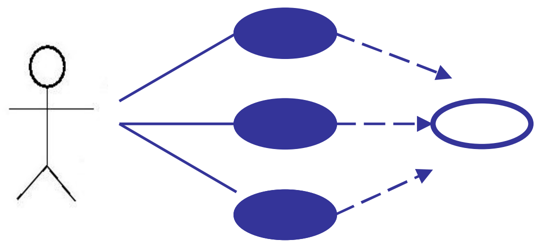
Experimentation:

Application Performance Model

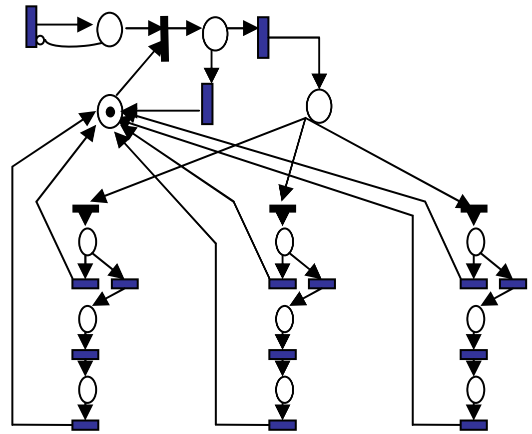


Experimentation: towards a *Application Security Performance Model*

- Application Model



- Application Performance Model

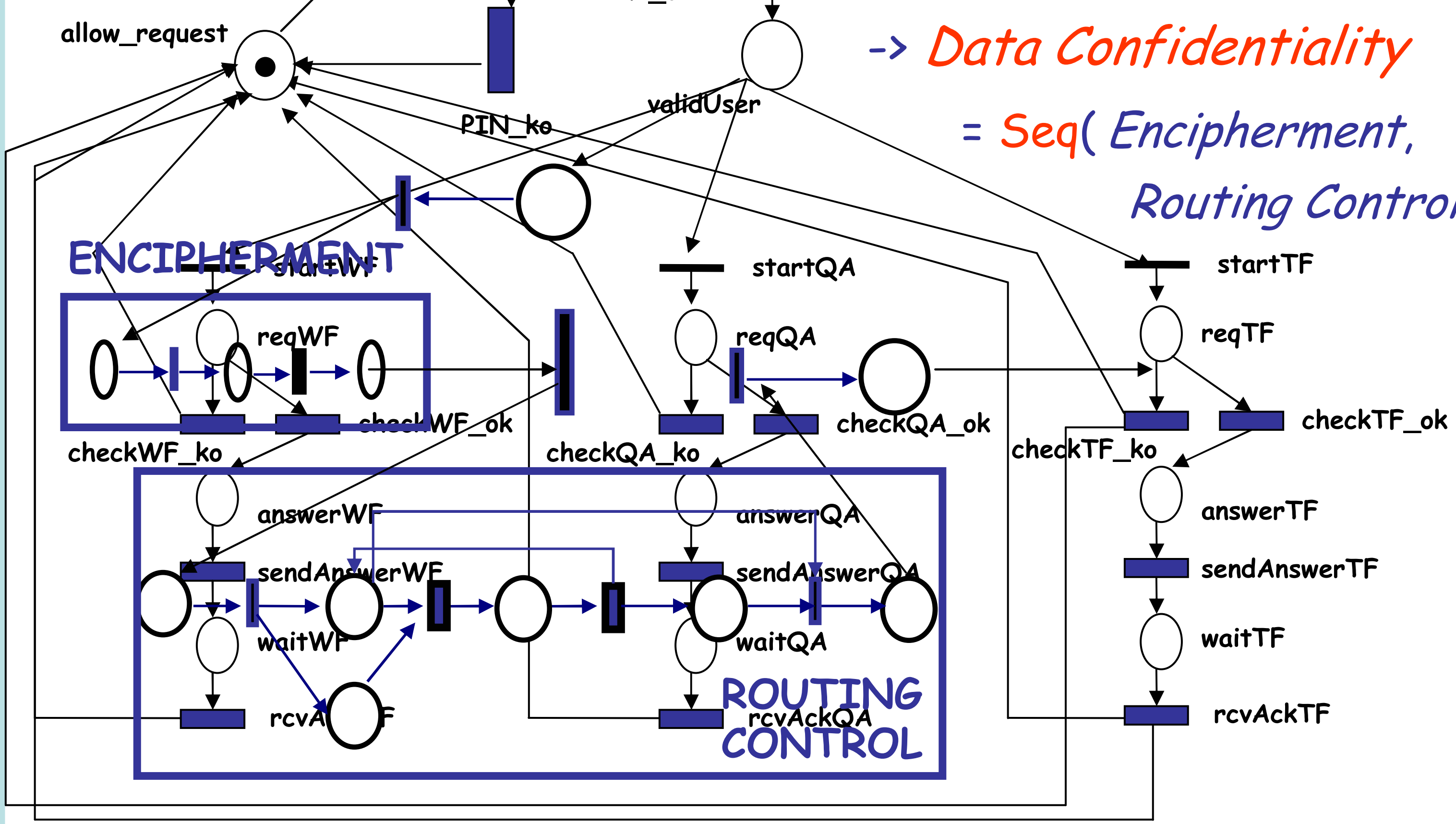


...and which ***Security Solutions*** ?

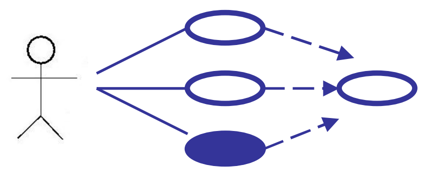
- Withdraw Funds -> *Availability*
- Query Account -> *Access Control*
- Transfer Funds -> *Data Confidentiality*

Experimentation:

Application Security Performance Model



Transfer Funds



-> Data Confidentiality

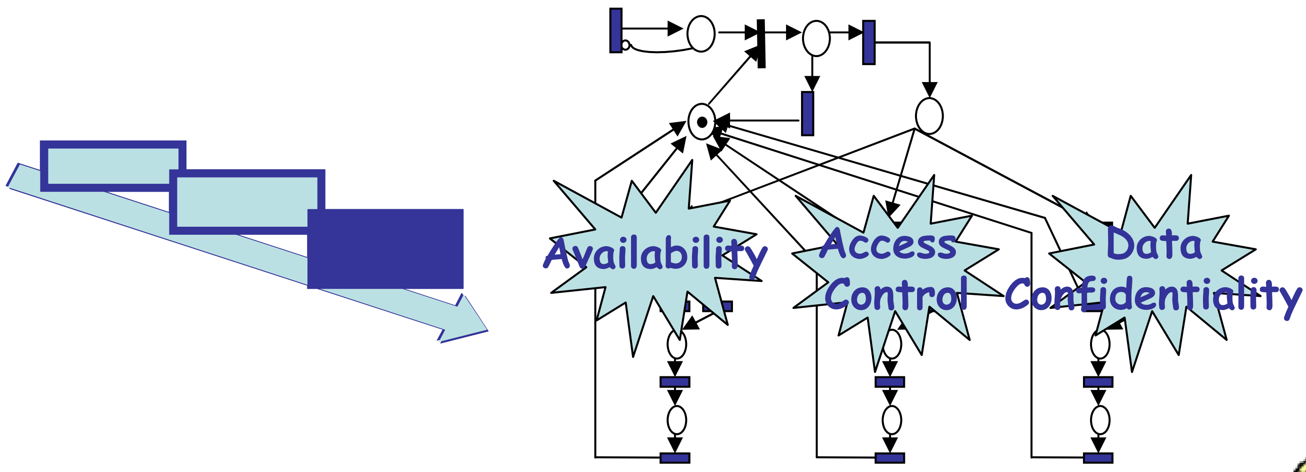
= Seq(Encipherment, Routing Control)

Experimentation

❑Let us compare two configurations:

1. Banking System

   (i.e. the *Application Performance Model*)

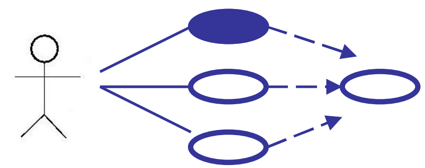


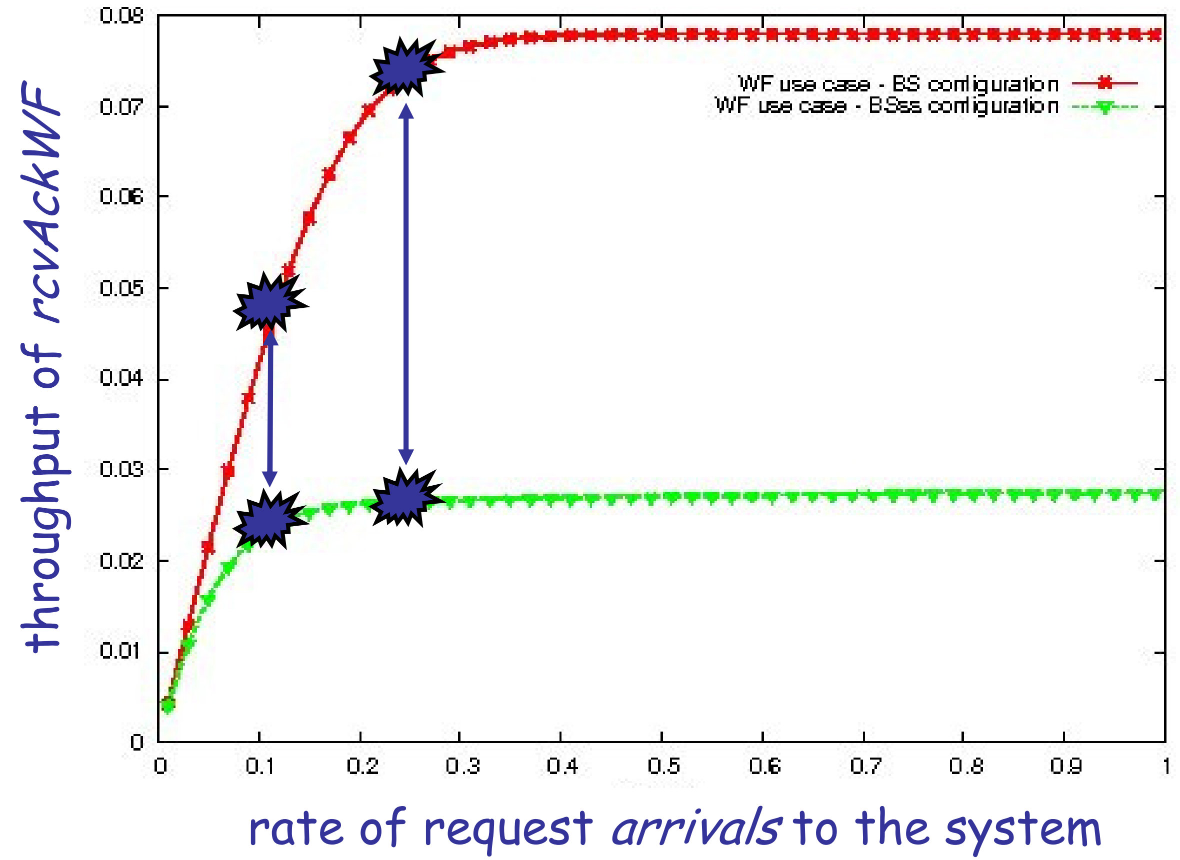
2. Banking System with Security Services

   (i.e. the *Application Security Performance Model*)

Experimental results

- Throughput Analysis (in WF use case)

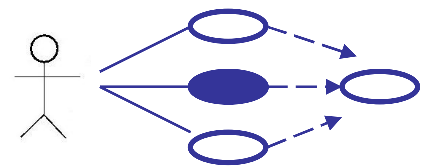


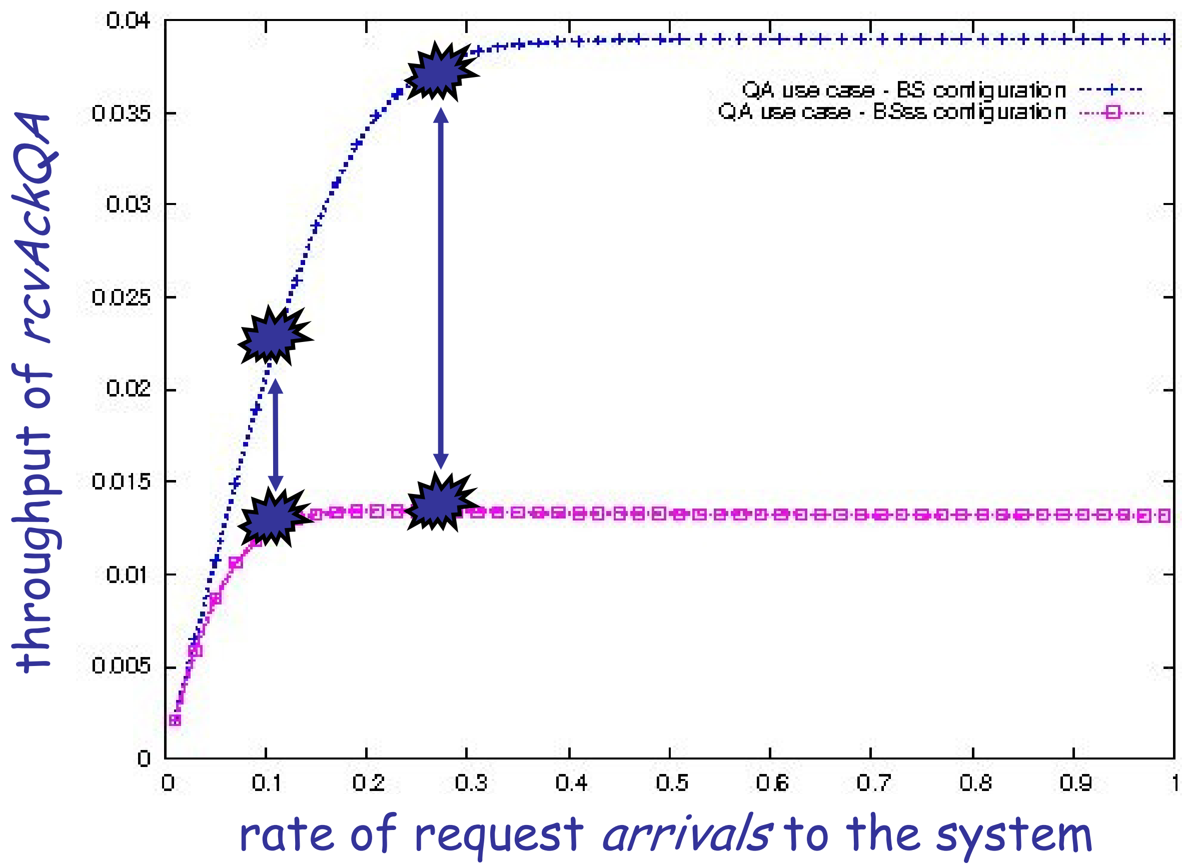


Withdraw Funds

Experimental results

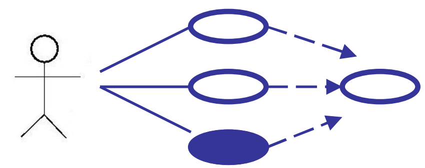
- Throughput Analysis (in QA use case)

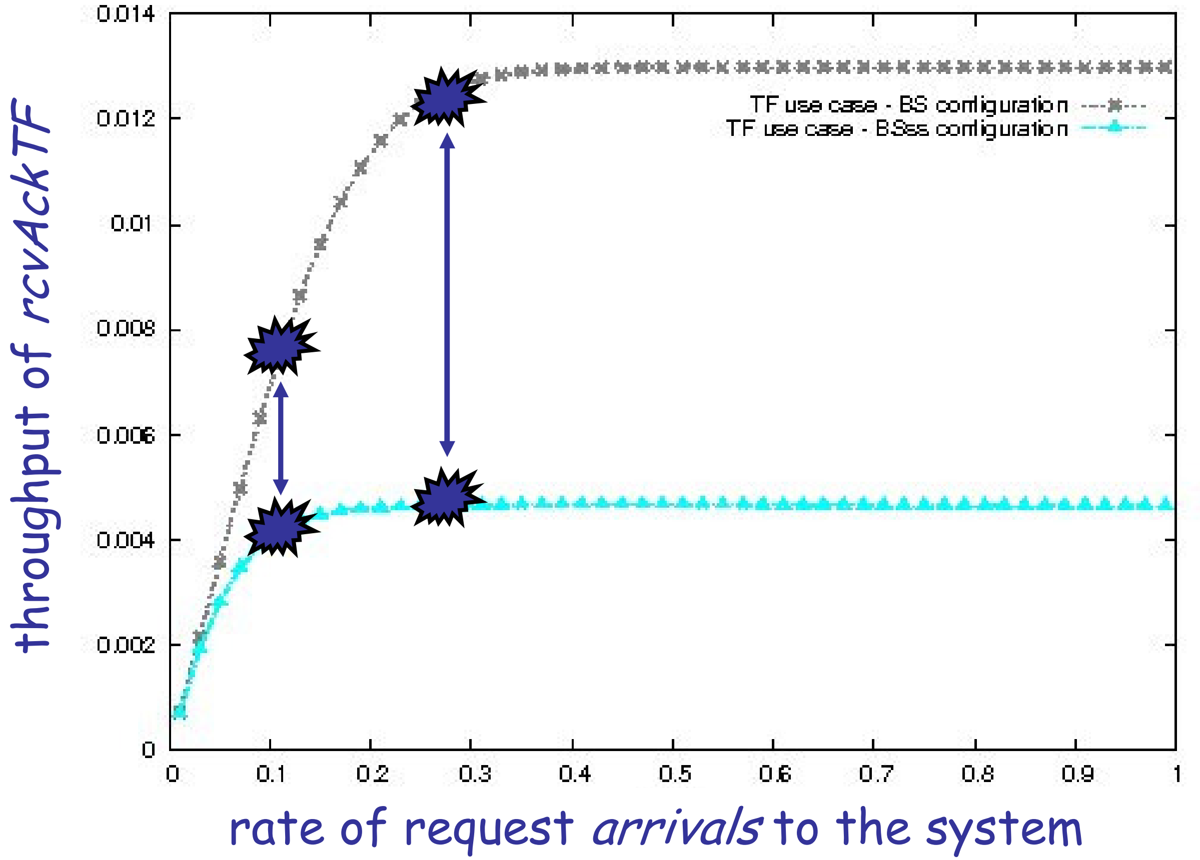




Experimental results

- ❑ Throughput Analysis (in TF use case)

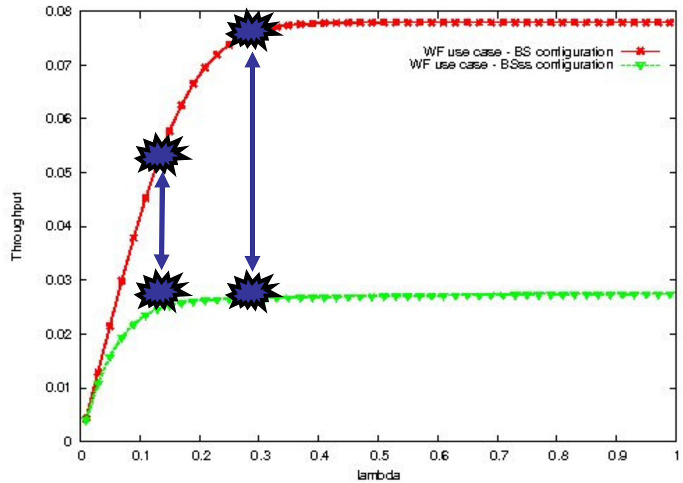




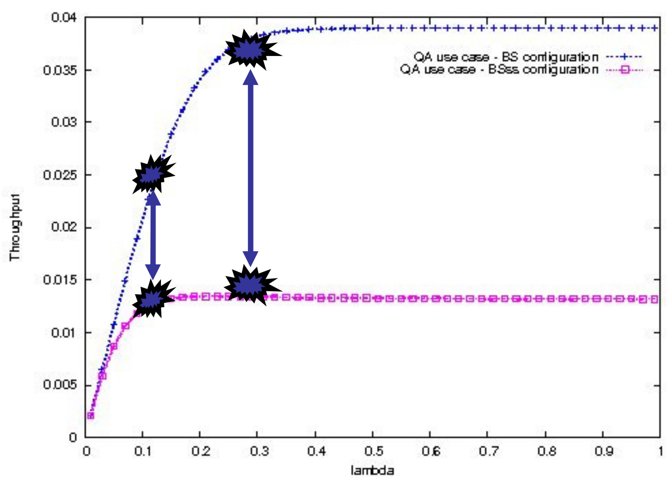
Experimental results

- Throughput Analysis (in use cases)

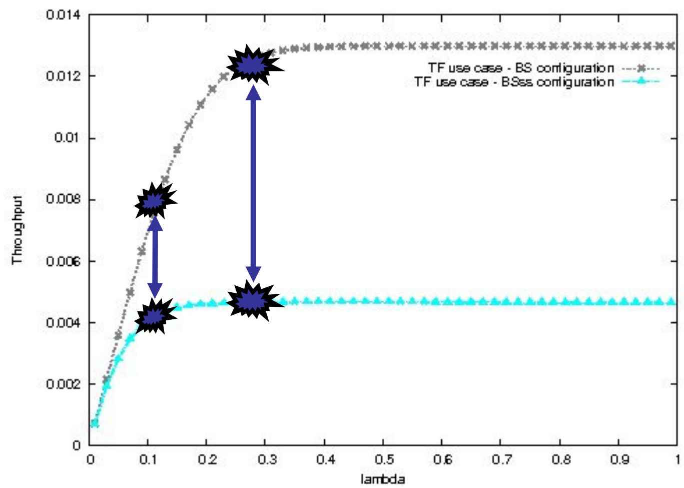
Withdraw Funds



Query Account



Transfer Funds



SUMMARY

- Non-functional analysis and software development
- Model-driven performance analysis
- Model-driven reliability analysis
- Model-driven tradeoff analysis
- Tool support
- Research perspectives

Our "zoo" of tools

http://sealabtools.di.univaq.it/SeaLab/index.html

- **MOSQUITO**
  MOdel driven conStruction of QUeuIng neTwOrks

- COBRA
  COmponent-Based Reliability Analysis

- DEER
  DEcision support for componEnt-based softwaRe

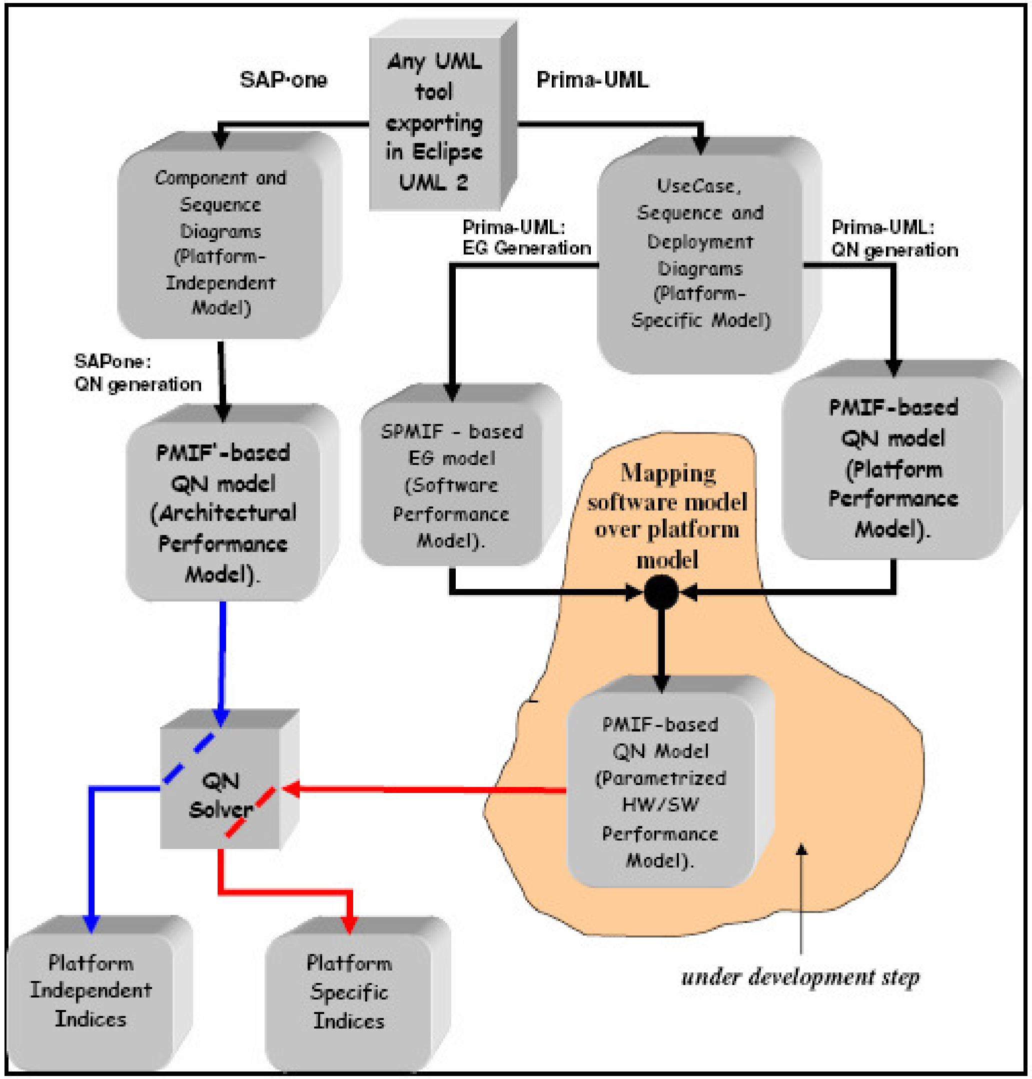
- **WEASEL**
  Web sErvice for Analyzing queueing networkS with multiplE soLvers

… and other "animals" yet to come

- SEAL
  Software architEcture uml modeling of Aemilia specification tooL

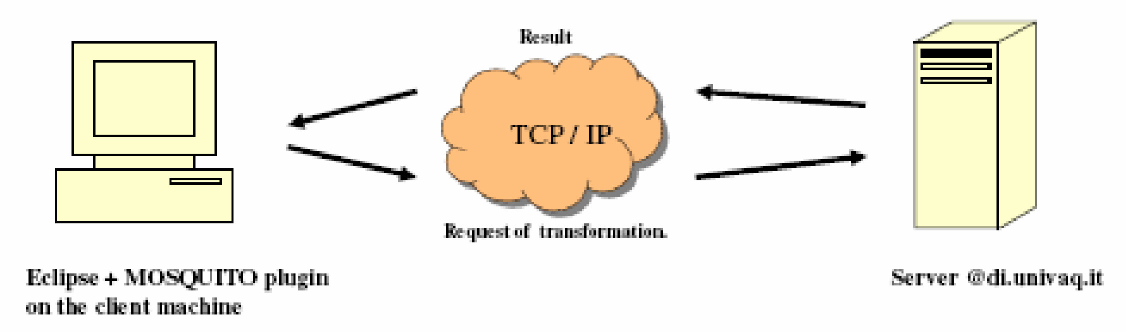
- GARFIEL
  Generation of ARchitectural Feedback from analysIs of pErformance resuLts

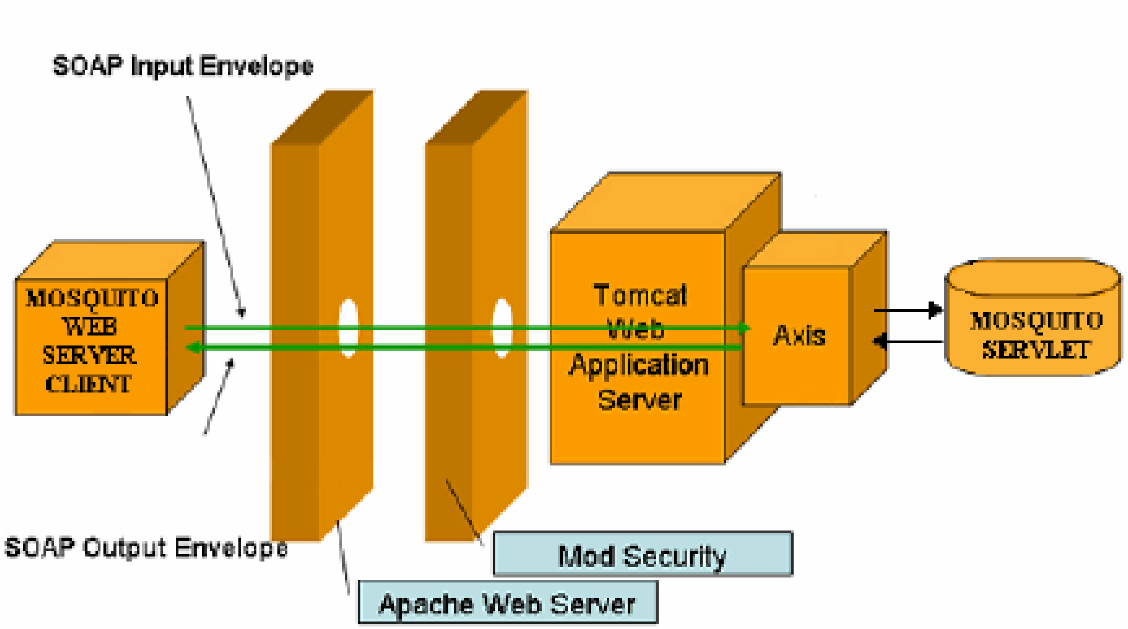
MOSQUITO



MOSQUITO is a tool that generates Queueing Network and Execution Graph models from annotated UML models according to the SAP·one and the Prima-UML methodologies.

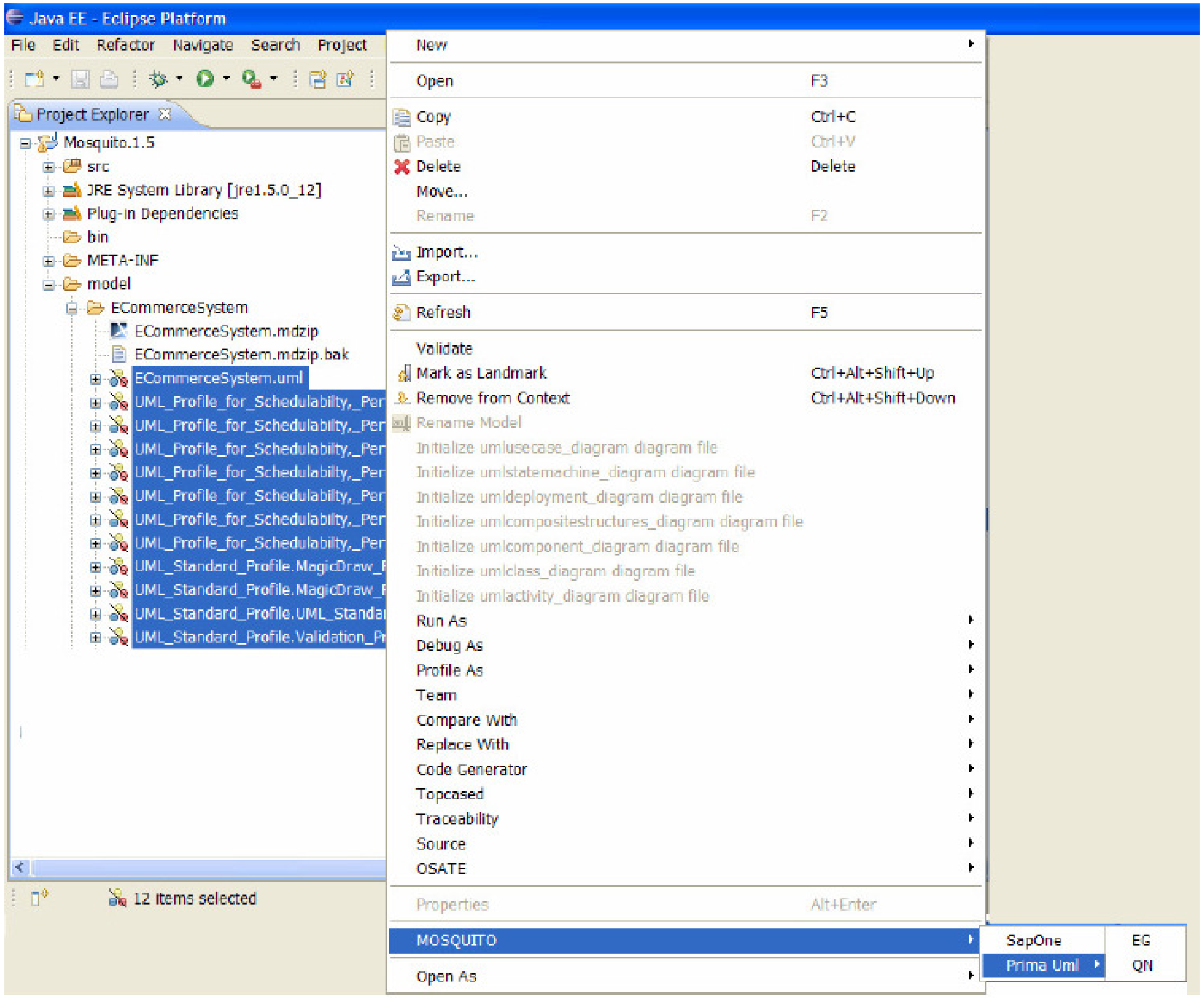
MOSQUITO





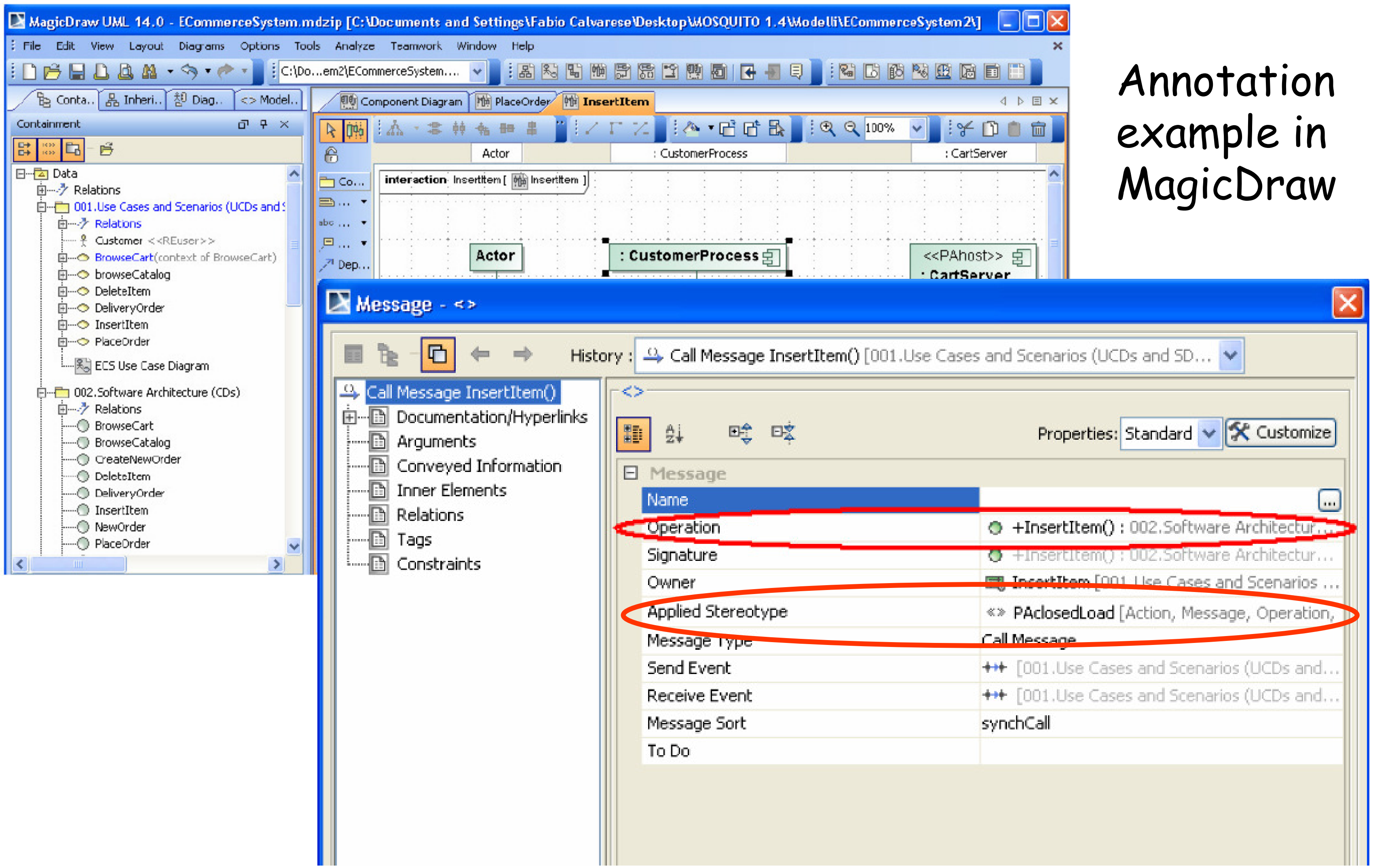
MOSQUITO has a client/server architecture.
A MOSQUITO downloadable plugin implements the client side that provides the functionality to invoke the web services provided on the server side at University of L'Aquila.

MOSQUITO



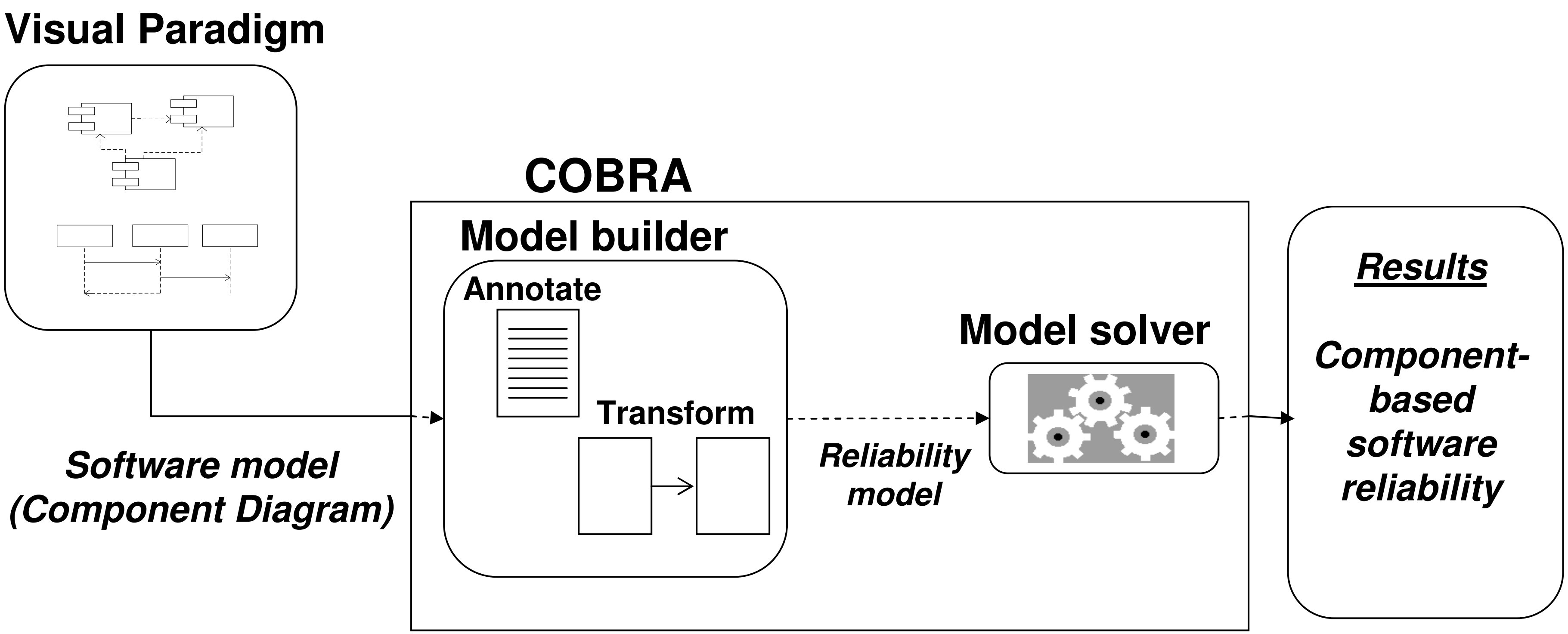
The extension point used for **MOSQUITO** interface is the Eclipse popup menu. After the creation of a model the user must only select the UML files (*.uml) and activate the desired transformation

MOSQUITO



Annotation example in MagicDraw

COBRA



COBRA is a tool that generates (and solves) reliability models for component-based software from annotated UML models

COBRA

The most recent reliability model we had as a target:

- is based on failure probability of software components
- includes error propagation
- does not consider connector and platform failures

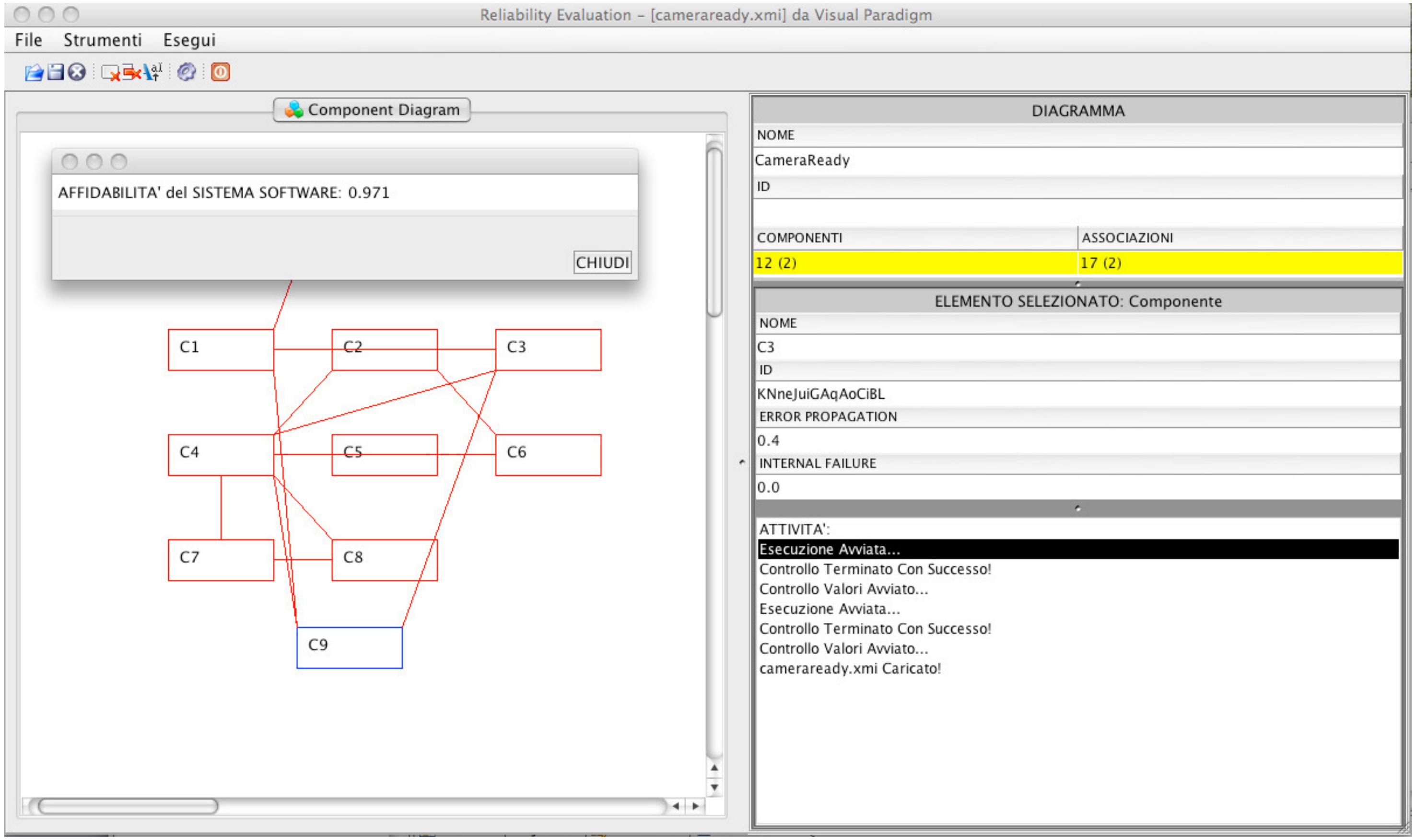
$err^{(k)}(i,j)$: probability that the application reaches comp. *j* after *k* control transfers, starting from comp. *i*, and *j* produces an erroneous output

$$err^{(k)}(i,j) = p^{(k)}(i,j)\cdot if(j) + ep(j)\cdot(1-if(j))\sum_{h=0}^{C} err^{(k-1)}(i,h)\,p(h,j)$$

$\mathbf{e}$: vector of the probabilities that the application (for each possible initial component) produces an erroneous output

$$\mathbf{e} = (\mathbf{I}-\mathbf{Q})^{-1}\cdot\mathbf{F}\cdot(\mathbf{I}-\mathbf{Q}\cdot\mathbf{R}\cdot((\mathbf{I}-\mathbf{F}))^{-1}\cdot\mathbf{c}$$

COBRA

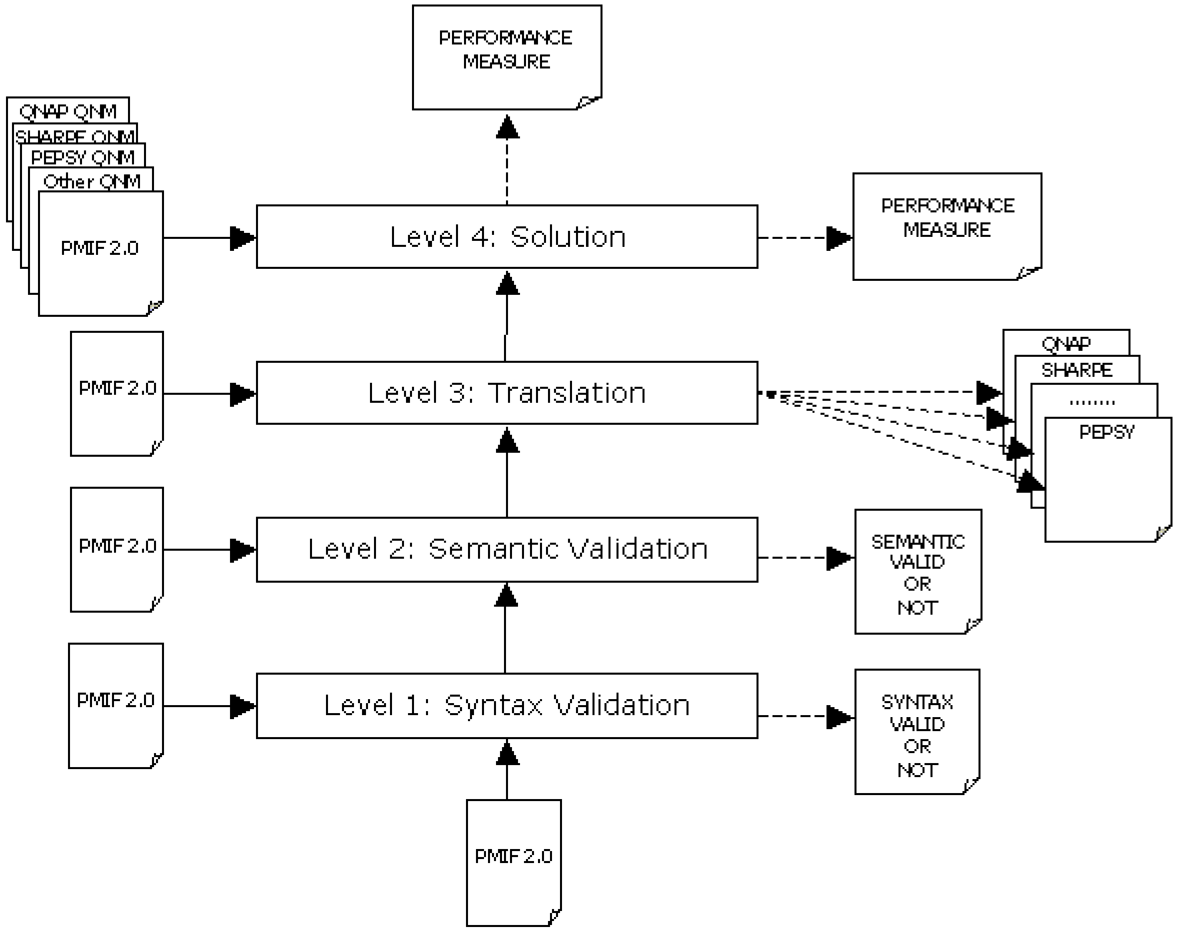


COBRA is an Eclipse plugin and provides a GUI to annotate UML models (at the moment imported from Visual Paradigm) and invoke the reliability model solution

DEER

DEER is a tool embedding (3 types of) transformations from UML and non-UML models to optimization models for supporting software component selection

WEASEL



WEASEL basically offers a Web Service that allows to solve Queueing Network models, specified in PMIF format, by invoking multiple external solution tools

WEASEL

- Up to now WEASEL provides input to **the following solvers**: QNAP, SHARPE, PDQ, PEPSY, MQNA1/MQNA2, PMVA, MVA-QFP, OPENQN/CLOSEDQN.

- Its architecture is **open to easily add the capability to interact with other solvers**. WEASEL is fully customizable and extensible, as **users can add new tools** and capabilities.

- WEASEL follows a simple client-server approach to accomplish this scope, **based on the web service standard SOAP protocol and WSDL specification language**.

WEASEL

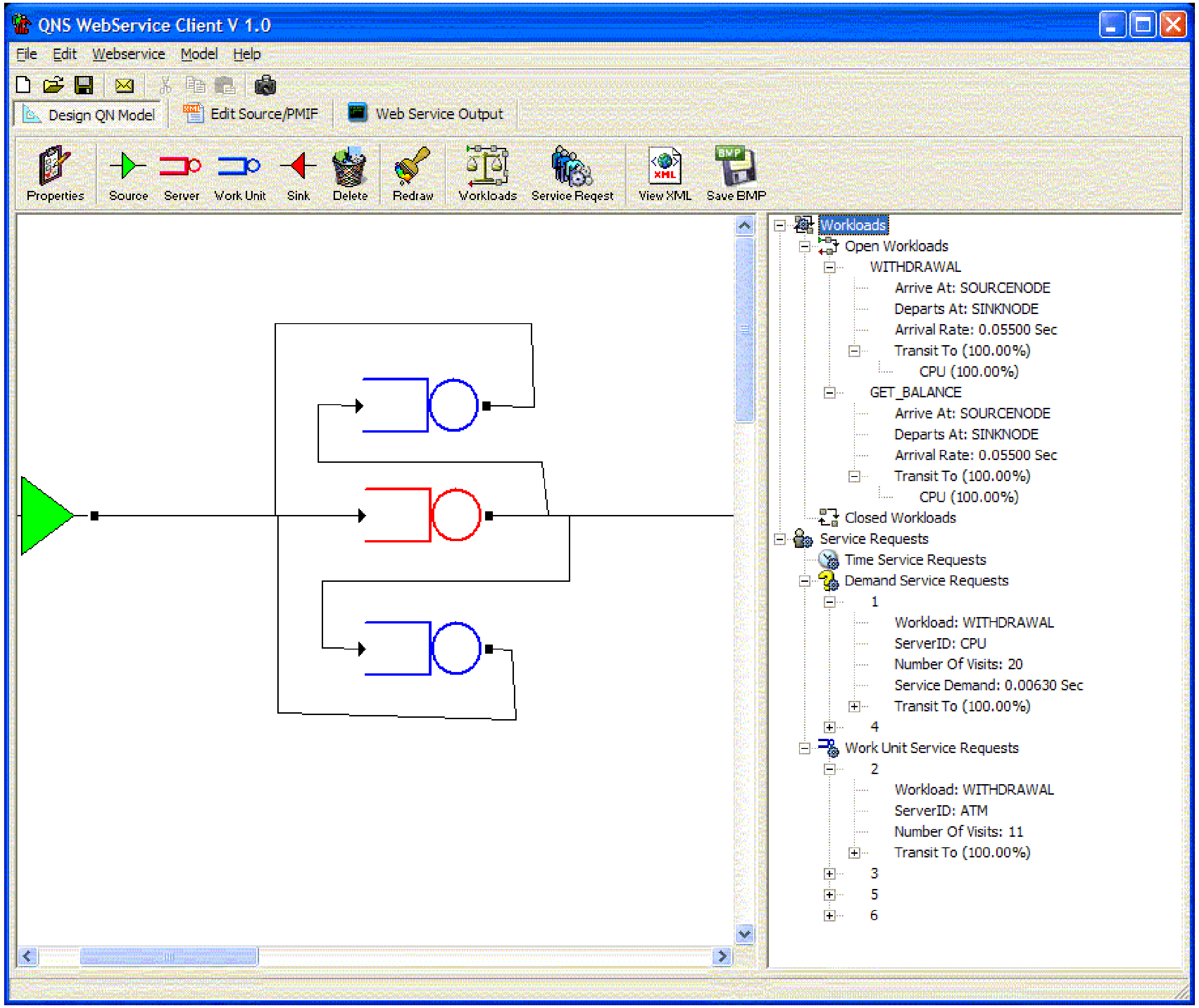
End users can develop their own web clients and implement them with any programming language that provides SOAP support.

Currently available WEASEL clients

- PHP-based command line interface
- AJAX (Asyncronous Javascript And Xml) – based web interface
- Delphi-based Windows application (**PMIF Editor**)

WEASEL

PMIF Editor screenshot



SUMMARY

- Non-functional analysis and software development
- Model-driven performance analysis
- Model-driven reliability analysis
- Model-driven tradeoff analysis
- Tool support
- Research perspectives

Research perspectives

- Non-functional **attributes** may be (strictly) **correlated**, so sophisticated models to **capture dependencies** are needed
- **Conflicts** between **non-functional and functional** analysis **results**

- **Automated interpretation** of analysis results and **feedback generation**
- **Keeping aligned software models and analysis models** (in both directions)

- Introducing an **unified representation of non-functional attributes** to favor tool interoperability

- **Role of pivot languages** in all previous problems

A new direction: Model synchronization and differences