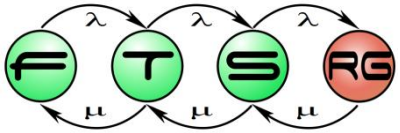
Standard platforms: Web services (and dependability)

László Gönczy

Dept. of Measurement and Information Systems



Content

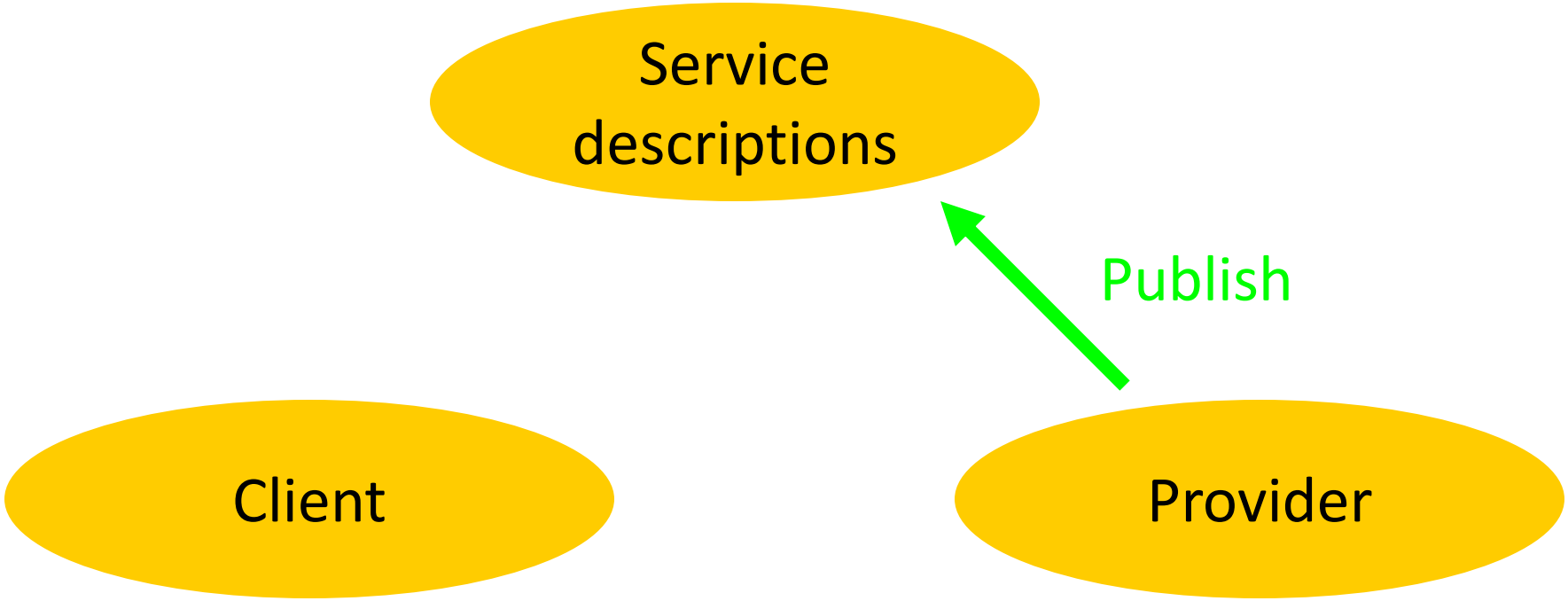
- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Web service architecture

- One possible way of „SOA”

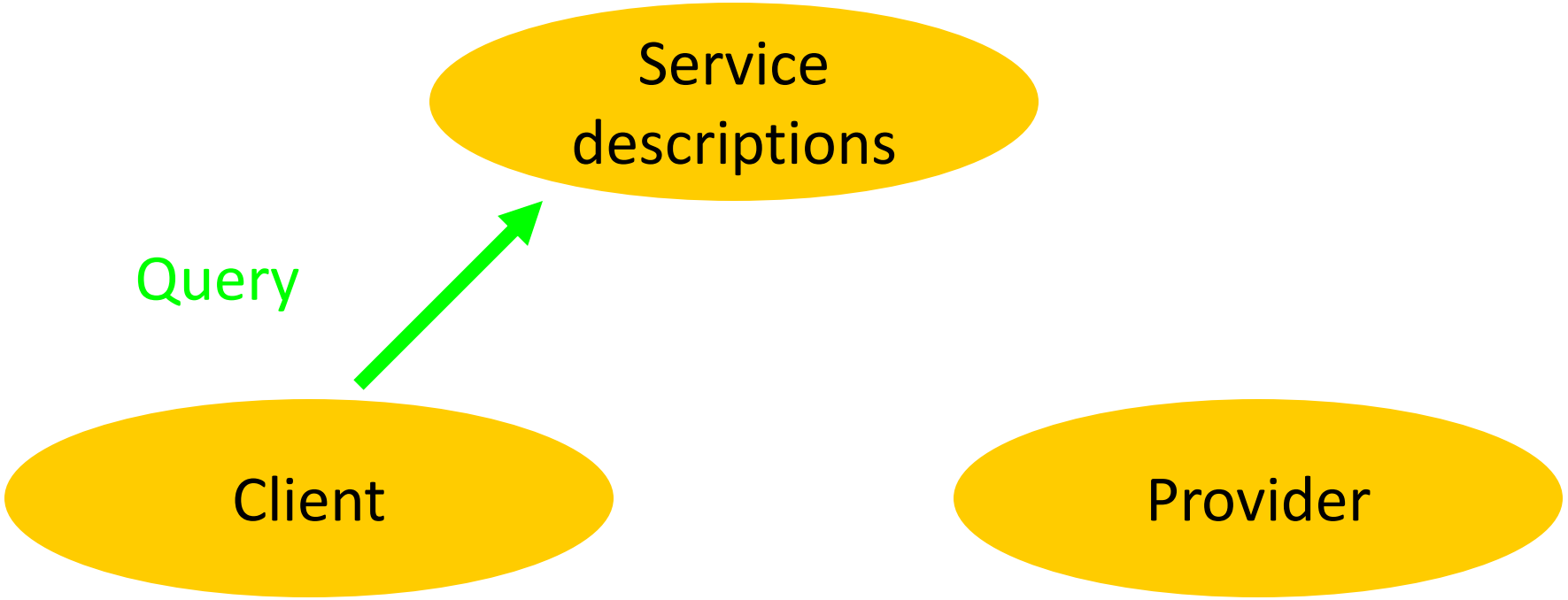
Web service architecture

- One possible way of „SOA”



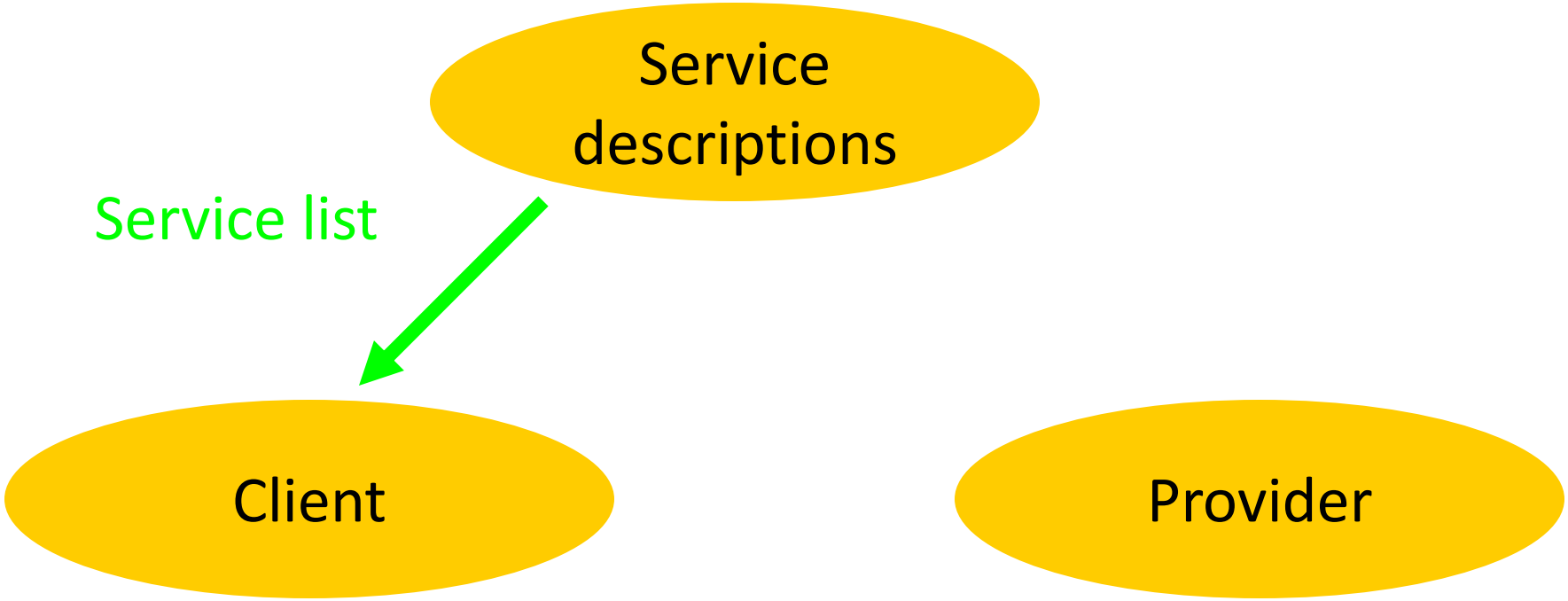
Web service architecture

- One possible way of „SOA”



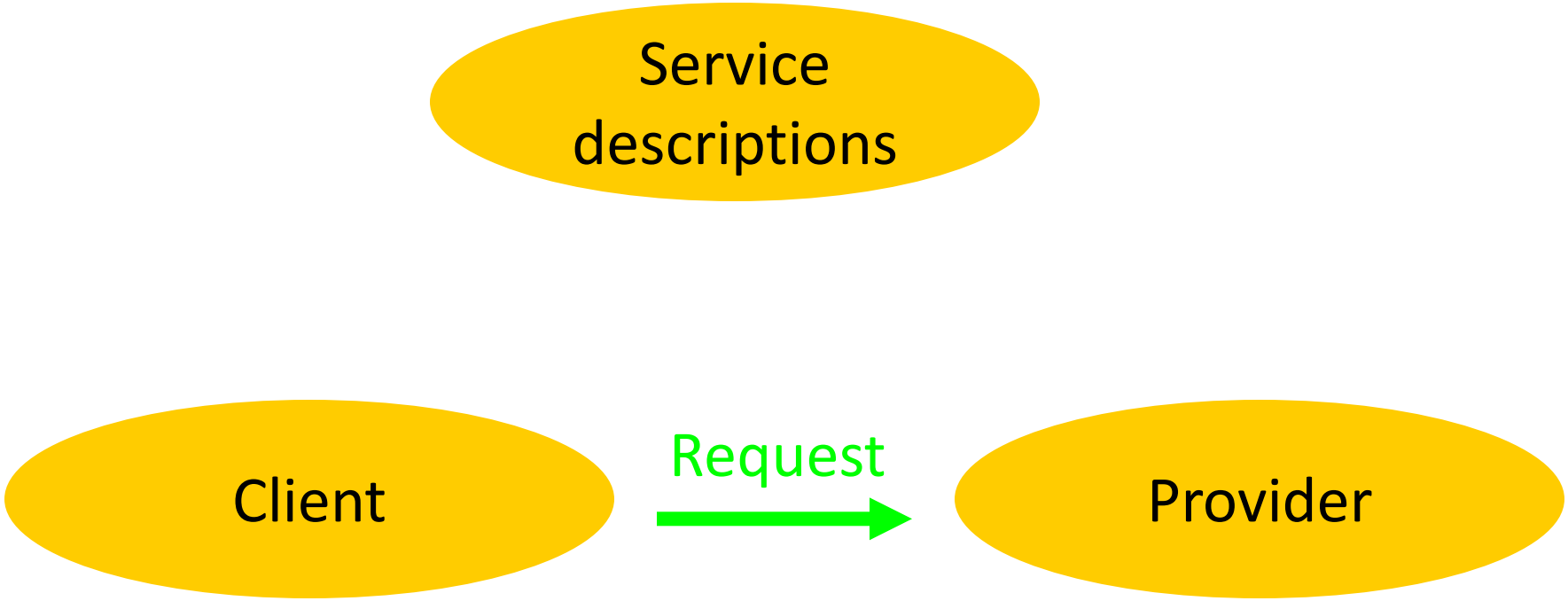
Web service architecture

- One possible way of „SOA”



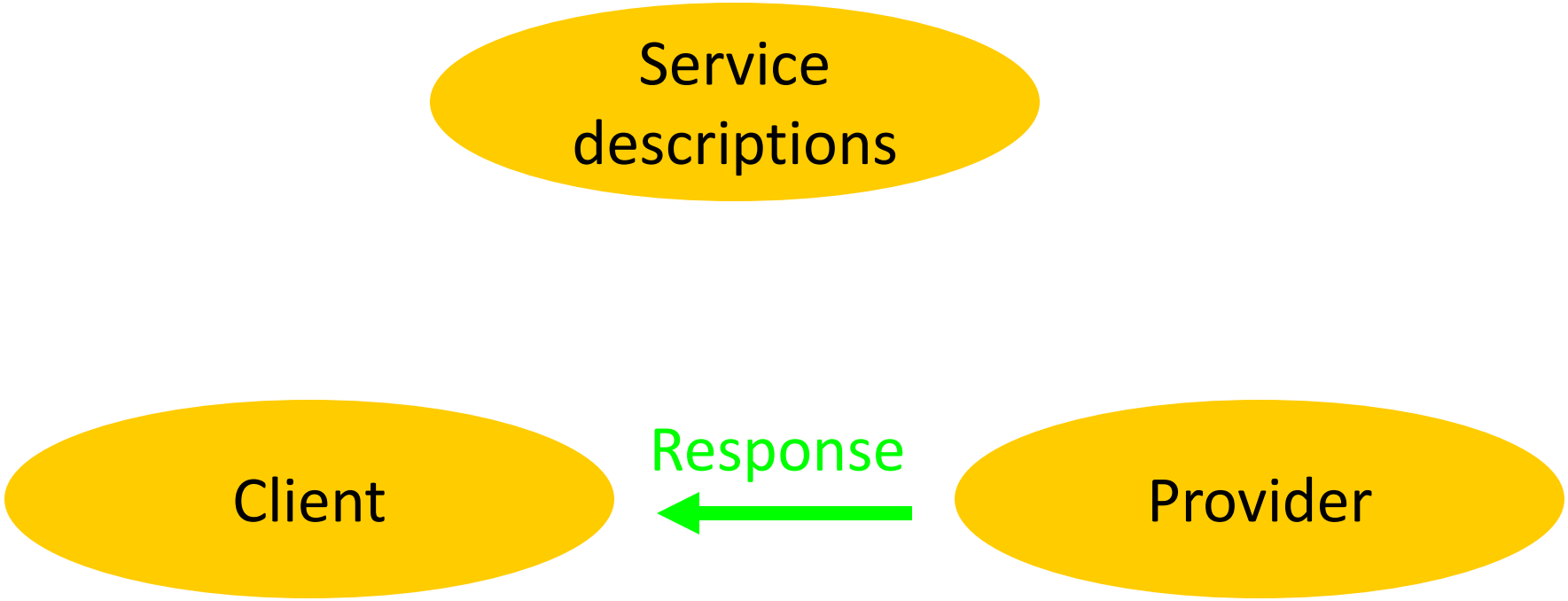
Web service architecture

- One possible way of „SOA”



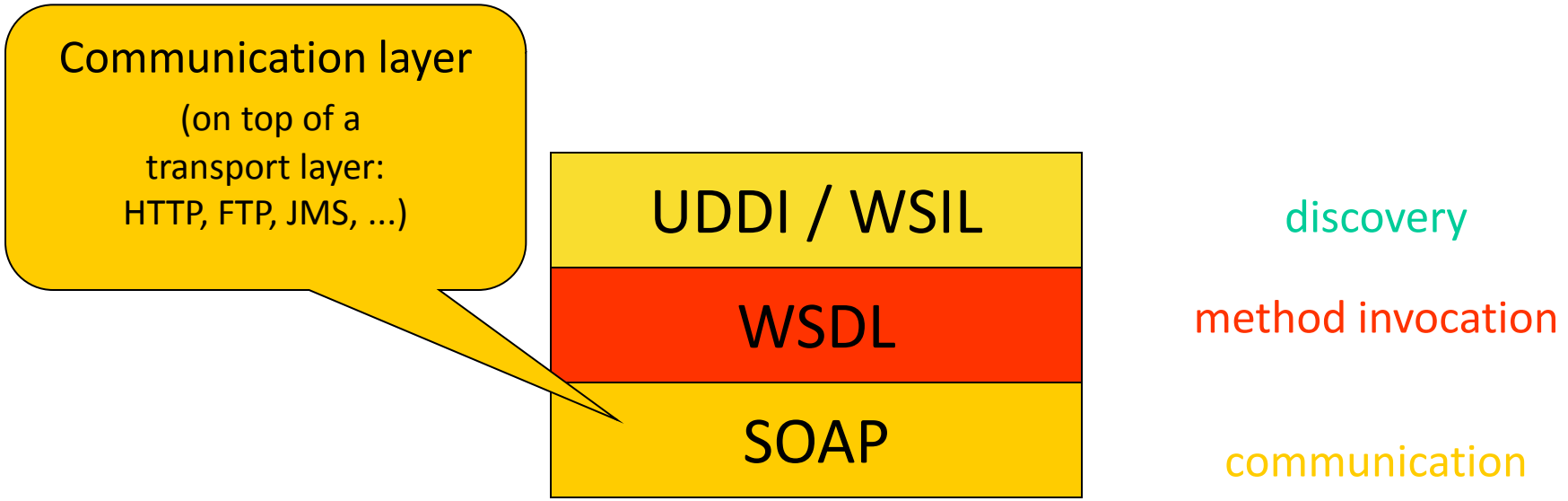
Web service architecture

- One possible way of „SOA”



Web Services standards

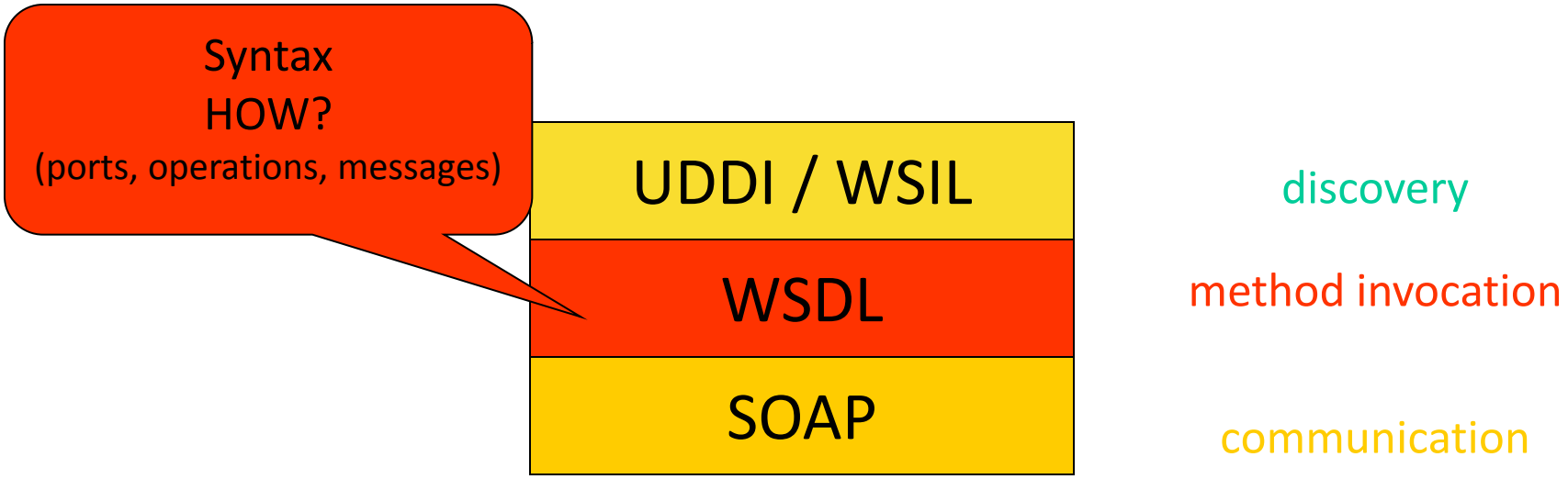
- XML languages
- Low level specs. of functionality



Web service stack

Web service standards

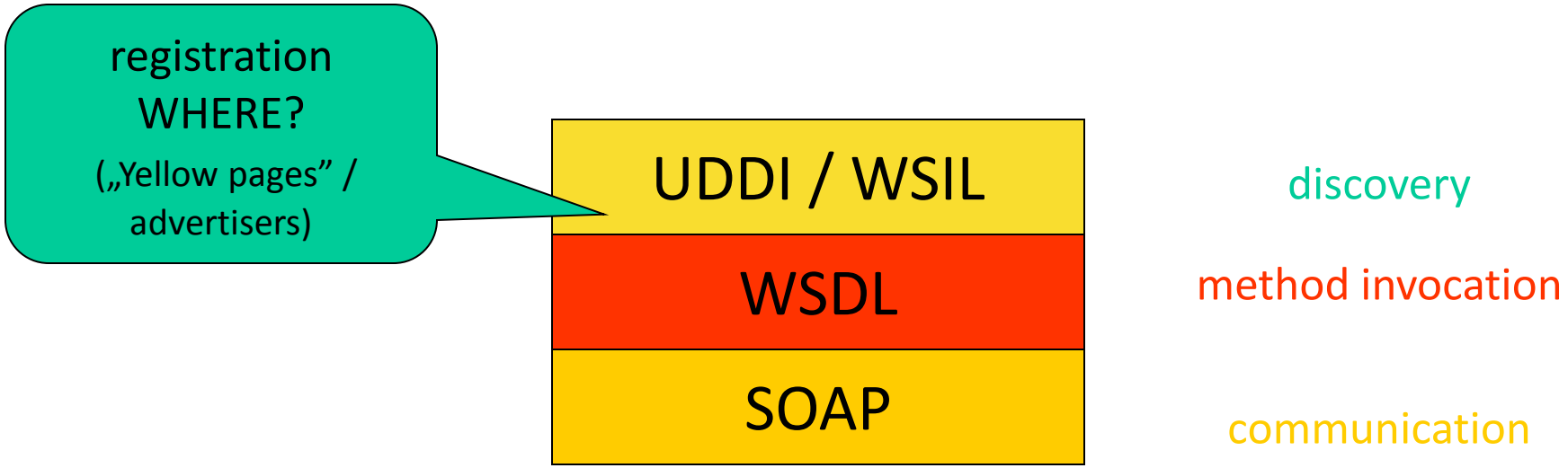
- XML languages
- Low level specs. of functionality



Web service stack

Web Service standards

- XML languages
- Low level specs. of functionality



Web service stack

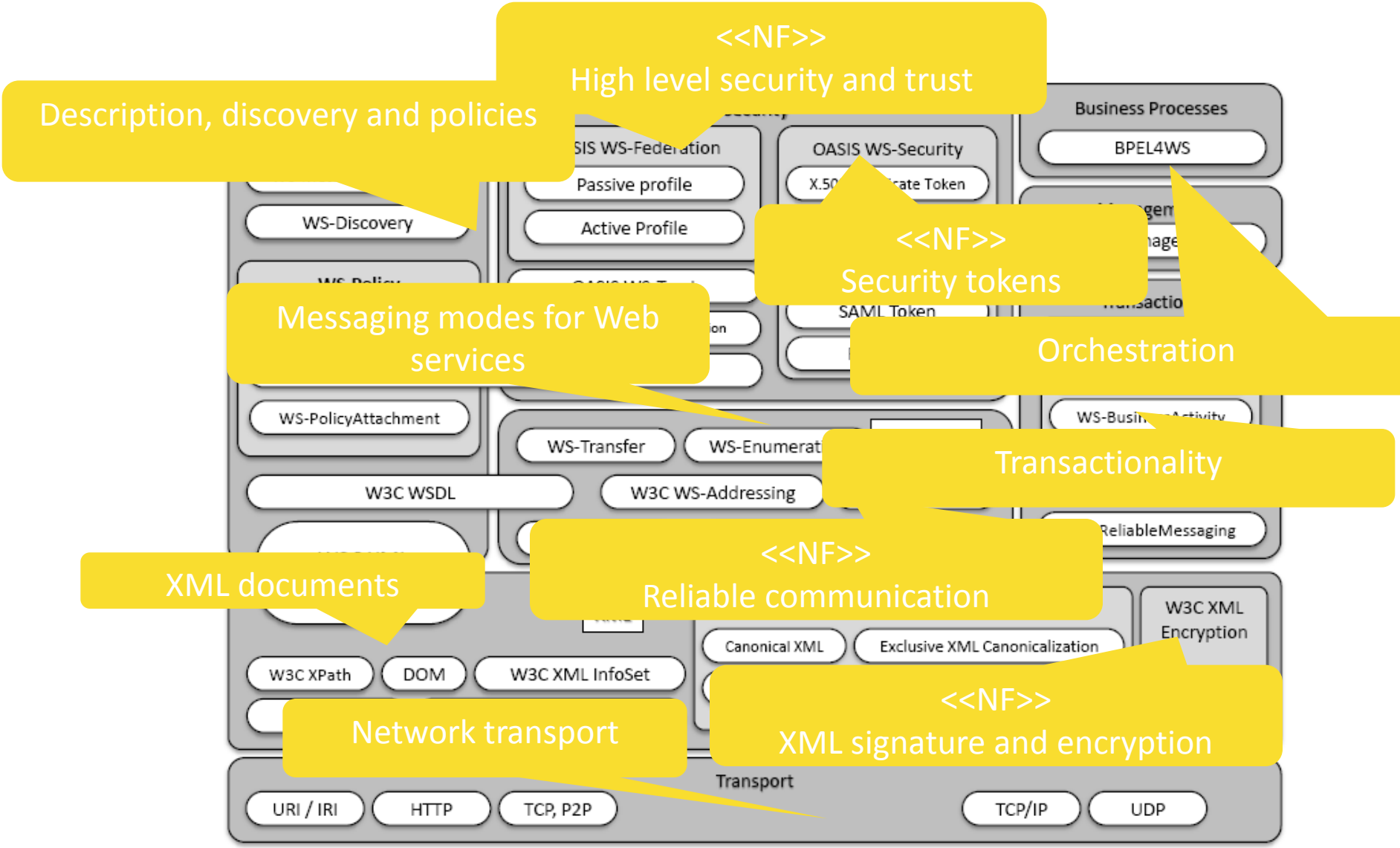
Objectives of service integration

- Service-Oriented Architectures (SOA):
  - Flexible and dynamic platform to deliver business services
- Requirements:
  - Reduced time-to-market
  - Increased quality of service
- Challenges
  - **Specification and querying** of services?
  - **Correctness and consistency** of service composition?
  - Continuous operation in **changing environment** (no service outages)?
  - Design for **justifiable SLA-compliance** (security, performance, reliability, availability)?
- To meet such non-functional requirements,
  - A service needs to be designed for reliability
  - Architectural level design decisions

What is „non-functional”?

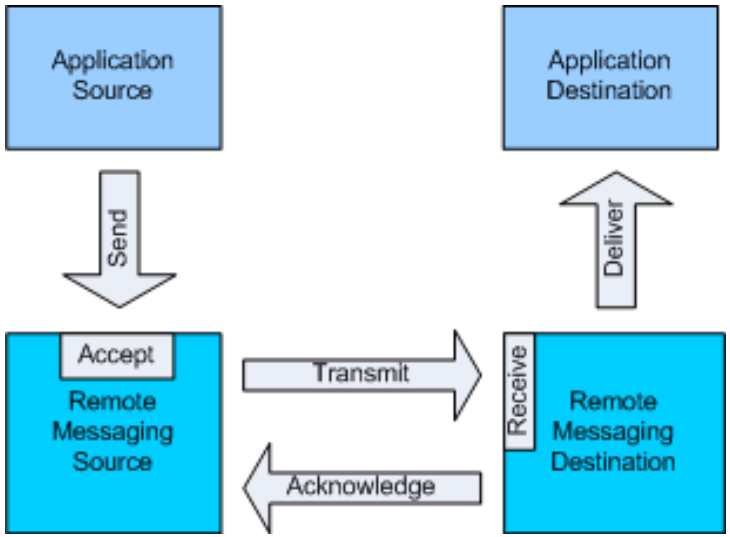
- Everything which is above the core functionality
  - Also called „extra-functional”
- Under what circumstances is a service provided?
  - When is it available?
  - What response time does it guarantee?
  - How many requests can be sent to the system (from how many clients)?
  - What prevents messages from being lost?
  - Can messages of a given service be tampered with?

The beautiful world of standards….



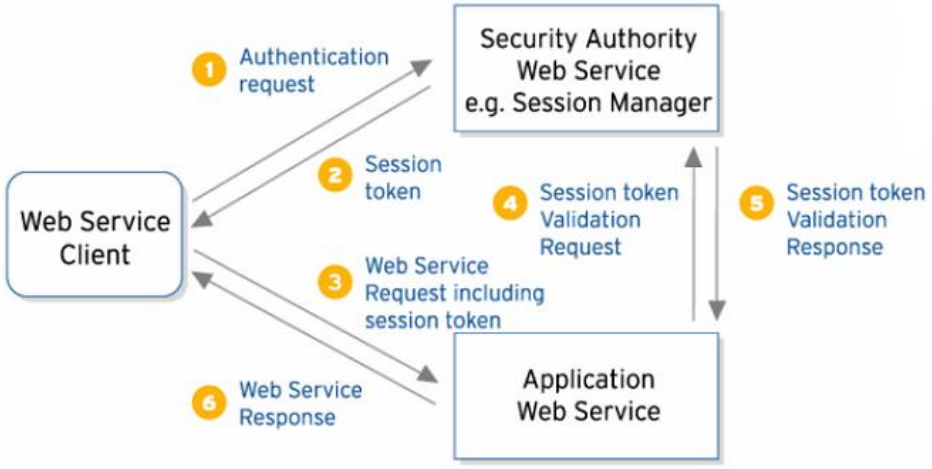
Reliable messaging: WS-RM

- „TCP protocol” for Web services
- What is reliable messaging?
  - Acknowledgements
  - Message ordering
  - Filtering duplicates
  - Guaranteed delivery
  - Timing paremeters
- Messaging semantics
  - At-least once
  - At-most-once
  - Exactly-once
- Convergence of multiple standards (MS, IBM)
- Implementations
  - RAMP (IBM WebSphere Application Server)
  - Apache Sandesha (Axis2)
  - Microsoft Windows Communication Foundation
  - Bea WebLogic (→Oracle)

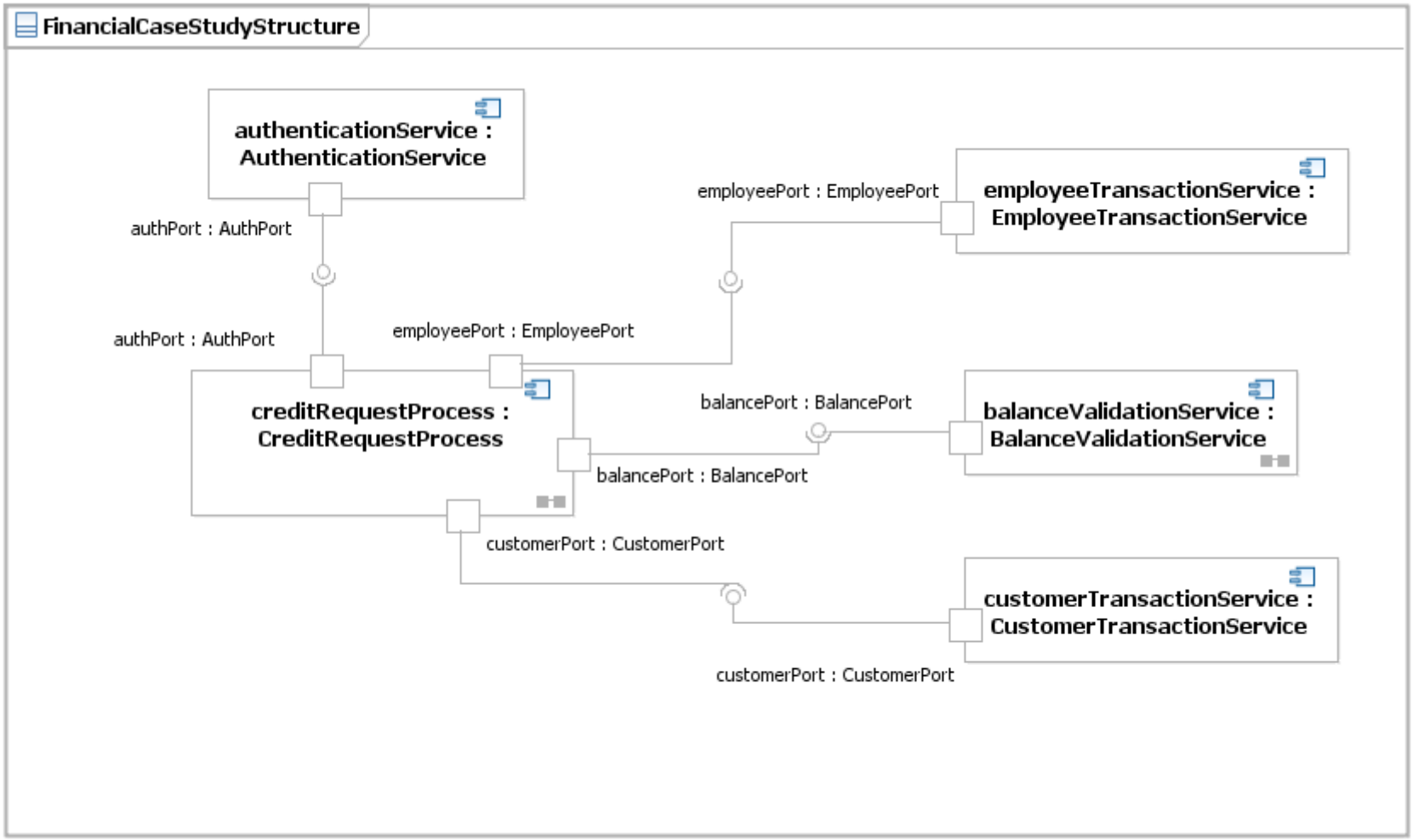


Standards:WS-Security

- Encryption for body and header
- Digital signature for body and header
- Authentication tokens
- Timestamps allow the user to specify timestamps for messages



Example: components of a financial case study



Example: finance case study

- **All services** should be available only via secure connections. This means digital signatures for the entire message and encryption of the message body. Messages sent to this service should be acknowledged.
- **Customer Transaction Service** should provide an answer to the customer about the receipt of his request soon, therefore its maximum response time should be no longer than 8 seconds.
- **Balance Validation Service** should send an acknowledgement of all incoming request. As this is a resource-intensive task, multiple instances of the same request should be identified and filtered out. Since the complete balance validation may require human interaction as well (depending on the business rules of the bank), a quick answer to all validation requests cannot be expected. However, some feedback about the initiation of the validation process should be sent back soon, with an average of 5 seconds and maximum value of 8 seconds. The throughput of this service is also of outmost importance, resulting in a requirement of 6 user requests per second. On the other hand, maximum throughput of the service is also bound due to the bank policies in 20 requests per second.

Example: finance case study

- **Authentication Service** is used by many applications, therefore its throughput can be a bottleneck in the system. To support a continuous service, a minimum throughput of 100.000 requests/hour is required. This throughput is used by multiple the applications relying on the authentication service, e.g., Credit Request Process.
- **Credit Request Process** is depending on the above services. Most of its requirements are derived from requirements of the invoked services (e.g., where and what to encrypt), but some are also implied by the customer portal interface. Such a requirement is the performance of this process and the non-repudiation of requests.

Content

- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Web service faults and effects

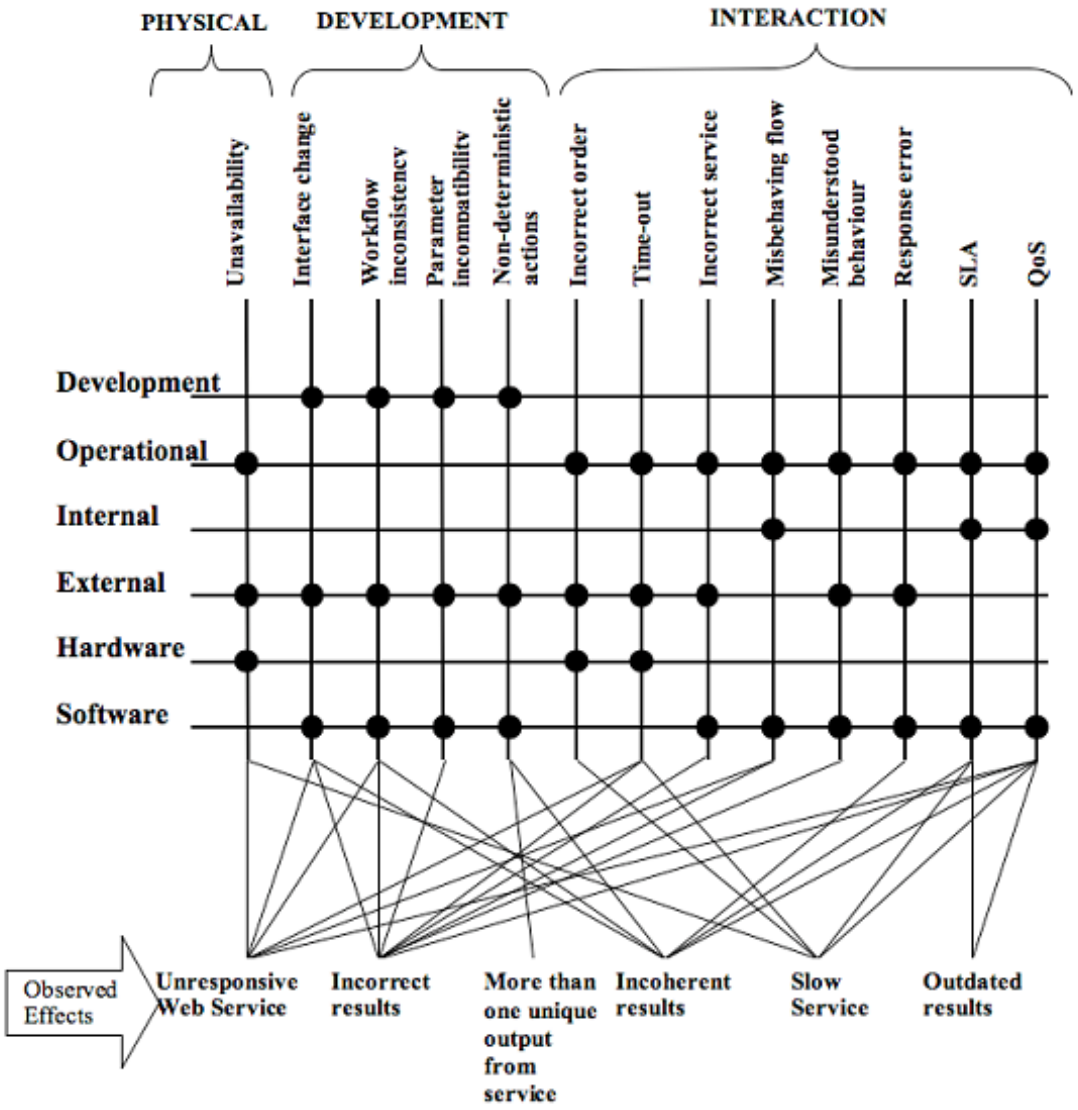


Fig. 6. Taxonomy of faults, combined with observed effects

Fault injection models

	Bernoulli	Random Variable	Stochastic Process
Service	—	[10, 11, 12, 13, 14, 15]	—
Platform	[16]	—	[17, 18, 19, 16]
		—	—
Application Server/ SOAP Framework		—	[20, 21]
Virtualisation	—	—	[22]
Storage/Databases	[23, 24]		[3]
Communication	—		—
Web Services Stack	—		—
WS Infrastructure	—		—
WS Transport	—	[7]	—
Internet Infrastructure	[27, 25]	[26]	[27, 25]
Internet Transport	[33, 34, 35, 16]	—	37,

Bernoulli distribution

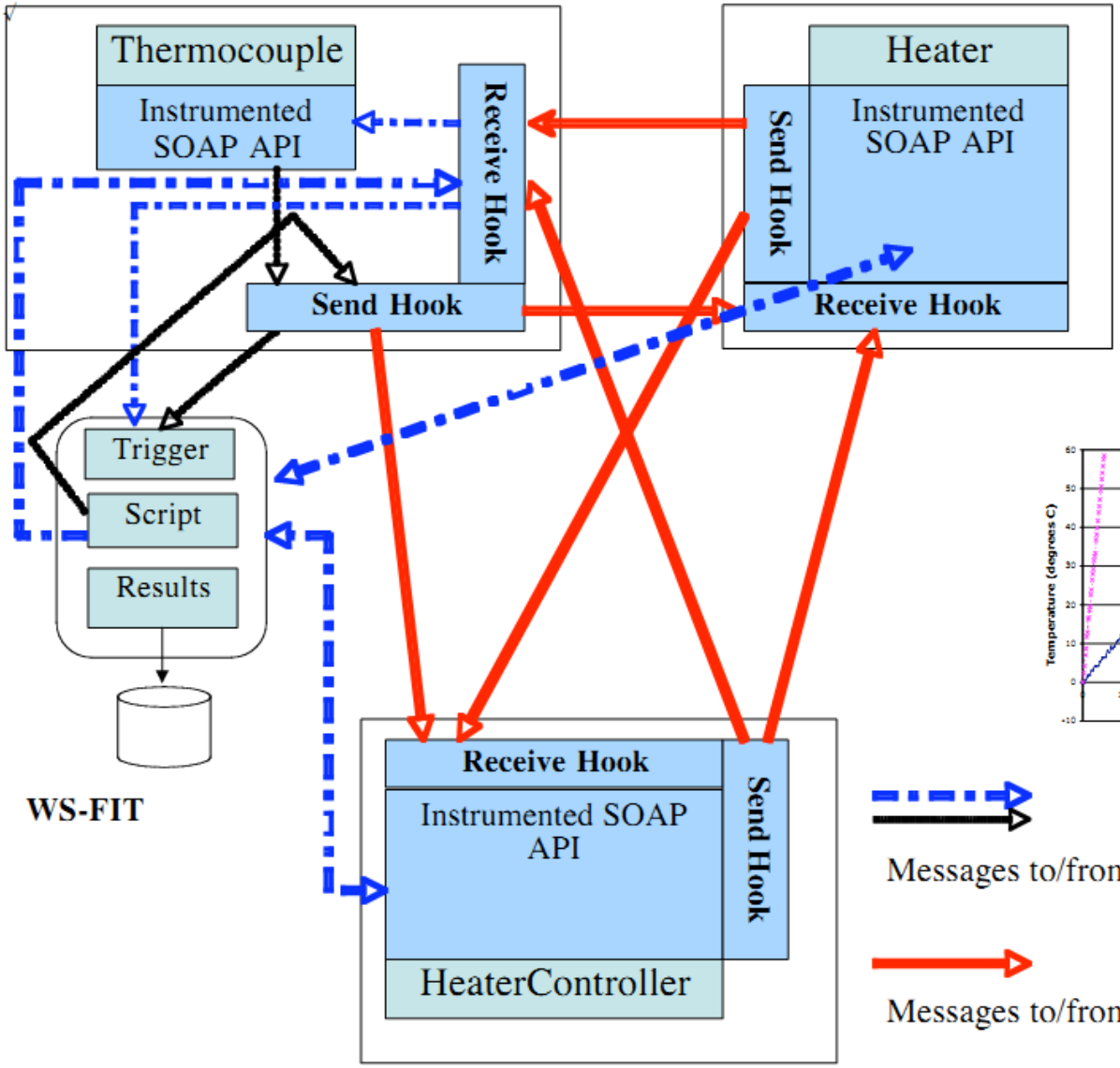
Random variable (distribution, density, mean+ quantile….)

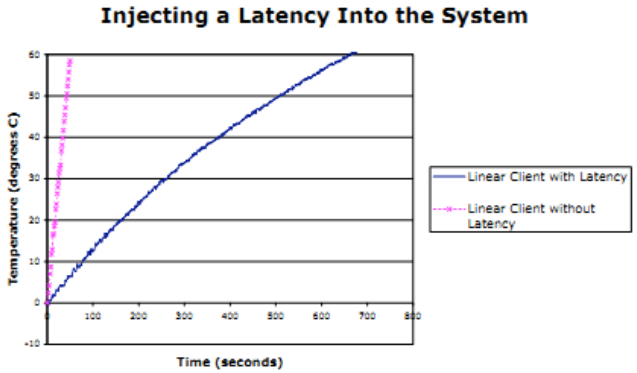
Stoachastic processes for state description

WS-FIT

- Web services fault injection technology
  - Univ. Durham, Univ. Leeds
- Standard network level fault injection techniques:
  - Easy to detect at the application/middleware level
- RPC level faults can be injected
  - WSDL = API
  - SOAP level fault injection
- Handling of middleware faults can be tested
  - E.g. Axis failure modes: connection refused, unknown host, wrong content type, XML parser errors…

Test case


Thermocouple
Instrumented SOAP API
Receive Hook
Send Hook
Heater
Instrumented SOAP API
Send Hook
Receive Hook
Trigger
Script
Results
WS-FIT
Receive Hook
Instrumented SOAP API
Send Hook
HeaterController
Messages to/from WS-FIT
Messages to/from SOAP Stacks


Injecting a Latency Into the System
Temperature (degrees C)
Time (seconds)
Linear Client with Latency
Linear Client without Latency

Security analysis

- Systematic attack based on WSDL
  - Public information
- „Brute force” attack (XML parsing)
  - Overloed: parsing comes bottleneck
- „XML injection”
  - Changing the parsing process
    - Eg. using XPath, XSTL, XQuery
- External reference-based attack
  - Linking a document
- SOAP protocol level attack
- Network level attack

Content

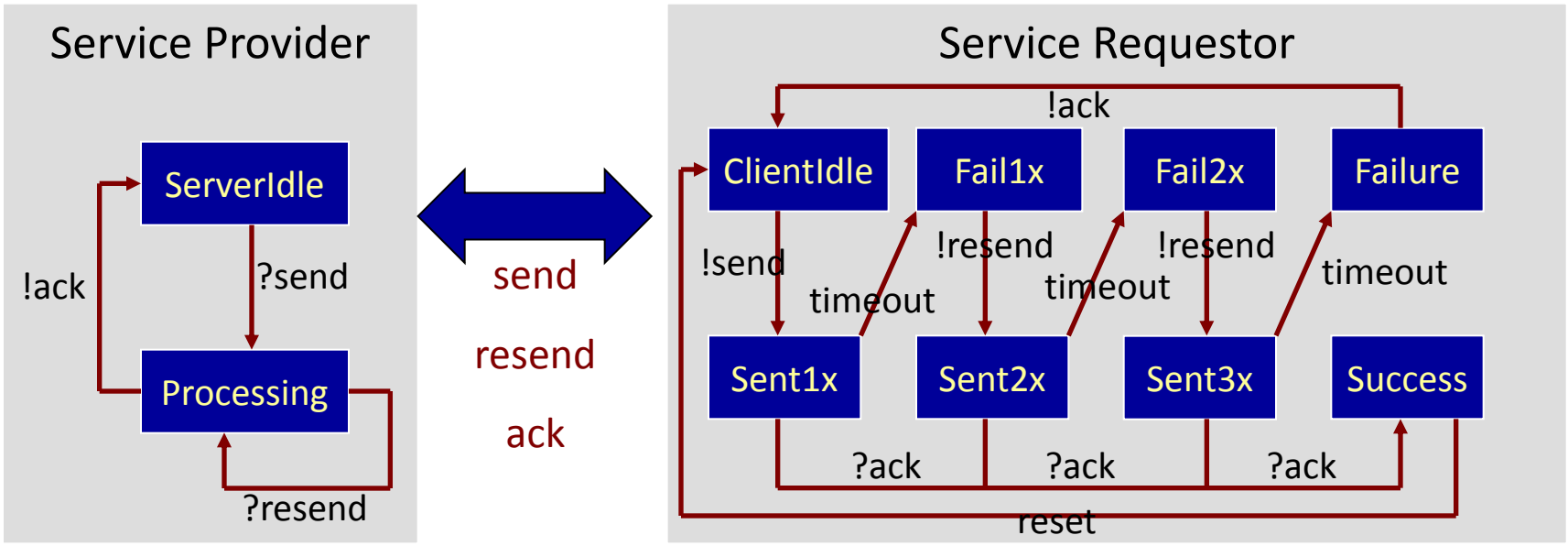
- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Performability analysis

- „Performability = Performance + Reliability”
- What happens if something goes wrong?
  - E.g., reliable communication middleware with re-sending can mask network faults but the **guarenteed** response time can be longer
  - E.g. if the acknowledgement interval is too small, false alerts are sent
- What is the „cost of reliability”?
- How to tune SLA parameters?

Performability model

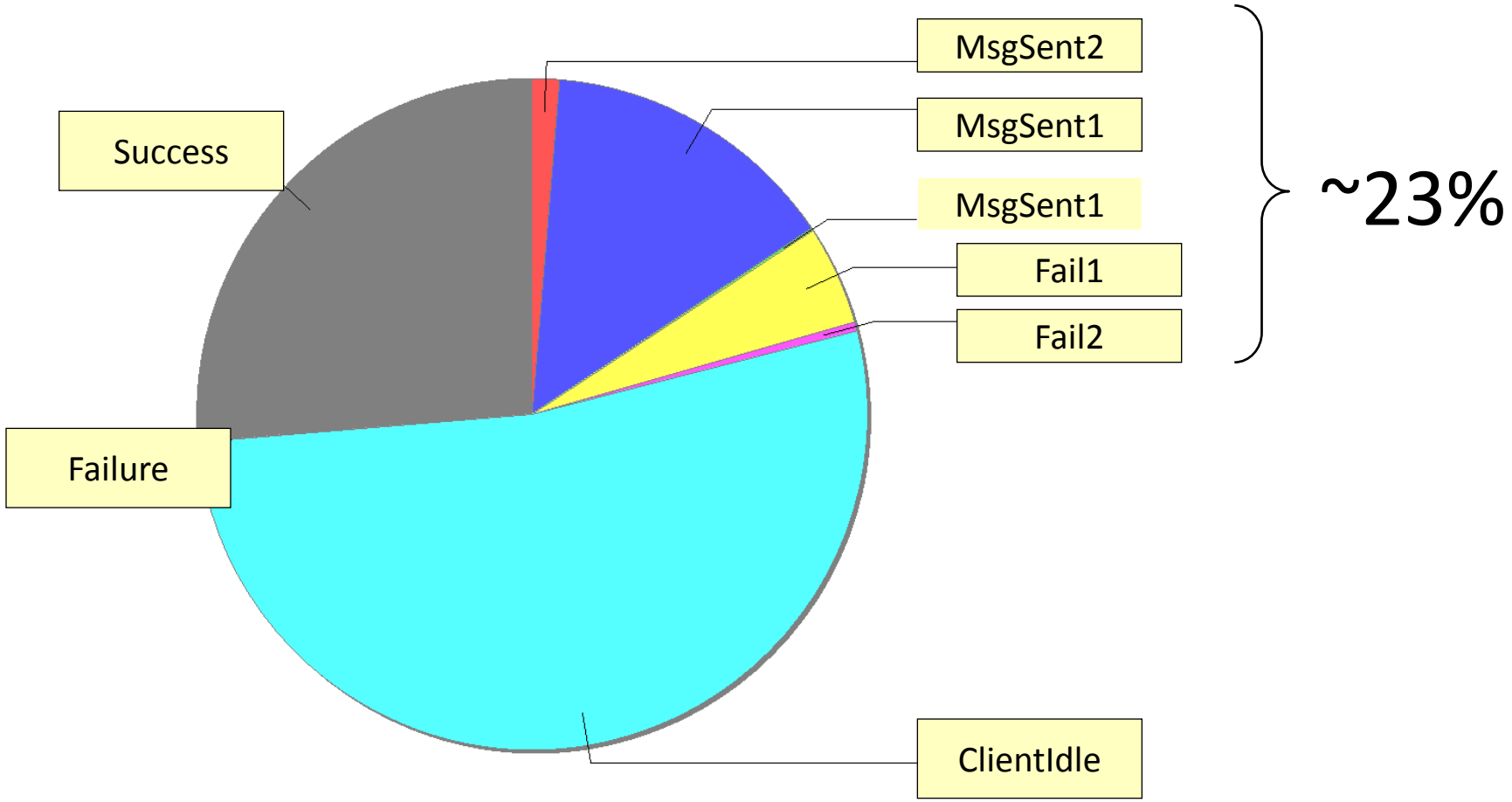
- Abstract behaviour of
  - Service provider
  - Service consumer
- Reliable messaging parameters (derived)
  - Number of resends
  - Parameters of **send**, **resend**, **ack** (exponential distribution)



Analysis results: Utilization

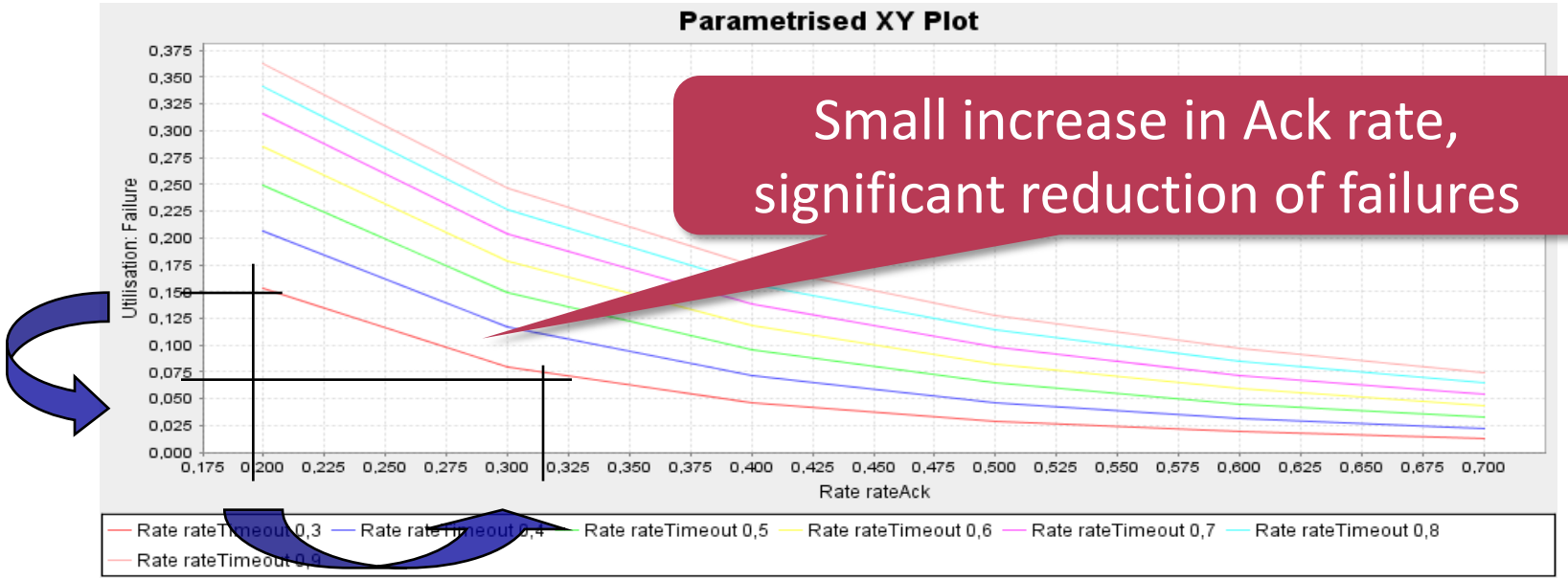
Steady state analysis of Throughput / Utilization

- What percentage is fault handling (when the client waits)?



Analysis results: Sensitivity analysis

Sensitivity Analysis: where to improve?
How does the probability of system-level failure change
if there is a change of the parameter of resend?



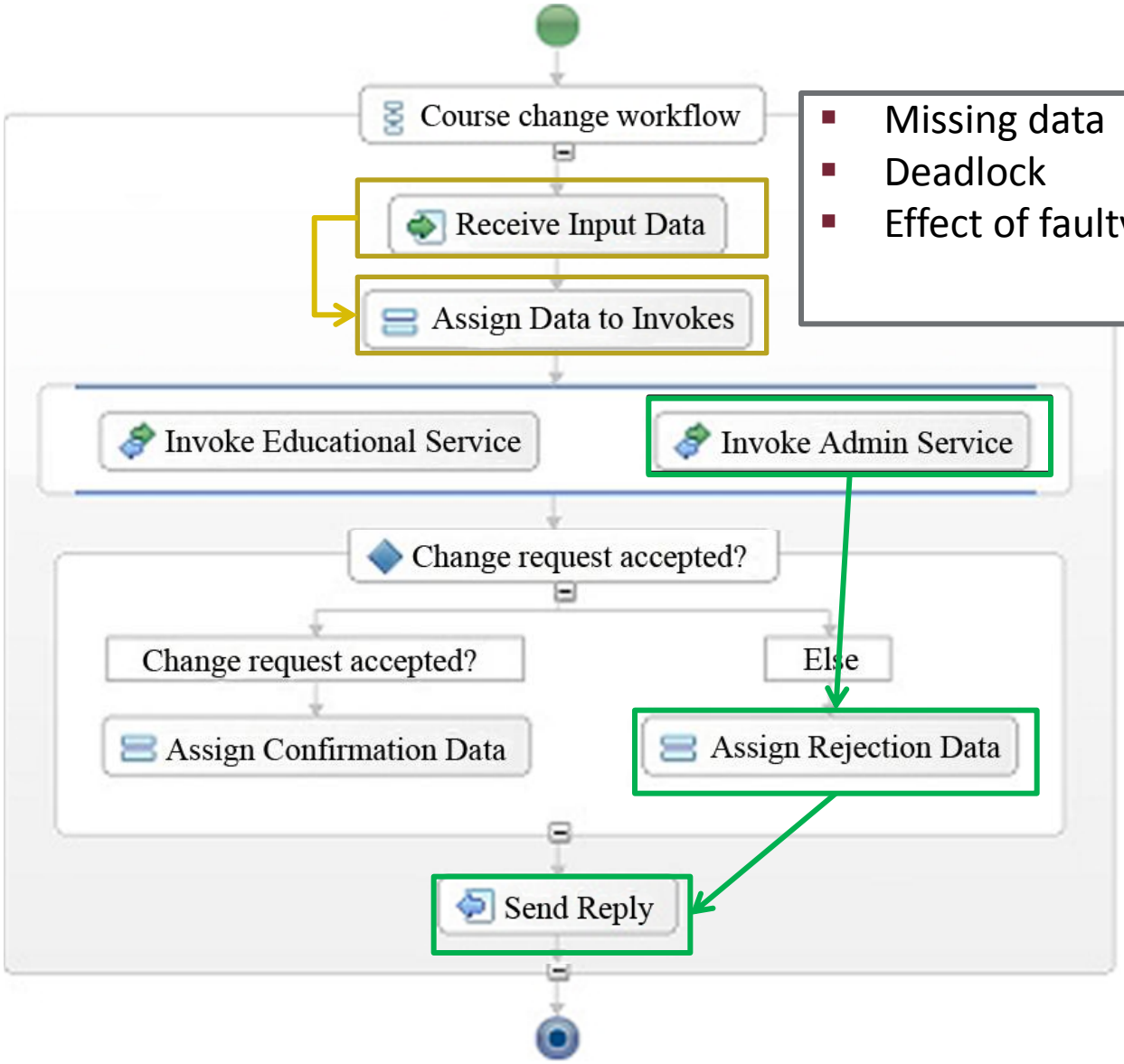
Content

- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Analysis of service compositions

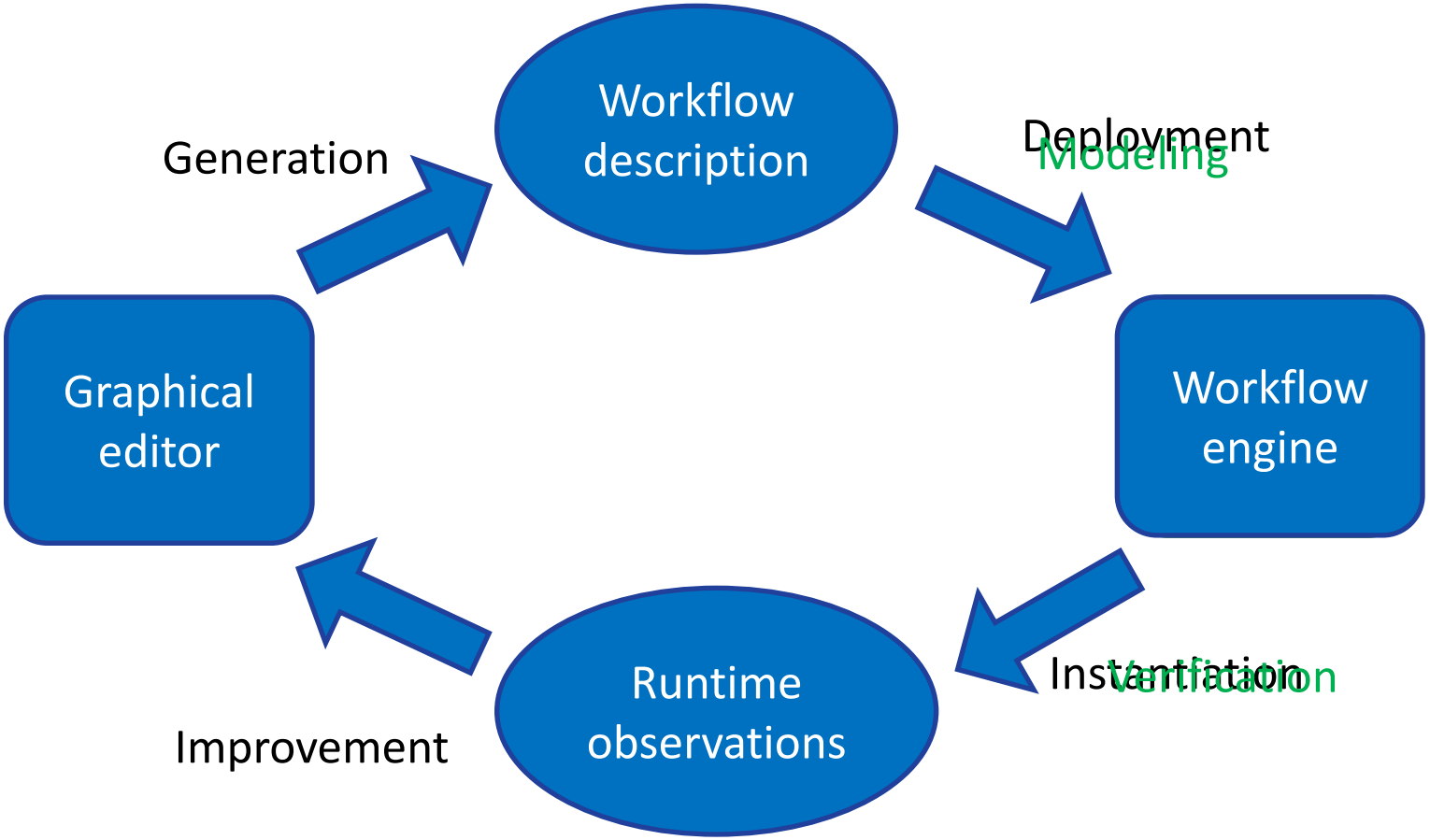
- Design tools offer only syntax check
- Static analysis in BPEL 2.0
  - Constraints on workflows (attributes, structure)
- Safety critical services
  - E.g. e-Health
- Ensure correct functioning of combined services

BPEL flaws

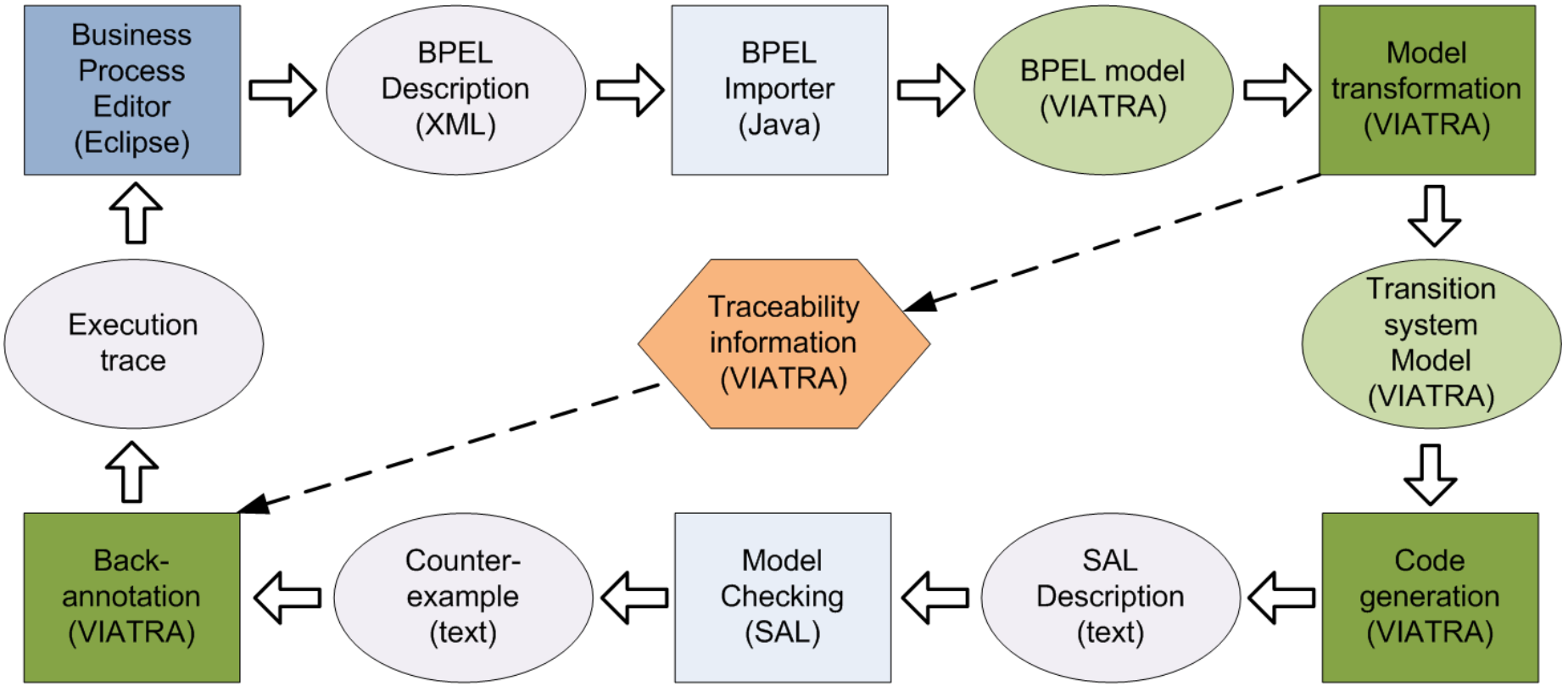


- Missing data
- Deadlock
- Effect of faulty data

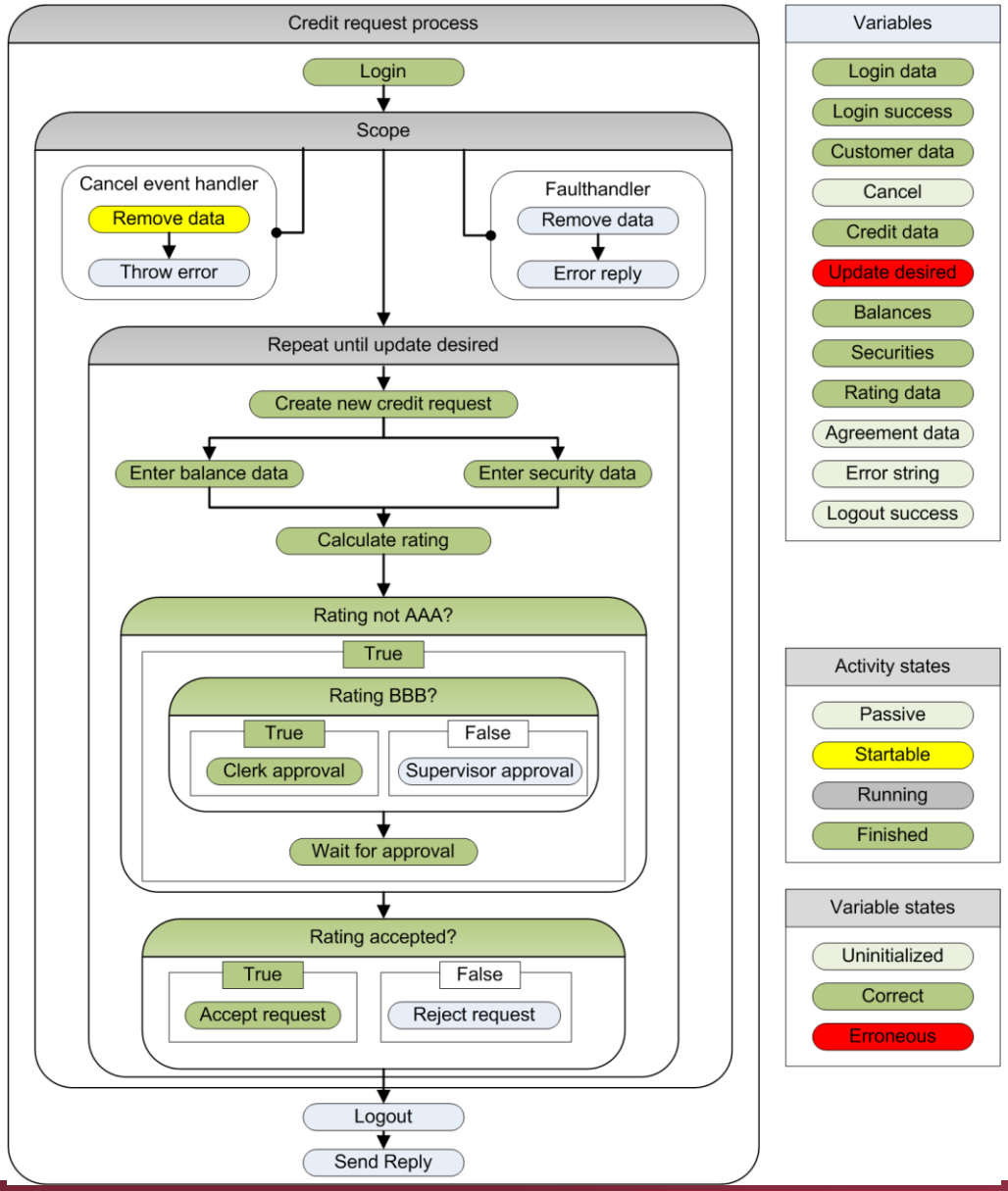
Workflow assessment



Transformation chain



Case study&back-annotation



Content

- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Performance analysis with PEPA

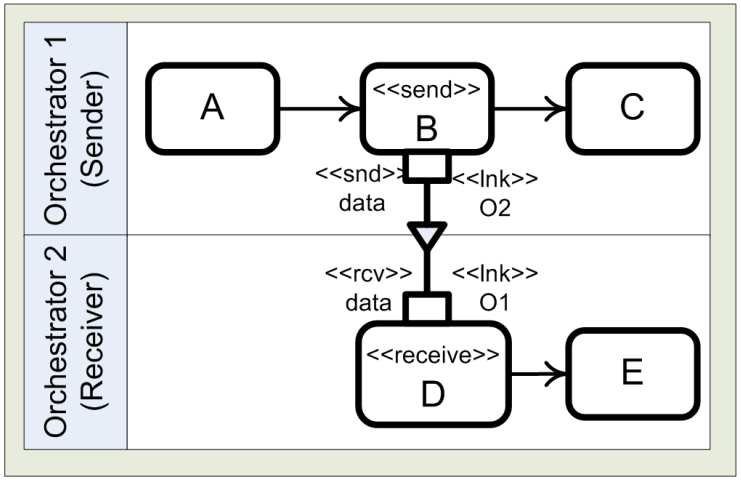
- PEPA is a formal language for quantitative analysis of systems
  - A model is expressed in terms of **components** which perform timed activities and **co-operate** with each other
    - Based on process algebras,
    - Synchronization, Parallel, sequential composition
    - Length of activities: Exponential distribution
    - Generates continuous-time Markov chain (or differential equations)
- PEPA supports **steady-state analysis** to answer questions such as:
  - What is the **percentage of time** that the local discovery server is idle in the long run? (*Utilisation Analysis*)
  - What is the **throughput** at which remote services are discovered? (*Throughput Analysis*)
  - What is the **probability** that the system does compensation upon notification of failure?

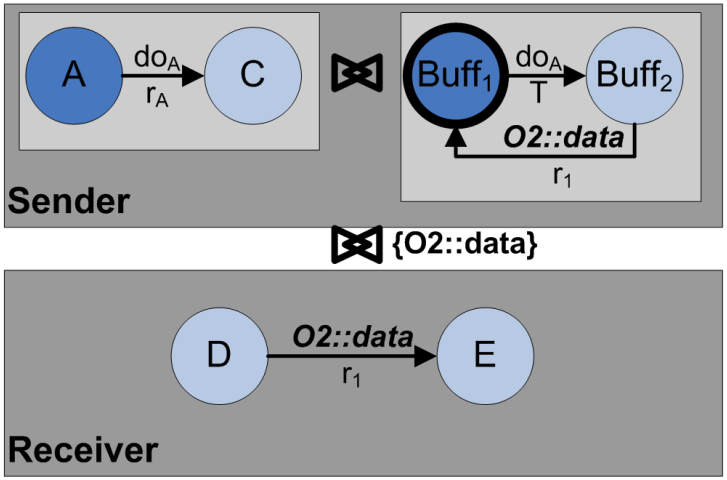
UML4SOA Performance Analysis

- The transformation of UML4SOA-annotated activity diagrams follows the rationale behind the treatment of *plain* activity diagrams:
  - An **Action Node** is stereotyped with *PaStep*, which indicates the rate of execution (with an exponential distribution). It is modelled as a prefix *(action, rate).Process*
  - A **Decision Node** is modelled with a PEPA choice *ProcessA + ProcessB*
  - A **Fork Node** activates one or more flows of behaviour, modelled as synchronising sequential components
  - A **Join Node** makes all incoming flows synchronise on the same activity, enforcing that the outgoing flow executes after all the incoming ones terminate

Performance Analysis with PEPA

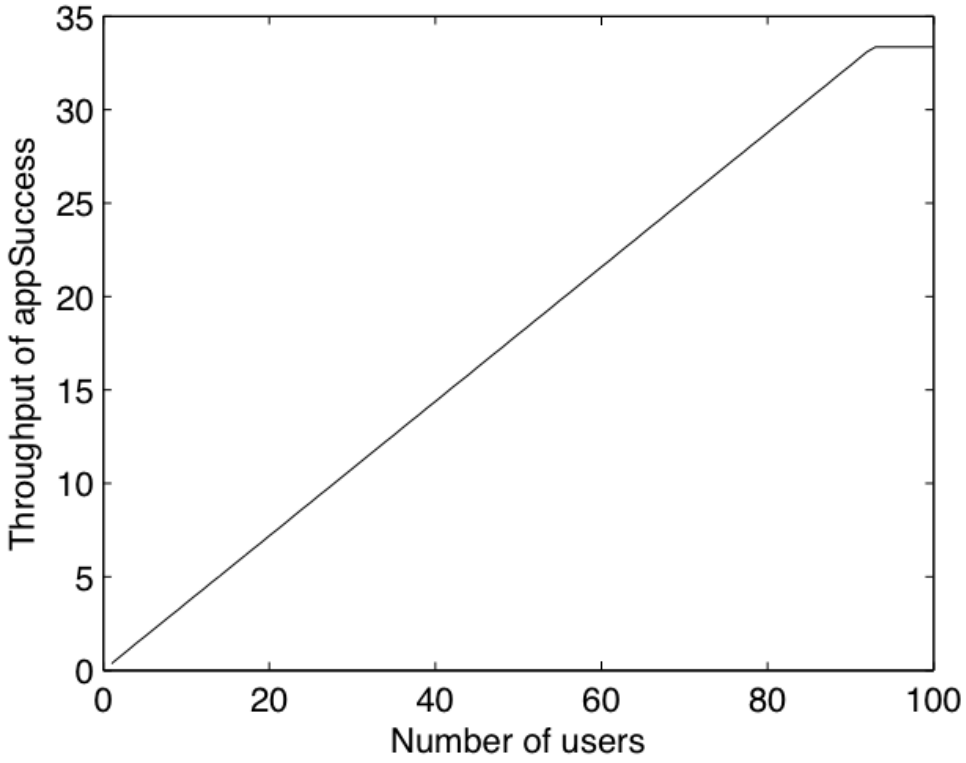
- Key UML4SOA-specific element: **communication between activity diagrams**





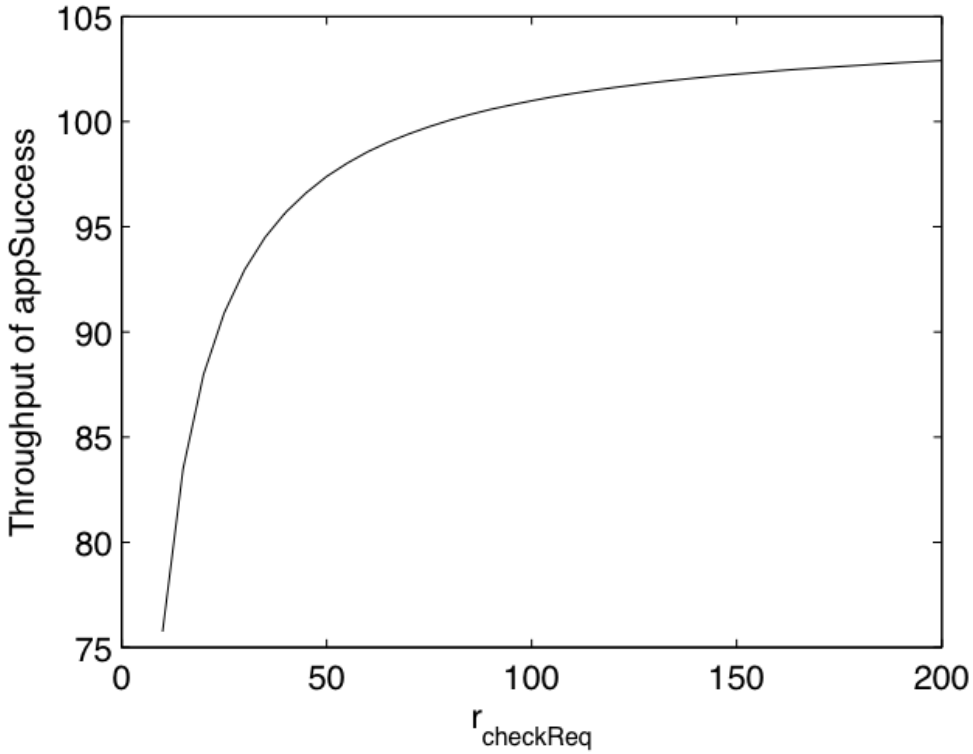
- Each activity is modelled as a distinct component
- Message exchange modelled as a shared activity (named after the sender's pins **<<snd>>** and **<<lnk>>**)
- Support for asynchronous and synchronous messages (according to the stereotypes **<<send>>**, **<<receive>>**, **<<reply>>**, etc.)
  - Here, the asynchronous message is dispatched by a *buffer* component and the sender does not wait while the communication is happening

Performance Results



Varying Workload. The number of applications processed successfully grows with the number of users. A bottleneck occurs for populations larger than 93.

Performance Results



Sensitivity Analysis. The number of applications processed depends on the rate at which the entry requirements are checked. However, for sufficiently large rates further increases do not impact significantly on the user-perceived system performance.

Content

- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Dependability analysis of composite services

Composite services

- Composed of basic service components
- Only partial control over the different services

Analysis of composite services

- According to SLA parameters of services
  (e.g. throughput, reliability, availability)
- User perceived service:
  potentially different service levels for different users
- Required parameters for the invoked services
- Guaranteed parameters for the main service

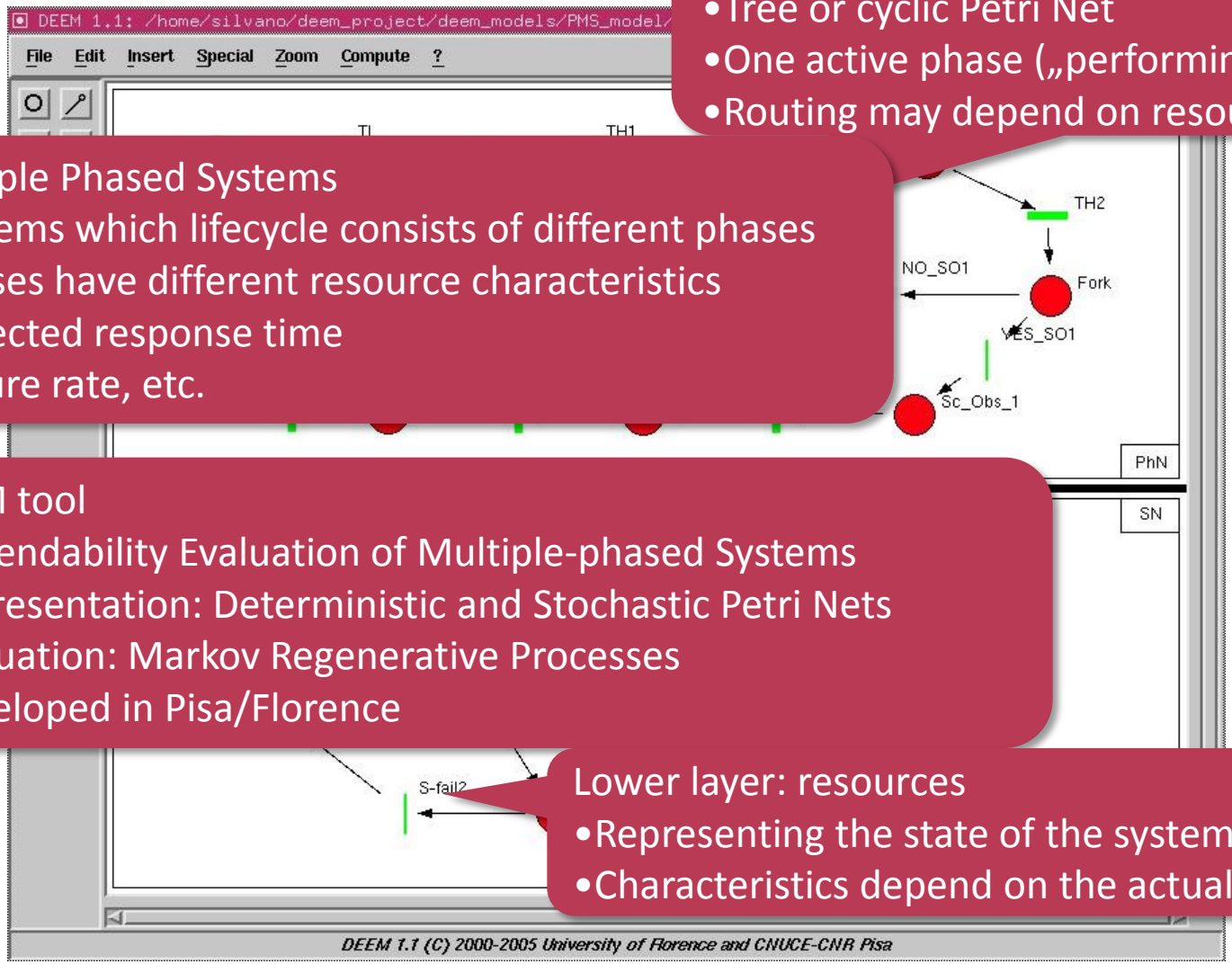
Non-functional analysis

- PREDICTION of
  - Dependability metrics for the services
  - Business impacts
- WHAT-IF analysis

Phased Mission Systems

Upper layer: phases
- Operational life
- Tree or cyclic Petri Net
- One active phase („performing action X”)
- Routing may depend on resource states

Multiple Phased Systems
- Systems which lifecycle consists of different phases
- Phases have different resource characteristics
- Expected response time
- Failure rate, etc.



DEEM tool
- Dependability Evaluation of Multiple-phased Systems
- Representation: Deterministic and Stochastic Petri Nets
- Evaluation: Markov Regenerative Processes
- Developed in Pisa/Florence

Lower layer: resources
- Representing the state of the system
- Characteristics depend on the actual phase

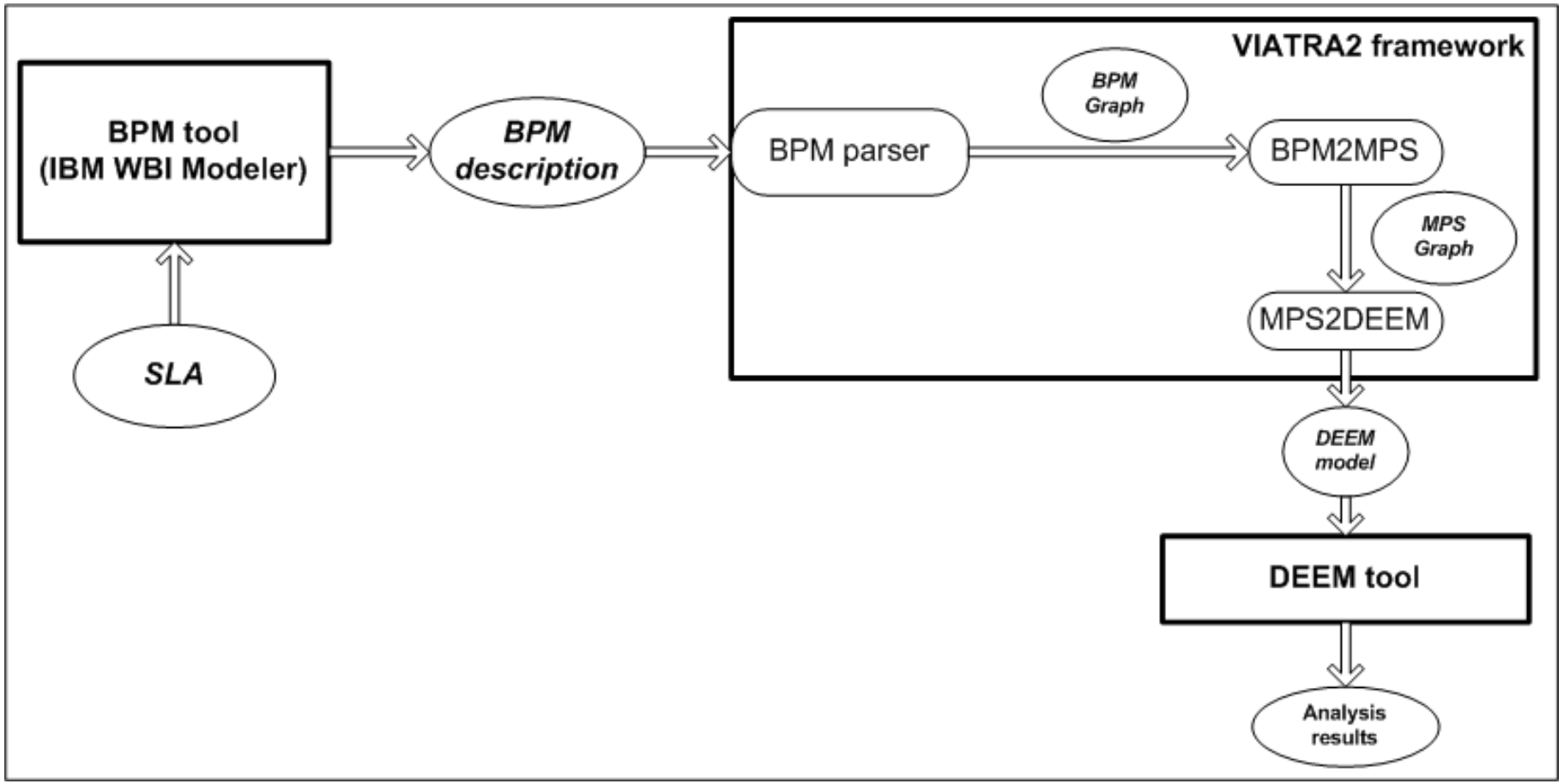
Example: Phased Mission Systems

- Stochastic modeling
- Phased operational life
- System changes during the phases
  - E.g. resource states
- System characteristics depend on the actual phase
  - E.g. phase-dependent failure rates
- Mission goal depends on system state
  - Degradation
- Dependability modeling and analysis
  - Described as GSPN
  - Originally for mission-critical systems

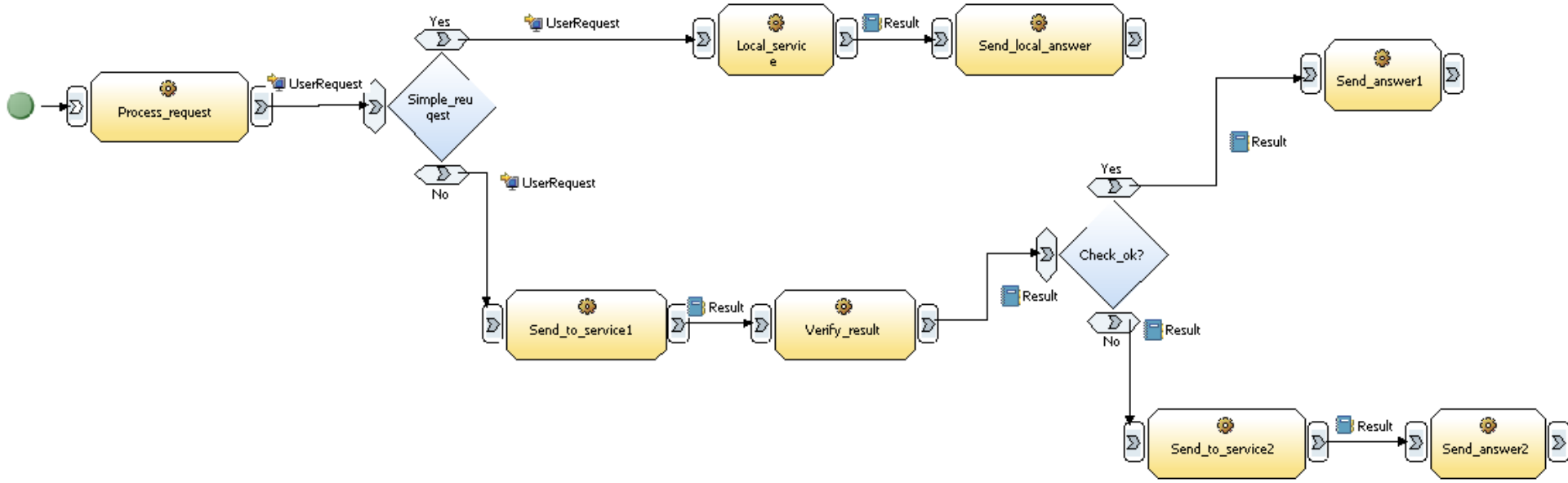
SOA service flows as PMS

- SOA service flow as PMS
- The operational life is built of distinct steps
  - Web service invocations are the phases
  - The dependability requirements of the phases are different
    - Based on Service Level Agreements
  - The execution of the phases depends on the result of previous steps
    - Restricted operation if a service invocations fails
- Dependability of the main service
- Bottleneck analysis
- Sensitivity analysis
  - Component's failure rate $\rightarrow$ System dependability

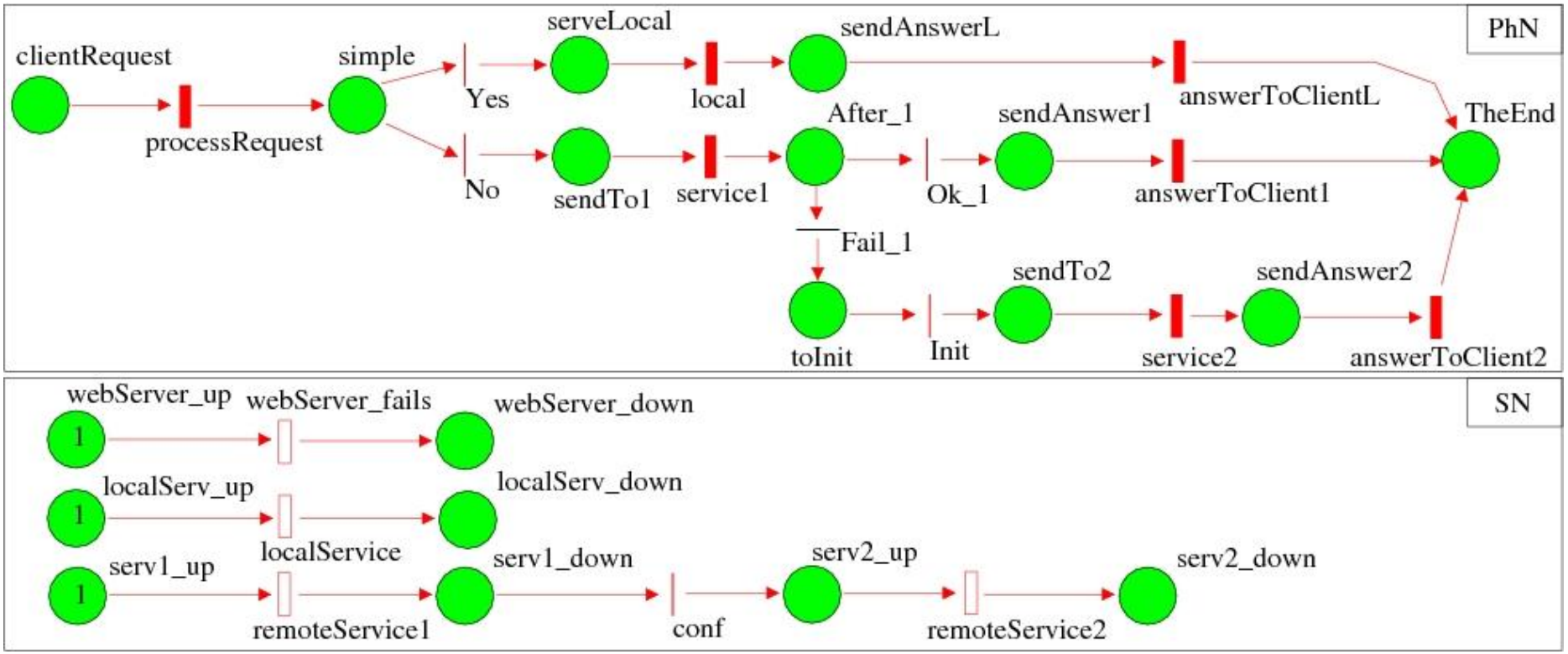
„Toolchain”


BPM tool
(IBM WBI Modeler)
SLA
BPM
description
VIATRA2 framework
BPM parser
BPM
Graph
BPM2MPS
MPS
Graph
MPS2DEEM
DEEM
model
DEEM tool
Analysis
results

An example Web service flow



The resulting PMS



Content

- Basics of Web services (WS-*)
- Fault classification &fault injection for Web services
- Performability analysis of WS-middleware
- Correctness of service compositions
- Performance analysis of service compositions
- Dependability analysis of service compositions
- Automated deployment

Why do we need development support?

- Evolving standards and platforms
- Configuration details not known for application developer
- The (XML) configuration is not portable if the model changes
- Consistency of large systems have to be ensured
- E.g. SLA → WS-* mapping
- Service intgerator can focus on business logic
- Helps „correct” modeling/development
  - Find incorrect/contradicting requirements
  - Domain/business specific requirements

Why model transformations?

- Why not XSLT, Jet, ….?
- They capture the problem at a higher level of abstraction
- Easier to maintain
- Development support (parse of engineering models like UML, incremental pattern matching for large models)
- Analysis support on intermediate models

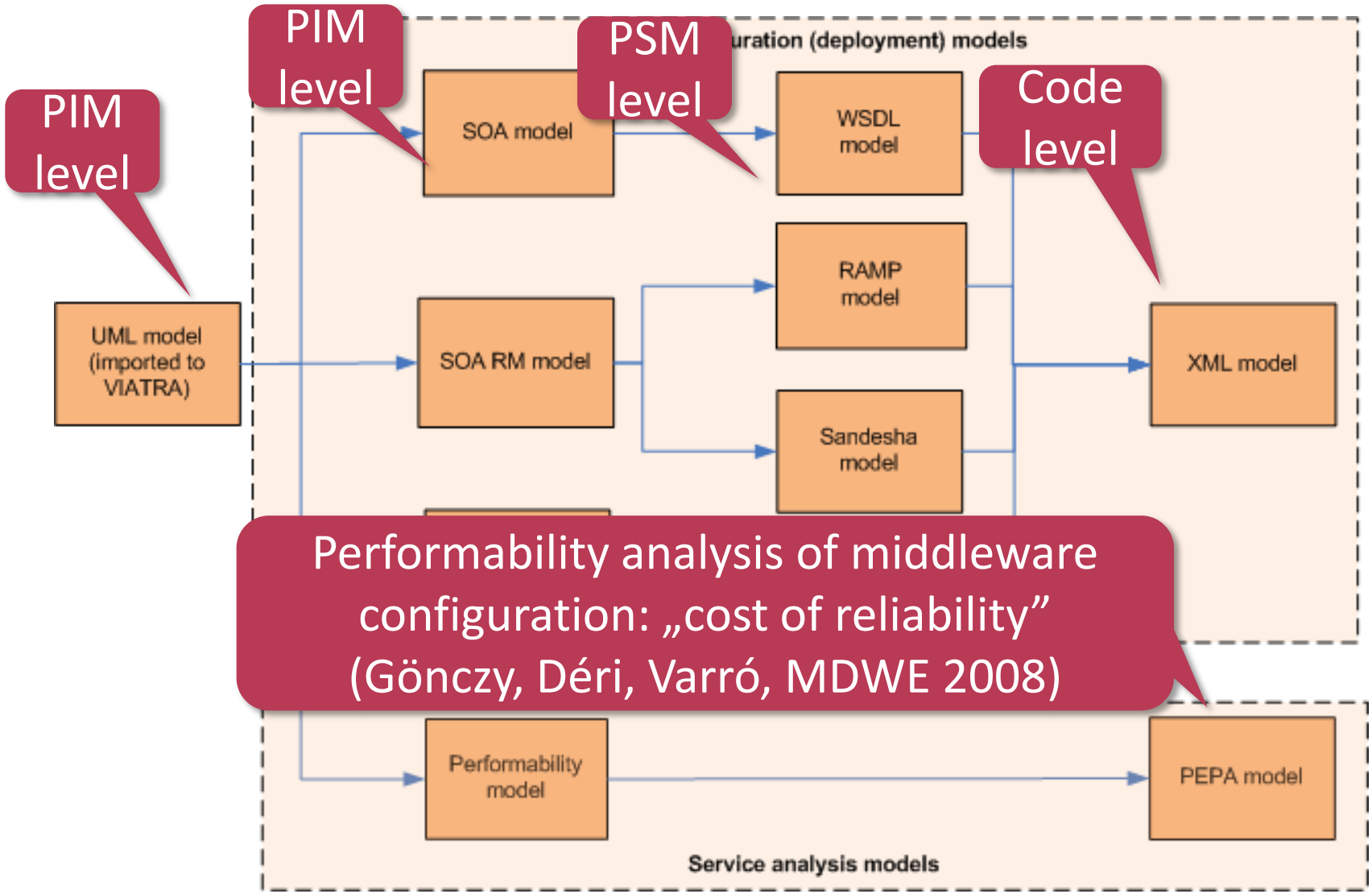
NFP in practice: deployment

- We addressed the problem of „PIM-PSM” mappings and code generation
  - Similar works in SENSORIA: MDD4SOA transformations
  - Focusing on orchestration
- Goal:
  - Have a flexible toolkit to generate middleware
-

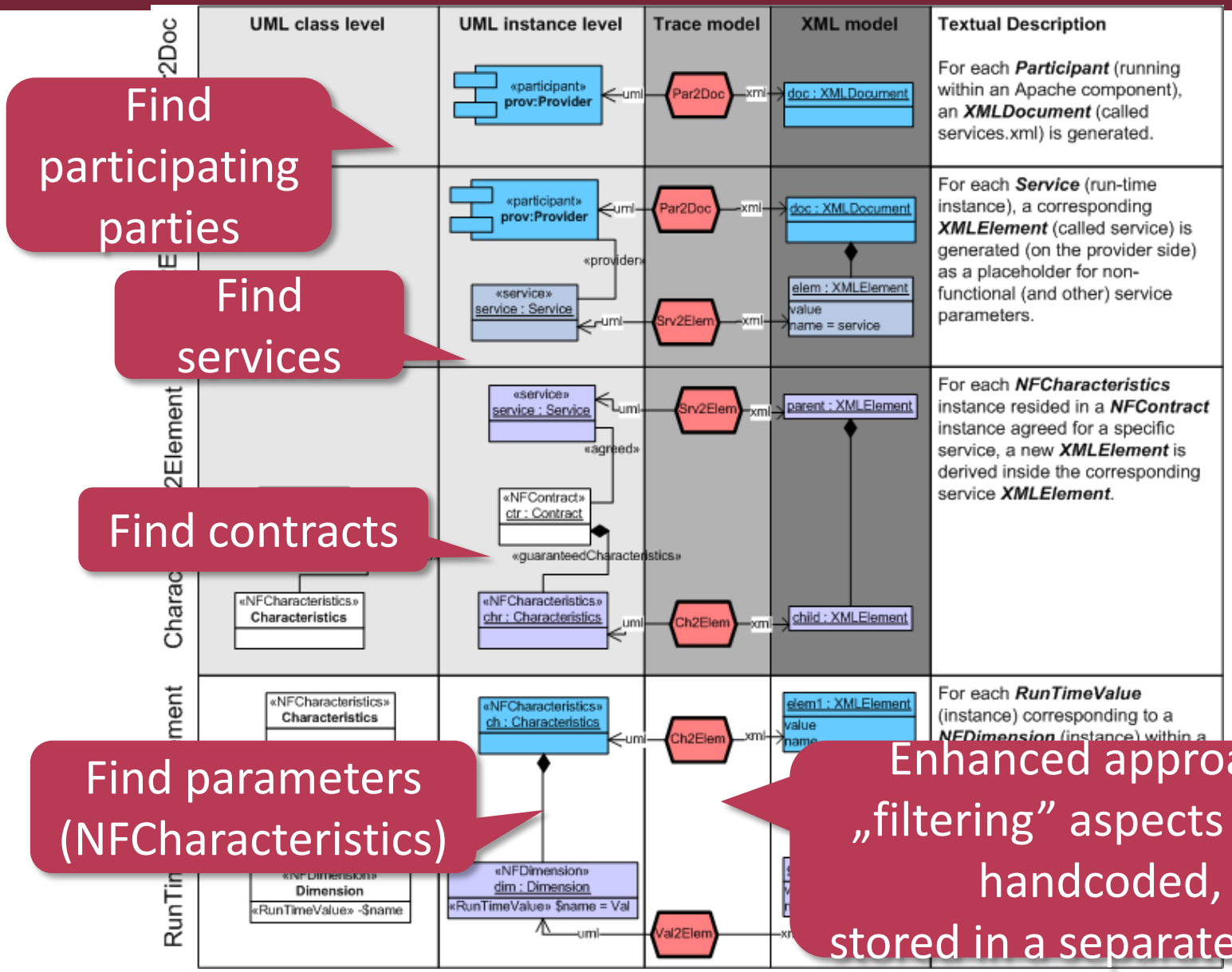
```
<reliabityParams>
    <InactivityTimeout>60</InactivityTimeout>
    <MaximumRetransmissionCount>3</MaximumRetransmissionCount>
    <ExponentialBackoff>false</ExponentialBackoff>
    <AcknowledgementInterval>100</AcknowledgementInterval>
    <RetransmissionInterval>10000</RetransmissionInterval>
    <SequenceRemovalTimeout>60</SequenceRemovalTimeout>
    <InvokeInOrder>true</InvokeInOrder>
</reliabityParams>
```

```
<SecurityParams>
    <AuthTokenType>username</AuthTokenType>
    <EncryptBody>true</EncryptBody>
    <EncryptSignature>false</EncryptSignature>
    <EncryptAlgorithm>default</EncryptAlgorithm>
    <SignBody>true</SignBody>
    <EncryptHeader>false</EncryptHeader>
    <UseTimeStamp>true</UseTimeStamp>
    <SignHeader>false</SignHeader>
    <SignAlgorithm>default</SignAlgorithm>
</SecurityParams>
```

Deployment-specific transformations



General deployment transformation overview



References

- L. Gönczy, Zs. Déri, and D. Varró. Model Driven Performability Analysis of Service Configurations with Reliable Messaging. In Proc. of Model Driven Web Engineering Workshop (MDWE) 2008,
- Jerry Preissler & David Bosschaert: Policy Support in Eclipse STP (www.eclipse.org/stp)
- Anish Karmarkar, Ashok Malhotra, David Booz, Service Component Architecture (SCA) Tutorial, 2007.
- L. Gönczy et al. Th04.b, Methodologies for MDA and Deployment, Second version. Deliverable of SENSORIA EU FP6 project, 2008.
- N. LOOKER et al.: SIMULATING ERRORS IN WEB SERVICES
- Alodib&Bordbar: A model-based approach to Fault diagnosis in Service oriented Architectures
- Reniecke, Wolter, Malek: A Survey on Fault-Models for QoS Studies of Service-Oriented Systems
- May Chan, Bishop, Steyn, Baresi and Guinea: A Fault Taxonomy for Web Service Composition