Realization of Models in Programming Languages: Achieving Non-Functional Properties Derived from the Models

Silvia Lizeth Tapia Tarifa

Precise Modeling and Analysis Group
Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo

sltarifa@ifi.uio.no

07.05.2014

Overview

- Software life cycle
- Requirements:
  Functional and non-functional (NFR)
- From design to operation:
  Models, systems, modeling languages & programming languages
- From models to programming languages:
  Example using a representative concrete approach
- Summary

Title of this lecture

Realization of models in programming languages: Achieving non-functional properties derived from the models

Overview

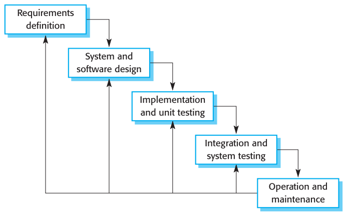
- Software life cycle
- Requirements: Functional and non-functional (NFR)
- From design to operation:
  Models, systems, modeling languages & programming languages
- From models to programming languages:
  Example using a representative concrete approach
- Summary

Software Life Cycle

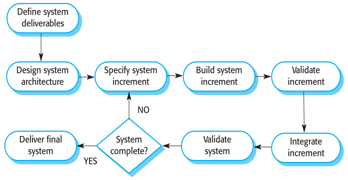
Software life cycle typically includes the following phases:

- Requirements
- Design
- Implementation
- Verification/Validation/Test
- Delivery/Deployment
- Operation & Maintenance

These phases may overlap or be performed iteratively


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Software life cycle



Iterative development process

Source: Software Engineering (7th Edition), Ian Sommerville and ISTQB glossary of testing terms 2.3

Overview

- Software life cycle
- Requirements: Functional and non-functional (NFR)
- From design to operation:
  Models, systems, modeling languages & programming languages
- From models to programming languages:
  Example using a representative concrete approach
- Summary

Requirements

Functional Requirements

- Describe what the system should (and should not) do
- Usually have localized effect (e.g., they affect only the part part of the software addressing the functionality defined by the requirement.)
- Example - consider an online university registration system:
  Students shall be able to apply for courses

Non-functional Requirements (NFRs)

- Describe how the system operates or how the functionality is exhibited
- Example - from the Online University:
  Easy to use, rapid user response, no Heartbleed bug

Source: Software Engineering (7th Edition), Ian Sommerville

More on Non-Functional Requirements (NFRs)

User and system NFRs:

- User NFRs: typically stated in natural language by the clients of a software application (e.g., easy to use)
- System NFRs: typically more detail and precise, it may be part of a contract between developers and clients (e.g., max. training time p.p. is 5h)

Some characteristics of NFRs:

- They are often global and often critical (e.g., aircraft systems)
- User NFRs are usually abstract and informally stated
  (e.g., rapid user response).
- They might conflict with each other
  (e.g., high performance and low budget)
- They might be difficult to validate even after deployment
  (e.g., maintainability)
- They are complex to deal with, etc.

More on Non-Functional Requirements (NFRs)

Classification:

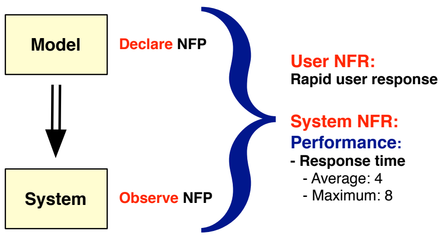
- Product requirement: product behavior (e.g., performance, usability)
- Organizational requirements: policies and procedures (e.g., standards)
- External requirements: external factors (e.g., interoperability, security)

Whenever possible: quantify NFRs
(e.g., performance by means of response time and throughput),
Example: User NFR: Rapid user response,
System NFR: Average response time, maximum response time

Sometimes it is not obvious how to quantify them
(e.g., maintainability)

Predicting Quantifiable Non-Functional Properties (NFPs)

Requirement: A thing that is needed or wanted
Property: An attribute, quality, or characteristic of something.



Model

Declare NFP

System

Observe NFP

User NFR:
Rapid user response

System NFR:
Performance:
- Response time
- Average: 4
- Maximum: 8

Acquire domain-specific information for predicting NFP

"Measurement and modeling are intimately linked because accurate measurement provides the parameter data which models need in order to make valuable predictions"

Source: Non-functional properties in the model-driven development of service-oriented systems, Gilmore *et al.*

Example: for performance:
Where will this application be utilized?
What are the performance features of this environment?, etc

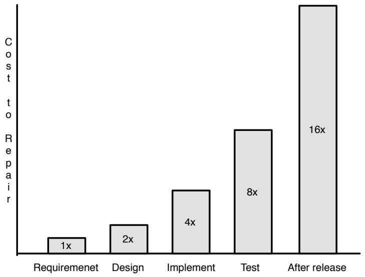
Dealing with Non-Functional Requirements

Product Oriented Approach

- Focus on evaluating the final application to determine whether it satisfy the NFRs
- Most common used approach
- May require redesign

Process Oriented Approach

- Integrates NFRs into the software development process
- Support for languages, methodologies and tools is currently on-going research



Cost to repair: multiplicative increases in cost.

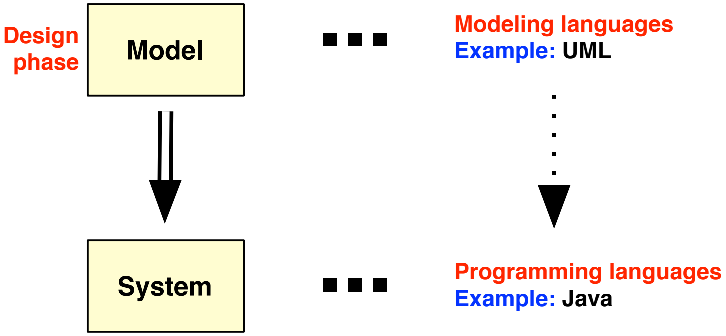
Sources:

- Quantifying Non-Functional Requirements: A Process Oriented Approach, Hill *et al.*
- A Framework for Building Non-Functional Software Architectures, Rosa *et al.*
- Foundation of Software Testing (3rd edition), Black *et al.*

Overview

- Software life cycle
- Requirements: Functional and non-functional (NFR)
- From design to operation: Process oriented approach
  Models, Systems, Modeling languages & Programming languages
- From models to programming languages:
  Example using a representative concrete approach
- Summary

From Design to Operation: Models and Systems



Design phase

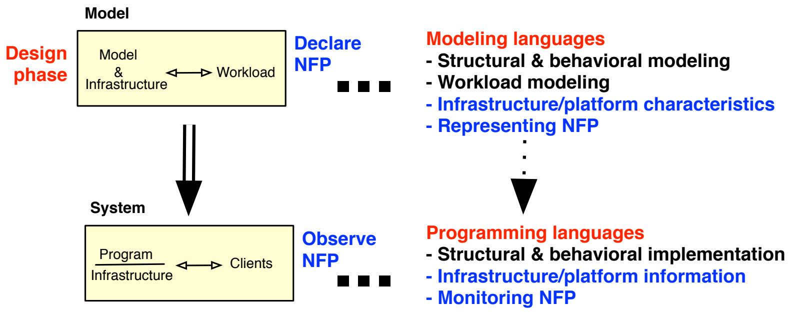
Model

Modeling languages
Example: UML

System

Programming languages
Example: Java

From Design to Operation: NFPs



Model

Design phase

Model & Infrastructure ⟷ Workload

Declare NFP ■ ■ ■

Modeling languages
- Structural & behavioral modeling
- Workload modeling
- Infrastructure/platform characteristics
- Representing NFP

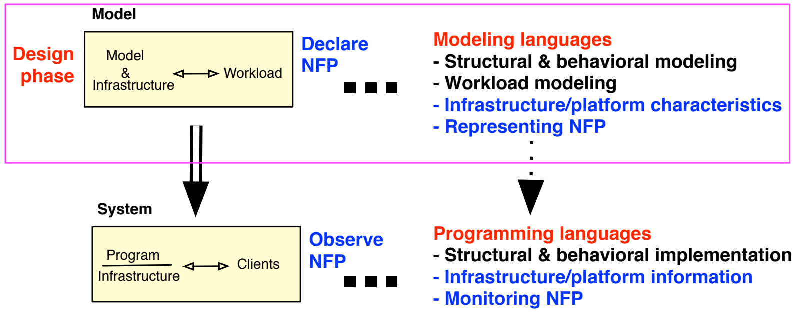
System

Program Infrastructure ⟷ Clients

Observe NFP ■ ■ ■

Programming languages
- Structural & behavioral implementation
- Infrastructure/platform information
- Monitoring NFP

Models and Modeling Languages



Model

Design phase

Model & Infrastructure ⟷ Workload

Declare NFP ■ ■ ■

Modeling languages
- Structural & behavioral modeling
- Workload modeling
- Infrastructure/platform characteristics
- Representing NFP

System

Program Infrastructure ⟷ Clients

Observe NFP ■ ■ ■

Programming languages
- Structural & behavioral implementation
- Infrastructure/platform information
- Monitoring NFP

Examples: Profiles for UML, UPPAAL, VDM++, etc.

Models and Modeling Languages (1)

Profiles for UML: extension mechanism for customizing UML models for particular domains and platforms

Examples:

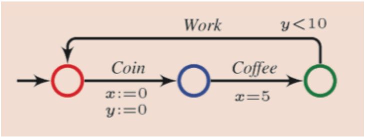
- UML4SOA-NFP:
  UML profile enhancing UML4SOA (a profile for service behavior, service protocols and orchestration) with non-functional properties
- UML profile for MARTE (Modeling and analysis of real-time embedded systems):
  Support for specification, design, and verification/validation of real-time and embedded systems. MARTE focuses on performance and schedulability analysis.
- UML-SPT:
  UML profile for schedulability, performance, and time

Models and Modeling Languages (2)

Timed automata: a finite automaton extended with a finite set of real-valued clocks

Example:

- UPPAAL:
  An integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata



Precisely five time units pass between coin insertion and coffee collection, and the time which passes between coin insertion and going back to work is less than 10 time units

Here x and y are timers representing platform characteristics

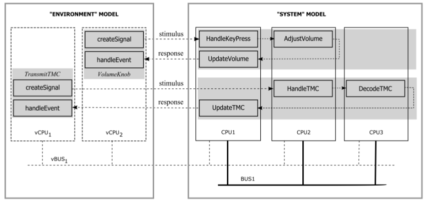
Models and Modeling Languages (3)

Modeling of embedded systems: system is embedded as part of a complete device, often including hardware and mechanical parts

Example:

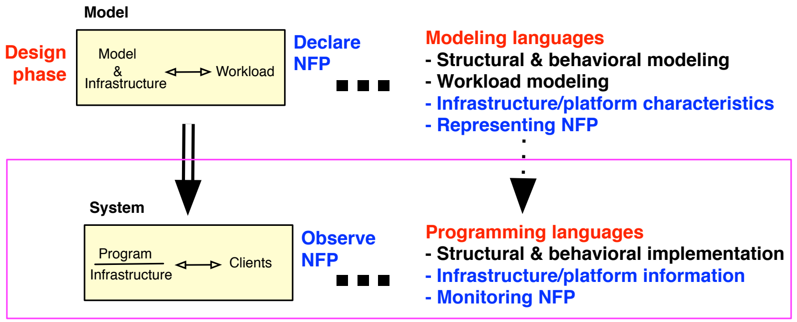
- Modeling and Validating Distributed Embedded Real-Time Systems with VDM++, Verhoef *et al.*, 2006

  Extend VDM with new language elements representing deployment characteristics, to enable the modeling of distributed real-time embedded systems



Buses and CPUs to represent deployment characteristics

Systems and Programming Languages



Declare NFP ■ ■ ■

Modeling languages
- Structural & behavioral modeling
- Workload modeling
- Infrastructure/platform characteristics
- Representing NFP

Observe NFP ■ ■ ■

Programming languages
- Structural & behavioral implementation
- Infrastructure/platform information
- Monitoring NFP

Examples: AspectJ, Java RTS, JRes, reflective middleware, etc.

Systems and Programming Languages (1)

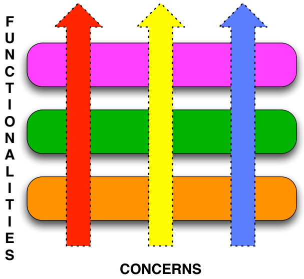
Aspect-oriented programming: programming methods and tools that support the modularization of (crosscutting) concerns at the level of the source code.

Examples:

- Aspect-Oriented Programming with AspectJ, Kiselev, 2003

  An extension of Java to support aspect oriented programming

- An evaluation of aspect-oriented programming for Java-based real-time systems development, Tsang *et al.*,2004



Systems and Programming Languages (2)

Real-time & programming languages:
specification of time in programming languages (e.g., hard deadlines)

Examples:

- An Approach to Platform Independent Real-Time Programming: (1) Formal Description, Hooman and Roosmalen, 2000

  An approach to enable the specification of timing constraints in programs.
  The approach is not language specific and the extension can be included in many existing programming languages.

- Real-Time Java Programming: With Java RTS, Bruno and Bollella, 2009

  Extends Java with various ways to specify time

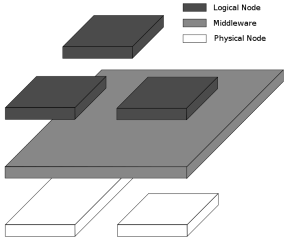
Systems and Programming Languages (3)

Middleware:
support for communication between components deployed in diverse platforms, implemented in different programming languages, etc.

Example:

- An Architecture for Next Generation Middleware, Blair *et al.*, 2009

  Design and implementation for a next generation reflective middleware platform to provide the desired level of configurability and openness



Systems and Programming Languages (4)

Resource-aware programming frameworks

Examples:

- Resource Aware Programming, Moreau and Queinnec, 2005

  A framework which allows users to monitor the resources used by their programs and to express policies for the management of such resources in the program.

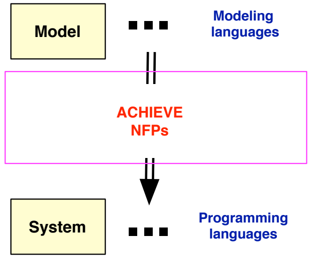
- JRes: A Resource Accounting Interface for Java, Czajkowski and von Eicken, 1998

  A flexible resource accounting interface for Java. The interface allows to account for heap memory, CPU time, and network resources consumed by individual threads or groups of threads.

Overview

- Software life cycle
- Requirements: Functional and non-functional (NFR)
- From design to operation: process oriented approach
  Models, Systems, Modeling languages & Programming languages
- From models to programming languages:
  Example using a representative concrete approach
- Summary

From Models to Programming Languages


Model

Modeling languages

ACHIEVE NFPs

System

Programming languages

Representative Example from On-Going Research in Software Engineering Practices (2011)

Non-functional properties in the model-driven development of service-oriented systems

Stephen Gilmore and László Gönczy and Nora Koch and Philip Mayer and Mirco Tribastone and Dániel Varró

Journal in Software & Systems Modeling, 2011

A model-driven approach for the development of service-oriented systems with explicit support for the specification of non-functional properties

Outline

- Service oriented architecture (SOA)
- Model driven development (MDD)
- High-level understanding of the approach

Service Oriented Architecture (SOA)

About SOA:

- Pattern for designing software and software architecture
- Separate functions into distinct software units called services
- Allow users to combine functionalities to form ad hoc web-based applications built almost entirely from existing software services

- Define how to integrate widely disparate applications for a web-based environment (independent of any vendor, product or technology)
- Aim at a loose coupling of services by means of the orchestration
- Orchestration:
  describe the arrangement and coordination of the different services

Service Oriented Architecture (SOA)

About the services:

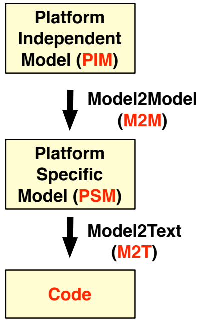
- Each service is designed to perform one or more functionalities
- Services are offered through interfaces
- The service interface describes the set of interactions supported by a service

- Service descriptions are published by service providers and services are invocable by a service requester according to a set of access policies

About the example approach:

- The orchestration is also defined as a service
- Modeling of NFP as contracts associated to the services
- NFP: security, performance and reliable connection

Model Driven Development (MDD)



Platform Independent Model (PIM)
Model2Model (M2M)
Platform Specific Model (PSM)
Model2Text (M2T)
Code

- In MDD, models are the predominant artifacts of the development process.
- MDD process consists of a chain of model transformations which starts with the models of the application (so-called PIM) and ends with a (sort of) code generation
- MDD uses different languages: modeling languages for the specification of the applications, and model transformation languages required for generating other models or code.

Modeling of Service Oriented Systems (SOS) - Approach

For functional requirements:

- SoaML: UML profile for describing the structure of SOS
  UML4SOA(proposed): SoaML + behavioral modeling + orchestration

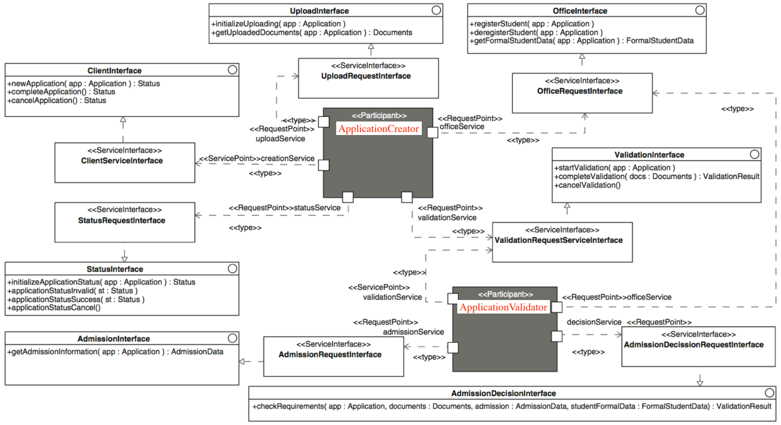
For non-functional requirements:

- UML4SOA-NFP(proposed): UML4SOA + NFP
- Some NFPs can be directly implemented by using web service standards (e.g., reliable messaging, security, logging, etc.)
  other NFPs are effected by the underlying platform (e.g., performance)
- For NFPs affected by the underlying platform:
  MARTE: UML + performance requirements annotations
  PEPA: quantitative analysis
- For the WS-standards: generation of deployment descriptors (XML files) based on standards (e.g., WS-Security, WS-ReliableMessaging, WS-Reliability)

Running Example: eUniversity Case Study

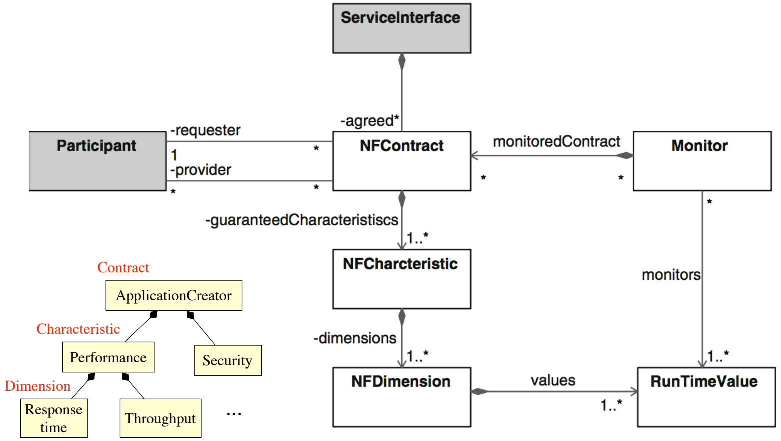
- eUniversity: all courses and paperwork are handled online
- Example focus: processing of a student application for a course of study
- Scenario: eUniversity website acts as a client to a service providing the functionality for handling a student application
- *ApplicationCreator*(Service): this functionality requires the orchestration of a set of different external services, e.g. student office, a service for the upload of documents, and a service to check the application (validation service)
- *ApplicationValidator*(Service): is itself also an orchestration of other services

eUniversity (UML4SOA)


UploadInterface
+initializeUploading(app : Application)
+getUploadedDocuments(app : Application) : Documents
OfficeInterface
+registerStudent(app : Application)
+deregisterStudent(app : Application)
+getFormalStudentData(app : Application) : FormalStudentData
ClientInterface
+newApplication(app : Application) : Status
+completeApplication() : Status
+cancelApplication() : Status
<<ServiceInterface>>
UploadRequestInterface
<<ServiceInterface>>
OfficeRequestInterface
<<type>>
<<type>>
<<RequestPoint>>
uploadService
<<Participant>>
ApplicationCreator
<<RequestPoint>>
officeService
<<type>>
<<ServiceInterface>>
ClientServiceInterface
<<ServicePoint>>creationService
<<type>>
ValidationInterface
+startValidation(app : Application)
+completeValidation(docs : Documents) : ValidationResult
+cancelValidation()
<<ServiceInterface>>
StatusRequestInterface
<<RequestPoint>>statusService
<<type>>
<<RequestPoint>>
validationService
<<type>>
<<ServiceInterface>>
ValidationRequestServiceInterface
<<type>>
StatusInterface
+initializeApplicationStatus(app : Application) : Status
+applicationStatusInvalid(st : Status)
+applicationStatusSuccess(st : Status)
+applicationStatusCancel()
<<ServicePoint>>
validationService
<<Participant>>
ApplicationValidator
<<RequestPoint>>officeService
<<RequestPoint>>
admissionService
decisionService <<RequestPoint>>
AdmissionInterface
+getAdmissionInformation(app : Application) : AdmissionData
<<ServiceInterface>>
AdmissionRequestInterface
<<type>>
<<type>>
<<ServiceInterface>>
AdmissionDecissionRequestInterface
AdmissionDecisionInterface
+checkRequirements(app : Application, documents : Documents, admission : AdmissionData, studentFormalData : FormalStudentData) : ValidationResult

ApplicationCreator: orchestration with student office, service for the upload of documents, *ApplicationValidator*, etc.

Extension: Adding NFP to SOS models (UML4SOA-NFP Metamodel)

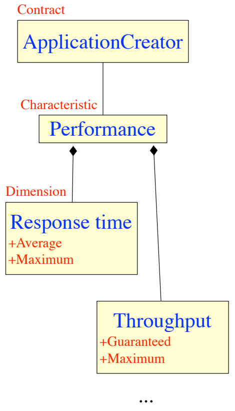


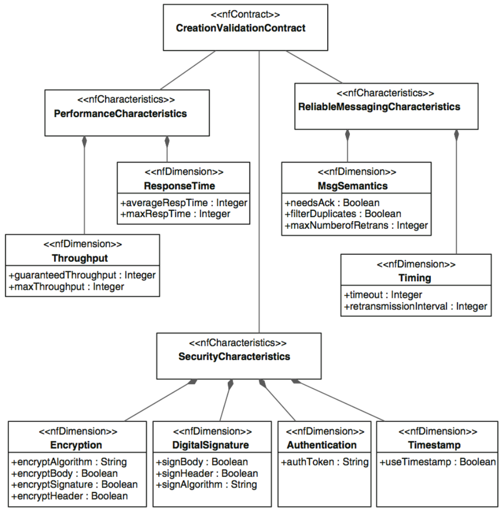
Running Example: eUniversity Case Study (NFR)

- The Client and the *ApplicationCreator* should communicate via a secure and reliable connection
- The document *UploadService* might be under heavy workload, therefore its throughput should be at least 10 requests/second with a 4s average response time
- All requests sent to the *ApplicationValidator* should be acknowledged
- As the validation service handles confidential data, all requests should be encrypted in order to protect the privacy of the students

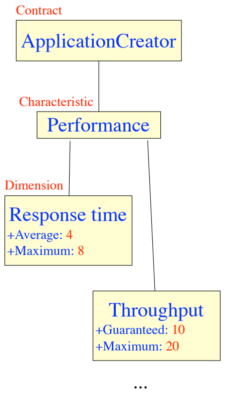
NFP for security, reliable connections and performance

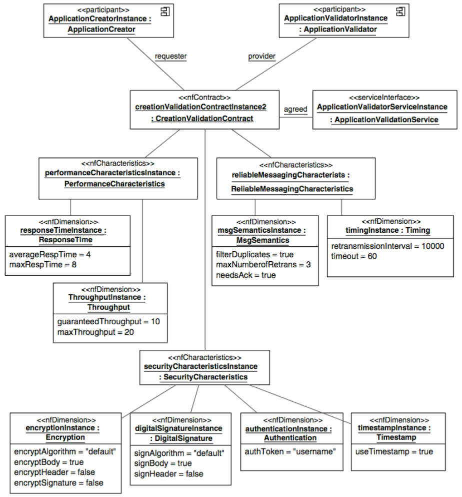
Adding NFP to eUniversity (UML4SOA-NFP)





Adding NFP to eUniversity: Concrete Configuration

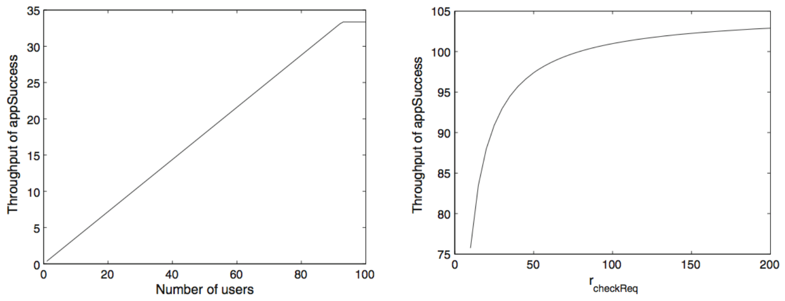




Early Estimation & Evaluation of Performance - Approach

- Automatic translation from UML4SOA-NFP and MARTE models into PEPA (as system equations)
- MARTE models include workloads and the execution rate (measurements) of actions
- PEPA is a formal language which allows the definition of models as a composition of interacting automata
- For the quantitative analysis, PEPA models are interpreted as continuous-time Markov chains

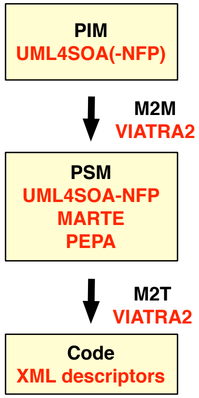
eUniversity: Performance Evaluation



Fixed rates, varying workload (Left): Workload analysis studies how the user population affects performance of the system. Non-degrading performance is observed for population sizes less than 93

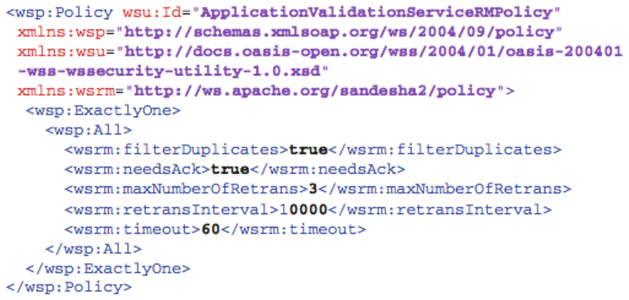
Fixed workload, varying rates (right): Increasing the activity rate corresponds to an increase in the system performance. Although the relationship is not linear. For the example an optimal gain is obtained for values around 50. Further increases, give smaller and smaller improvement.

Automating Service Deployment by Model Transformation



- Automated Transformations were implemented in the VIATRA2 framework
- VIATRA2: tool that supports the design and execution of model transformations
- Transformations are defined by graph transformation rules and abstract state machines
- NFP are captured at a low implementation-level by using dedicated XML deployment descriptors
- PIM models: input UML4SOA(-NFP) Profile
- PSM models: internal service models are generated within the model transformation tool. These are then processed in order to create descriptor models
- Target XML files: descriptor models are the basis of XML file generation. XML files are directly usable as configuration descriptors on standard platforms

eUniversity: Deployment Descriptor Fragment in XML



```xml
<wsp:Policy wsu:Id="ApplicationValidationServiceRMPolicy"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401
 -wss-wssecurity-utility-1.0.xsd"
 xmlns:wsrm="http://ws.apache.org/sandesha2/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsrm:filterDuplicates>true</wsrm:filterDuplicates>
      <wsrm:needsAck>true</wsrm:needsAck>
      <wsrm:maxNumberOfRetrans>3</wsrm:maxNumberOfRetrans>
      <wsrm:retransInterval>10000</wsrm:retransInterval>
      <wsrm:timeout>60</wsrm:timeout>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Overview

- Software life cycle
- Requirements: Functional and non-functional (NFR)
- From design to operation: process oriented approach
  Models, Systems, Modeling languages & Programming languages
- From models to programming languages:
  Example using a representative concrete approach
- Summary

Summary

- Achieving NFRs derived from models is an on-going research field
- NFRs are often global, critical, not compositional and might conflict with each other
- For achieving NFPs derived from models a process oriented approach is needed
- Modeling languages need a way to represent infrastructure/platform characteristics for some NFPs
- For quantitative NFPs, system measurements are needed to make predictions (e.g., for performance these measurements capture the infrastructure/platform characteristics)
- Programming languages need a way to obtain infrastructure/platform information for some NFPs
- Monitors could be used to observe that systems respect NFPs (contracts)
- We have looked at a concrete example from a representative approach to an on-going research topic

Main Sources

Software Engineering (7th Edition),
Ian Sommerville, 2004

Foundations of Software Testing ISTQB Certification (3rd edition),
Rex Black and Erik van Veenendaal and Dorothy Graham, 2012

Non-functional Properties in the Model-Driven Development of Service-Oriented Systems,
Stephen Gilmore and László Gönczy and Nora Koch and Philip Mayer and Mirco Tribastone and Dániel Varró,
Journal of Software and Systems Modeling, 2011

A Framework for Building Non-functional Software Architectures,
Nelson S. Rosa and George R. R. Justo and Paulo R. F. Cunha,
ACM Symposium on Applied Computing, 2001

THANK YOU