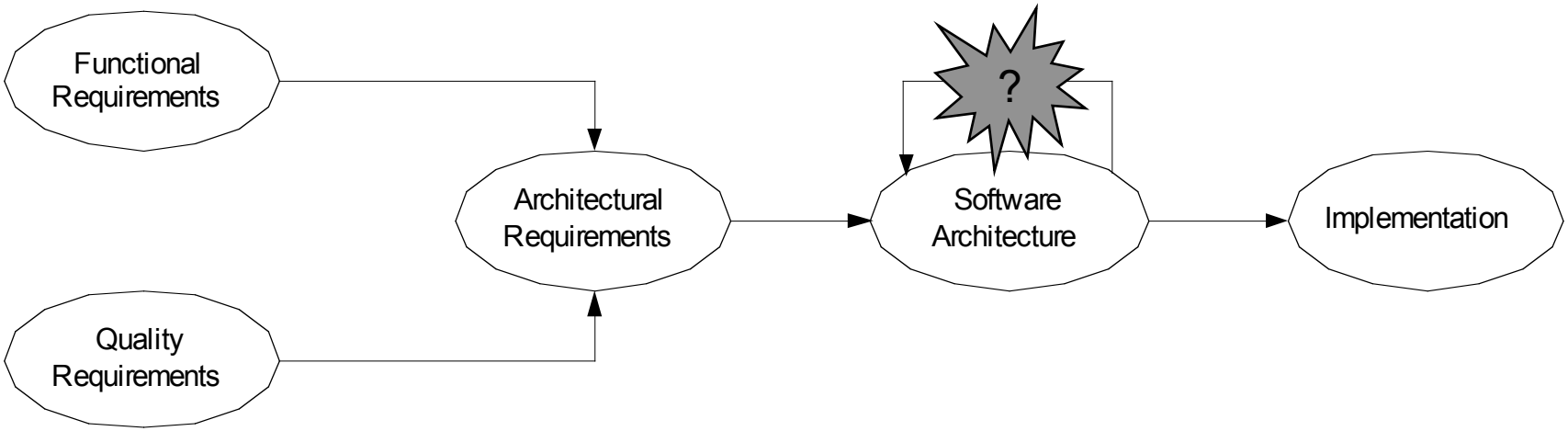
AARHUS UNIVERSITET

Architectural Evaluation

Overview

Questioning & Measuring Techniques

The (Very) Simplified Development Process Again



NB!

- Creative, iterative, incremental, parallel, … process
- Architecture-Business Cycle

What *can* you reasonably evaluate given a software architecture (description)? How?

What are the limits of architectural evaluation?

Rationale for Architectural Evaluation

Software architecture allows or precludes almost all system quality attributes

Architecture needs to be designed early

- E.g., prerequisite for work assignment
- Cf. RUP

Architecture embodies fundamental design decisions

- Hard/costly to change these
- The later the change, the costlier it is

Need to evaluate impact of architectural design on quality attributes

- *Should* be possible…

Cheap techniques for architectural evaluation exist

When?

Typically when architecture is designed, design not implemented

- I.e., early in a cycle in an iteration
- E.g., “We have designed this architecture will it possibly fit the quality requirements?”

Could also be earlier…

- E.g., “Will any architecture meet these quality requirements?”
  - “You can have any combination of features the Air Ministry desires, so long as you do not also require that the resulting airplane fly” (Messerschmitt)
- Evaluate design decisions
  - Either design decisions made or considered
  - Architecture description may not be available
- Completeness and fidelity is product of state of architectural description
  - Thus evaluations may be (more) misleading in this phase

Could also be later…

- E.g., ”We have these requirements; could this COTS system be customized to fulfill them?”
- Evaluate architecture for existing system
  - Possible purchase, redesign, evolution, …

Who?

Roles

- Evaluation team
  - Preferably not process staff
    - Objectivity reasons
  - Often architect himself…
- Project stakeholders
  - Articulate requirements
  - “Garden variety” and decision makers
  - Producers
    - Software architect, developer, maintainer, integrator, standards expert, performance engineer, security expert, project manager, reuse czar, …
  - Consumers
    - Customer, end user, application builder (for product line), persons in problem domain, …
  - Servicers
    - System administrator, network administrator, service representatives, …

Needs depend on the actual type of evaluation

- From little stakeholder involvement to continuous involvement

Overview


Architecture Analysis
questioning techniques
what?
any project area
any state of completeness
types
questionnaires
open questions
process
product
apply to all architectures
checklists
detailed questions
based on experience
usually domain-specific
scenario-based methods
scenario =
description of interaction with system from point of view of stakeholder
(Quality Attribute Workshops (QAWs))
Architecture Tradeoff Analysis Method (ATAM)
Software Architecture Analysis Method (SAAM)
Active Reviews for Intermediary Design (ARID)
Survivable Network Analysis method (SNA)
Cost Benefit Analysis Method (CBAM)
quality-based methods
architectural Software Quality Assurance (aSQA)
measuring techniques
what?
need artifact to do measures on
focus on specific quality attributes
types
metrics
quantitative interpretations
based on particular observable measurements
simulations, prototypes, experiments
automated tools/ADLs
adapted techniques
Rate-Monotonic Analysis

Questioning Techniques

AARHUS UNIVERSITET

Analytical

- Any project artifact
- Typically in the form of a review
- Any state of completeness

Subtypes

- Questionnaires
- Checklists
- Scenario-based methods

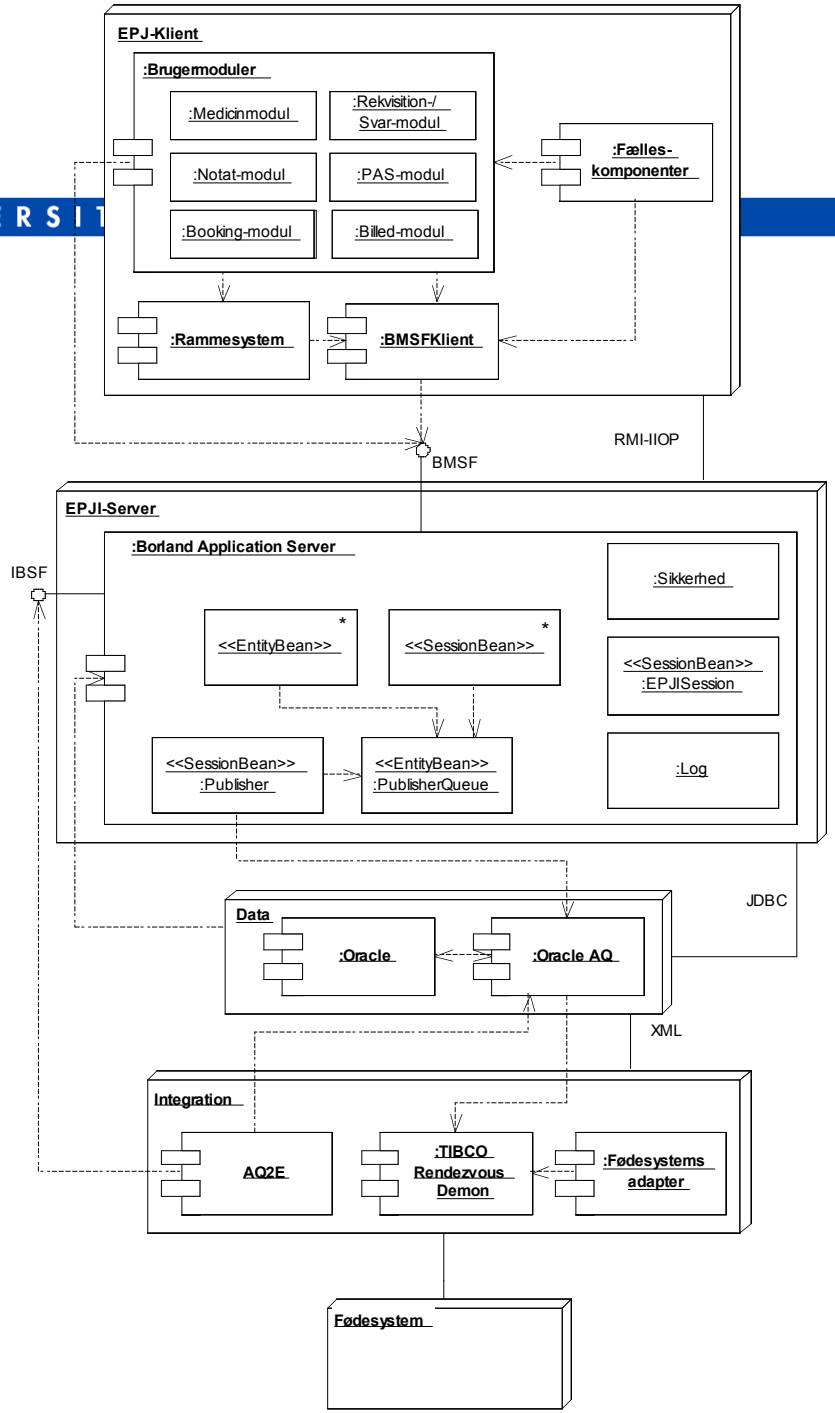
Questionnaires

List of relatively open questions
- Apply to all architectures

Process and product questions
- Cf. Capability Maturity Model (CMM) for software

E.g., Kruchten checklist of typical risks

- The architecture is forced to match the current organization. For example, is there a database built into the design because one is needed or because there is a database group that's looking for work?
- There are more than 25 top-level architecture components. This design will likely be too complex for the architects to maintain intellectual control over, let alone the developers to whom they hand it.
- One requirement drives the rest of the design. In one system we evaluated, the requirement for ultra-high availability was the primary impetus behind the architecture. When that requirement was eventually relaxed, the architecture was then much too complex for the task at hand. Focusing on a single requirement tends to give others short shrift.
- The architecture depends on alternatives in the operating system. This makes the architecture mortally vulnerable to inevitable operating system upgrades; this obvious design flaw occurs surprisingly often.
- Proprietary components are being used when standard components would do. This renders the architecture dependent on a single supplier.
- The component definition comes from the hardware division. Hardware changes as systems evolve, and hardware components tend to merge into more general-purpose processors or split into special-purpose devices. Keeping the software independent of the hardware organization insulates it from these changes.
- There is redundancy not needed for reliability. For instance, the presence of two databases, two start-up routines, or two error locking procedures suggests that the designers could not agree, resulting in unnecessary complexity and a system much harder to maintain.
- The design is exception driven; the emphasis is on the extensibility and not core commonalities.
- The development unit is unable to identify an architect for the system.


EPJ-Klient
:Brugermoduler
:Medicinmodul
:Rekvisition-/ Svar-modul
:Notat-modul
:PAS-modul
:Booking-modul
:Billed-modul
:Fælles-komponenter
:Rammesystem
:BMSFKlient
BMSF
RMI-IIOP
EPJI-Server
:Borland Application Server
IBSF
:Sikkerhed
<<EntityBean>>
<<SessionBean>>
<<SessionBean>> :EPJISession
<<SessionBean>> :Publisher
<<EntityBean>> :PublisherQueue
:Log
JDBC
Data
:Oracle
:Oracle AQ
XML
Integration
AQ2E
:TIBCO Rendezvous Demon
:Fødesystems adapter
Fødesystem

Checklists (1)

Detailed questions

- Domain-specific
- Based on experience
- E.g., a checklist used when integrating information system [Hansen and Christensen, 2004]

Development issues	
Issues	**Questions**
Technical Platform	Which technical platform is used by the component/the receiving platform? This may, e.g., be component architectures (J2EE, .NET etc.) or communication technologies (Java RMI, CORBA, publish/subscribe etc.)
Domain Model	Which assumptions does the component have about data modelling of its domain? For example: if a component is manipulating medical data of patients, the data needs to be compatible to a model that the receiving system understands
Services	Is the functionality of the component sufficient for the requirements that the receiving system has? For example: a specific date component enables visual picking of dates in the receiving system, but the date component cannot be configured with public holidays, which is important for the receiving system
Source Code	Which programming languages have been used in each system?
Interface/Provided	Is the interface of the component compatible with the requirements of the receiving system? This includes aspects such as method signatures and data format of parameters
Interface/Required	Which requirements does the components have towards the functionality of the receiving system? Specifically, which interfaces are provided by the system? The same issues as above are relevant
Persistence	What does the component require in order to store and retrieve data? Are these requirements compatible with the receiving system?
Build Management	How is the build process handled, when the component is changed? Is this process compatible with the one of the receiving system? For example: the component is built using make scripts and a fixed directory structure whereas the components of the receiving system are built using and integrated development environment that expects another structure
User Interface	Which architectures have been used for building the graphical user interfaces? Are these compatible? Technical incompatibilities as well as aesthetical incompatibilities may be of importance

Checklists (2)

Deployment issues	
Issues	**Questions**
Operation	How is the component to be deployed? Does the deployment structure fit the deployment structure of the receiving system? For example: Does the component consist of a number of binary components that are supposed to run in a web client and on an application server whereas the receiving platform does not have a web interface.
Concurrency	Which threading model has been used by the components? Is this compatible with the model of the receiving system. Subdimensions are synchronization and thread handling. For example: the receiving system has a single-threaded architecture, whereas the component to be reengineered is multi-threaded
Protocol	Which dynamic protocol is used by the component? Is this compatible with the dynamic protocol of the receiving system? For example: the receiving platform assumes that interaction with the component is done via method calls, but the component assumes callbacks using an Observer pattern
Packaging	Which physical form does the component have? Is this form compatible with the form used in the receiving system? For example: The component is contained in a single JAR archive or a number of DLLs that need to be deployed on a number of different machines

Organizational issues	
Issues	**Questions**
Ownership	Has module source code ownership been determined? What model has been chosen?
Maintenance	Has a procedure been established for how changes and bugfixes are communicated between organizations? Is the component a single basedlined source or has the development been split?
Division of Work	Has the responsibility for the reengineering effort been defined between the component vendor and the receiving framework vendor?

Scenario-Based Methods

Scenarios

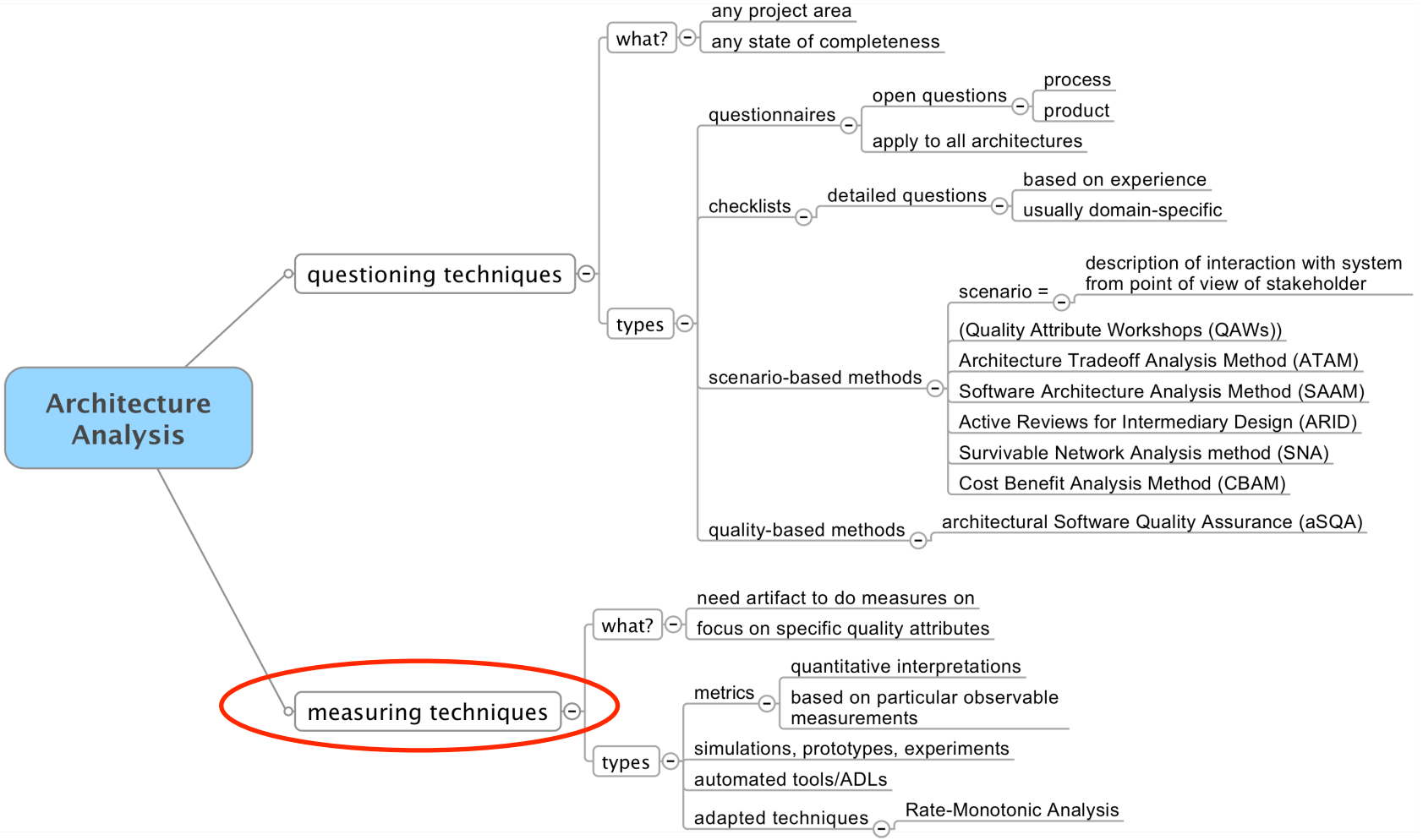
- Description of interaction with system from the point of with of a stakeholder
  - System-specific – developed as part of project
  - Cf. quality-attribute scenarios as in [Bass et al., 2003]

Types

- *Quality Attribute Workshops (QAW)*
  - Structured way of involving stakeholders in scenario generation and prioritization
- *Architecture Tradeoff Analysis Method (ATAM)*
  - Creates *utility trees* to represent quality attribute requirements
  - Analysis of architectural decisions to identify sensitivity points, tradeoffs, and risks
- Software Architecture Analysis Method (SAAM)
  - Brainstorming of modifiability and functionality scenarios
  - Scenario walkthrough to verify functionality support and estimate change costs
- Active Reviews for Intermediary Designs (ARID)
  - Active Design Review of software architecture
    - E.g., “Is the performance of each component adequately specified” vs. “For each component, write down its maximum execution time and list the shared resources that it may consume”
- Survivable Network Analysis method (SNA)
  - Focus on survivability as quality attribute
  - Process
    - Determine essential components based on essential services and assets
    - Map intrusion scenarios onto architecture to find “soft-spot” components (essential, but vulnerable)
    - Analyze these wrt
      - » Resistance
      - » Recognition
      - » Recovery

      of/from attacks

Overview


Architecture Analysis
questioning techniques
what?
any project area
any state of completeness
types
questionnaires
open questions
process
product
apply to all architectures
checklists
detailed questions
based on experience
usually domain-specific
scenario-based methods
scenario =
description of interaction with system from point of view of stakeholder
(Quality Attribute Workshops (QAWs))
Architecture Tradeoff Analysis Method (ATAM)
Software Architecture Analysis Method (SAAM)
Active Reviews for Intermediary Design (ARID)
Survivable Network Analysis method (SNA)
Cost Benefit Analysis Method (CBAM)
quality-based methods
architectural Software Quality Assurance (aSQA)
measuring techniques
what?
need artifact to do measures on
focus on specific quality attributes
types
metrics
quantitative interpretations
based on particular observable measurements
simulations, prototypes, experiments
automated tools/ADLs
adapted techniques
Rate-Monotonic Analysis

Measurement Techniques

May answer specific quality attribute scenarios
- Cf. experimental vs. exploratory prototyping

Prerequisite
- Artifacts to do measurements on…

Types
- Metrics
- Simulation, prototypes, experiments
- Domain-specific analyses
  - E.g., Rate-Monotonic Analysis
- ADL-based

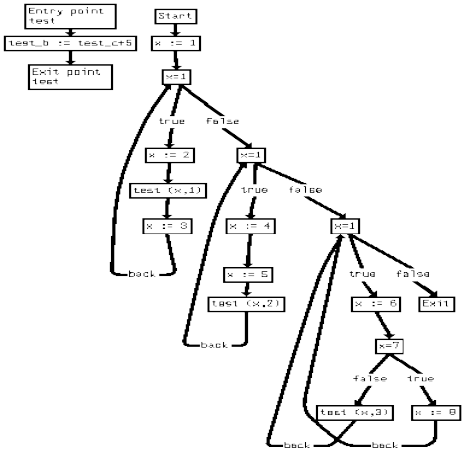
Metrics

Quantitative interpretation of observable measurement of software architecture

- Cf. the ISO/IEC 9126 quality standard

E.g., measuring complexity to predict modifiability

- Real-time object-oriented telecommunication systems
  - Number of events reacted to
  - Number of {asynchronous, synchronous} calls made
  - Number of component clusters
    - Object decompositions units, e.g., car as wheels, transmission, steering, …
  - Depth of inheritance tree
  - …



E.g., flow metrics to predict reliability

- Call-graph-metrics
  - Number of modules used by module
  - Total calls to others modules
  - Unique calls to other modules
- Control-flow metrics
  - Number of if-then conditional arcs
  - Number of loop arcs
  - Cyclomatic complexity

given control flow graph

Simulations, Prototypes, Experiments

Architectural prototyping: [Bardram et al., 2004]

- More later...

Domain-Specific: Rate-Monotonic Analysis

Static, quantitative analysis

- Ensuring dependability in hard real-time systems

Preemptive multitasking, execute task with highest priority

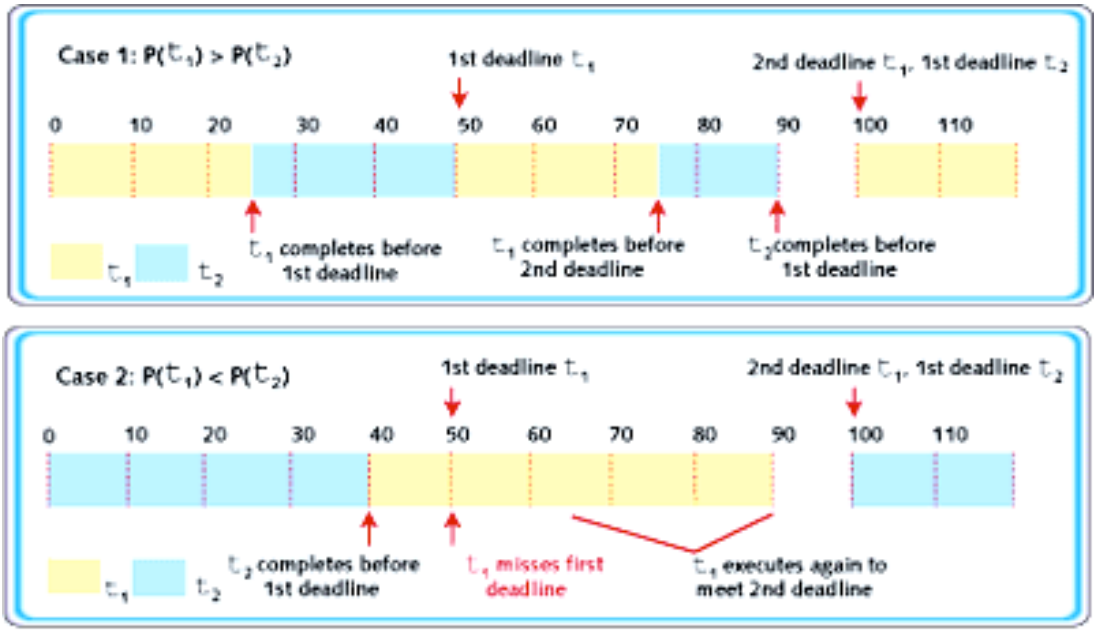
Basic idea

- Assign priority to each process according to its period
  - The shorter the period/deadline, the higher the priority

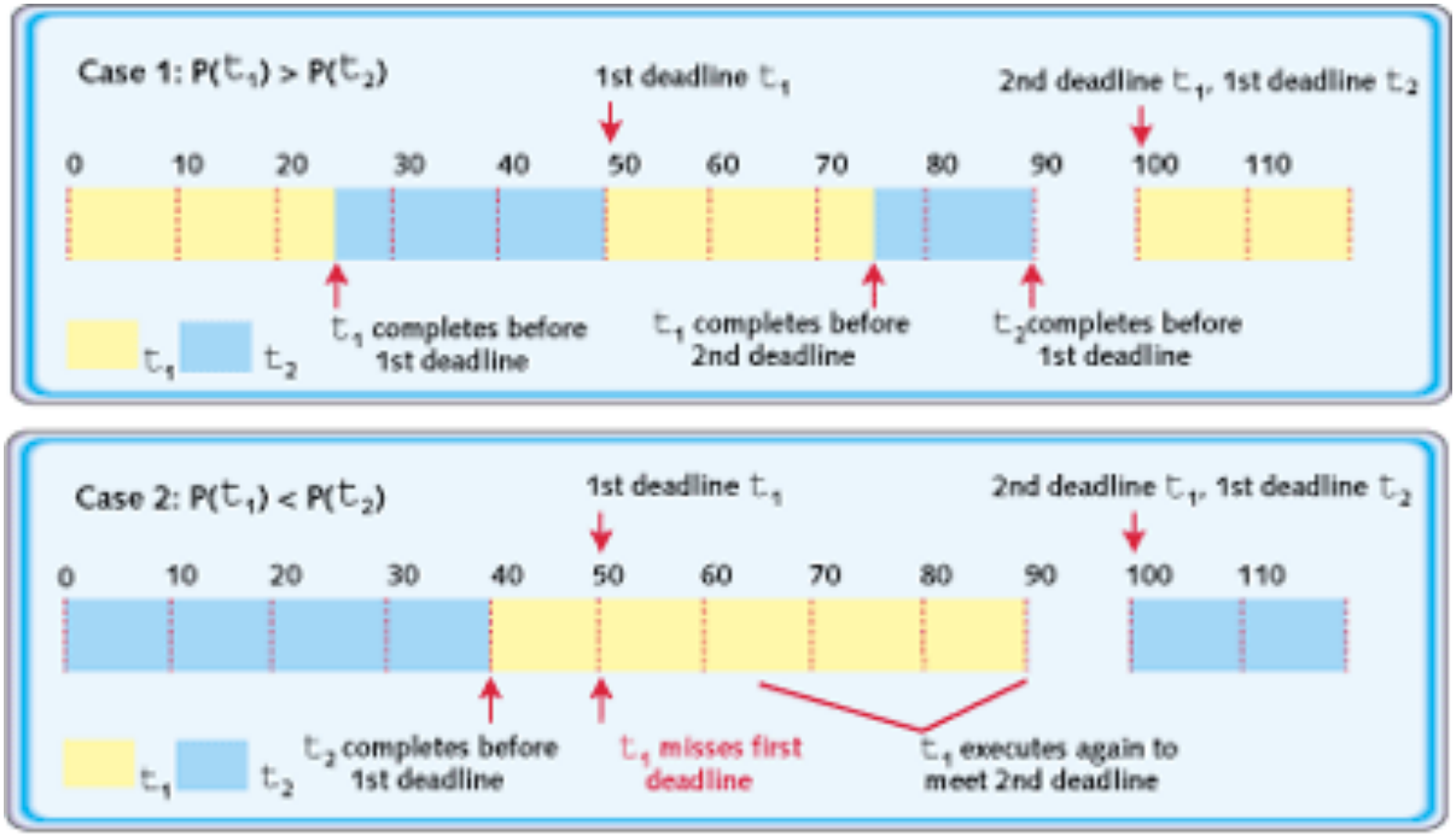
RMA ensures schedulability of processes

- No process misses execution deadline
- Worst-case schedule bound: W(n) = n * (2^(1/n) - 1)
  - ln 2, n -> oo

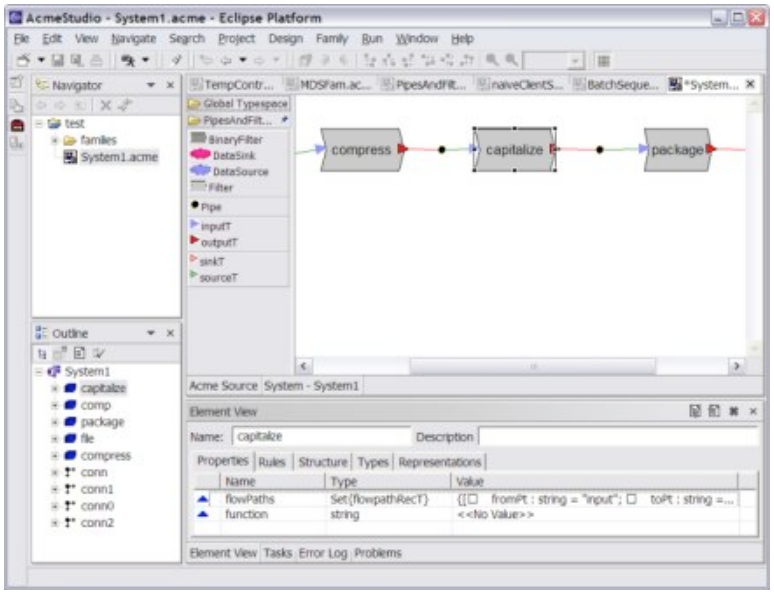
Guarantees schedulability if concurrency model is observed in implementation



Rate-Monotonic Analysis



Automated Tools



Programming language-based

- ArchJava

UML-based

- Simulation
- Semantic checks
- Code generation to ensure conformance
- …

ADL-based

[Medvidovic & Taylor, 2000]

AcmeStudio

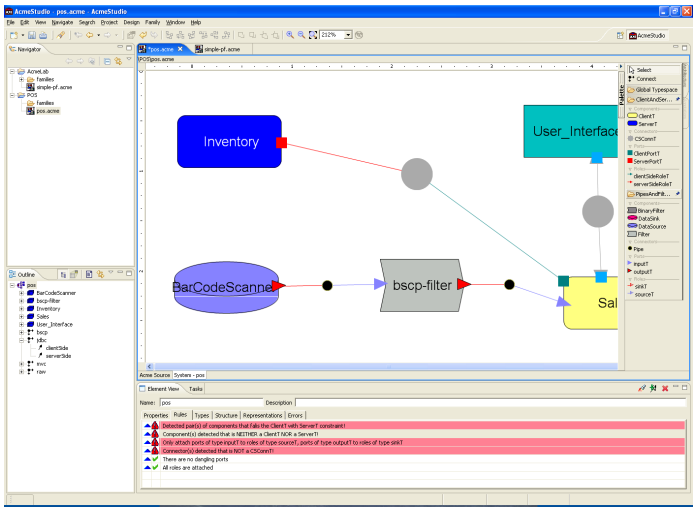
Graphical editor for architectural designs

- Eclipse plug-in perspective
- Nice tutorials available

Based on the *Acme ADL*

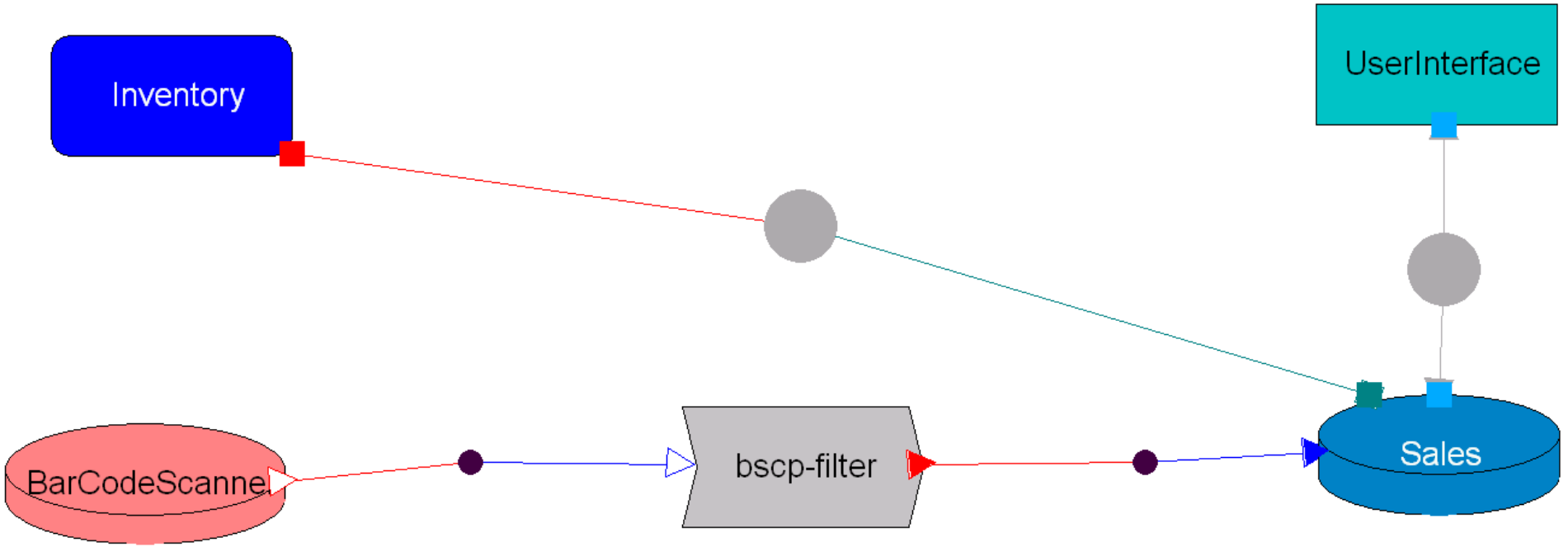
- Generic representation of structures common to many ADLs
  - Lowest common denominator…
- Focus on *styles*

Armani constraint checker of architectural rules integrated



We will meet AcmeStudio again in the lecture on *architectural reflection*

ACME Example (1)



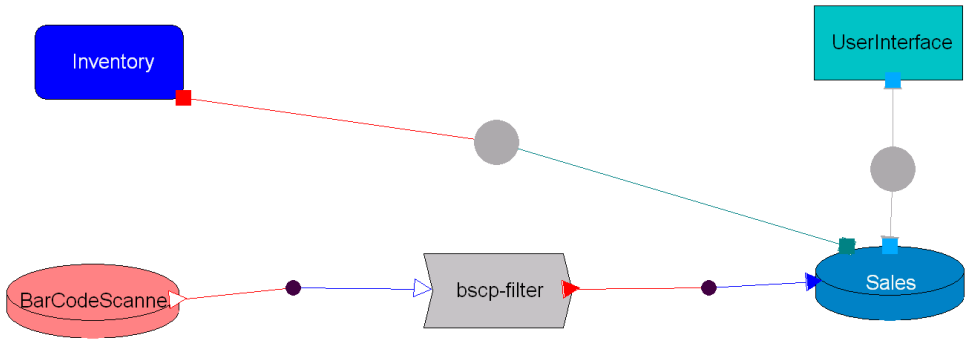
ACME Example (2)


```
import $AS_GLOBAL_PATH/families/PipesAndFiltersFam.acme;

import $AS_GLOBAL_PATH/families/ClientAndServerFam.acme;

System POS : PipesAndFiltersFam, ClientAndServerFam = new PipesAndFiltersFam, ClientAndServerFam extended with {

    Component BarCodeScanner : DataSource = new DataSource extended with {
    }
    Component bscp-filter : Filter = new Filter extended with {
    }
    Component Sales : ClientT, DataSink = new ClientT extended with {
    }
    Component Inventory : ServerT = new ServerT extended with {
    }
    Component UserInterface  = {
    }
    Connector Pipe0 : Pipe = new Pipe extended with {
    }
    Connector Pipe1 : Pipe = new Pipe extended with {
    }
    Connector CSConnT0 : CSConnT = new CSConnT extended with {
    }
    Connector Connector0  = {
    }
    Attachment Sales.inputT0 to Pipe1.source;
    Attachment Sales.p0 to Connector0.role0;
    Attachment UserInterface.Port0 to Connector0.role1;
    Attachment bscp-filter.input to Pipe0.source;
    Attachment BarCodeScanner.output to Pipe0.sink;
    Attachment bscp-filter.output to Pipe1.sink;
    Attachment Inventory.receiveRequest to CSConnT0.serverSide;
    Attachment Sales.sendRequest to CSConnT0.clientSide;
}
```



Summary

Wide range of evaluation approaches exist

- Questioning techniques
- Measuring techniques

Establishing *suitability* of architecture as main goal

- Does architecture fulfill quality goals?
- Is architecture buildable within project constraints?

Utility of approaches dependent on context

- Project state
- Expertise
- Tools used
- Domain
- …