Overview of Software Architectures (SA)

Barbora Bühnová

Faculty of Informatics, Masaryk University
Brno, Czech Republic

LaSArIS Seminar

October 1, 2009

Aim of the talk

Three levels of topics

- Concepts = general principles
- Technologies = realization of concepts
- Tools = support of the technologies

This talk discusses

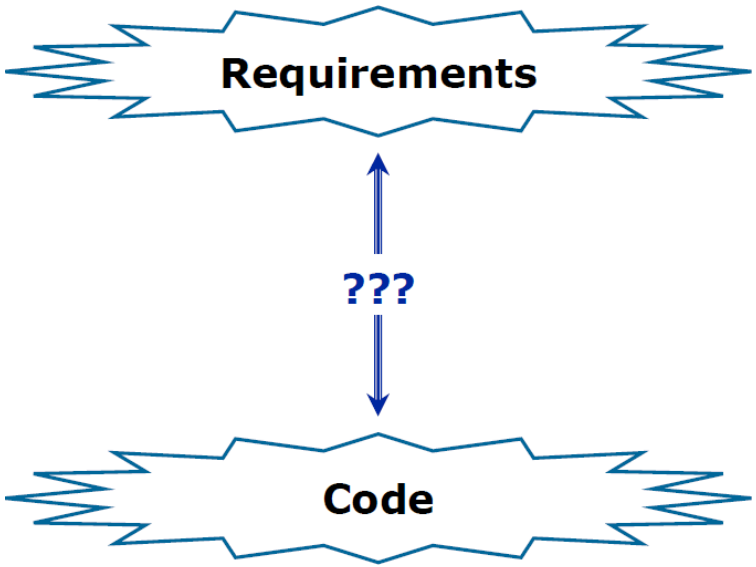
- Concepts of software architectures
- Topics related to software architectures

Notice: Some of the slides are inspired by Software Architecture lectures of Uni Karlsruhe, ©R. Reussner

The problem

Why software architectures?

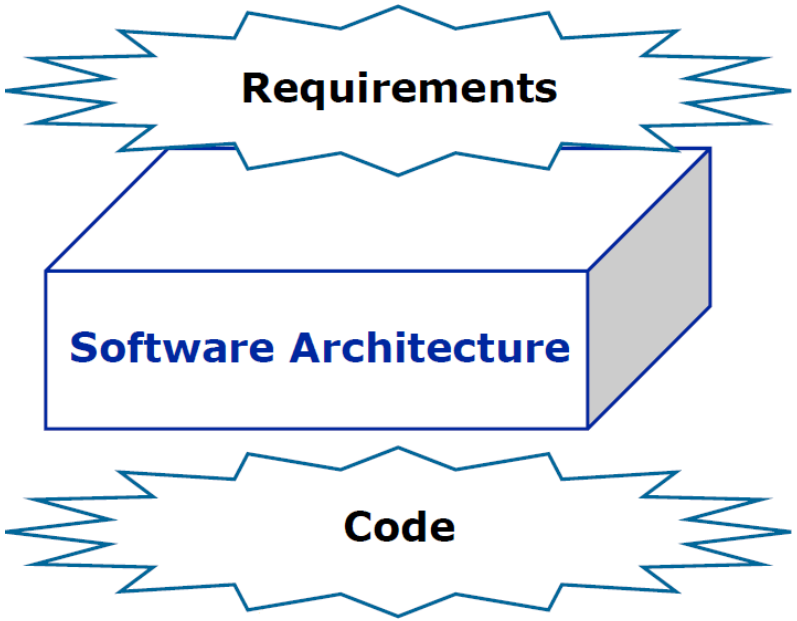
- How to bridge the gap between requirements and code?



The solution

The role of software architecture

- System-level abstractions
- Coarse-grained structure of the system



What constitutes a Software architecture?

Three core views that need to be described

- Module view = system components
  i.e. computational software units, often concurrent (tasks, threads)
- Connector view = communication styles
  e.g. pipe-and-filter, shared-data, publish-subscribe, client-server (synchronous vs. asynchronous)
- Allocation view = mapping to hardware (or software) resources

Advantages of an explicit Architecture

Stakeholder communication

- Architecture may be used as a focus of discussion by system stakeholders

System analysis

- Prediction of the quality attributes of the architecture

Large-scale reuse

- The architecture may be reusable across a range of systems
- Existing components can be considered during design

Project planning

- Cost estimation, mile-stone organisation, dependency analysis, etc.

Architecture development

Architecture design

- Developer roles, development process, correctness by construction

Architectural patterns

- Layers, Peer-to-Peer, Client-Server, Pipe & Filter, ...

Middleware architectures

- Run-time environments realizing architectural concepts

Architecture design and evaluation

Modelling of SW architectures

- Component, Class, Sequence and Deployment diagrams of Unified Modelling Language (UML)

Documenting SW architectures

- Interface descriptions

Evaluation of SW architectures

- Qualitative vs. quantitative properties

Concepts in software architectures I.

Component-based development

- System assembly out of prefabricated components

Service-oriented architecture (SOA)

- Autonomy of services, loose coupling, interoperability, outsourcing

Aspect-oriented architecture

- Integration of crosscutting concerns

Model-driven architecture (MDA)

- M2M transformations and completions

Concepts in software architectures II.

Embedded control systems

- Role of sensors and actuators

Software product lines

- Families of products with similar core, variation points

Dynamic and self-adapting architectures

- Reaction to changes in system usage, fault tolerance

Components around us

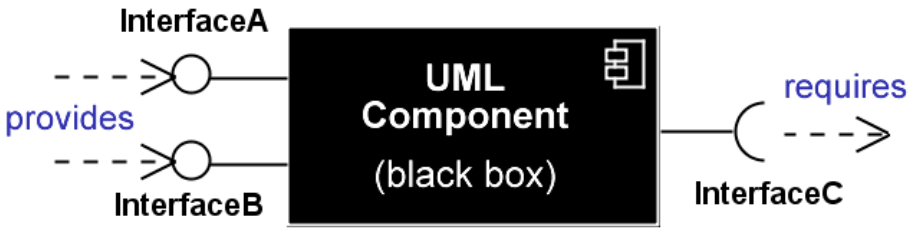
Mechanical components are all around

- **Cars** assembled from engine, gearbox, wheels, tires, breaks, ...
- **Computers** assembled from processor, memory, sound card, monitor, keyboard, ...

And what about **software**?

Software components

A **software component** is a contractually specified building block for software which can be readily composed by third parties without understanding its internal structure. *[Reussner]*



Main characteristics:

- Encapsulation
- Interfaces (services)
- Compositionality
- Client anonymity
- Ready to use
- Black-box reusability

Other characteristics:

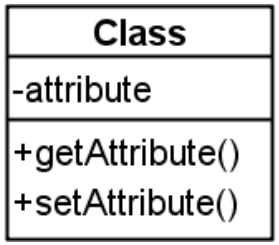
- Language independence
- Platform independence
- Configurability

Component vs. Class

Component:

- Executable run-time entity
- Described by interfaces (IDL)
- May contain several classes (hierarchical)
- No code available (black-box/grey-box)
- Developed separately for reuse
- Deployment context changes after compilation

Class:



- Design-time entity
- Source code available (white-box)
- In most cases designed for one system
- Stable deployment context after compilation

Component Models

A **component model** defines specific representation, interaction, composition, and other standards for software components.

[Heineman and Councill]

Existing component models have different views on

- What constitutes a **component**
  - run-time [COM/.NET, Fractal] vs. design-time [KobrA, SOFA, PCM]
  - dynamic [EJB, CCM, Darwin] vs. static [Wright, SOFA]
- Description of **interfaces** (services)
  - signatures [EJB, COM/.NET, CCM], pre/post-conditions [KobrA]
  - protocols of valid/performed call sequences [PCM, SOFA, Wright]
- Component **composition**
  - synchronous [Darwin, SOFA] vs. asynchronous [EJB, COM/.NET, CCM]
  - flat [EJB, COM/.NET, CCM] vs. hierarchical [Fractal, SOFA, Koala, ACME]

Why component-based development?

Component libraries

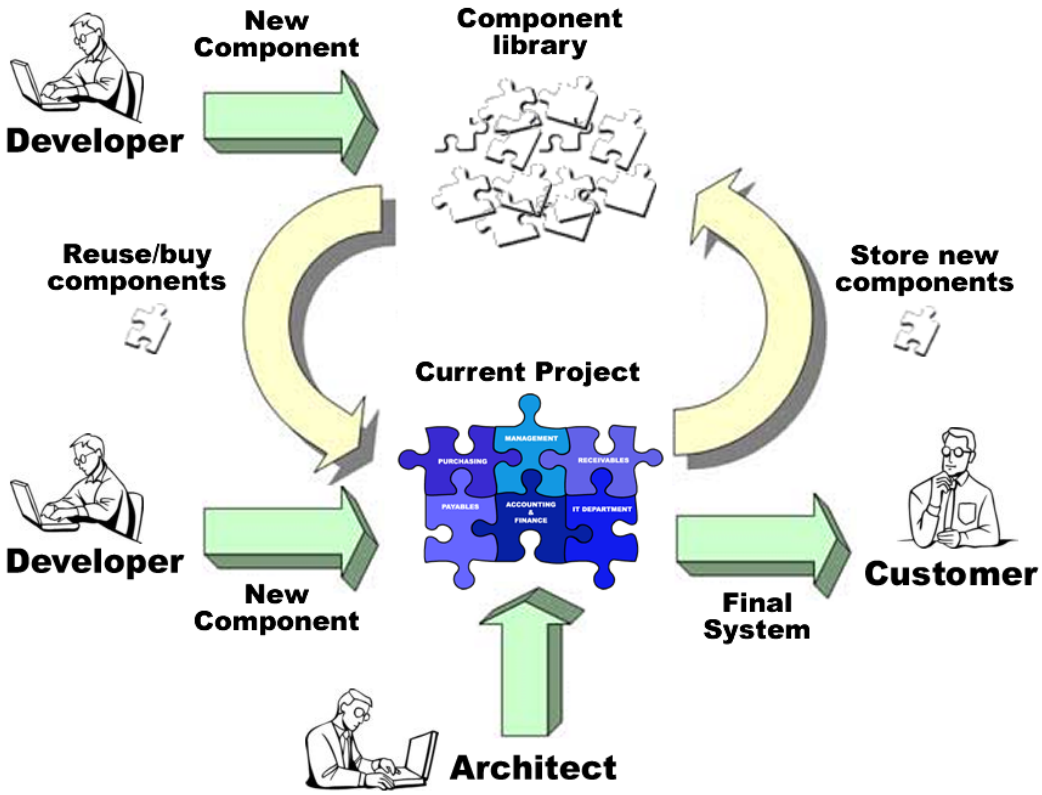
- Faster development
- Lower risk

Reuse of components

- Lower cost
- Higher quality

Maintainability

- Comprehensibility
- Configurability
- Language independence
- Flexibility w.r.t component update



Challenges of component-based development

General difficulties

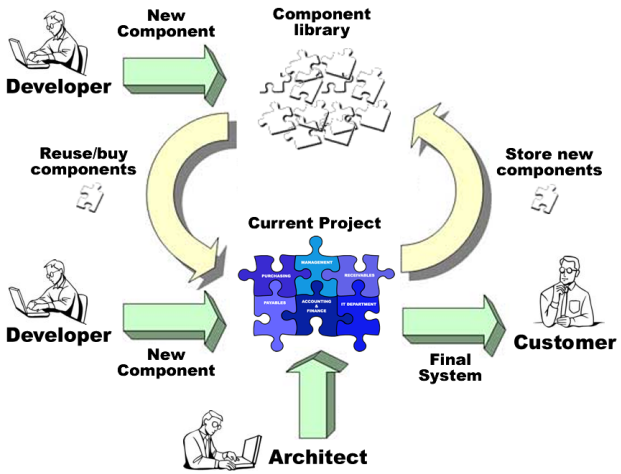
- Components delivered by different vendors
- Components developed for different environments
- Component environment changes dynamically

Correctness of a single component

- Unknown deployment context
- Unknown component usage

Correctness of a composite system

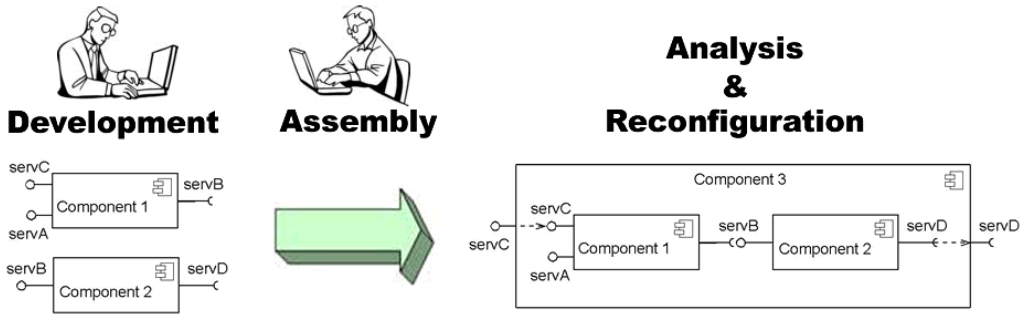
- Independently developed components
- No compilation after assembly
- Common updates and reconfiguration



Challenges of component-based development

The aim of CBSE – Engineering of **correct** and **high-quality** component-based systems on the level of

1. Single component
2. Assembly process
3. Composite system



1. Correctness of a component

- Context and usage independent/dependent
- Component certification
  - of component properties
  - of behavioural description

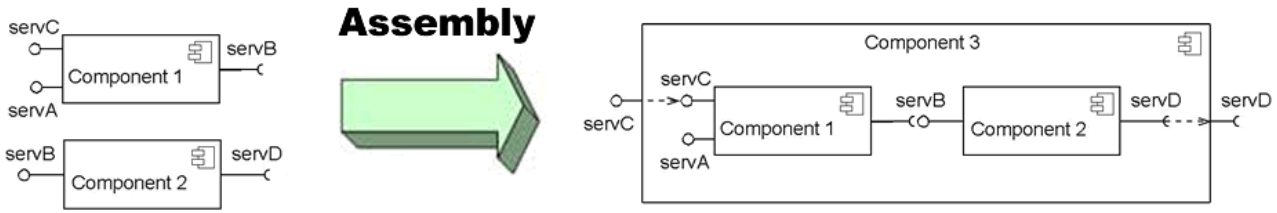
2. Assembly process

Component compatibility

- Syntactic – on a signature level
- Semantic – guarantee of correct interaction
- Automatically generated adaptors and wrappers

Component assembly

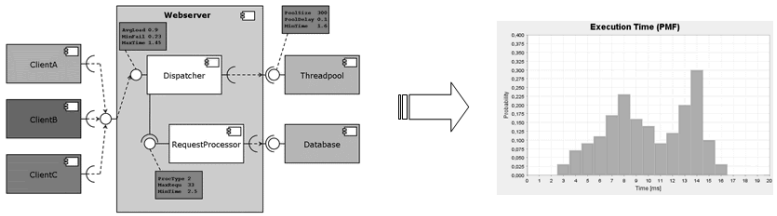
- Components first – assembly of existing components
- Architecture first – decomposition of system specification to component specifications, and search for the implementations
- Correctness by construction



3. Composite system

System analysis

- Verification of coordination errors
  - Compositional verification (top-down, bottom-up)
  - Assume-guarantee verification
- Prediction of extra-functional properties (performance, reliability, etc.)
  - Markov-chain analysis
  - Simulation-based methods



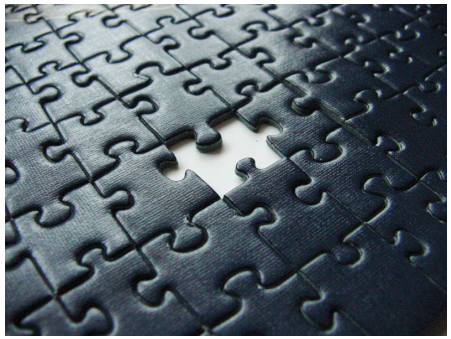
System evolution

- Component update – safe substitutability
- Reconfiguration – regression verification and testing

Lessons learned

Concepts of Software Architectures

- What is a software architecture?
- Why software architectures?
- What are the topics?



Component-Based Software Engineering

- What are software components?
- Why component-based development?
- What are the challenges of CBSE?

Thank you

Thank you for your attention!
Any **questions?**