Language for reconfiguring runtime infrastructure of component-based systems

Michal Malohlava, Tomáš Bureš

DISTRIBUTED SYSTEMS RESEARCH GROUP
http://dsrg.mff.cuni.cz

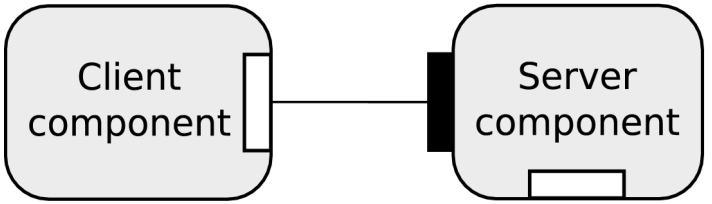
CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

Outline

- Introduction
  - Component-based software engineering
- Motivation
  - Execution environment for component-based applications
    - Where is the problem?
- Approach proposal
  - Language for describing infrastructure
    - Lightweight execution environment as the language interpreter
- Conclusion

Introduction

- Component-based systems
  - Modularity
  - Separation of concerns
  - Component re-use

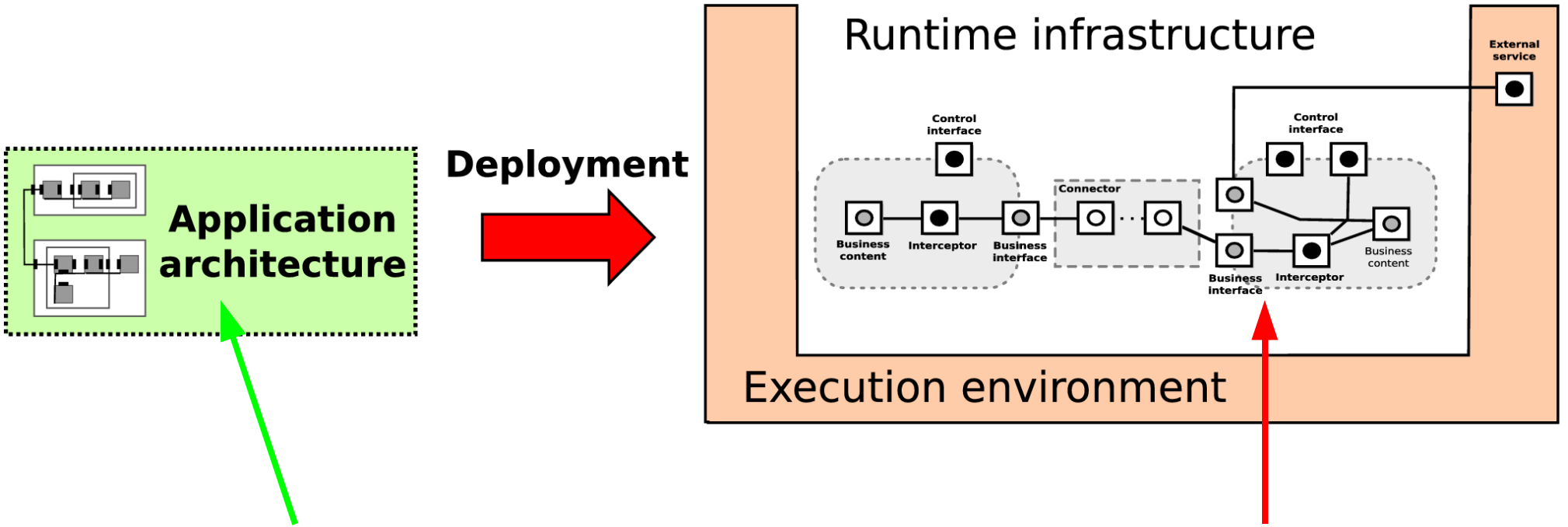


- Development process of component-based applications
  - Design
  - Implementation
  - Deployment
  - Runtime

Intro – component systems

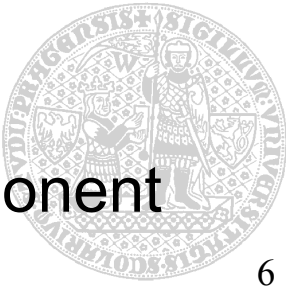
- Design level
  - Composition, behavior, non-functional properties specification
- Runtime level
  - Exec. environment where components *live*
  - Execution environment configuration
    - Control aspects, mebrane (SOFA 2, Fractal)
    - Container (EJB, Spring)
    - Statically generated infrastructure (Koala)

Intro - runtime



Execution environment

- Reflected complexities
  - Target domain (enterprise, embedded, RT)
  - Optimization
    - Artifacts skipping, merging
  - External services (data sources, transaction managers)
  - Distribution
  - Introspection
    - Depth (business level, control level,...)
  - Dynamic reconfiguration
    - Dynamic update, creation/destroy of component



Execution environment

- Com
  - Runtimes are often
    - Monolithic
      - **Component model instantiation logic is hard-coded**
    - Non-trivial to developed and update
      - Error prone
      - No concept of verification
    - Designated for a specific domain
      - Enterprise, Embedded, Real-time (RTSJ)
    - **How can we deal with these problems?**
  - Creation/Destroy of new component

Lightweight runtime idea

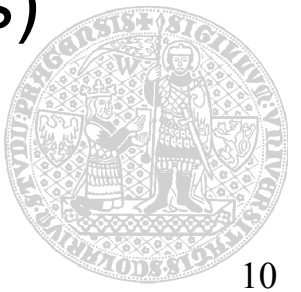
- **Displace instantiation/reconfiguration logic**
  - **Knowledge of component model**
- **Execution environment as a script interpreter**
  - **Instantiation scripts**
  - **Reconfiguration scripts**

Lightweight runtime requirements

- How to achieve the goals?
  - 1. step: unified & simple runtime infrastructure model
    - Instead of a component model
  - 2. step: process of infrastructure instantiation and reconfiguration
    - Described be a proposed language

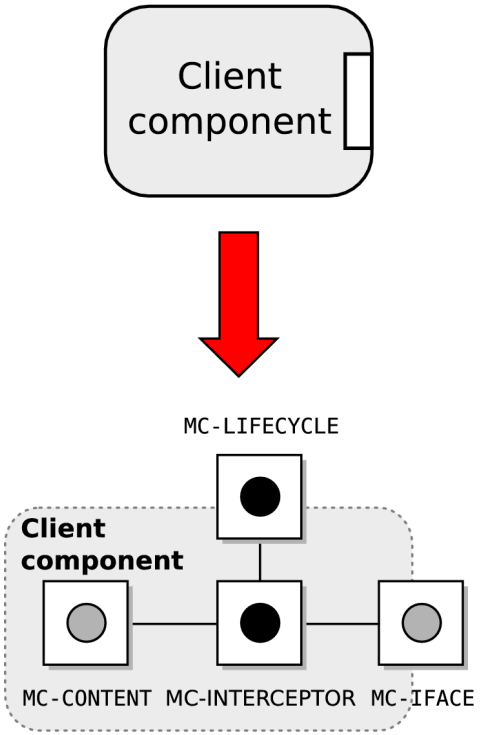
SOFA microcomponent model

- **Microcomponents**
  - Infrastructure model
  - Simple component model
    - No control part, no distribution, flat
- At runtime: microcomponent represents
  - Control part (via aspects) of components
    - Control interfaces, interceptors
  - Connectors
  - Business code (implementation, interfaces)
  - External services



Infrastructure instantiation

- *"When, what, in which order?"*
- **High-level operation is**
  - Creation of component instance



    - Creation of component's business interface
    - Instantiating the component architecture
    - Applying control aspects
      - Creation of interceptors
      - Creation of control interfaces
    - Instantiation connector end-points
    - Connecting all artifacts
    - Initialization of all artifacts

All created artifacts are microcomponents

Infrastructure reconfiguration

- **But instantiation is not enough**
  - **Dynamic reconfiguration**
- SOFA 2 reconfiguration patterns
  - Dynamic update
  - Factory/Removal pattern
- Component modes
  - Reconfiguration of a component's internals
- Reconfiguration = execution of a script

Proposed micro-operation

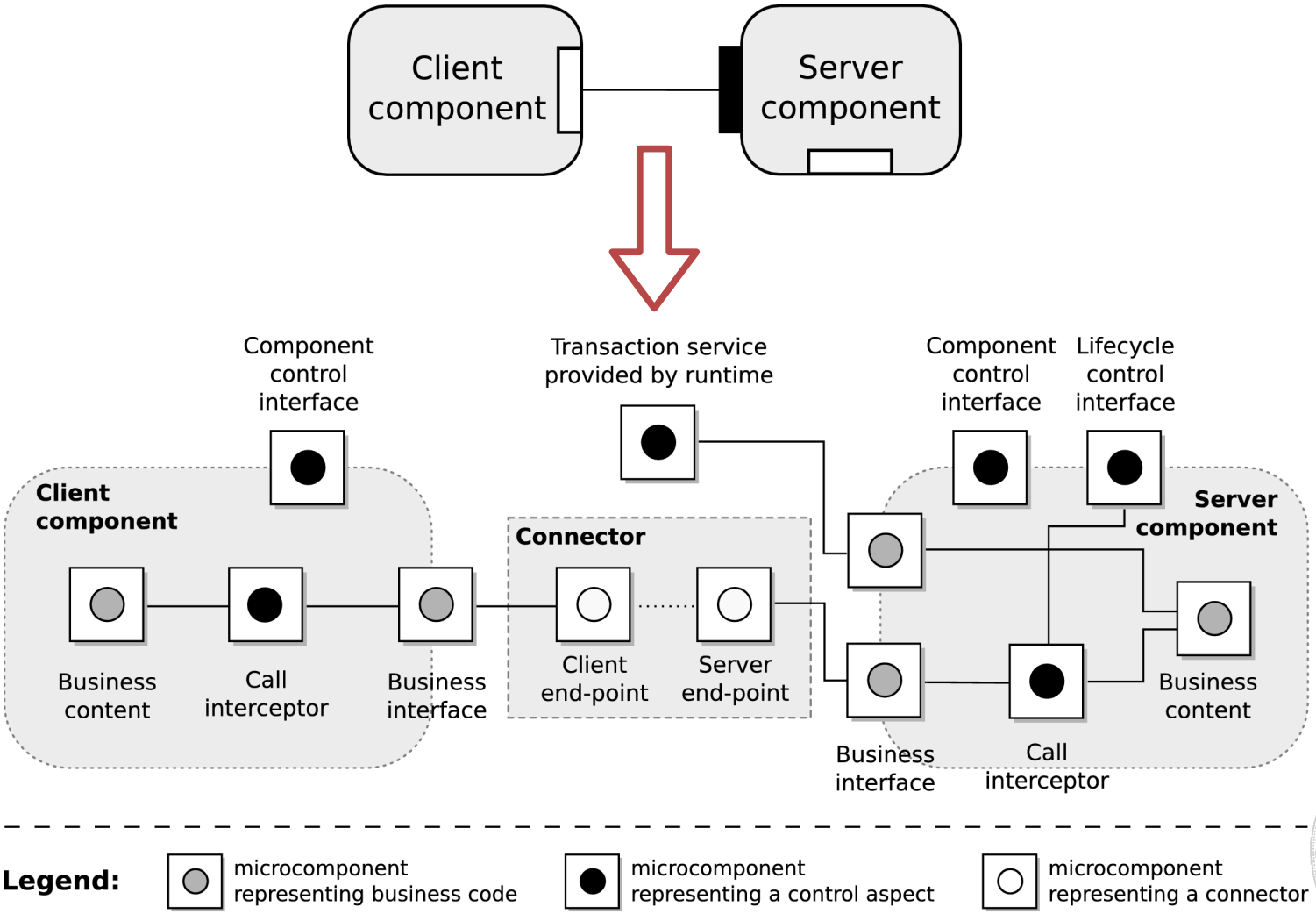
- **Proposed infrastructure *micro-operation***
  - Creating/Destroying of microcomponent
  - Generating/Loading code
  - Initialization of microcomponent
  - Binding/Unbinding micro-interfaces
  - Saving/restoring a microcomponet state
  - Calling control interface operation
  - Calling external tool producing micro-script
    - e.g. connector solver
- Semantics depends on the execution environment

Runtime environment scenarios

- Utilization scenarios
  - Runtime as instantiation tool
  - Runtime as a compiler
    - Produces a binary image of the application

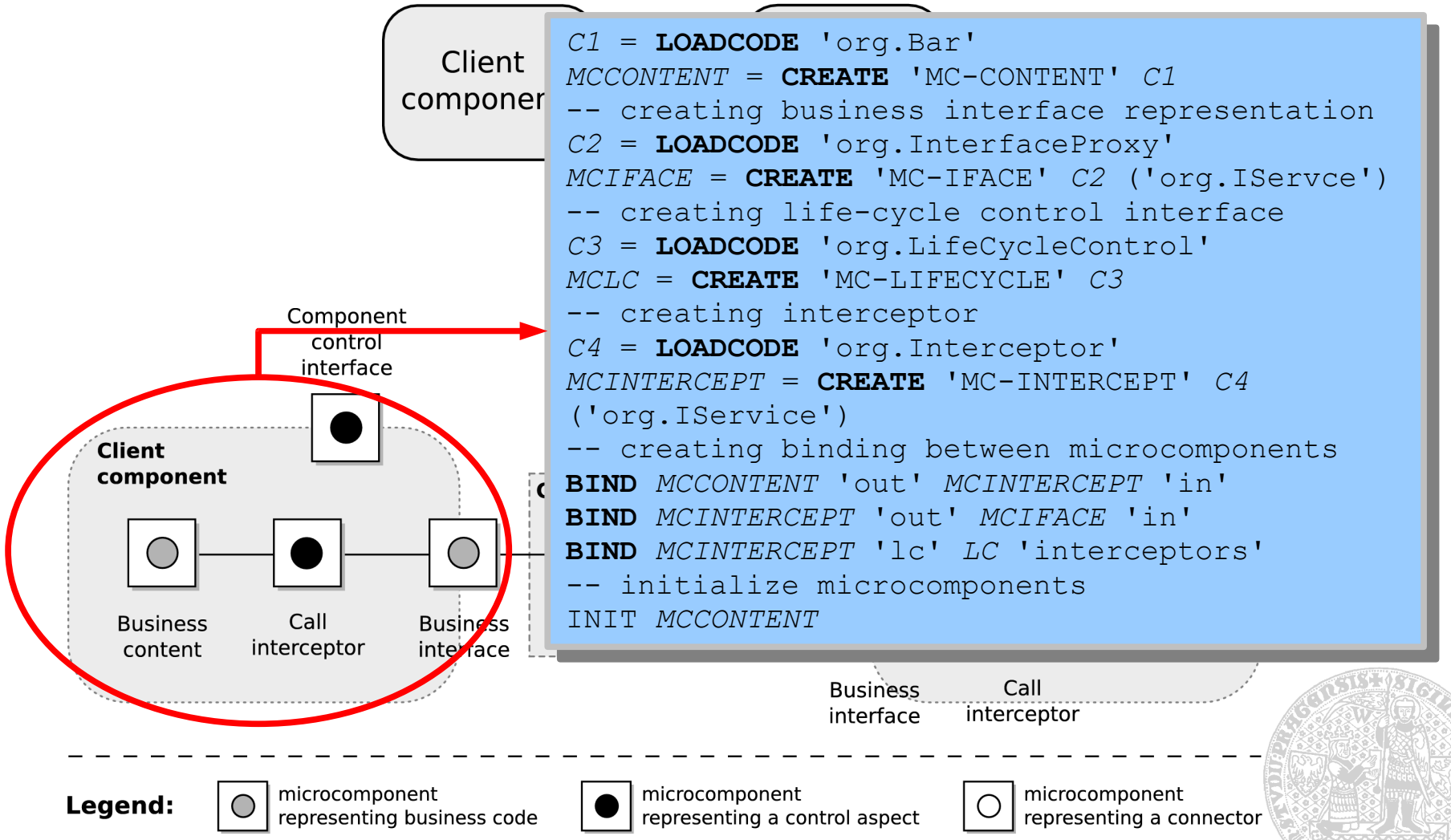
Example – infrastructure instantiation

- Instantiation of runtime infrastructure



Example – infrastructure instantiation

- Instantiation of runtime infrastructure



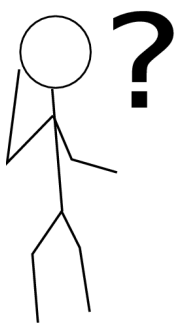
Conclusion

- Lightweight runtime approach
  - Runtime as a language interpreter
- Language for instantiation and reconfiguration of runtime infrastructure
  - Separation of a logic of component model from execution environment

Future Challenges

- Different runtimes for SOFA 2
  - Java for distributed applications
  - C/C++ for embedded devices
- Automatic preparation of target runtime
  - Product lines of a runtime
- Utilization of model-to-model & model-to-text tools (e.g OAW) to automatize:
  - Step-by-step refinement of an application architecture
  - Architecture diff script generation

Questions?



Thank you for your attention.