



# SOFA 2 Component Model

---

Tomas Bures, Petr Hnetynka, Michal Malohlava

**CHARLES UNIVERSITY PRAGUE**  
Faculty of Mathematics  
and Physics  
Czech Republic

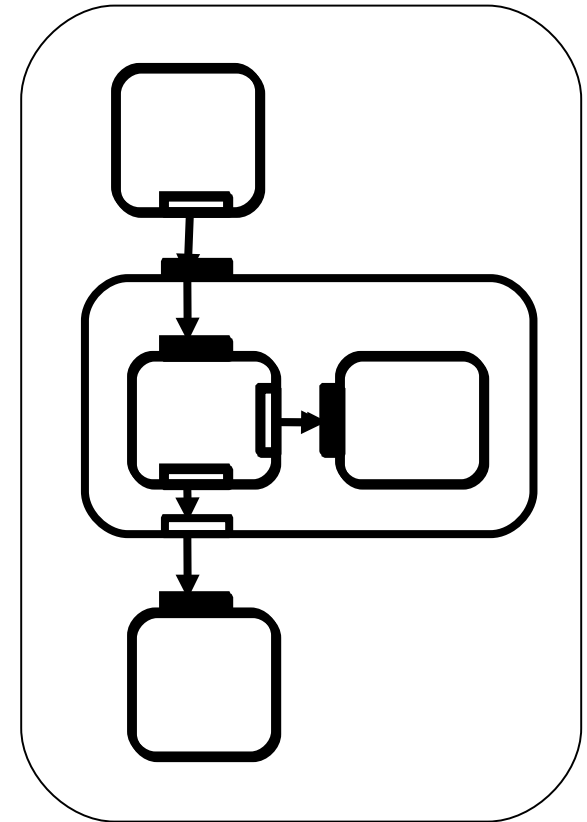


# SOFA 2 components

- SOFA 2 – a component system with hierarchical component model
- Based on SOFA (1)
  - in fact a prototype
- <http://sofa.ow2.org/>
- LGPL license

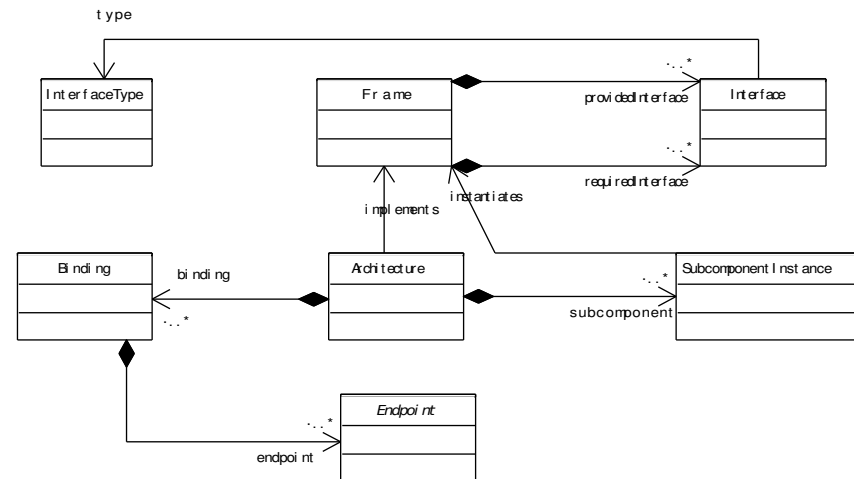
# SOFA 2 components

- Components defined by
  - component type (called frame)
    - defines component's provided and required interfaces
      - interfaces defined by interface types
    - defines component's behavior
  - component implementation (called architecture)
    - glass-box view
    - implements a frame
    - either primitive or composite
      - directly implemented or composed of other components
  - at runtime – architectures are instantiated
  - there can be multiple instances of a single architecture



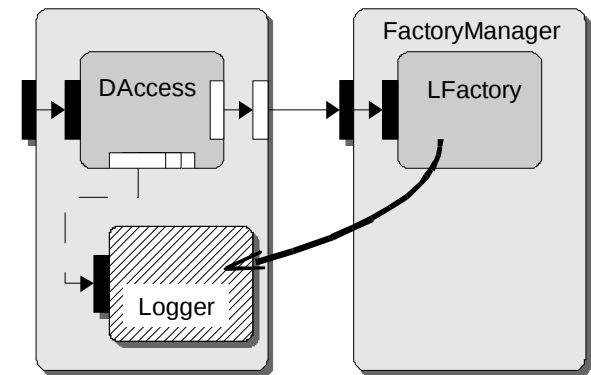
# SOFA 2 meta-model

- Defines all SOFA 2 abstractions
  - frames, architectures,...
- Advantages
  - automated generation of a repository
  - simple support of MDA
- Meta-model core



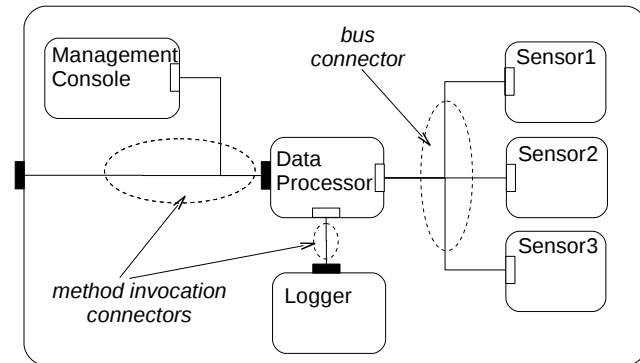
# Dynamic architectures

- Changes of application architecture at runtime
- Controlled reconfiguration via reconfiguration patterns
  - factory pattern
  - removal pattern
  - utility interfaces pattern
- Marked by annotations on frames and interfaces



# Connectors

- Bindings among components
  - at design time
    - links with properties
      - secure, logging,...
    - and communication style
      - method invocations, shared memory, messaging,...
  - at deployment time
    - automatically generated based on
      - properties,
      - connected interfaces,
      - configuration of runtime environment
    - allow for transparently distributed applications

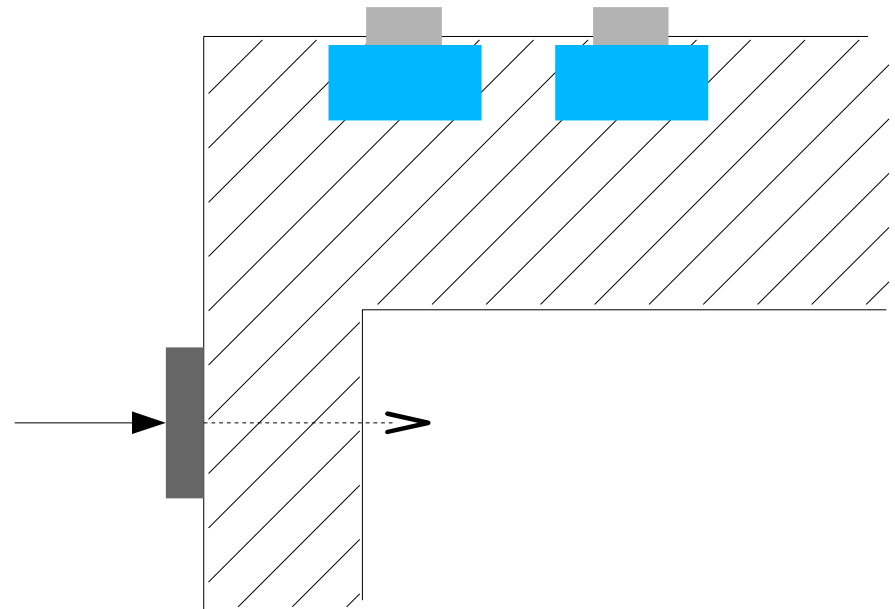


# Control part of components

- Explicitly separated control and business parts of components
- Control part
  - controls non-functional properties
  - provides so-called *controllers*
    - interfaces managing non-functional properties
  - modular and fully extensible
    - applied as *aspects* at deployment time
    - composed of *microcomponents*
- Business part
  - primitive components – an implementation provided by developers
  - composite components – subcomponents

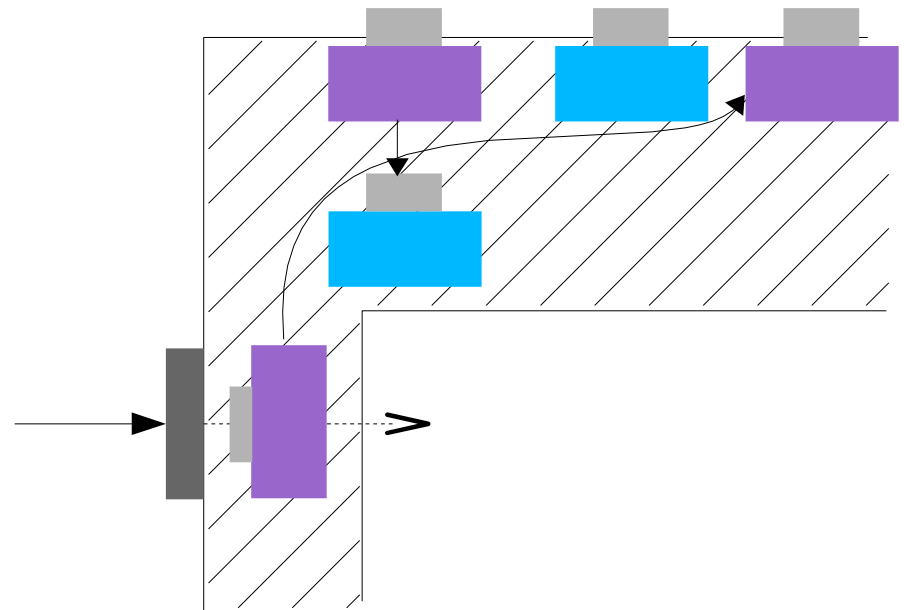
# Aspects

- Aspect ~ extension of the control part
  - definition of microcomponents
  - instantiation patterns
- Core aspect
  - present in all components
  - controllers
    - lifecycle
    - binding



# Aspects

- Aspect ~ extension of the control part
  - definition of microcomponents
  - instantiation patterns
- Core aspect
  - present in all components
  - controllers
    - lifecycle
    - binding



# Microcomponent model

- Aspects defined using a microcomponent model
  - Very minimalistic
  - Flat
  - No connectors
  - No distribution
  - No control part
  - From the implementation view
    - microcomponent ~ class

# Behavior specification and verification

- Component types have associated a behavior specification
  - any type of behavior specification
- Primarily supported – ***Behavior protocols***
  - verification of correctness of component composition
  - verification of correctness of primitive component implementation
- General support for modular verification of implementation against low-level properties (deadlocks)

# Behavior specification (example)

```
Component LightDisplay {
  types {
    States = {LIGHT_ENABLED, LIGHT_DISABLED}
    Events = {EXPRESS_MODE_ENABLE, EXPRESS_MODE_DISABLE}
  }
  vars {
    States state = LIGHT_ENABLED
  }
  behavior {
    ?LightDisplayControllerEventHandlerIf.onEvent(Events e) {
      switch (e) {
        EXPRESS_MODE_ENABLE: { state <- LIGHT_ENABLED }
        EXPRESS_MODE_DISABLE: { state <- LIGHT_DISABLED }
      }
    }*
  }
}
```

# SOFA 2 component/application lifecycle

## 1. Development

- composing existing components together
  - components stored in the repository
- newly developed ones also stored in the repository

## 2. Assembly

- subcomponents primarily defined by frames
- recursively replacing frames by architectures
  - enables [product-line development](#)

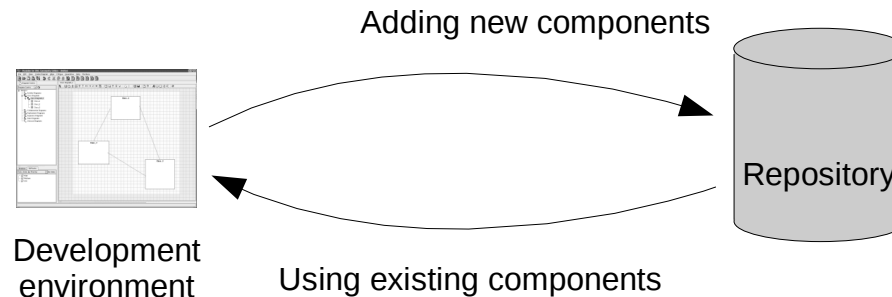
## 3. Deployment and executing

- where particular components have to be executed
  - information stored in a deployment plan
- connector generation
- applying control aspects

- execution

# Component development

- Components – stored in the repository
  - component types, implementations, and also code
- Development of an application
  - composing components into new composite components and
  - adding new (primitive) components
- Each element stored in the repository can exist in multiple versions



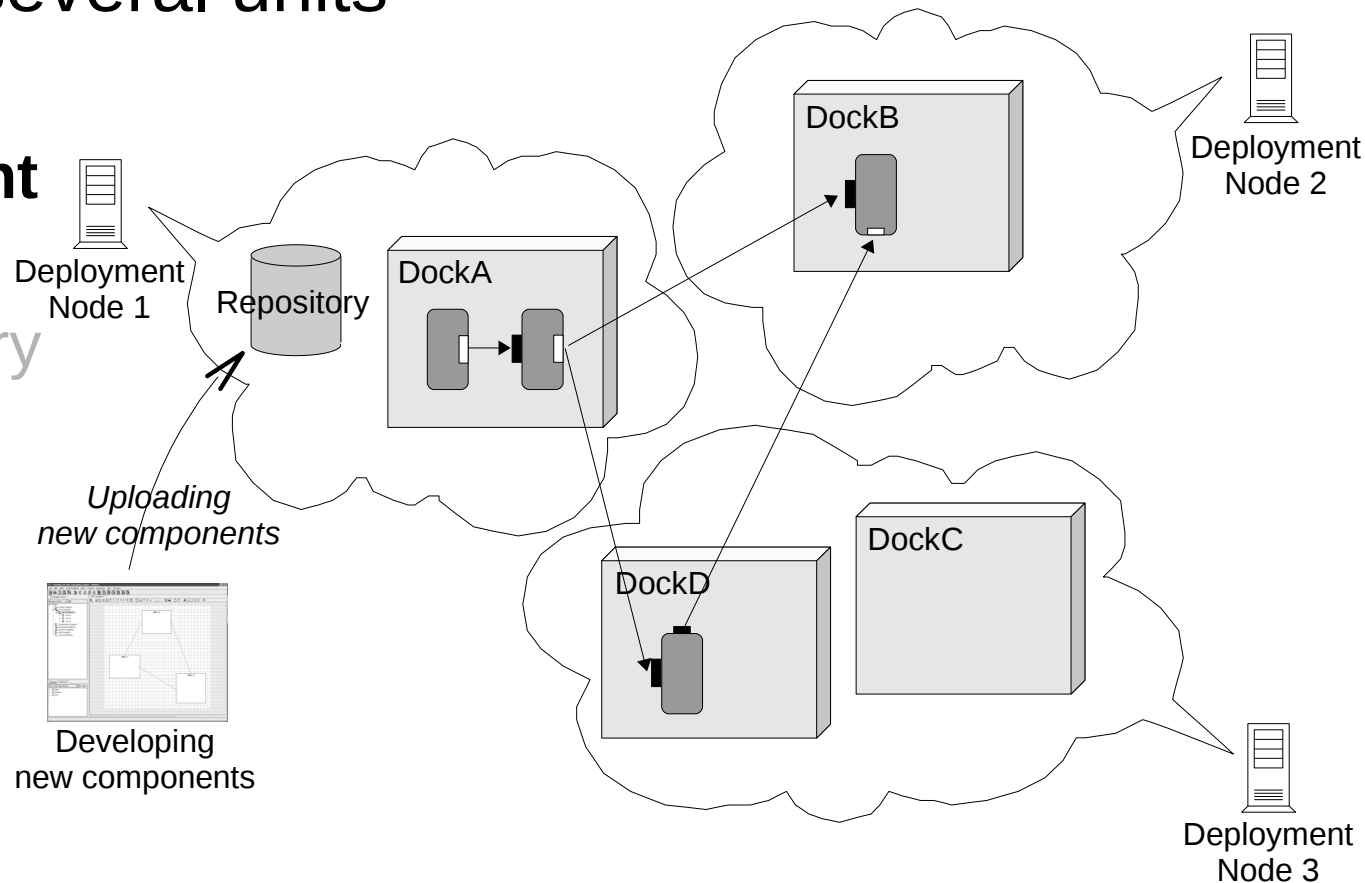
# System assembly, Deployment

- Assembly
  - Subcomponents in architectures can be defined by component types only
  - Assembly process recursively specifies subcomponents till primitive components
    - at the end – an application is a tree
      - non-leaf nodes – composite components
      - leaf nodes – primitive components
- Deployment

# Runtime environment

- Runtime environment (called SOFAnode) consists of several units

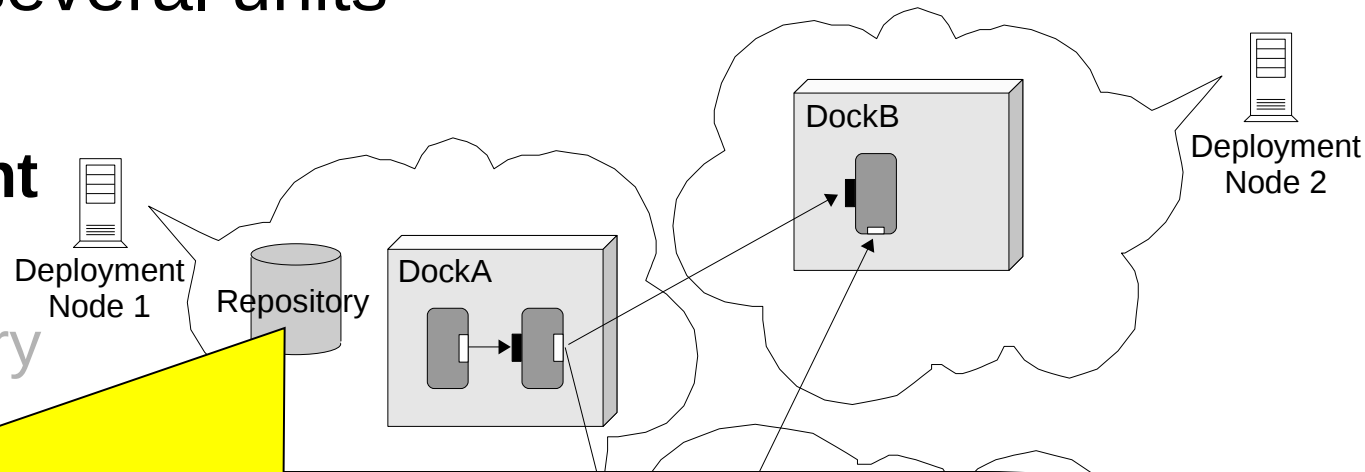
- repository
- deployment docks
- dock registry
- global connector manager



# Runtime environment

- Runtime environment (called SOFAnode) consists of several units

- repository
- deployment docks
- dock registry
- global

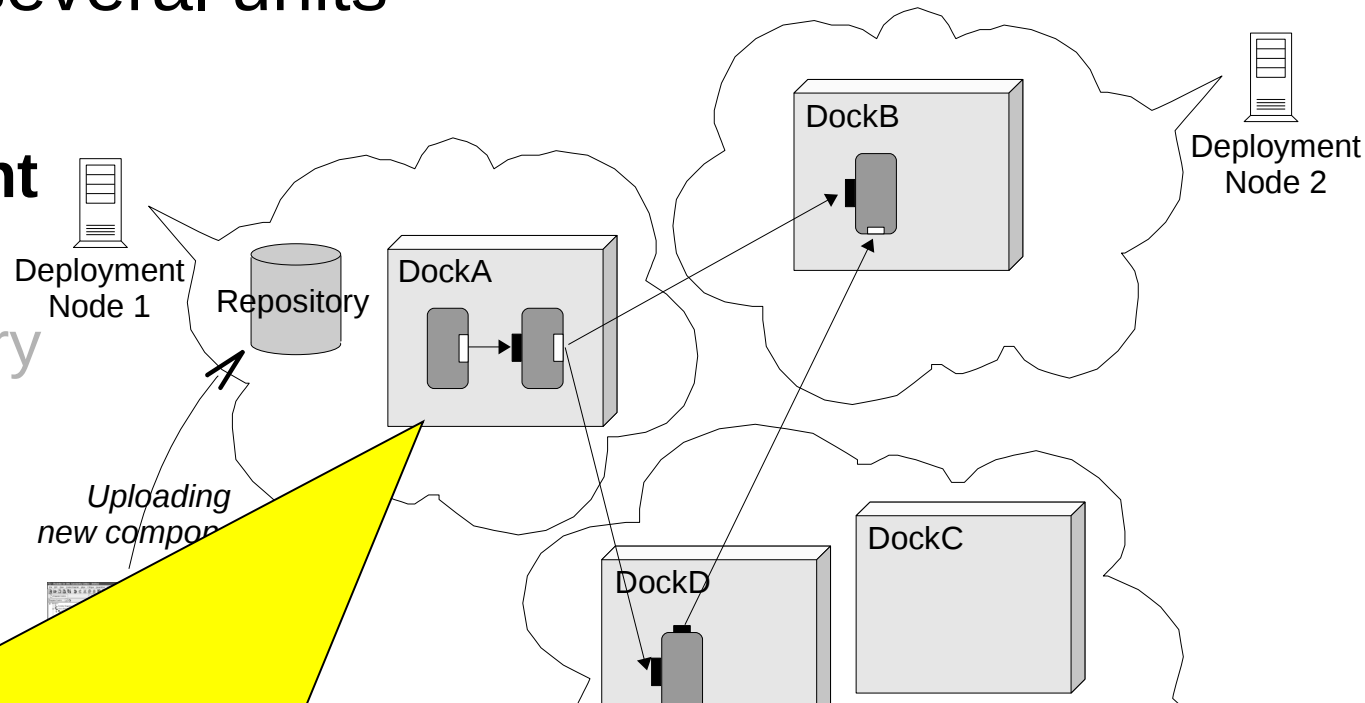


- Generated from the EMF meta-model
  - Eclipse Modeling Framework
    - de-facto standard for defining meta-models
- Remotely accessible
- Supports versioning
- Used at both development time and run time

# Runtime environment

- Runtime environment (called SOFAnode) consists of several units

- repository
- deployment docks
- dock registry
- global connector manager



- A container executing components
- Automatically downloads code of components from the repository



# Future/current work

- Extra-functional properties of components
  - performance models
  - resource consumption
- Embedded systems
  - SOFA HI (High Integrity)
  - attributes needed for scheduling (analysis)