SOFA 2 Component Model

Petr Hnětynka, Tomáš Bureš, František Plášil

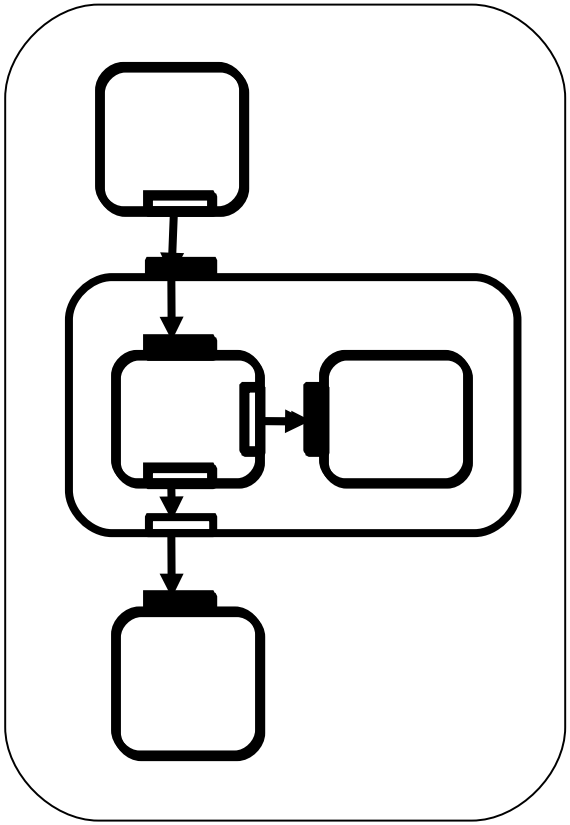
CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics
and Physics
Czech Republic

History

- Second generation of the SOFA component model
  - based on original SOFA (1)
    - evolution over years => issues (with support of different features)
- autumn 2006
  - initial design of SOFA 2
    - properly balanced features
    - completely redesigned
- summer 2007
  - publicly available implementation
    - supporting complete both system and component life-cycle

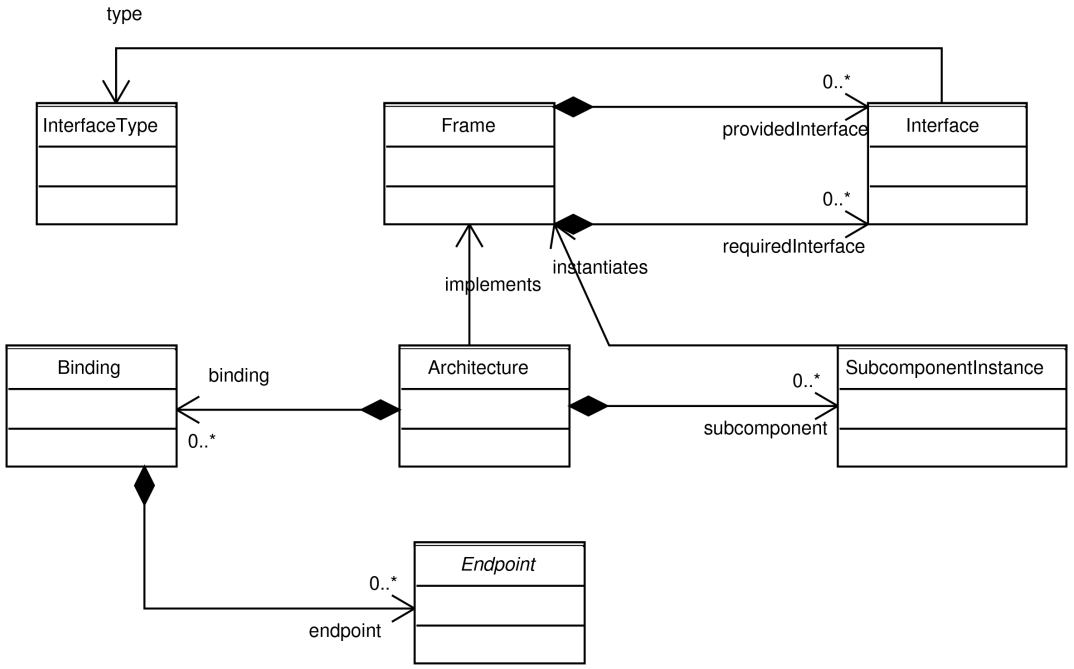
SOFA 2 components

- 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



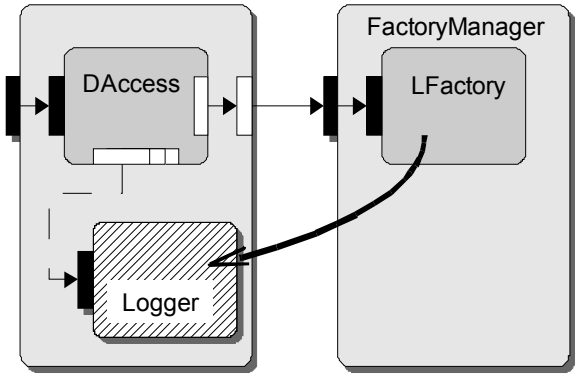
SOFA 2 meta-model

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



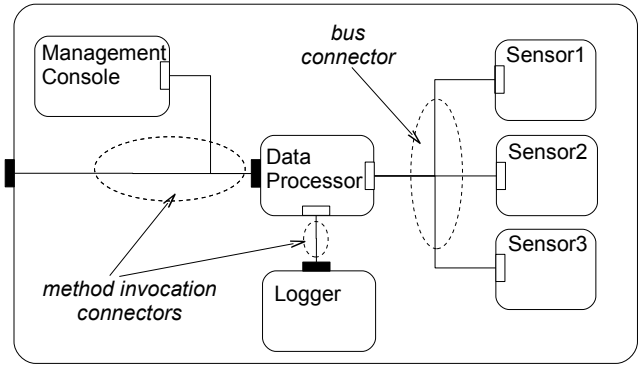
Dynamic architectures

- 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

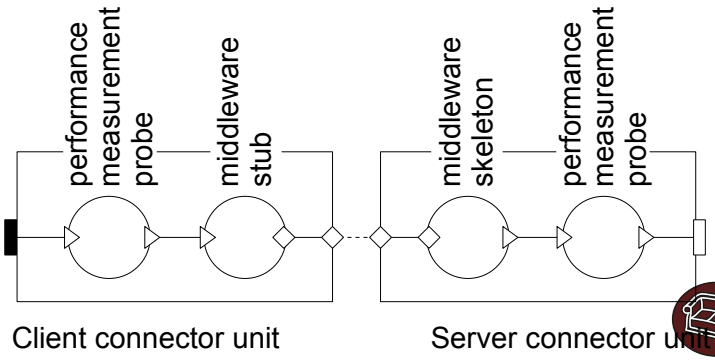
- Bindings among components
  - at design time
    - links with properties
      - secure, logging,...
    - and communication style
      - method invocations, shared memory, messaging,...



  - at deployment time
    - automatically generated based on
      - properties,
      - connected interfaces,
      - configuration of runtime environment
    - allow transparently distributed applications

Connectors

- Allows transparently distributed applications
  - developers do not bother with the networking
  - generated using suitable middleware
- Internal structure
  - set of connector elements
  - connector generator builds a suitable connector architecture and uses predefined elements
    - (connector architectures are either predefined or can be defined by special language)
- An example of connector architecture
  - method-invocation style

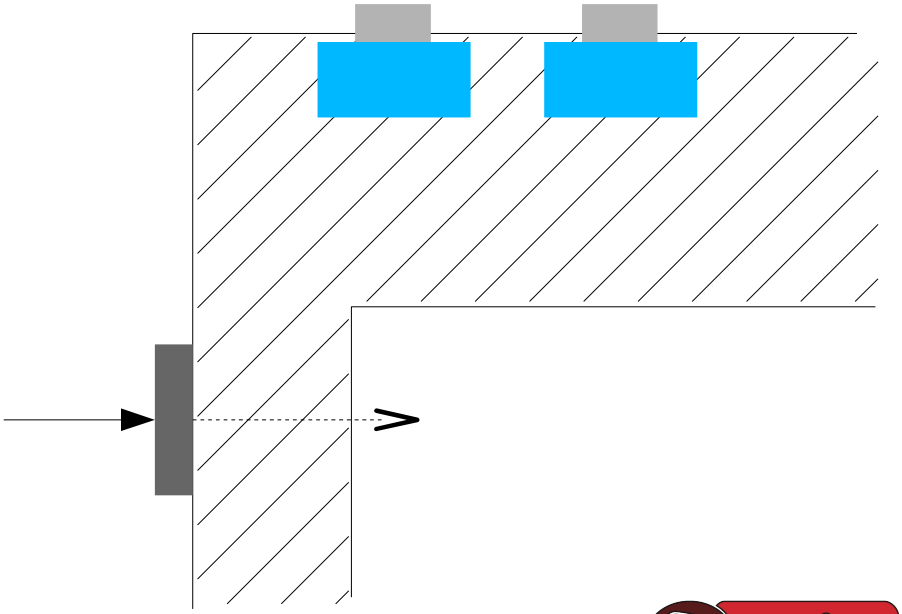


Control part of components

- 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
- Business part
  - primitive components – an implementation provided by developers
  - composite components – subcomponents

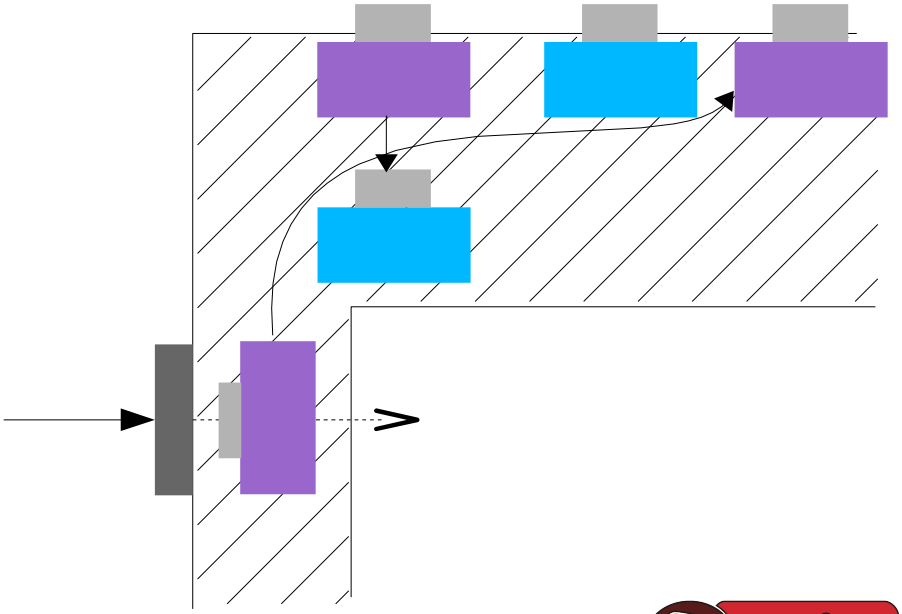
Control aspects

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



Control aspects

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

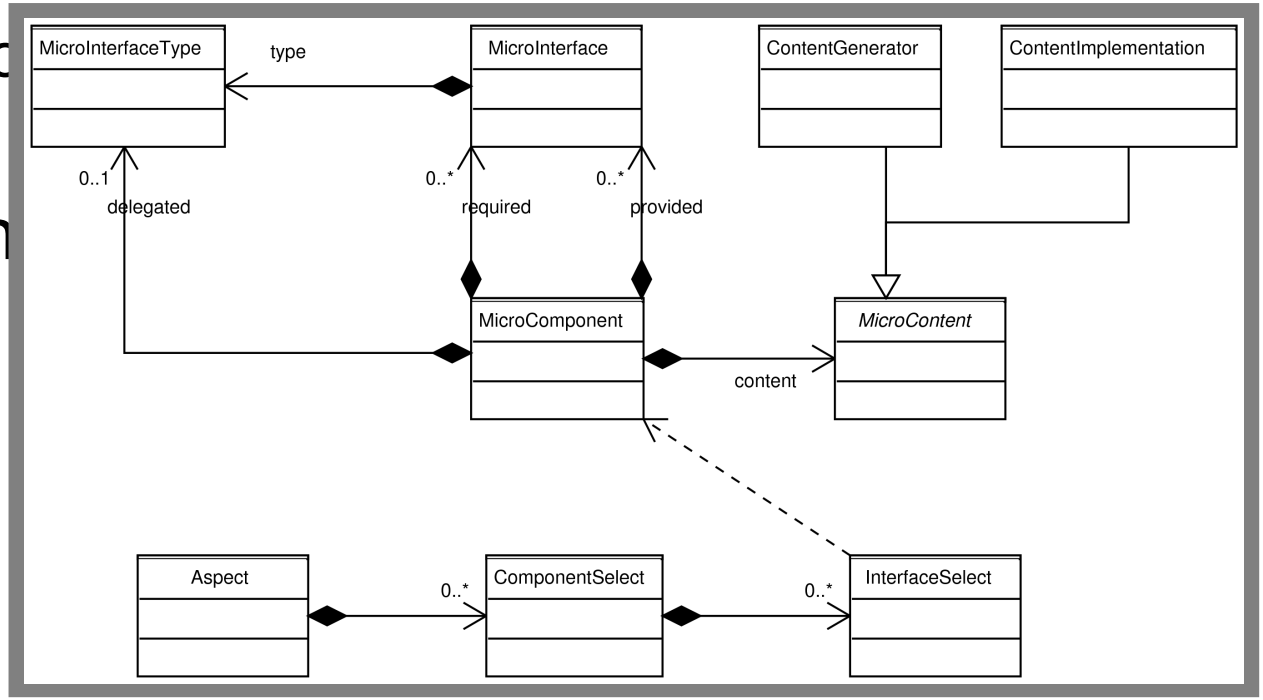


Microcomponent model

- Very minimalistic
  - Flat
  - No connectors
  - No distribution
  - No control part
  - From the implementation view
    - microcomponent ~ class
- Defined again by meta-model

Microcomponent model

- Very minimalistic
  - Flat
  - No connectors
  - No distribution
  - No control part
  - Fro
    -
- Defin



Behavior specification

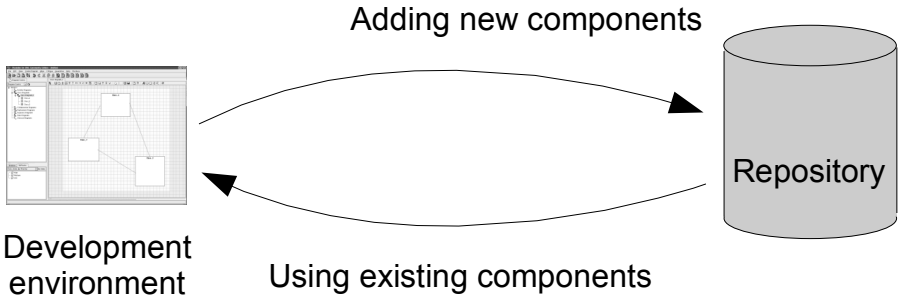
- Component types have associated a behavior specification
  - any type of behavior specification
- Behavior protocols, Extended behavior protocols
  - ... Pavel ...

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
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



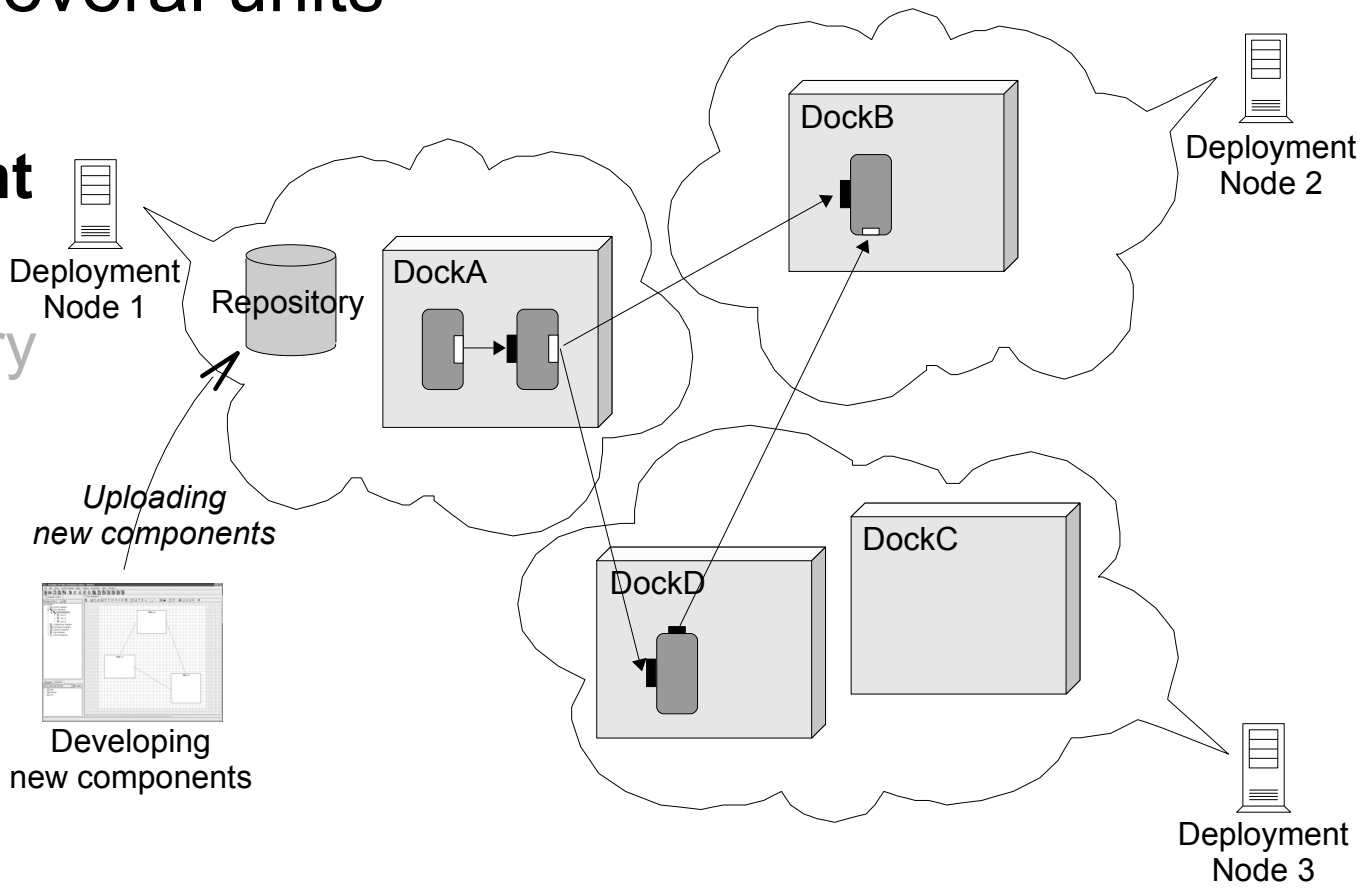
System assembly, Deployment

- 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

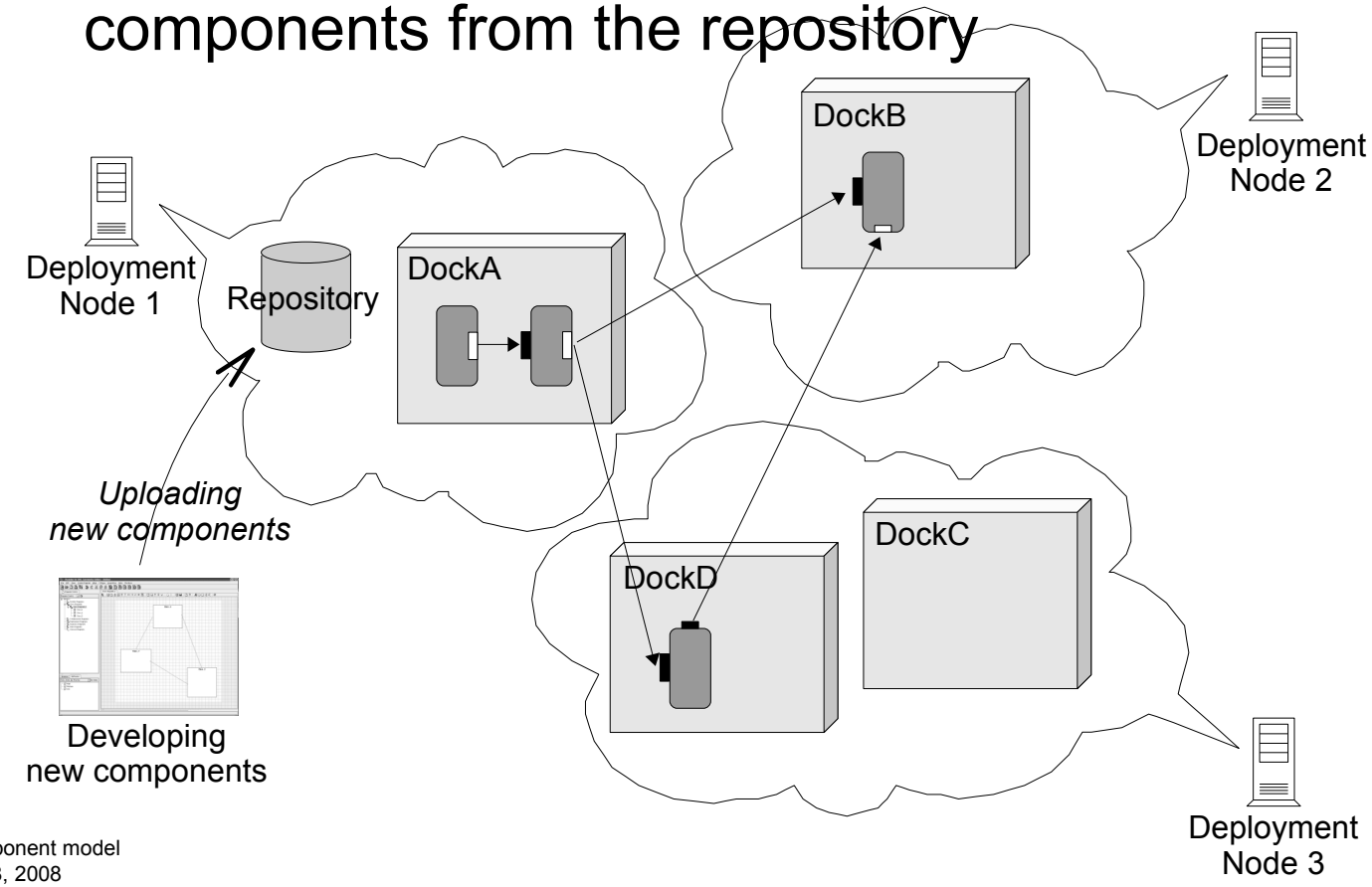


Repository

- Generated from the EMF meta-model
- Remotely accessible
- Supports “transactions” (via cloning)
- Repositories of different SOFAnodes can exchange content among themselves
  - trading of components

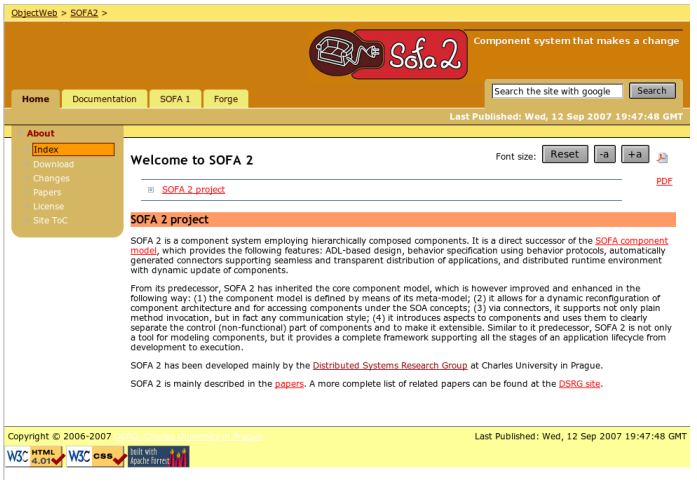
Runtime environment

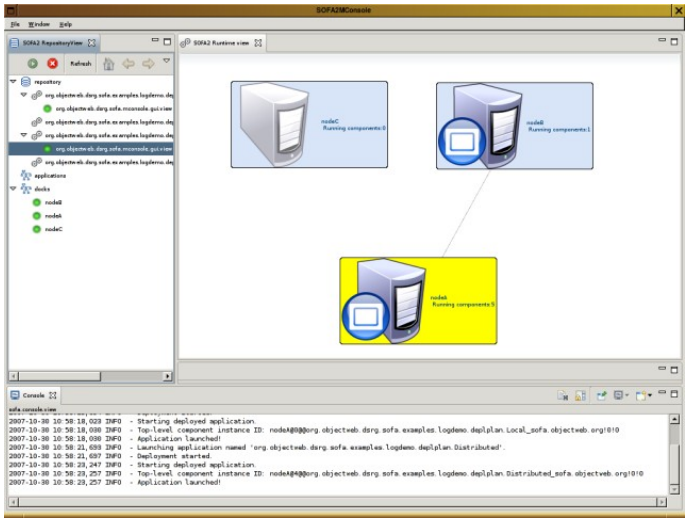
- Deployment dock (~container)
  - a container executing components
  - automatically downloads code of components from the repository



Current status

- Implementation available
  - http://sofa.ow2.org/
  - LGPL license
- All features are implemented
  - repository, docks, ...
  - Cushion
    - a modular development tool (command line)
    - supports creating all elements (types, implementations, assemblies)
    - verification of composition and behavior





Work in progress (almost finished)

- Graphical designer of components
  - composition of already developed components
  - developing new ones
  - as an Eclipse plugin
- Integration with OSGi platform
  - transparent using of OSGi services as SOFA 2 components
    - thanks to support of dynamic architectures
  - using both local (in-dock) and remote OSGi services
  - (Thinking about opposite direction of integration)

Real-life example – Multimedia center