μSOFA

Michal Malohlava, Tomas Bures
Department of Distributed and Dependable Systems
http://d3s.mff.cuni.cz/
CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics
Before we start...
Current status of the SOFA
SOFA status – finished improvements

• Modularization
  ▪ Meta-projects
    • sofa-core, sofa-j, sofa-hi, sofa-ide, mconsole, cushion, ...
  ▪ Modules
    • Sofa-repository, Sofa-j-dock, sofa-**

• New build system based on Ivy
  ▪ Build server at india.ms.mff.cuni.cz
    • Build and assembly SOFA distribution
    • Build process for Eclipse tools in progress
SOFA status – improvements in progress

• Master thesis
  ▪ SOFA HI (PH & PH - coming soon:-)
  ▪ J2ME support (PH)
  ▪ Cloud computing (PH)
  ▪ Component migration (PH)
  ▪ Dynamic languages support (MM)
  ▪ Connector generator improvements (MM)
  ▪ Code generation infrastructure (TB)
  ▪ Reconfiguration patterns (TB)

• Bachelor thesis
  ▪ MConsole improvements (MM)
  ▪ SOFA development dashboard (TB)
  ▪ ...

D3S Seminar, 21st April 2010, Michal Malohlava
SOFA implementation status

• SOFA implementation is still more complex and larger

• It is not easy to maintain and integrate students’ changes
  ▪ Only 1-2 persons (thanks PH!)

• It is not easy to extend
  ▪ Implement non-pervasive extensions of runtime is almost impossible
Example: SOFA OSGi support

- **Situation:** application requires/provides OSGi service

  - **Model**
    - Component is marked by an annotation

  - **Component control layer**
    - Annotation is translated into an OSGi aspect

  - **SOFA runtime platform**
    - OSGi aspect support is hard-coded into runtime platform (~ dock)
Other hard-coded variation points

- Component instantiation
- Access to repository
  - Hardcoded into runtime

- Configurable target implementation technology
  - Only Java, why not OSGi, or Guice?

- Constrained dynamic reconfigurations in runtime infrastructure (e.g., entities)
  - Impossible
Critical demands

• Highly configurable & extensible runtime infrastructure
  ▪ Only features which are really required

• Understandable runtime infrastructure

• Use contemporary tools & technologies for implementation
  ▪ E.g., Google Guice for runtime composition, iPojo for generation of OSGi applications, ...

• Generate more code automatically
  ▪ More focus on models and their transformations than a target implementation

• Perhaps more changes:
  ▪ Lighter development process
    ▪ E.g., only local repository
  ▪ Definitely we need tests
    ▪ It's hard to maintain & develop complex system without them (manual testing is tedious and time consuming)
  ▪ ...

D3S Seminar, 21st April 2010, Michal Malohlava
The objective

Highly configurable & extensible runtime infrastructure
• Modeling layer between high-level application architecture and target implementation
• Why we need another meta-model?
  ▪ A gap between a high-level architecture model and runtime infrastructure
Note: parallel motivation for uSOFA

• There are still enough meta-levels to speak about :-)  

• Idea of meta-component system  
  ▪ Also requires highly configurable runtime  
  ▪ TODO picture here  
    • Model weaving  
    • Transformation from high-level design -> micro-components -> instantiation/code generation
uSOFA meta-*

- Micro meta-model
- Aspect meta-model
- Runtime platform meta-model
uSOFA

• High-level components translated into a micro-components

• Translation can enforce functionality in runtime platform

• TODO obrazek jak si predstavujeme vztah mezi aplikaci a runtime platformou
  ▪ Je to jen model
  ▪ Za runtimu je to uzce propojene

• Chci to modelovat vzavislosti na sobe
  ▪ Viz osgi motivace
• Micro model
  ▪ Flat
  ▪ No control, no distribution layers
  ▪ Suitable for code generation
  ▪ Reflects “idealized” runtime infrastructure

• Micro-components model
  ▪ Application control layer
    • Control functionality (life-cycle), introspection, non-functional properties
  ▪ Runtime platform
    • Application services (transaction service, timers), technology requirements (runtime platform bootstrap)

• Micro-component stands as
  ▪ A regular component with prov/req interfaces (iface type is known)
  ▪ An interceptor delegating calls (iface type is unknown)
  ▪ An implementer of component control interface (iface type is known)

• Micro-component consists of
  ▪ Content
    • Regular
    • Generated
  ▪ Micro-interfaces
    • Regular
    • Delegation
uSOFA – aspects meta-model

• Adopts idea of AOP
  ▪ Aspect = point-cut + advice
    • Point-cut – defined place in architecture where the advice is applied
    • Advice = new micro-*

• Two types of aspects
  ▪ Component aspect
  ▪ Platform aspect

• Dependencies via aspects expressed via joint-points
  ▪ Component control interfaces
  ▪ Named join-points – defined points of extensibility
uSOFA – aspects meta-model

• Component aspect
  - Select component or/and interface, join-point (=point-cut)
  - Add new micro-components & bindings or control interface (=advice)
  - May require application of platform aspect
  - E.g., OSGi service publisher/consumer, script aspect, dynamic update, threading, platform service, introspection
• Platform aspect
  ▪ Selects a join-point
  ▪ Extends functionality of runtime platform via new
    • Micro-components & bindings
    • Join-points
uSOFA – platform meta-model

- Model of platform consists of
  - Predefined micro-components
    - Bootstrap
  - Micro-components added by platform aspects
uSOFA – model construction
It’s time for ....

... DEMO
It’s time for ....

... DEMO

Only example
Example

• Simple watch application

  ▪ Timer active component
    • continuously update display component

  ▪ Display
    • Display shows actual time via HttpService
High-level design

Ticker

@activeComponent(period=1s)

Fax

Display

@requireService(HttpService)

TickerImpl

HttpDisplayImpl

Runtime platform
uSOFA model

• This is just a runtime infrastructure model

• Can be instantiated and executed or

• Can be compiled into a selected implementation technology (OSGi, Guice)
  ▪ Runtime platform as a compiler
    • Each micro-component generates snippet of code
• OSGi application synthetizer
  ▪ Application = instance of micro-model

  ▪ Constructed runtime platform
    • OSGi compiler
      ▪ Micro-components produce OSGi bundles, wrapper code

  ▪ Result
    • OSGi application
There is always work to be done...

• Platform variation
  ▪ More mature use-cases

• Challenges (not only ours)
  ▪ Model debugging
    ▪ Debugging aspects
    ▪ For model debugging in Eclipse
  ▪ Models @ runtime
    ▪ Synchronizing changes in runtime infrastructure with model
  ▪ Meta-component system
Thank you for your attention
Used images

- http://www.sxc.hu/photo/1268866
- http://www.sxc.hu/photo/673581