1. Why should you be interested?
   Overview of the current SOFA 2 controllers
   Cool features for easy component development

2. What I’m expecting from you?
   Feedback, comments
   Notify me when I’m speaking fast/slowly/softly
1. Motivation & Goals
Why dynamic (scripting) languages?

Long turnaround of SOFA 2 component development cycle
- Code compilation, uploading, application restarts, …
SOFA 2 component development cycle

1. **Commit**
2. **Implementation**
3. **Compilation**
4. **Upload**
5. **Application start**
6. **Application shutdown**
7. **ADL definitions**
Why dynamic (scripting) languages?

Long turnaround of SOFA 2 component development cycle
- Code compilation, uploading, application restarts, …

Implementation changes at runtime
- Rapid prototyping

Are more suitable for some tasks
- GUI, parsing, …

Easy to learn
Goals

1. Support for scripted primitive components
   - Seamless integration with the original development cycle
   - Uses Java Scripting API

2. Implemented using component aspects
   - To avoid changes in the core runtime implementation
   - Scripting support should be an optional extension

3. Support for implementation changes at runtime
   - Command line tool

4. Component aspect mechanism evaluation
2. SOFA 2 component aspects overview
Component aspects & micro-architecture

Runtime extension mechanism

Implements the control logic
- All basic controllers implemented as aspects

Orthogonal to the business logic
- Similar to AspectJ

Aspects implemented using micro-architecture elements
- Simple flat non-distributed component model

Several important concepts
- Control interfaces
- Delegation chains
- Interceptors
Micro-architecture example
3. Dynamic language support implementation
Overall architecture

The component content represents the script code
- Created automatically
- Generic component content

The script code is executed in a designated aspect
- Script Aspect

The code updates are implemented in a separate aspect
- Update Script Aspect
- Optional
**Script Aspect**

Executes the component’s script code
- Java Scripting Framework
- JavaScript (Rhino), Python (Jython)

Provided interface calls are redirected using an interceptor
- Script Interceptor

Integrated with the Lifecycle and Component aspects

**Script** control interface
Execution of the script code in an aspect
Script Aspect architecture
Update Script Aspect

Ensures the remote changes of the component’s code

Extends the Script Aspect

Stores changes back in the repository

Integrated with the Component aspect

**UpdateScript** control interface

- Accessed remotely from a console application
Update Script Aspect architecture

Script : MIScript
UpdateScript : MIUpdateScript
Component : MIComponent

MUpdateScriptProxy
update-script : MIUpdateScript
UpdateScript delegation chain

Component impl.
Component delegation chain

listener : MIComponent

MUpdateScriptImpl
script:MIScript

Provided business interface
Standalone microcomponents
Required business interface

Provided interface delegation chain
Component Content
Required interface delegation chain
Features

- Similar specification/implementation to Java components
- Implicit access to Java classes from dependent c. bundles
- Support for script module importing
- Code updates can be stored into repository
- Replacing Java component by scripted one at runtime
- Fully optional, non-intrusive implementation
- General support for similar extensions (C# components)
Limitations

Implementation is dependent on the used scripting engine
- Class loading, module importing

Doesn’t support checked exceptions
- Not supported by adaptation layer of the used scripting engines

Doesn’t allow to change the component type
- Not supported for the Java components either
4. Example
Example architecture

Tester

log:Log

:Log

Logger
package org...examples.script.logdemo.iface;

public interface Log {
    void log(String message);
}
Tester architecture definition

```xml
<?xml version="1.0"?>

<architecture

    name="org...script.logdemo.arch.Tester"

    frame="sofatype://org...script.logdemo.frame.Tester"

    impl="org...script.logdemo.arch.Tester.js"

    lang="JavaScript"

/>...

</architecture>
```
function main() {
  var ret = 0;
  try {
    ret = log.log("Hello world from script.");
    ...
    java.lang.Thread.sleep(5000);
  } catch (e) { ... }
}

function start() {
  v = new java.langRunnable() {
    run: function() {
      while(!end) {
        main();
      }
    }
  }
  try {
    t = new java.lang.Thread(v);
    t.start();
  } catch (e) { ... }
}

function stop() { end = true; }
Logger implementation (Python)

```python
SOFAPythonImporter.loadCurrentCodeBundle()

from org.logdemo.arch.Printer import LogPrinter

printer = LogPrinter()

def log(message):
    printer.printLog(message)

...

class LogPrinter:
    def printLog(self, message):
        print "LOG: " + message
```
5. Evaluation
What was hard/interesting?

Execution of the component’s code in an aspect
- Interceptor ordering
- Transparency with respect to other controllers
- Detection of non-scripted components

Integration of SOFA 2 environment and scripting engines
- Class loading of renamed classes
- Script module importing
Improved development cycle

1. Dynamic implementation updates
2. Making changes in the implementation
3. Saving of dynamic changes
4. Implementation of a new component or changes in the ADL definitions
5. Application start
6. Architecture implementation
7. Upload
8. Making changes in the implementation
9. Commit
10. ADL definitions
6. Future tasks
Scripting support future tasks

Integration of the code updates into the SOFA IDE
Support for additional scripting languages
Versioning of the dynamic code updates
SOFA micro-architecture future tasks

Interceptor ordering specification
  ■ Aspect ordering is not enough

Component aspect extension mechanism
  ■ Extension points, ...

Fully configurable component instantiation
  ■ User-supplied component factories
Summary

1. Motivation & Goals
2. SOFA 2 component aspects overview
3. Dynamic language support architecture
4. Example
5. Evaluation
6. Future tasks
Conclusion

Easy implementation of scripted components

Good integration with the original runtime

- Both Java and scripted components in one application

A general approach to integration of new implementation technologies
Questions?