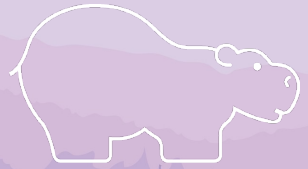
APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Using Apache Commons SCXML 2.0

a general purpose and standards based state machine engine

Presented For The Apache Foundation By
LINUX FOUNDATION



APACHECON
DENVER
WESTIN DENVER DOWNTOWN
APRIL 7-9, 2014

Ate Douma

R&D Platform Architect @ Hippo, Amsterdam Open Source CMS http://www.onehippo.org
10 years ASF Committer, 7 years ASF Member
Committer @ Apache Commons
Committer & PMC Member @ Apache Airavata, Incubator, Portals, Rave, Wicket & Wookie
Mentor @ Apache Streams (incubating)
ate@apache.org / a.douma@onehippo.com / @atedouma

Presented For The Apache Foundation By
LINUX FOUNDATION

Agenda

- SCXML – quick introduction
- Why use SCXML?
- A brief history of Commons SCXML
- Commons SCXML 2.0
  - Features (trunk)
  - A simple demo
  - High level architecture
  - Using custom Actions
  - Use case: Hippo CMS document workflow
  - Roadmap

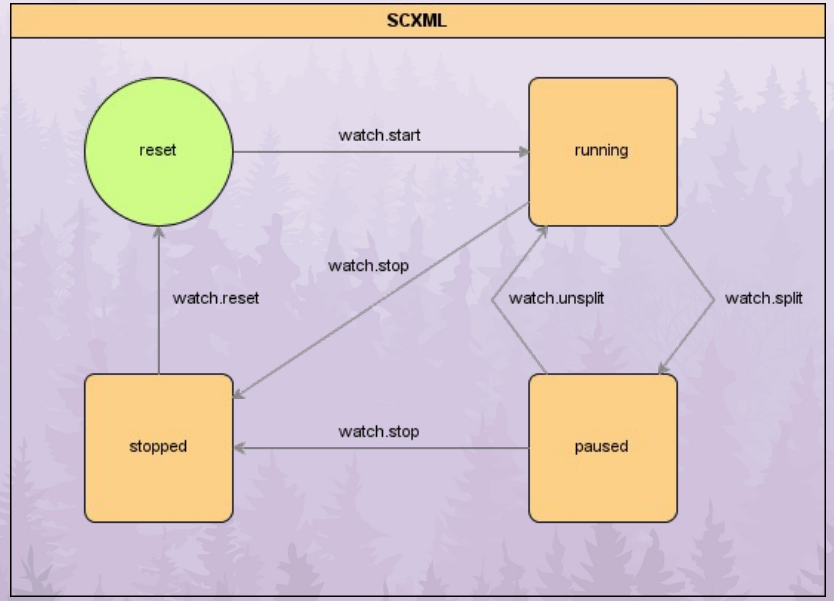
SCXML

State **C**hart **XML**: State Machine Notation for Control Abstraction

- an XML-based markup language which provides a generic state-machine based ***execution environment*** based on CCXML and Harel State charts.
- Developed by the W3C: http://www.w3.org/TR/scxml/
- First Draft July 5, 2005. Latest Candidate Recommendation March 13, 2014
- Defines the XML *and* the logical algorithm for execution – an engine specification

A typical example: a stopwatch


```
<?xml version="1.0" encoding="UTF-8"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0" initial="reset">
  <state id="reset">
    <transition event="watch.start" target="running"/>
  </state>
  <state id="running">
    <transition event="watch.split" target="paused"/>
    <transition event="watch.stop" target="stopped"/>
  </state>
  <state id="paused">
    <transition event="watch.unsplit" target="running"/>
    <transition event="watch.stop" target="stopped"/>
  </state>
  <state id="stopped">
    <transition event="watch.reset" target="reset"/>
  </state>
</scxml>
```



A bit more complex: a microwave


```
<datamodel>
  <data id="cook_time" expr="5"/> <data id="door_closed" expr="true"/> <data id="timer" expr="0"/>
</datamodel>

<parallel id="oven">

  <state id="engine">
    <initial>
      <transition target="off"/>
    </initial>
    <state id="off">
      <transition event="turn.on" target="on"/>
    </state>
    <state id="on">
      <initial>
        <transition target="idle"/>
      </initial>
      <transition event="turn.off" target="off"/>
      <transition cond="timer &gt;= cook_time" target="off"/>
      <state id="idle">
        <transition cond="In('closed')" target="cooking"/>
      </state>
      <state id="cooking">
        <transition cond="In('open')" target="idle"/>
        <transition event="time">
          <assign location="timer" expr="timer + 1"/>
        </transition>
      </state>
    </state>
  </state>

</parallel>
```

```
<state id="door">
  <initial>
     <transition target="closed"/>
  </initial>
  <state id="closed">
     <transition event="door.open" target="open"/>
  </state>
  <state id="open">
    <transition event="door.close" target="closed"/>
   </state>
</state>
```

Why use SCXML?

- Almost everyone uses some form of 'state machine', implicitly
- Trivial state management typically becomes less trivial over time
- Often the realization a proper state machine is needed, kicks in 'too late'
- Use a generalized finite state machine like SCXML to:
  - better validate and test the state management and transition logic
  - encapsulate and generalize the processing logic, promote re-use
  - simplify and standardize the usage
  - support easier extend and customize
  - hook up 'listeners' into the state machine for extra/pre/post processing
- The 'overhead' of using a proper state machine is almost always neglectable

Apache Commons SCXML

A brief history

- Started in 2005
- First release 0.5 in 2007, last release 0.9 in 2010 ...
- First, and still only, open source Java implementation of the specification
  (there are several implementations though in Python, Ruby, C++, Javascript, Lua, etc.)
- Version 0.9 (still) used quite a lot, including some commercial products
- But ... the project got 'stalled' in 2010 with very little/no activity
- The SCXML specification in the meantime however changed quite a bit...
- Status until 4th quarter 2013: project seemed 'dead in the water', ready for the attic?
  No :)
- Development has picked up again, rapidly working towards specification alignment
- Next version will be Commons SCXML 2.0, with some radical (breaking) changes!

Commons SCXML 2.0
features (trunk)

- SCXML processing algorithm now based on latest specification!
- Full support for SCXML core model elements
- SCXML datamodel and external communication support limited or not compliant (yet)
- Hierarchical datamodel contexts, external 'root' context can be used to inject your data
- Complete in memory Java model, 'bootstrap' you own SCXML model through code
- Manual external event processing – run Commons SCXML embedded 'in process'
- Apache Commons JEXL, Groovy and (limited) Javascript & XPath expressions
- SCXMLListener notification support for receiving callbacks during event processing
- Runtime SCXML instance state can itself be (de)serialized and persisted
- Easy to use and add custom 'actions'
- ***And most importantly: easy to embed, extend and customize!***

Commons SCXML 2.0

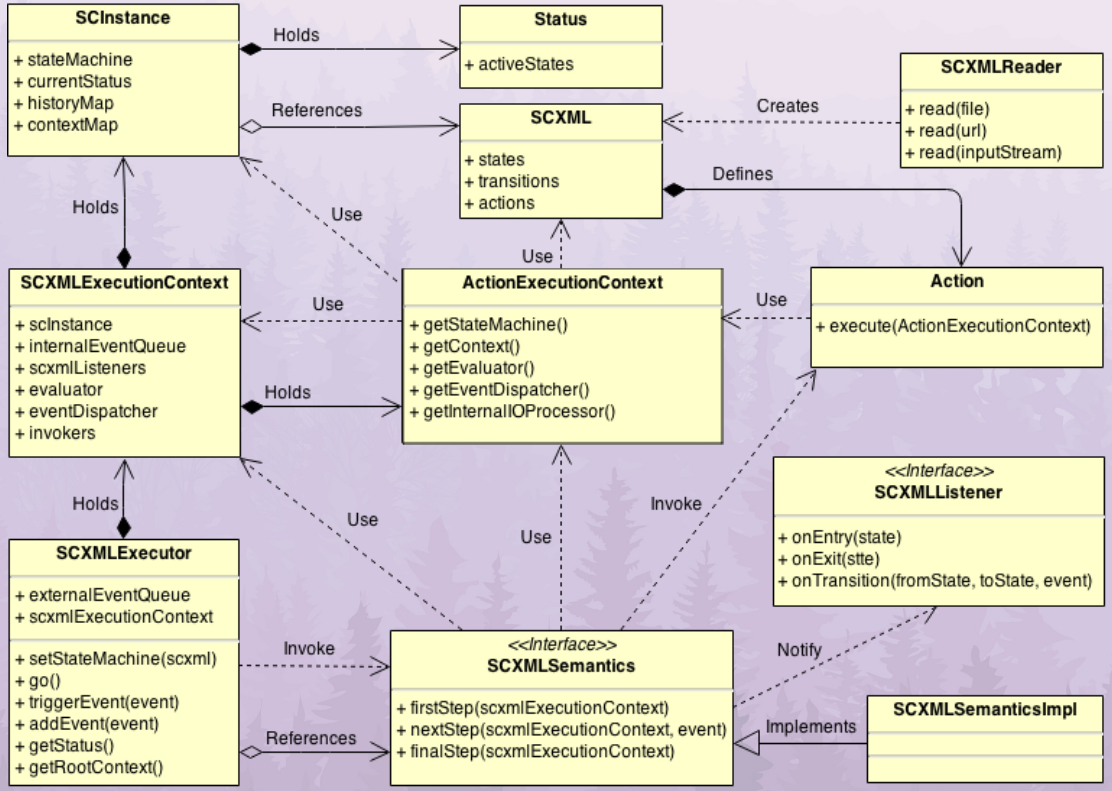
A simple demo

Showing a stopwatch example:

- embedded in a Swing application, with a custom SCXMLListener connected to:
- scxmlgui – A graphical editor for SCXML finite state machines.
  https://code.google.com/p/scxmlgui/ (Apache License 2.0)

Commons SCXML 2.0

High level architecture



Commons SCXML 2.0

Using custom actions

- *Note: all build-in SCXML executable content elements are Actions*
  plenty of examples to go by: `<if>`, `<foreach>`, `<script>`, `<log>`, `<send>`, etc
- Using custom Actions allows 'just about anything' to be wired into the SCXML processing
- *Just make sure they are Serializable, stateless and thread safe!*

Configuring custom Actions using a list of CustomAction wrappers for the SCXMLReader:

```
List<CustomAction> customActions = new ArrayList<CustomAction>();
CustomAction ca = new CustomAction( "http://my.custom-actions.domain"    // custom namespace
                                  , "action"                             // custom XML element local name
                                  , MyAction.class);                     // the actual custom Action class
customActions.add(ca);

SCXML scxml = SCXMLReader.read(url, new SCXMLReader.Configuration(null, null, customActions));
```

Commons SCXML 2.0

Using custom actions

Writing the custom Action:

```
public class MyAction extends Action {
    private static final long serialVersionUID = 1L;
    private String expr;

    public final void setExpr(final String expr) { this.expr = expr; }

    @Override
    public void execute(ActionExecutionContext exctx) throws ModelException, SCXMLExpressionException {
        Context ctx = exctx.getContext(getParentEnterableState());
        ctx.setLocal(getNamespacesKey(), getNamespaces());
        System.out.println("MyAction.expression result: " + exctx.getEvaluator().eval(ctx, expr).toString());
        ctx.setLocal(getNamespacesKey(), null);
    }
}
```

Using the custom Action:

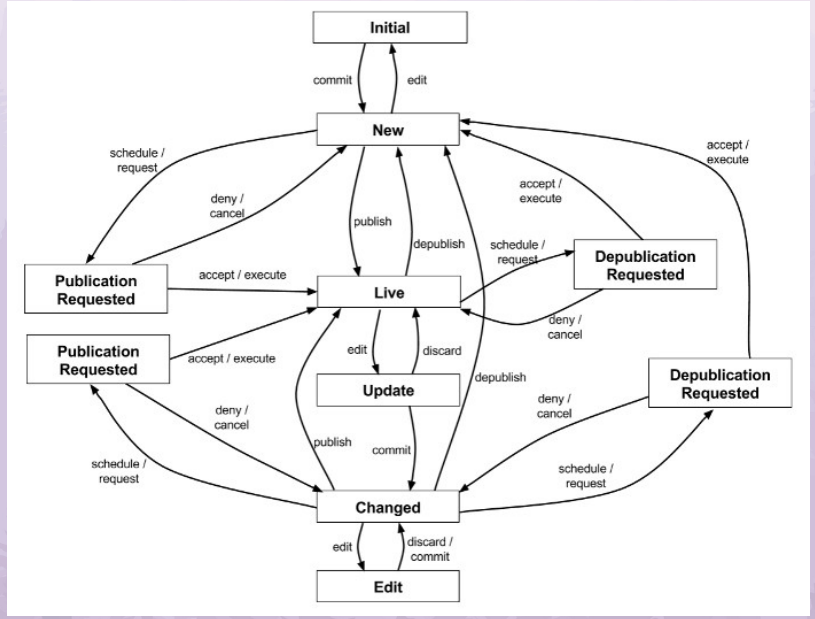
```
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0"
       xmlns:my="http://my.custom-actions.domain" initial="custom">
  <final id="custom">
    <onentry>
      <my:action expr="1+1"/>
    </onentry>
  </final>
</scxml>
```

Commons SCXML 2.0

Use case: Hippo CMS document workflow

Hippo CMS is an open source Content Management System using Commons SCXML for its document workflow:

- used to be 'hand coded', which was rather difficult to extend and customize
- content and workflow state is stored in a JCR (Apache Jackrabbit based) repository
- workflow process configuration (SCXML) is now also stored in the repository
- Many workflow processes can be executing concurrently



Commons SCXML 2.0

Use case: Hippo CMS document workflow

Implementation details:

- http://svn.onehippo.org/repos/hippo/hippo-cms7/repository/trunk/workflow/ \
  ./src/main/java/org/onehippo/repository/
  (open source, Apache License 2.0)
- Uses custom SCXMLWorkflowContext and SCXMLWorkflowData objects as 'bridge' between the JCR Repository 'external' context and the SCXML 'internal' context
- Uses Groovy as expression language
- Uses custom SCXML Actions as worflow operation 'commands'
- Dynamic workflow: workflow definitions can be customized and extended at runtime
- SCXML (internal) state is *not* persisted: the external (JCR) data is leading as well as highly volatile, so the state machine is restarted on every invocation

Commons SCXML 2.0

Use case: Hippo CMS document workflow

Quick demo

Commons SCXML 2.0
Roadmap

http://commons.apache.org/proper/commons-scxml/roadmap.html

- Milestone 0: Cleanup (done, 2014-03-11)
  - Dropped support for outdated/broken features
- Milestone 1: SCXML processing Algorithm (done, 2014-04-03)
  - Complete redesign and implementation of SCXMLSemantics and architecture
- Milestone 2: Datamodel and expression language(s)
  - Align with the specification, better support for ECMAScript and (optionally) XPath
- Milestone 3: External communications
  - Full support for <invoke> and <send> elements
- Release 2.0

That's all folks

Please check out the project!

We are very open to contributions and participation

and will welcome any help.

So if you're interested: join the community!

The project:	http://commons.apache.org/proper/commons-scxml/
The community:	http://commons.apache.org/proper/commons-scxml/mail-lists.html
The specification:	http://www.w3.org/TR/scxml/