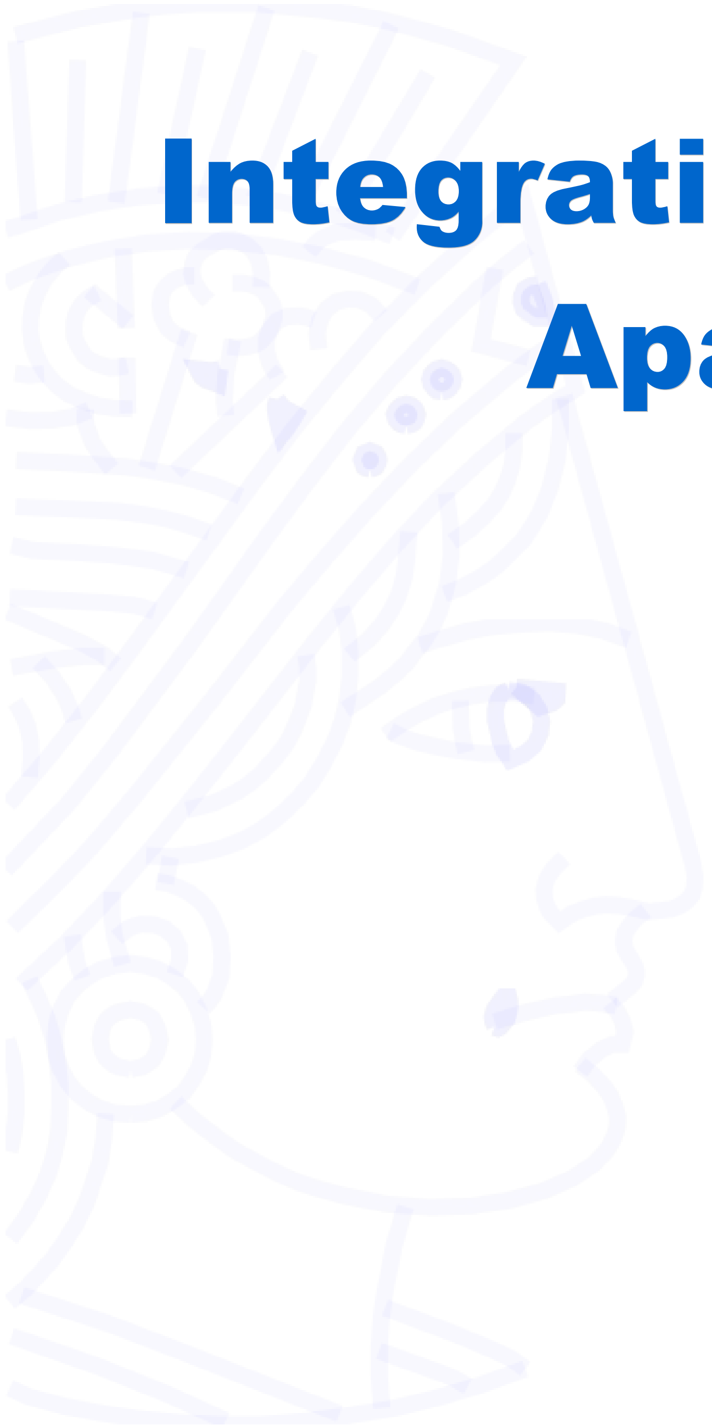
Integrating Databases with Apache Cocoon

Christian Haul
haul@apache.org



Outline

- prerequisites
- JDBC2 vs JDBC3
- (J2EE)
- plain JDBC
- ESQL
- SQL transformer
- actions
- object-relational-bridge
- how to chose

Who ...

- 1997-2003 teaching assistant Technische Universität Darmstadt – Databases and Distributed Systems Group
- 2000-2003 student labs Cocoon + Databases
- since 2001 committer
- mainly ESQL (with Torsten Curdt) / database actions (and input modules)

Prerequisites

- basic Cocoon knowledge
- XSP
- Java
- sitemap
- SQL
- databases aka relational model

Database Connectivity

- [2.1] have database “block”
- add `driver.jar` to libs
- for Avalon connection pools
  - add driver to `web.xml`
  - add database URL to `cocoon.xconf`
  - restart cocoon
  - usable from
    - ESQL
    - SQL transformer
    - Flow
    - custom Avalon components

Step-By-Step (1)

```
> cp mydbms.jar $COCOON/WEB-INF/libs
```

web.xml

```
<webapp>
   ...
   <init-param>
      <param-name>load-class</param-name>
      <param-value>
         org.hsqldb.jdbcDriver
         com.mydbms.Driver
      </param-value>
   </init-param>
   ...
</webapp>
```

Step-By-Step (2)

cocoon.xconf

```
<cocoon version=”2.0”>
 ...
 <datasources>
  <jdbc logger=”some.logger” name=”mydb”>
   <pool-controler min=”5” max=”10”/>
   <dburl>jdbc:mydbms:mydb://host:port</dburl>
   <user>username</user>
   <password>******</password>
  </jdbc>
 </datasources>
 ...
</cocoon>
```

Connectivity (2)

- ESQL need not use connection pools
- connection pools can use J2EE data source
- special options for ORACLE
- special pool for INFORMIX
- see Avalon Excalibur for details

JDBC2 vs JDBC3

- or J2SDK 1.4 versus older versions
- incompatible API change
- JDBC2 compliant connection class does not implement all methods of JDBC3 interface
- never mix versions
- [2.1] uses delegation instead of inheritance: problem solved

Checking Connectivity

- things to look out for in core.log
- no suitable driver
  - driver not loaded or connection error
- attempts to connect on startup
- keep alives using simple query
  - may fail

Database View

- two tables

```
users (name, uname, uid)
users_groups (uid, gid)
groups (gname, gid)
```

note: some DBMSs i.e. PostgreSQL don't like those names.... (reserved words)

JDBC (1) Overview

- no restrictions for own JAVA code
- need to implement `Composable / Servicable` interface for Avalon connection pools
  - use from within a `Component`
  - obtain data source selector from `ComponentManager`
  - obtain data source from selector
  - do JDBC calls
  - release data source
  - release selector
- in future Cocoon will use Avalon Fortress, no selector needed any more

JDBC (2) Summary

- good stuff
  - everything is under your control
- downside
  - not leveraging the framework
  - everything needs to be done manually
  - access to Avalon pools only from Avalon components

ESQL (1) Overview

- logicsheet for use with XSP
- used to create a generator
- thin layer on top of JDBC
  - JDBC knowledge required
- supports almost all features your DBMS offers
- typical use: selects

page.xsp

ESQL(2) Example

```
<xsp:page language="java"
 xmlns:xsp="http://apache.org/xsp"
 xmlns:xsp-request="http://apache.org/cocoon/request/2.0"
 xmlns:esql="http://apache.org/cocoon/SQL/v2">
 <page>
  <esql:connection>
   <esql:pool>mydb</esql:pool>
   <esql:execute-query>
    <esql:query>select name from users where serial=
     <esql:parameter type="int">
      <xsp-request:parameter name="serial"/></esql:parameter>
    </esql:query>
    <esql:results>
      <ol><esql:row-results>
       <li><esql:get-string column="1"/></li>
      <esql:row-results></ol>
    </esql:results>
   </esql:execute-query>
  </esql:connection>
 </page>
</xsp:page>
```

ESQL (3) Ex. Result

result

```
<page>
 <ol>
  <li>John Doe</li>
  <li>Maria Stuart</li>
 </ol>
</page>
```

Warning

- never trust input from the client side without validation!
- `PreparedStatements` are part of your defence line

ESQL (4) Dynamic Elements

- connection
- query (SQL injection!)
- `PreparedStatement` parameters (value only)
- start row / number of rows to display
- column in `<esql:get-XXX/>`
- access to `ResultSet`
- access to `ResultSetMetaData`
- conditional branches:
    - error, no results, more results, ...

ESQL (5) Grouping / No Results

```
<esql:execute-query>
 <esql:query>
  SELECT * FROM users NATURAL JOIN user_groups
                      NATURAL JOIN groups ORDER BY gid
 </esql:query>
 <esql:results>
  <esql:row-results>
   <esql:group group-on="gname">
    <h1><esql:get-string column="gname"/></h1>
    <ul>
     <esql:member>
      <li><esql:get-string column="name"/></li>
     </esql:member>
    </ul>
   </esql:group>
  </esql:row-results>
  <esql:no-results>
   <h1>Warning....</h1>
  </esql:no-results>
 </esql:results>
</esql:execute-query>
```

ESQL(6) Summary

- good stuff
  - stored procedures
  - grouping
  - nesting
  - paging
  - XML attributes
  - arbitrary SQL
  - fast - compiles to Java
  - low overhead
  - fast prototyping
  - XSP actions
- downside
  - JDBC knowledge required for advanced usage
  - no abstraction layer
  - different views in one page
  - mixes concerns
  - XSP
  - XSP has size limit of 64K byte code

ESQL Advice

- hide ESQL in custom taglib
- don't modify the database unless in XSP actions
- use column number, not column name
- use for complex query results
- or for simple applications
- or prototyping

Transformer (1) Overview

- similar to ESQL
  - but syntax different :-(
- no compilation needed
- more dynamic queries

Transformer (2) Example

page.xml

```
<page xmlns:sql="http://apache.org/cocoon/SQL/2.0">
 <sql:execute-query>
  <sql:query name="cocoonUsers">
   select * from users where
   serial=<sql:substitute-value sql:name="serial"/>
  </sql:query>
 </sql:execute-query>
</page>
```

sitemap

```
<map:transformer type="sql">
 <map:parameter name="use-connection" value="mydb"/>
 <map:parameter name="show-nr-of-rows" value="true"/>
 <map:parameter name="serial" value="{request-param:serial}"/>
</map:transformer>
```

Transformer (3) Ex. Result

result

```
<page xmlns:sql="http://apache.org/cocoon/SQL/2.0">
 <rowset nrofrows="1" name="cocoonUsers"
         xmlns="http://apache.org/cocoon/SQL/2.0">
  <row>
   <name>John Doe</name>
   <uname>jdoe</uname>
   <serial>123456789</serial>
  </row>
 </rowset>
</page>
```

Transformer (4) Summary

- good stuff
  - stored procedures
  - nesting
  - updates / inserts / deletes
  - automatic inclusion of XML from DB
  - caching if late in pipeline
  - better separation than XSP
- downside
  - failures / no alternative paths
  - complex layouts require XSLT
  - no computations
  - limited value escaping

Transformer Advice

- similar to ESQL
- don't use to modify database
- use for display
- or simple applications

Actions (1) Overview

- two flavours
  - “original” and “modular”
  - will not talk about the original ones
- fail / success in sitemap
  - chose different pipelines
- meta data in extra file
- auto generated SQL
- use input modules
  - i.e. values from request, session, auth-fw, ...
- auto increment needs module support in `cocoon.xconf`

Actions (2) Ex. Sitemap

sitemap.xmap

```
<map:sitemap ...>
 ...
 <map:action name="mod-db-add”
      src="o.a.c.acting.modular.DatabaseAddAction">
  <descriptor>database.xml</descriptor>
 </map:action>

 ...
 <map:match pattern=”add-user-groups”>
  <map:act type="mod-db-add">
   <map:parameter name="table-set" value="user+groups"/>
   <map:redirect-to uri=”success”/>
  </map:act>
  <map:redirect-to uri=”failure”/>
 </map:match>

 ...

</map:sitemap>
```

Actions (3) Ex. Descriptor

database.xml

```
<metadata>

 <table name="users" alias="users">
  <keys>
   <key name="serial" type="int" autoincrement="true"/>
  <keys>
  <values>
   <value name="name" type="string"/>
   <value name="uname" type="string"/>
  </values>
 </table>

 <table-set>
  <table name="users"/>
 </table-set>

</metadata>
```

Actions (4) Mult. Rows / Modes

```
<table name="user_groups">
 <keys>
  <key name="uid" type="int">
   <mode name="request-param" type="request"/>
   <mode name="request-attr" type="attrib">
    <parameter>
     org.apache.cocoon.components.modules.output.OutputModule:user.uid[0]
    </parameter>
   </mode>
  </key>
  <key name="gid" set="master" type="int">
   <mode name="request-param" type="all"/>
  </key>
 </keys>
</table>

<table-set name="user+groups">
 <table name="user"/>
 <table name="user_groups" others-mode="attrib"/>
</table-set>
```

Actions (5) Advanced Usage

- results available in sitemap as `table.col[row]`
- results available through output module e.g. as session attributes
- automatic parameter names

  `table.col[row]`

- specify information sources using <mode/>

  `table-set @others-mode="..."` selects different mode

- multiple operations using

  `@set="[master|slave]"`

Actions (6) Summary

- good stuff
  - extra file for meta data
  - different pages for different views
  - automatic SQL
  - update / delete / select / “query”
  - multiple rows / tables
  - auto increments
  - easy prototyping
  - automatic type conversions
  - OR data types
  - input from various sources
- downside
  - extra file for meta data
  - adds complexity to sitemap
  - transactions span only similar operations (add, delete, update)
  - no stored procedures*
  - requires connection pool

Actions Advice

- use to modify database
- use transformer / ESQL for display
- combines well with simple HTML forms
- avoid complex logic

Bridges (1) Overview

- actions++
- stay in object oriented world
- integrates well with business logic in beans
  - cf container managed persistence (CMP)
- automatic SQL
- well known OpenSource bridges
  - OJB [http://db.apache.org/ojb]
  - Hibernate [http://hibernate.sf.net]
  - Castor [http://castor.exolab.org]
  - ...

Bridges (2) Overview cont.

- no integration with Avalon connection pools (yet)
- separate connection (pool) configured outside Cocoon
- meta data in mapping file
- generate persistent classes or mapping e.g. with xDoclet [http://xDoclet.sf.net] or from database meta data e.g. Druid [http://Druid.sf.net]
- Castor: CastorTransformer inserts bean
- JXTemplateTransformer inserts bean
- XSP (don't!)
- use Flow as excellent glue

Bridges (3) Usage

- general steps
  - get connection
  - lookup using an OQL / create object
  - work with object
  - persist object
  - end transaction

Bridges (4) Example Mapping

OJB

```
<class-descriptor class="my.own.stuff.User"
 proxy="dynamic" table="User">

 <field-descriptor name="uName" column="uname"
  jdbc-type="VARCHAR" indexed="true"/>

 <field-descriptor name="name" column="name"
  jdbc-type="VARCHAR"/>

 <field-descriptor name="uid" column="uid"
  jdbc-type="INTEGER" autoincrement="true"/>

</class-descriptor>
```

Bridges (5) Example Class

using xDoclet (OJB contrib)

```
/**
 * @ojb.class
 * @ojb.index name="NAME_UNIQUE"
 *             unique="true"
 *             fields="uName"
 */
public class User {

 /** @ojb.field name="name" length="100" */
 String name;

 /** @ojb.field name="uname" length="8" */
 String uName;

 /** @ojb.field name="uid" autoincrement="true" */
 int uid;
 ...
}
```

Bridges (6) Example Flow

```
broker = PersistenceBrokerFactory
                    .defaultPersistenceBroker();
newUser = new User();
newUser.name = "John Doe";
...
broker.beginTransaction();
broker.store(newUser);
broker.commitTransaction()
```

Bridges (7) Summary

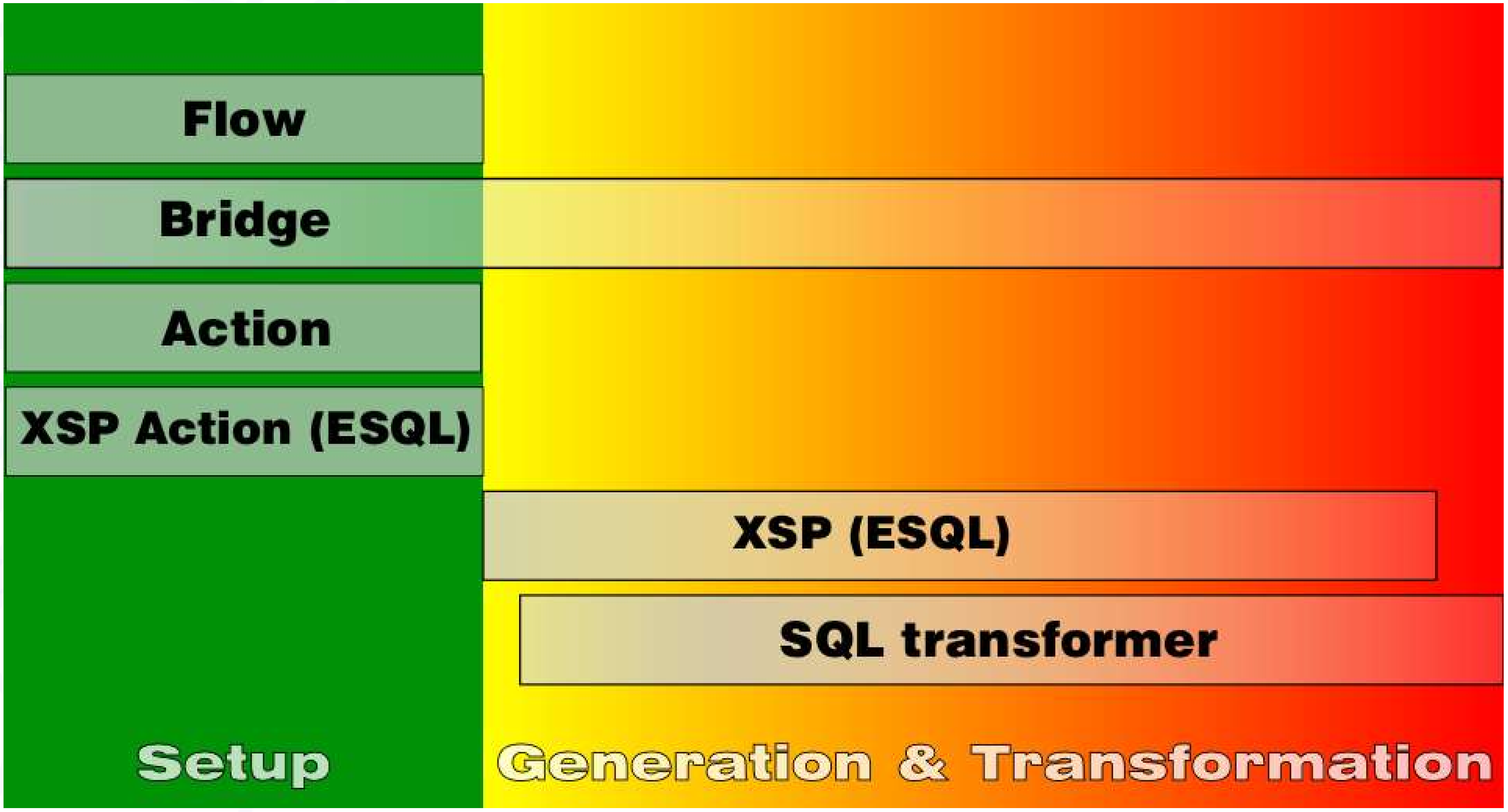
- good stuff
  - powerful
  - integrates well with e.g. Woody + Flow
  - no SQL
  - easy migration from actions
  - usable outside Cocoon / web server
  - meta data through xDoclet or Druid
- downside
  - requires POJOs
  - OQL not as “standard” as SQL
  - slightly more complex

Flow?

- call any JAVA code from Flow
- esp. persistence layer i.e. OR-bridge
- call actions through `legacy.js` (not yet)
- `Database.js` (currently in petstore sample)
  - obtain connection from pool
  - JDBC + some convenience

    i.e. `update() / query()`

Round Up (1)


Flow
Bridge
Action
XSP Action (ESQL)
XSP (ESQL)
SQL transformer
Setup
Generation & Transformation

Round Up (2)

- different solutions for different problems
- keep your pages simple
- use actions + ESQL / transformer + HTML forms
  - for small and throw away projects
  - without complex flow / mainly display
- use flow + bridge / J2EE + Woody
  - for complex applications
  - for maintainable projects

Resources to check out

- http://wiki.cocoondev.org
  - loads of info on database connectivity
  - including usage of Hibernate and OJB
- http://cocoon.apache.org
  - docs
  - javadocs
  - tutorials
  - samples
- http://db.apache.org/ojb
  - general OJB tutorials

The End

?