Apache Cocoon GetTogether

7/11/2003

Ghent, Belgium

Brought to you by:

Sponsored by:

Apache Cocoon GetTogether

Schedule

07/10/2003

08:45-09:30	*Coffee and welcome - sponsored by HP*
09:30-10:15	Introduction ***Stefano Mazzocchi***
10:15-11:00	Integrating database with Cocoon ***Christian Haul***
11:00-11:30	*Break - sponsored by HP*
11:30-12:15	Flow and woody ***Sylvain wallez***
12:15-12:45	Pipelined Fugues ***David Casal***
12:45-13:45	*Lunch*
13:45-14:30	CMS with Cocoon: Lenya ***Michael Wechner***
14:30-15:15	Lightweight tools for successful projects: the Open Source Way ***Bertrand Delacretaz***
15:15-16:30	*Break - sponsored by HP*
16:30-17:30	Cocoon & WebDAV ***Gianugo Rabellino & Matthew Langham***
17:30-18:30	Townhall and Panel Q&A ***All***
	Reception - sponsored by HP

Apache Cocoon GetTogether

Introduction

Stefano Mazzocchi

what cocoon is

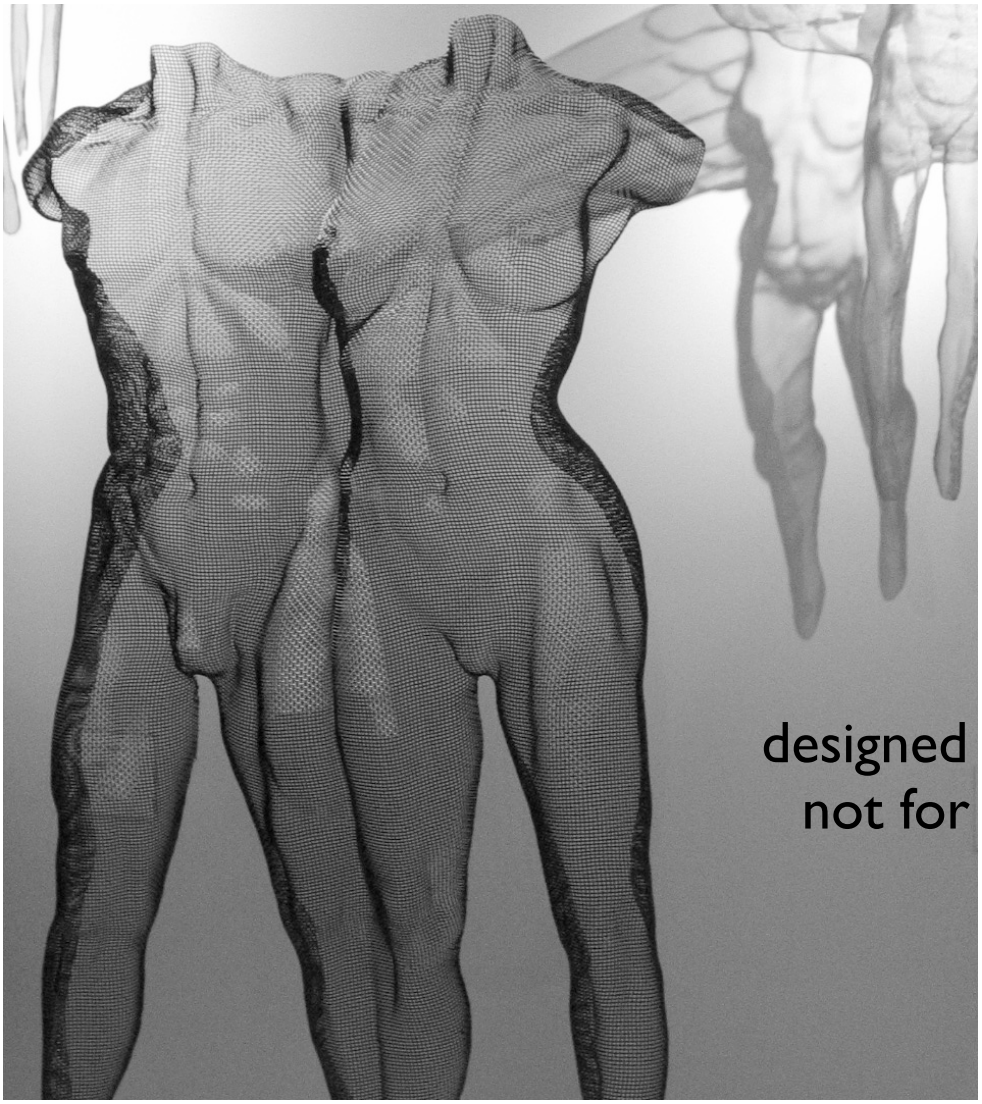
a visual journey

Part 1

philosophy

Now, more than ever, we need people who can lead humanity towards technologies that improve society, rather than technologies that simply improve over technology itself.

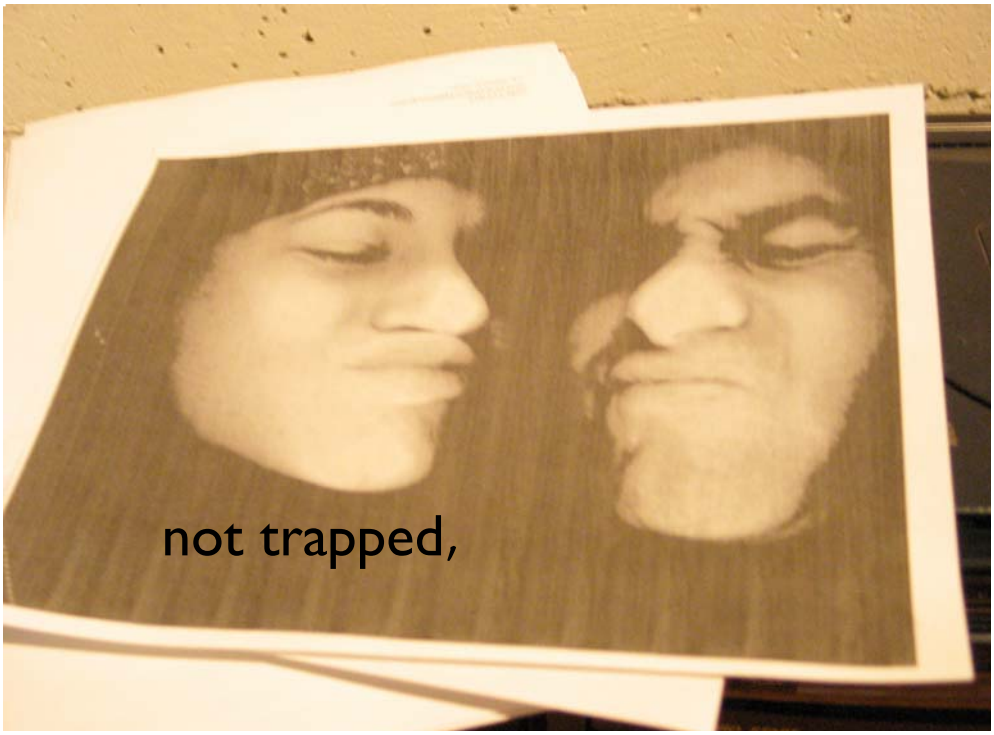
John Maeda



designed for people,
not for machines

not afraid
of looking at things
from a different angle

Guided,



by open standards

looks at the past
for inspiration

no fear of innovating

Balanced between

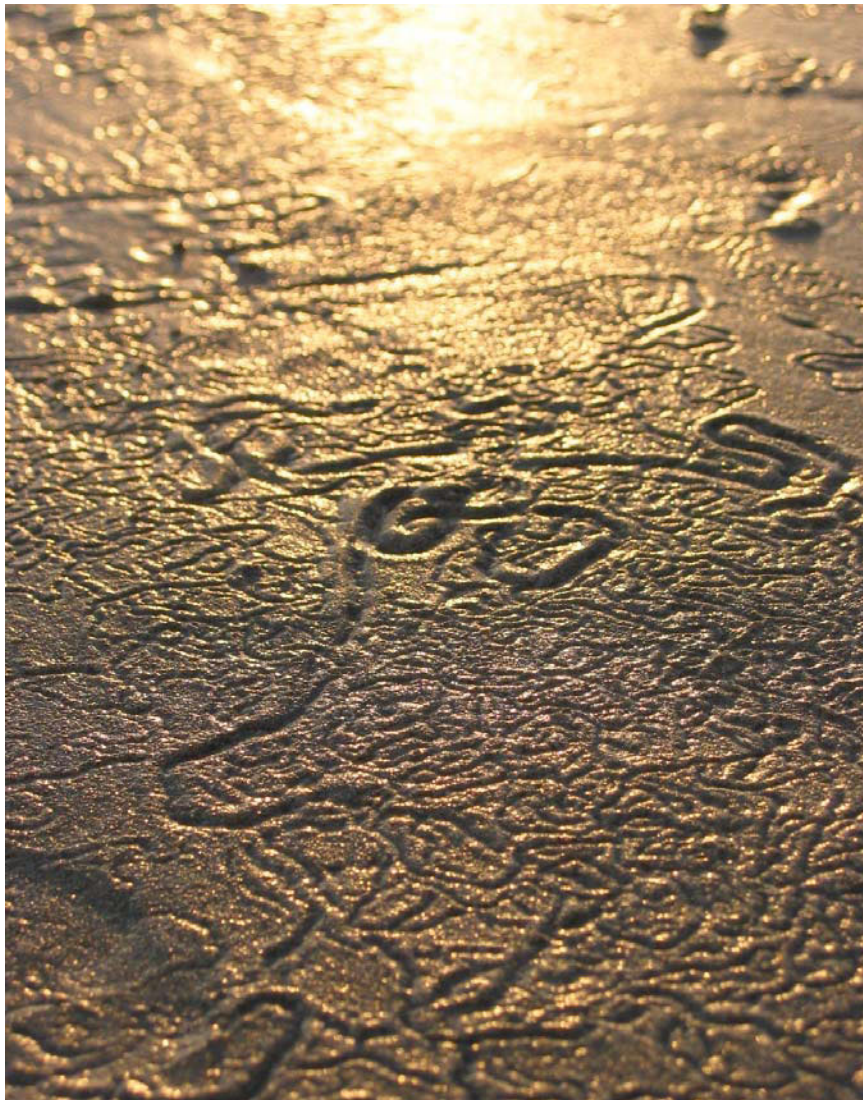
innovation and compatibility

Modular

yet solid

design

and community



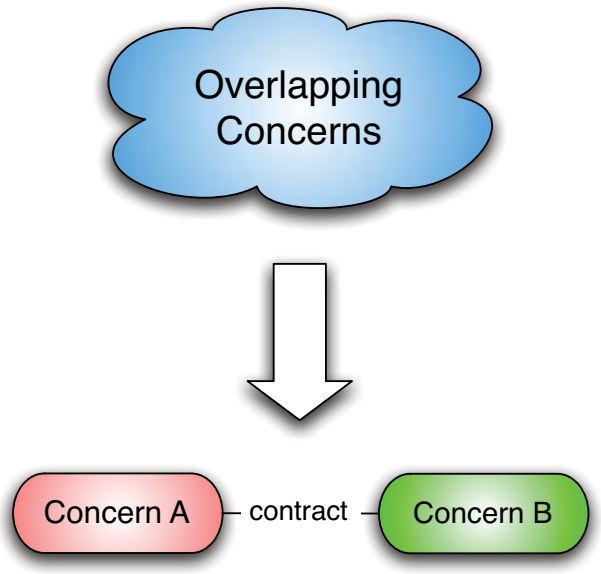
Part 2
Concepts

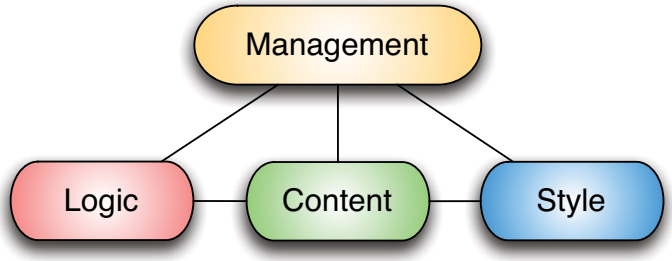
Creativity is making the complicated simple

Charles Mingus

Separation of Concerns

Separation of Concerns (SoC)





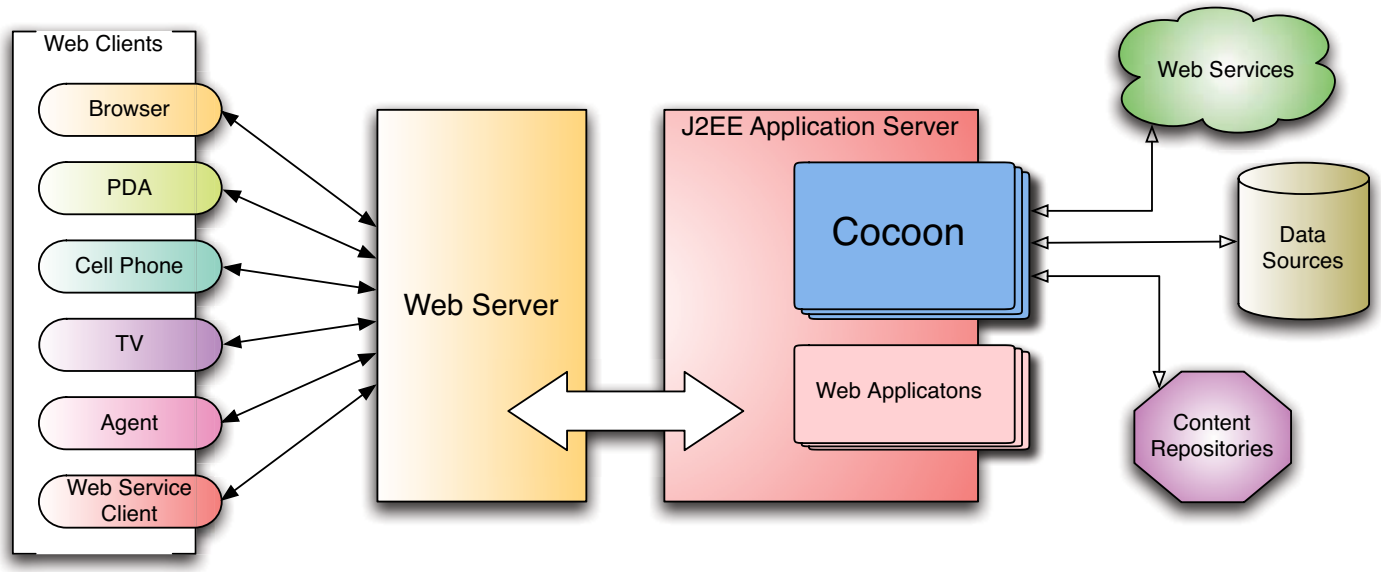
The Pyramid of Contracts

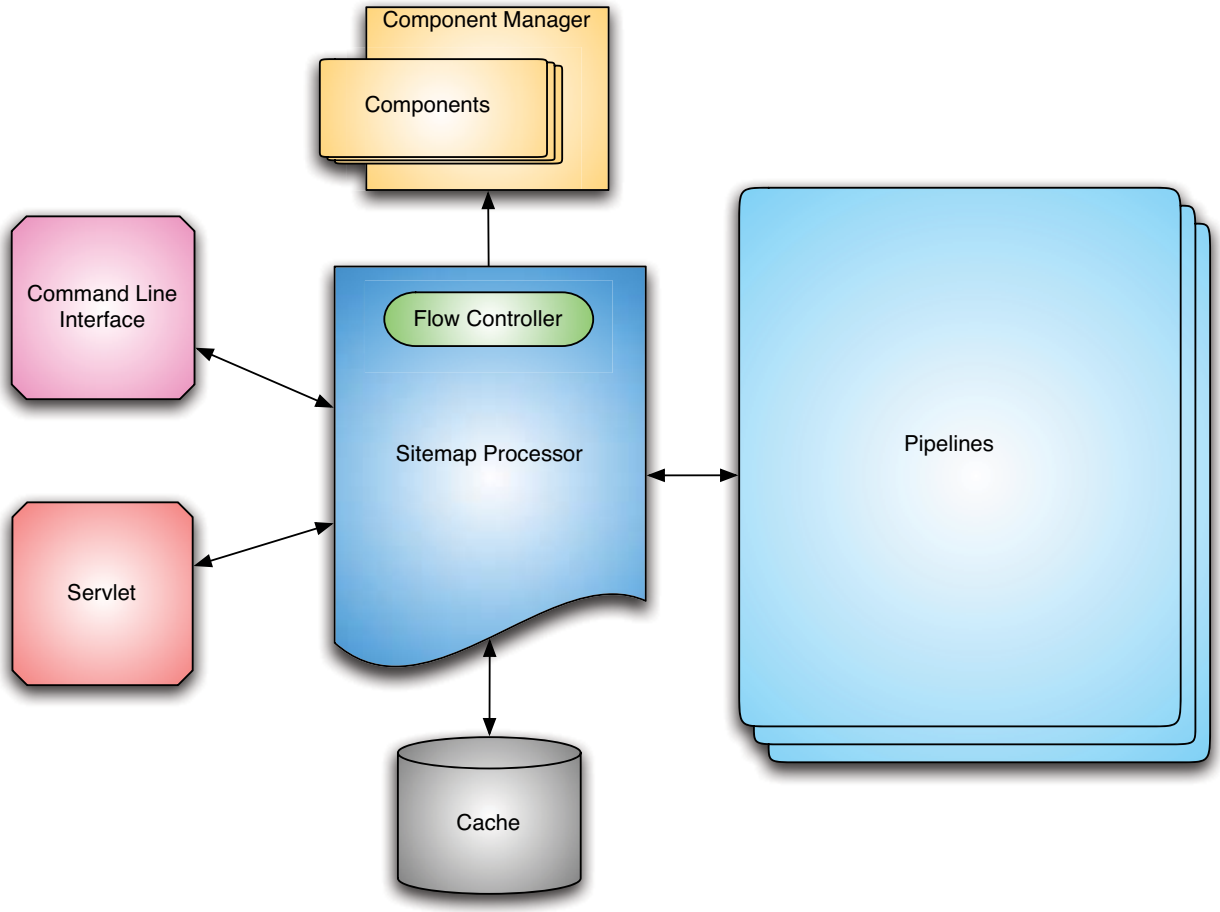
design Part 3

To create, one must first question everything.

Eileen Gray

Architecture

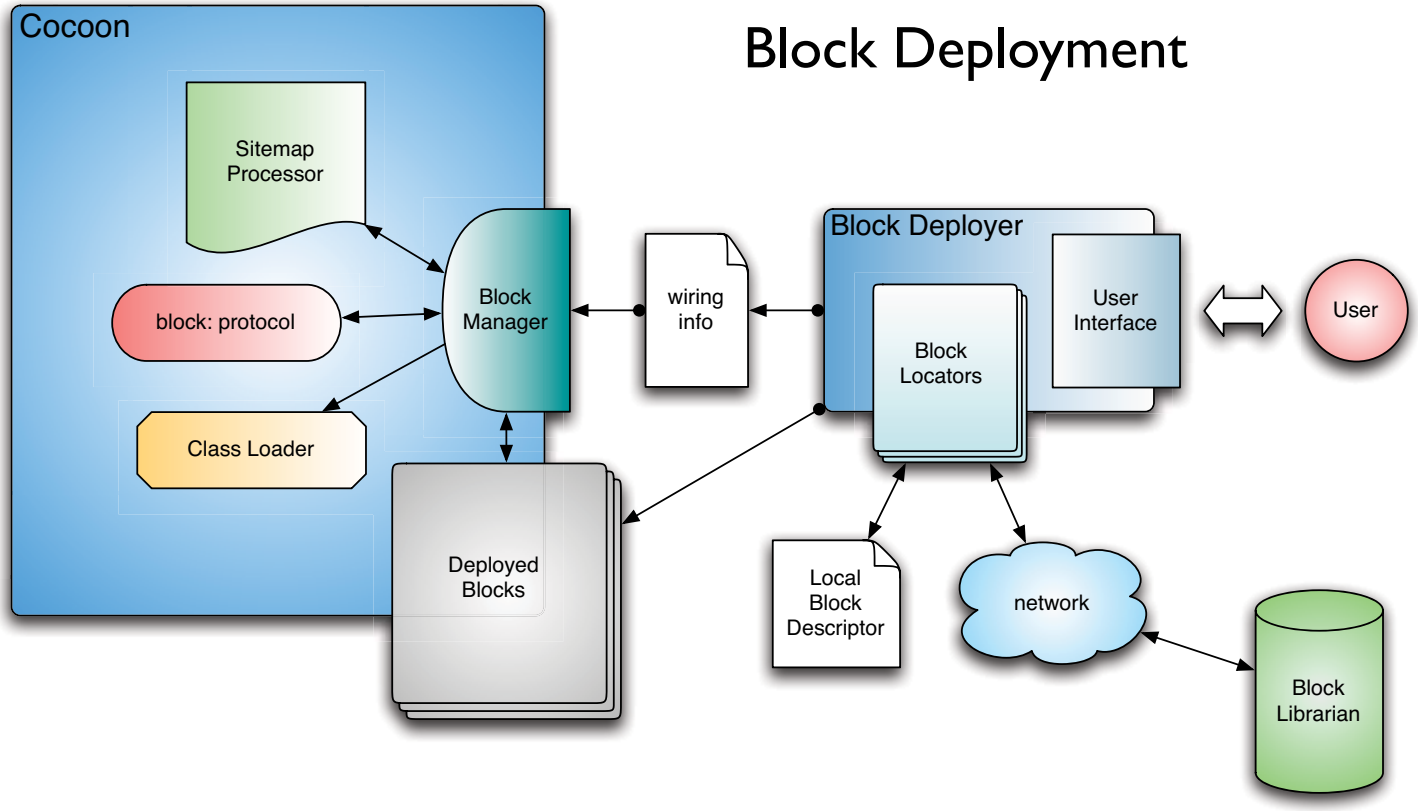



Component Manager
Components
Command Line Interface
Flow Controller
Sitemap Processor
Pipelines
Servlet
Cache

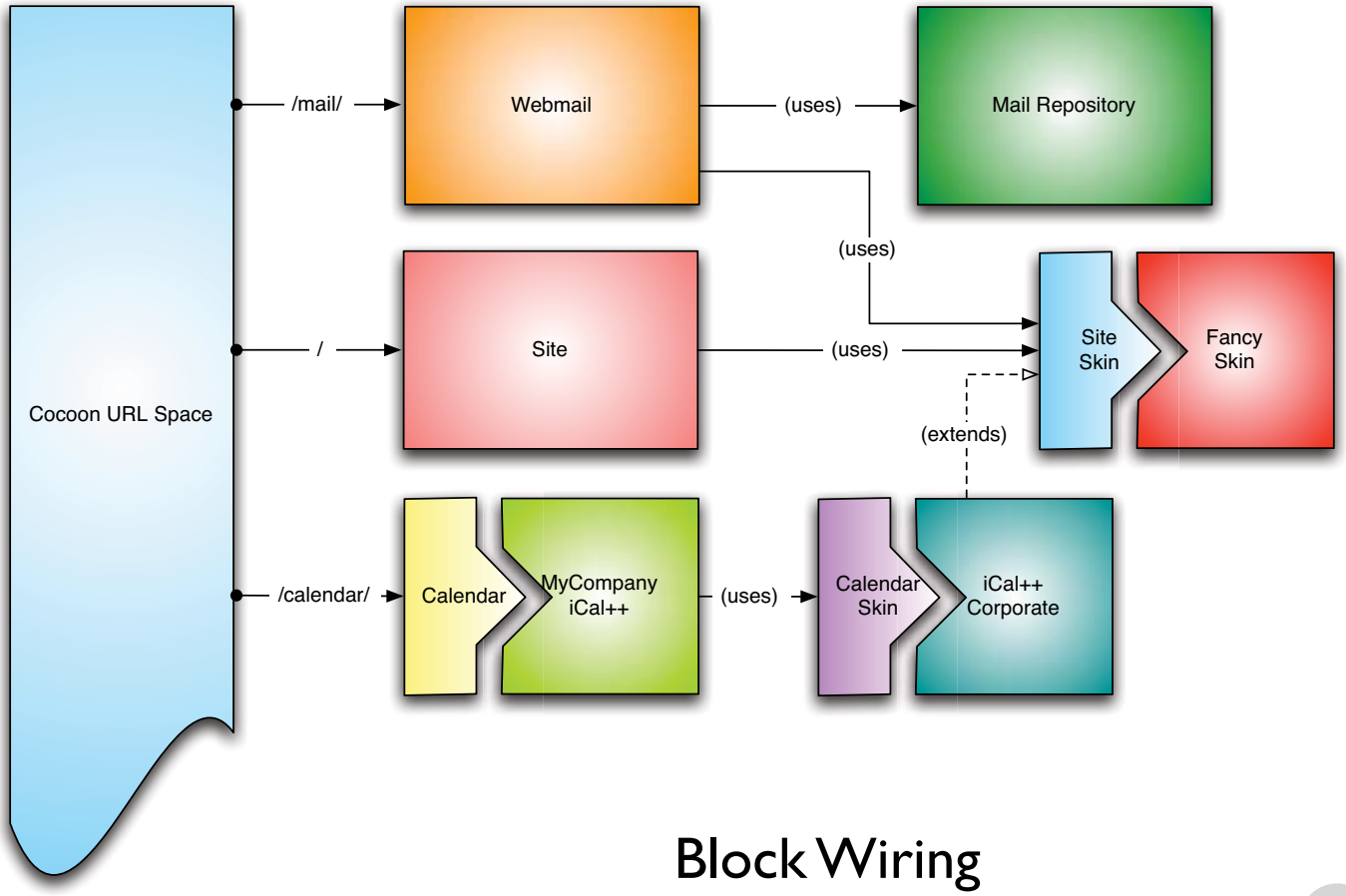

cool !!
yes
SUNDAY

Blocks

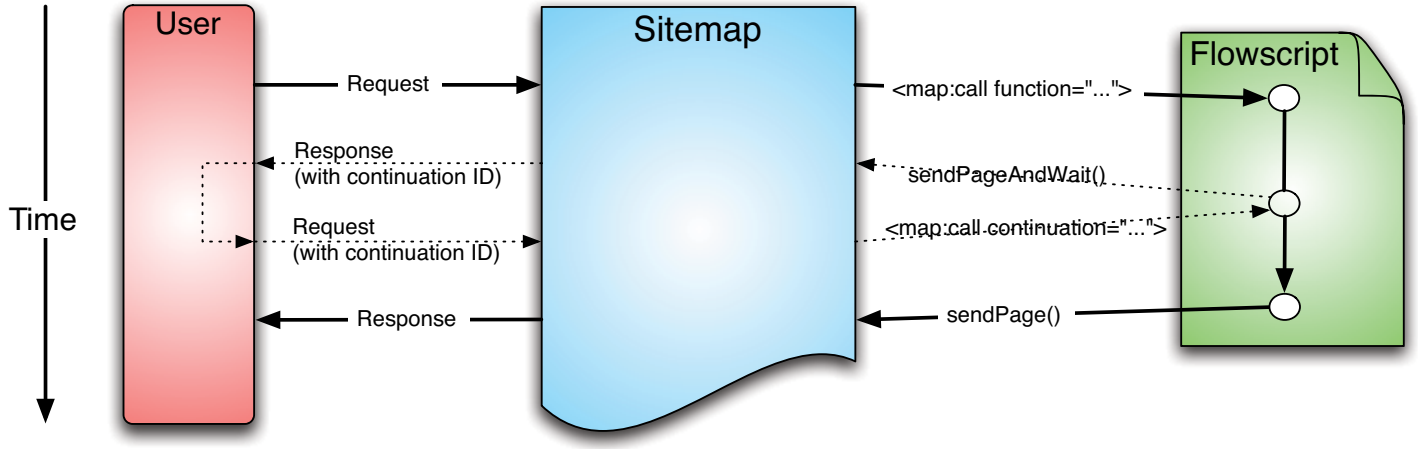
Block Deployment

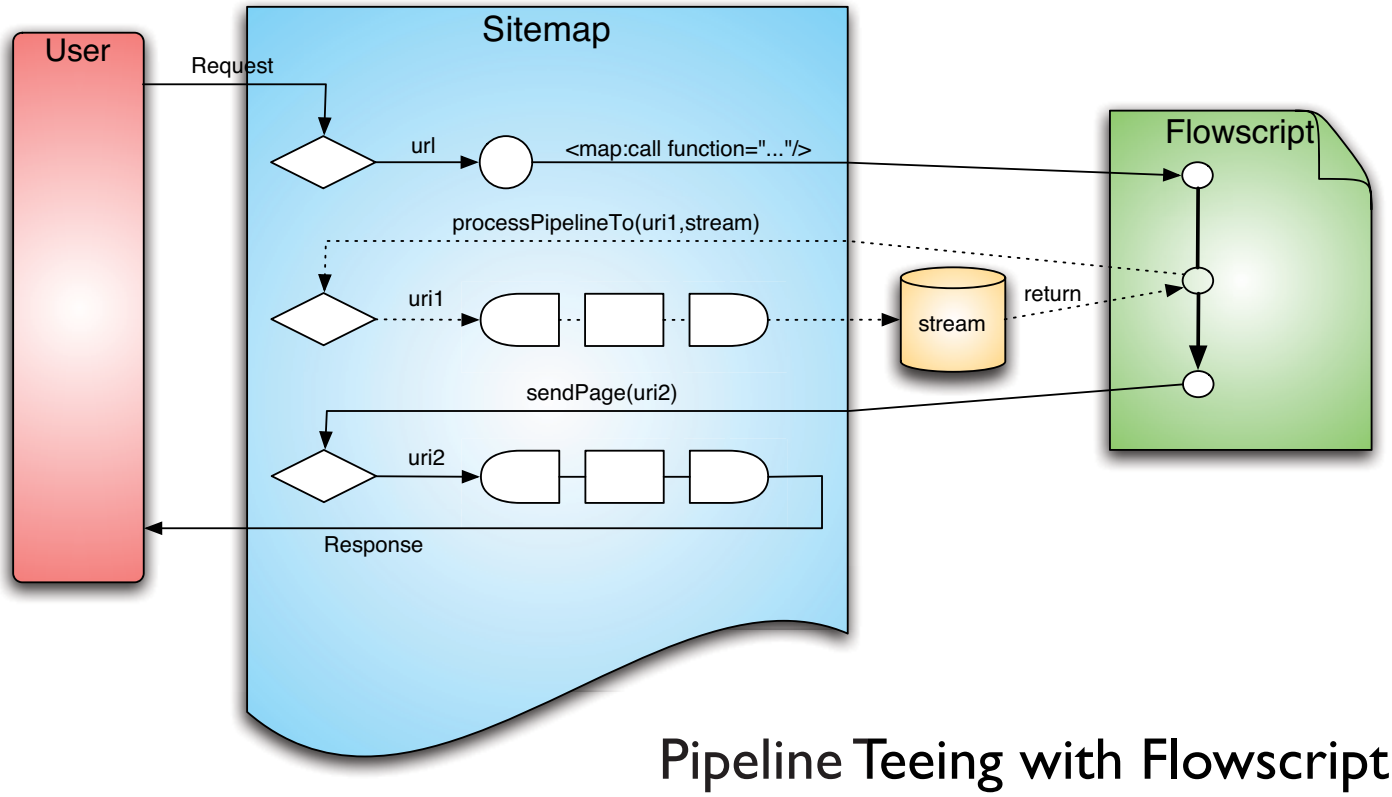


Block Wiring

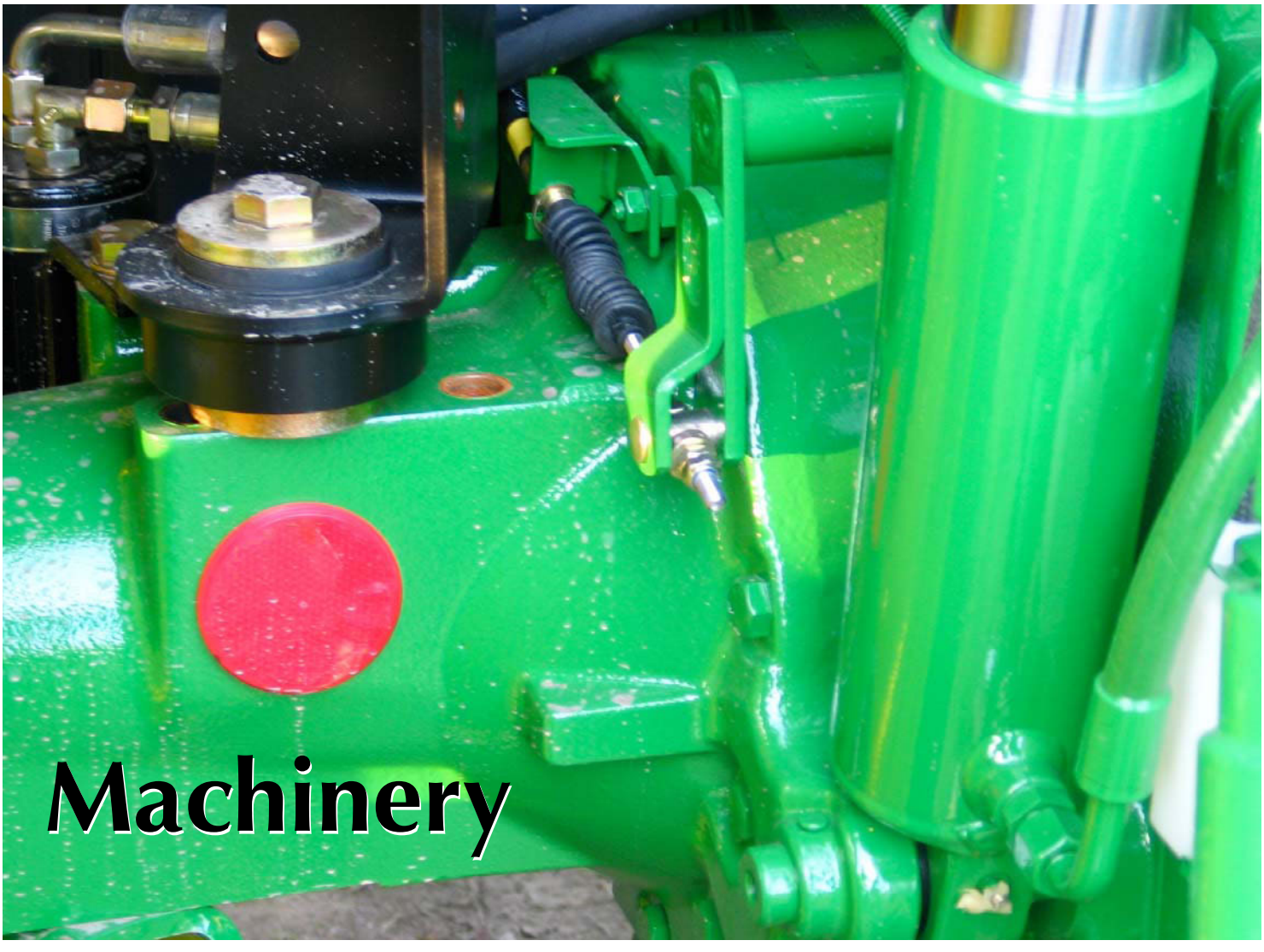


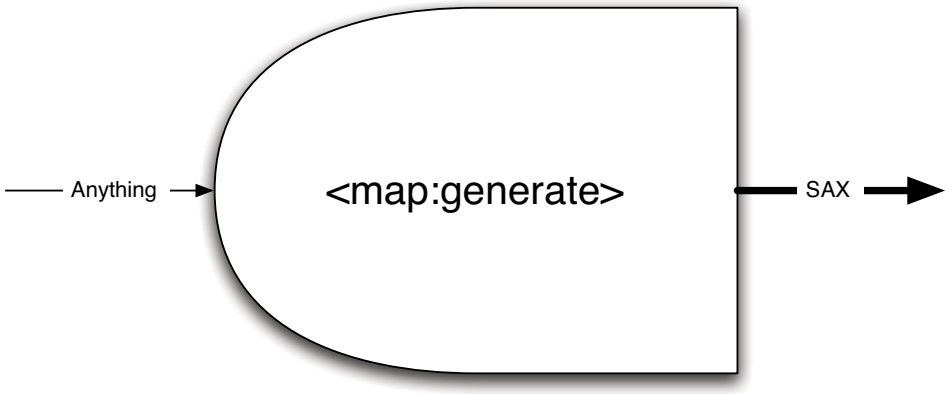
Flowscript with Continuations

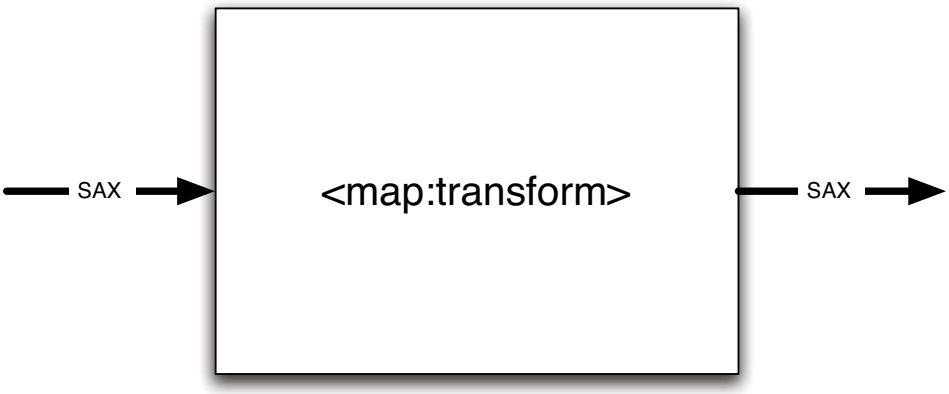


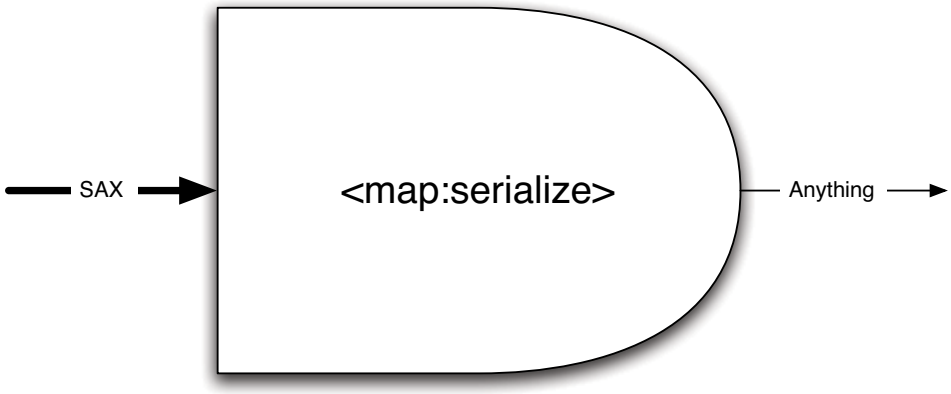


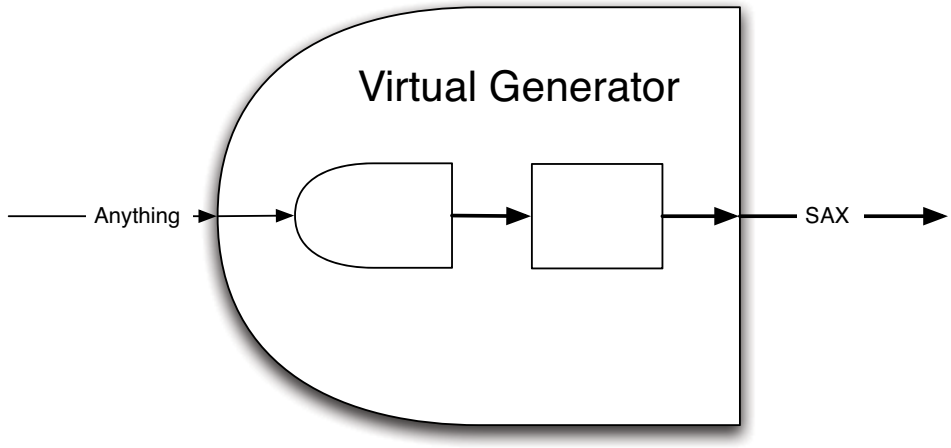
Pipeline Teeing with Flowscript

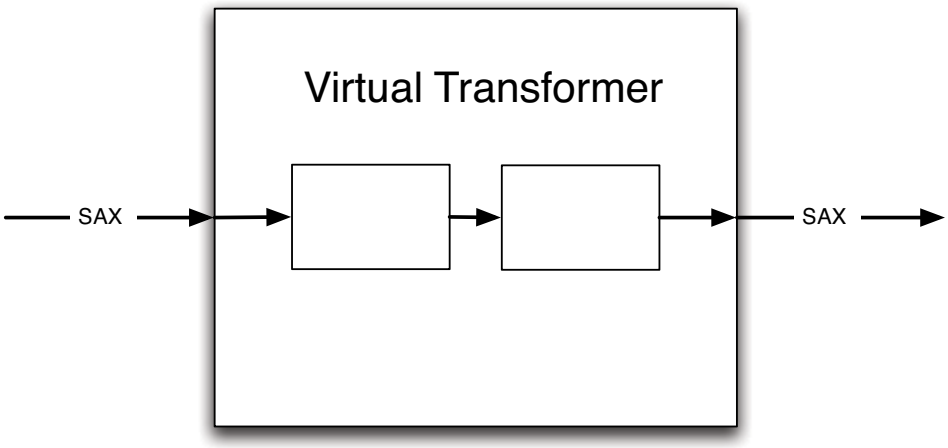


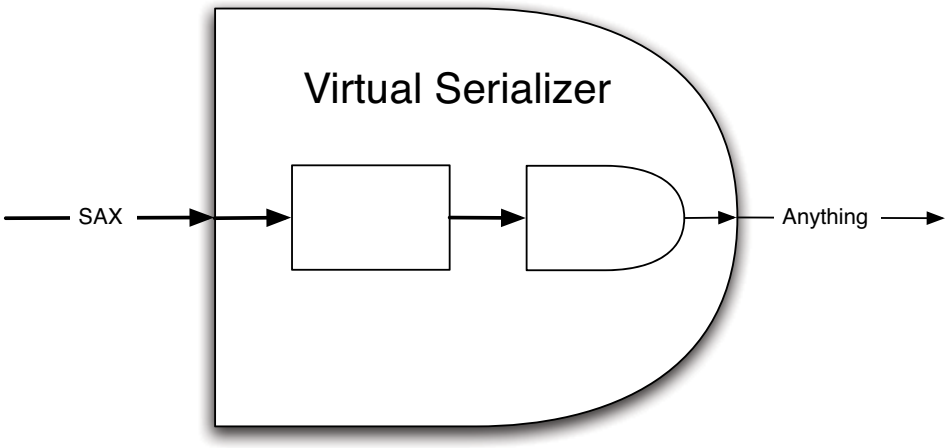

Anything
<map:generate>
SAX

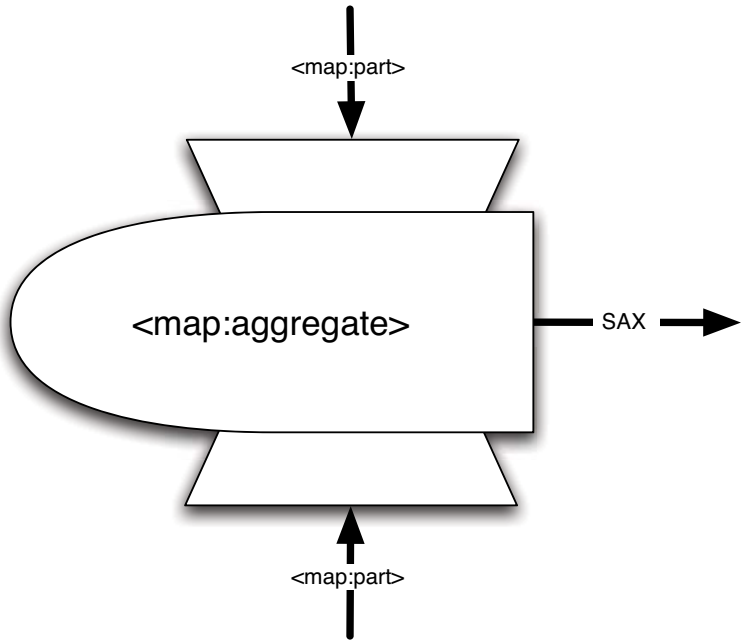

SAX
<map:transform>
SAX


SAX
<map:serialize>
Anything

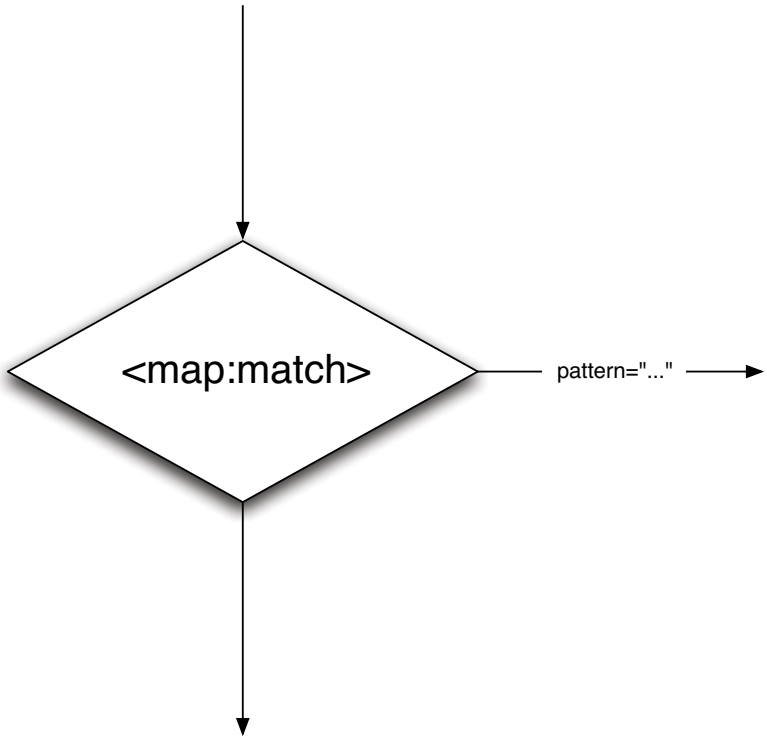

Virtual Generator
Anything
SAX

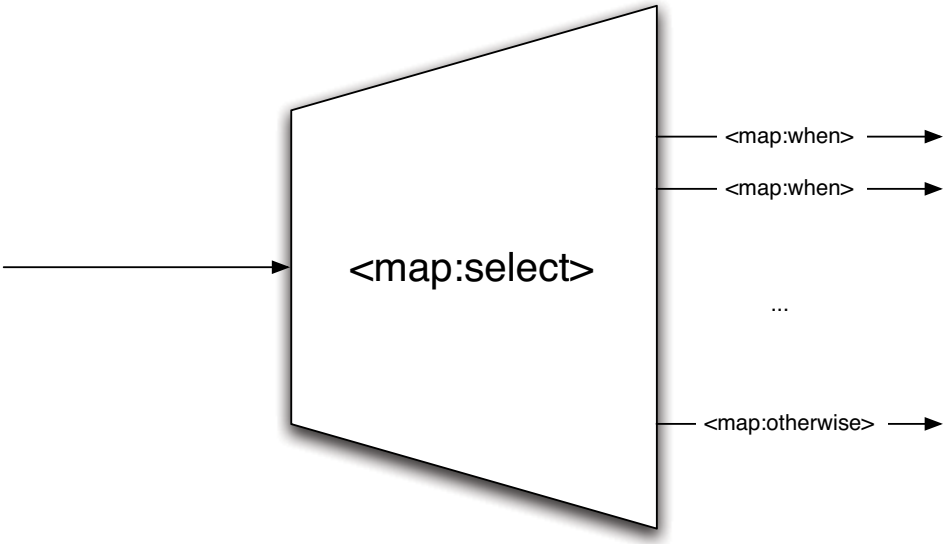

Virtual Transformer
SAX
SAX

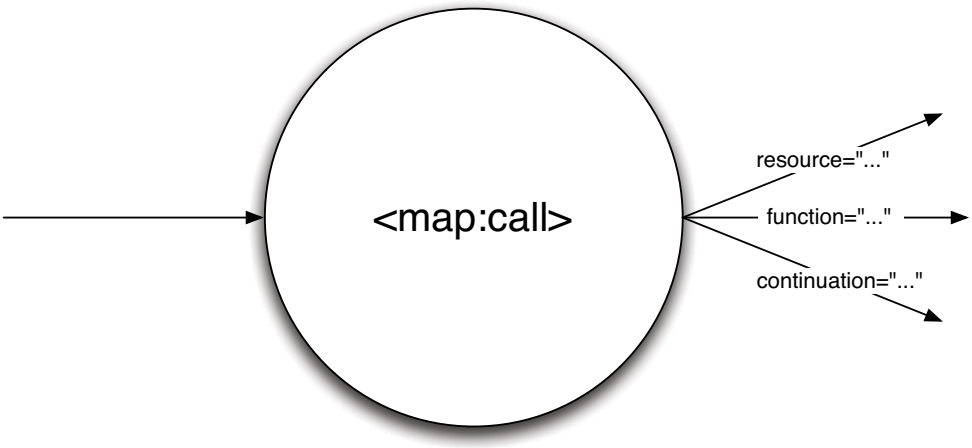

Virtual Serializer
SAX
Anything

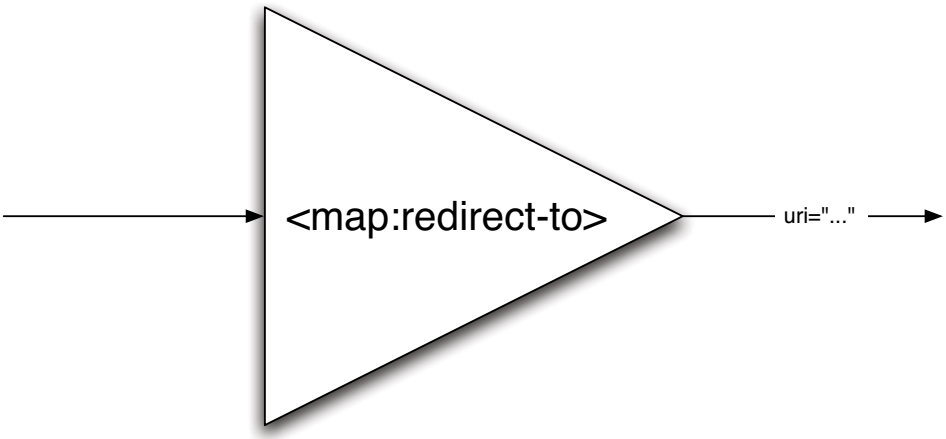

<map:part>
<map:aggregate>
SAX
<map:part>

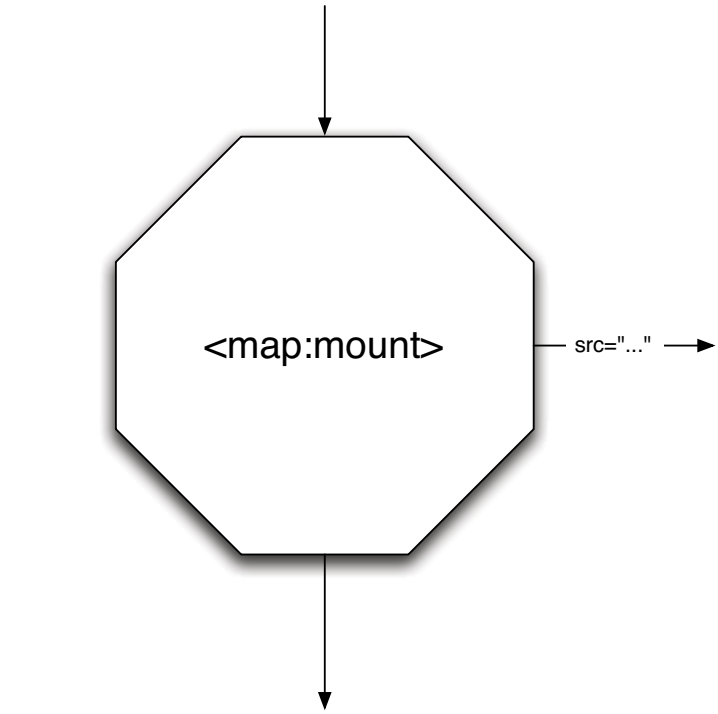

Anything
<map:read>
Anything

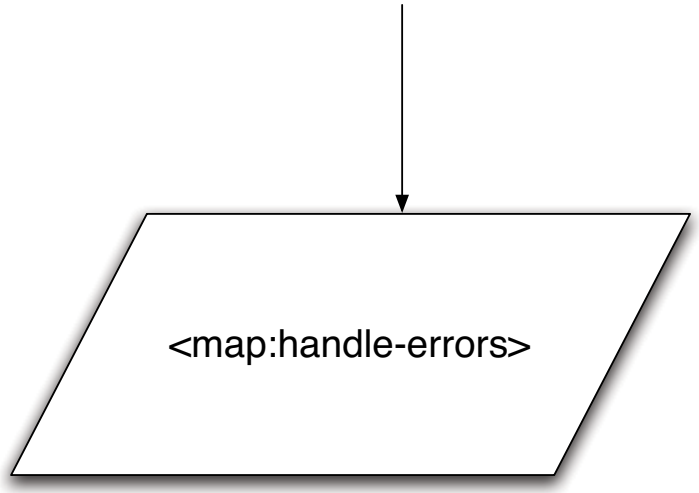

<map:match>
pattern="..."

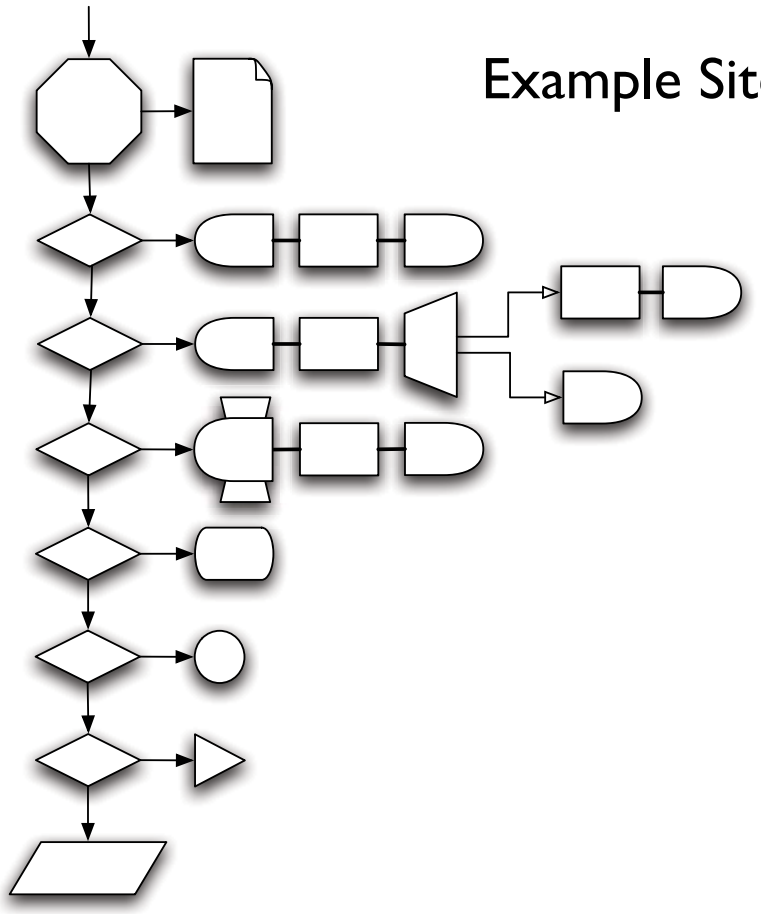

<map:select>
<map:when>
<map:when>
...
<map:otherwise>


<map:call>
resource="..."
function="..."
continuation="..."


<map:redirect-to>
uri="..."


<map:mount>
src="..."


<map:handle-errors>



Example Sitemap

Concepts, pictures and drawings by

Stefano Mazzocchi

<stefano@apache.org>

When you know a thing, to hold that you know it; and when you do not know a thing, to allow that you do not know it - this is knowledge.

Confucius

The end

Integrating Databases with Cocoon

Christian Haul

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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>
```

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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>
```

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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>
```

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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>
```

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

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()
```

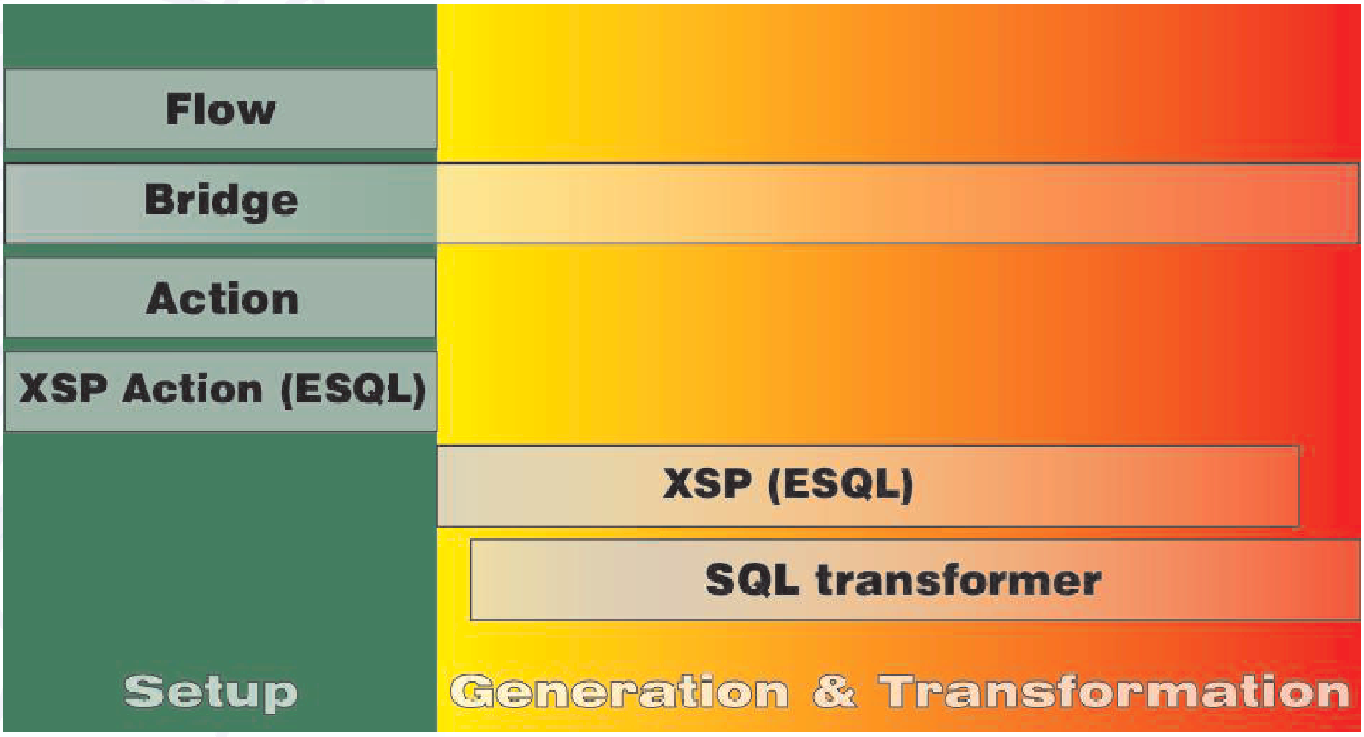

Bridges (7) Summary

- 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)



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

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

The End

?

GetTogether 2003 Ghent, Belgium 200301007 © C. Haul <haul@apache.org>

Flow and Woody

Sylvain Wallez

ANYWARE TECHNOLOGIES

Flowscript & Woody

Webapps made easy with Cocoon

Sylvain Wallez
http://apache.org/~sylvain

www.anyware-tech.com

Flowscript intro

Flow control in Cocoon

- Aren't actions enough ?
  - Yes, but they require state management
  - Quickly becomes complex, hard to understand and to maintain

  → Actions are the traditional "MVC" controller
- Flowscript is a controller…
  - Calls business logic and chooses the view
- …but also more
  - Keeps the application state
  - Describes page flow as a sequential program
  - Easily defines complex interactions

Flowscript intro

Flow script example

```
var cart;
var user;

function checkout()
{
  while(user == null) {
    cocoon.sendPageAndWait("login.html");

    user = UserRegistry.getUser(cocoon.request.get("name"));
  }
  cocoon.sendPageAndWait("shippingAddress.html", {who: user});

  var address = cocoon.request.get("address");
  cocoon.sendPageAndWait("creditCard.html");

  var creditCard = cocoon.request.get("creditCard");
  cocoon.sendPageAndWait("confirmOrder.html");

  EnterpriseSystem.placeOrder(user, cart, address, creditCard);
  cocoon.sendPage("orderPlaced.html");
}
```


Flowscript intro

Why JavaScript ?

- Simpler than Java, although powerful
- Integrates well with Java
- Well-known in the web world
- Allows faster roundtrips (save and reload)
- Supports *continuations*

Calling the view

cocoon.sendPage()

- **`cocoon.sendPage`** invokes the output page (view) with two arguments
  - The view URL, relative to current sitemap
  - A context object made available to the view
    → Can be a Java or JavaScript object

```
cocoon.sendPage("checkout.html",
        {user: loggedUser, email: address});
```

- **`cocoon.sendPage("view.html")`** is like redirecting to "cocoon:/view.html"
- Control then comes back to the script
  → Should normally terminate

Calling the view

cocoon.sendPageAndWait()

- Similar to **`cocoon.sendPage`**
  - Invoke the view with a context object
- The script is *suspended* after the view is generated
  → the whole execution stack saved in a *continuation* object
- Flow between pages becomes *sequential code*
  → No more complicated state automata

Continuations

What is it ?

- Contents of a continuation:
  - Stack of function calls
  - Value of local variables
  - → Most often a lightweight object

→ Creating a continuation does not halt a thread !!

- A continuation object is associated with a unique identifier available to the view
  - → Later used to "resurrect" it

Continuations

Sample flow script revisited

⇨ saved continuations

```
var cart;
var user;

function checkout()
{
  while(user == null) {
    cocoon.sendPageAndWait("login.html");

    user = UserRegistry.getUser(cocoon.request.get("name"));
  }
  cocoon.sendPageAndWait("shippingAddress.html", {who: user});

  var address = cocoon.request.get("address");
  cocoon.sendPageAndWait("creditCard.html");

  var creditCard = cocoon.request.get("creditCard");
  cocoon.sendPageAndWait("confirmOrder.html");

  EnterpriseSystem.placeOrder(user, cart, address, creditCard);
  cocoon.sendPage("orderPlaced.html");
}
```

View layer

How to define the view ?

- It's a regular Cocoon pipeline
  → Preferably in an `internal-only="true"` pipeline
- Two generators providing tight integration
  - JXTemplateGenerator
  - JPath XSP logicsheet

  → Easy access to the context object

View layer

JXTemplateGenerator

- An XML template language inspired by JSTL
- Doesn't allow code, but only access to context variables
  → Simpler than XSP
- Flow values are provided as variables :

```
sendPage("checkout.html",
         {"customer": user, "cart": cart});
```

```
<p>Welcome, #{customer/firstName}</p>
```

- Two expressions languages:
  Jexl with `${…}` and JXPath with `#{…}`

View layer

JXTemplate example

```
Your cart:
<ul>
  <jx:forEach var="item" items="${cart.items}">
    <li>${item.quantity} ${item.name}</li>
  </jx:forEach>
</ul>
<a href="kont/${continuation.id}">Continue</a>
```

```
Your cart:
<ul>
  <li>3 Cocoon T-Shirt</li>
  <li>1 Washing machine</li>
</ul>
<a href="kont/bf6c433aa3148f8ca083f18a83813f81">Continue</a>
```

Will "resurrect" the flow script

Putting it all together

```
<map:pipelines>
  <map:pipeline>

    <map:match pattern="checkout">
      <map:call function="checkout"/>
    </map:match>

    <map:match pattern="kont/*">
      <map:call continuation="{1}"/>
    </map:match>

  </map:pipeline>

  <map:pipeline internal-only="true">

    <map:match pattern="*.html"/>
      <map:generate type="jxt" src="{1}.xml"/>
      <map:transform src="page2html.xsl"/>
      <map:serialize/>
    </map:match>
…/…
```

❶ Call a flow function

❸ Resurrect a continuation

❷ View called by the flow

Putting it all together

FOM: the Flow Object Model

- Provided by the "cocoon" global object
- Access to the environment
  - "`request`", "`response`", "`session`" & "`context`" properties
  - "`parameters`" : sitemap parameters
- Access to the framework
  - Logging, using Avalon components
- Page flow control
  - `cocoon.sendPage()`, `cocoon.sendPageAndWait()`
  - `cocoon.redirectTo()`

Session variables

Global scope = session scope

- Global variables are attached to the session
  - Saved across top-level function invocations
  - Specific to each user

  → Removes most of the needs for session attributes !

Session variables

Example

Shows the login screen only if needed

Won't pass through if not logged in !

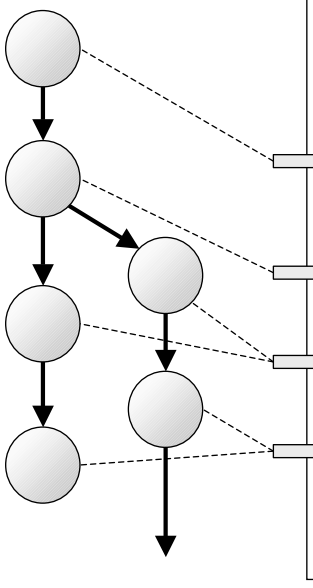
Just clear user info to log out

```
var user = null;

function login() {
  while (user == null) {
     sendPageAndWait("login.html");
     user = UserRegistry.getUser(
       cocoon.request.getParameter("name"),
       cocoon.request.getParameter("password") );
  }
}

function placeOrder() {
  login();
  Accounting.placeOrder(user);
  sendPage("orderPlaced.html");
}

function logout() {
  user = null;
  sendPage("bye.html");
}
```


Managing continuations

Continuation trees

- Browser "back" or "new window"



...

```
var cart;
var user;
function checkout()
{
  while(user == null) {
    cocoon.sendPageAndWait("login.html");

    user = UserRegistry.getUser(cocoon.request.get("name"));
  }
  cocoon.sendPageAndWait("shippingAddress.html", {who: user});

  var address = cocoon.request.get("address");
  cocoon.sendPageAndWait("creditCard.html");

  var creditCard = cocoon.request.get("creditCard");
  cocoon.sendPageAndWait("confirmOrder.html");

  EnterpriseSystem.placeOrder(user, cart, address, creditCard);
  cocoon.sendPage("orderPlaced.html");
}
```

Managing continuations

Continuation trees

- Browser "back" : the previous path is lost
- No fear : a continuation is lightweight
  - → Reference to the parent continuation
  - → Local variables since the parent continuation
- Browser "new window"
  - Creates a new branch
  - → Allows "what if ?" navigation in the application

Managing continuations

Expiring continuations

- Manual expiration :
  - **`sendPageAndWait()`** returns its continuation
  - **`k.invalidate()`** invalidates the continuation and its subtree :

```
var k = sendPageAndWait("start.html");
...
BusinessService.commit();
// Cannot go back again
k.invalidate();
```

  - → Again, avoids complicated state management
- Automatic expiration
  - An inactive continuation expires after a delay

Conclusion

Flow script

- Gives control back to the server
  → We always know "where" the browser is
- Allows sophisticated flow screens
  → No need for state automata
- Increases security and robustness
  → Forbids direct access to form submission URLs
  → Handles "back" and "new window"

Questions ?
Answers !

(next: Woody)

Woody intro

The need for form handling

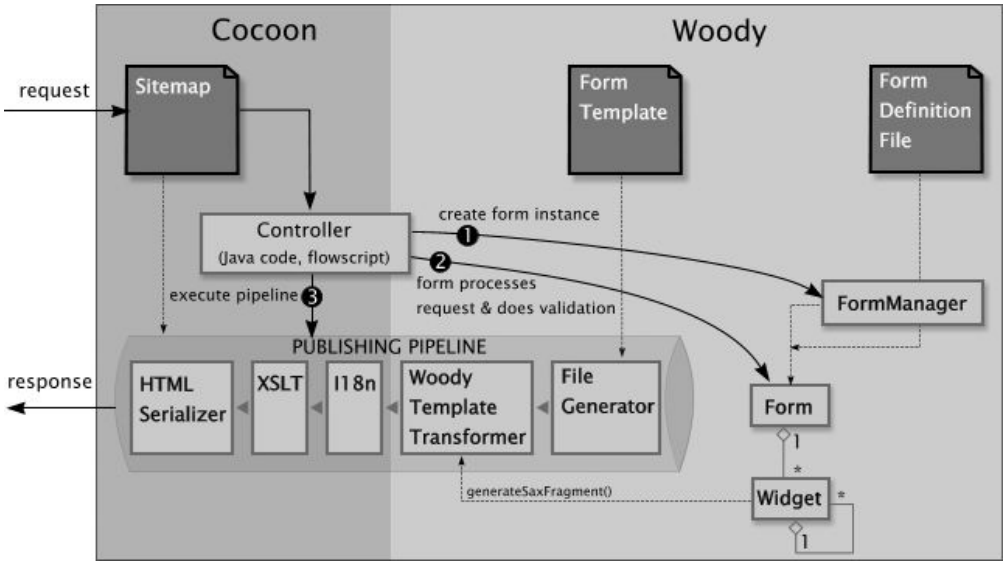
- Cocoon started as a publication framework
  - → Many pages, limited user feedback
  - → Content was mostly written "outside"
- Evolution towards a general-purpose web framework
  - → Published content has to be managed
  - → Used for more and more data-centric applications

→ Need for good form handling features

- Various attempts before Woody:
  FormValidatorAction, Precept, XMLForm, JXForms

Woody principles

The big picture



Woody principles

The form object model

- Composed of "widgets"
  - Represents "something" that appears in the form
  - Can read, parse and validate itself
  - Can output its XML representation
- Some widgets are non-terminal
  → Support for tables and rows

Woody principles

Form definition overview

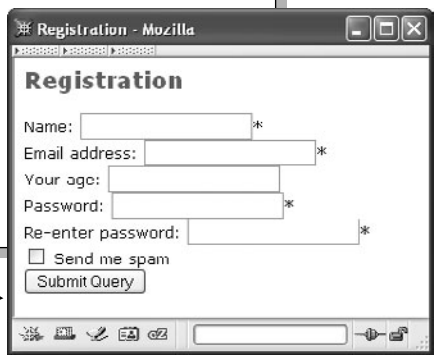
```
<wd:form xmlns:wd="http://apache.org/cocoon/woody/definition/1.0">

  <wd:field id="name" required="true">
    <wd:label>Name:</wd:label>
    <wd:datatype base="string">
      <wd:validation>
        <wd:length min="2"/>
      </wd:validation>
    </wd:datatype>
  </wd:field>

  <wd:field id="email" required="true">
    <wd:label>Email address:</wd:label>
    <wd:datatype base="string">
      <wd:validation>
        <wd:email/>
      </wd:validation>
    </wd:datatype>
  </wd:field>

  …/…
</wd:form>
```

Woody principles

Form template overview

- Embeds widget references in target markup

```
<html xmlns:wt="http://apache.org/cocoon/woody/template/1.0">
  <head>
    <title>Registration</title>
  </head>
  <body>
    <h1>Registration</h1>
    <wt:form-template action="registration" method="POST">
      <wt:widget-label id="name"/>
      <wt:widget id="name"/>
      <br/>
      <wt:widget-label id="email"/>
      <wt:widget id="email"/>
      <br/>
      …/…
      <input type="submit"/>
    </wt:form-template>
  </body>
</html>
```



The form definition file

Widgets

- Available widgets
  - <wd:form> : the main form widget
  - <wd:field> : "atomic" input field
  - <wd:booleanfield> : boolean input
  - <wd:mutivaluefield> : multiple selection in a list
  - <wd:repeater> : collection of widgets
  - <wd:output> : non-modifiable value
  - <wd:action> : action button (intra-form)
  - <wd:submit> : submit button (exits the form)
- They're all defined in cocoon.xconf
  → Add your own if needed

The form definition file

The <wd:field> widget

- Definition overview

```
<wd:field id="..." required="true|false">
  <wd:label>...</wd:label>
  <wd:datatype base="...">
     [...]
  </wd:datatype>
  <wd:selection-list>
     [...]
  </wd:selection-list>
</wd:field>
```

- The label can contain abitrary markup

```
<wd:label>Your <b>name</b></wd:label>
```

  - Including i18n references

```
<wd:label>
  <i18n:text key="name-field-label"/>
</wd:label>
```


The form definition file

Defining the data type of a field

- Mandatory "base" type
  - Defines the Java type
  - "string", "long", "decimal", "date", "boolean"
    → Pluggable components : add your own !
- Optional conversion and validation

Parsing and formatting

Validation

```
<wd:datatype base="...">
  <wd:convertor>
     [...]
  </wd:convertor>
  <wd:validation>
     [...]
  </wd:validation>
</wd:datatype>
```

The form definition file

Data type parsing and formatting

- Each base type has a set of converters
  → Pluggable components : add your own !
- Example : date's "formatting" converter
  - based on **`java.text.SimpleDateFormat`**
  → locale-dependent patterns

```
<wd:datatype base="date">
  <wd:convertor type="formatting">
    <wd:patterns>
      <wd:pattern>yyyy-MM-dd</wd:pattern>
      <wd:pattern locale="en">MM/dd/yyyy</wd:pattern>
      <wd:pattern locale="fr">dd/MM/yyyy</wd:pattern>
      <wd:pattern locale="nl-BE">dd/MM/yyyy</wd:pattern>
      <wd:pattern locale="de">dd.MM.yyyy</wd:pattern>
    </wd:patterns>
  </wd:convertor>
</wd:datatype>
```


The form definition file

Data type validation

- A validation rule checks value validity
  - length, range, regexp, creditcard, assert, email
  → Pluggable components : add your own !
- A datatype can have several validation rules

```
<wd:field id="email">
  <wd:datatype base="string">
    <wd:validation>
      <wd:email/>
      <wd:length max='100'>
        <wd:failmessage>Your address it too long!</wd:failmessage>
      </wd:length>
    </wd:validation>
  </wd:datatype>
</wd:field>
```

The form definition file

Selection lists

- Provide enumerations to the user
  - List of items having a value, with optional label

```
<wd:field name="OS">
  <wd:datatype base="string"/>
  <wd:selection-list>
    <wd:item value="Linux"/>
    <wd:item value="Windows"/>
    <wd:item value="Mac OS"/>
    <wd:item value="Solaris"/>
    <wd:item value="other">
      <wd:label><i18n:text key="other"/></wd:label>
    </wd:item>
  </wd:selection-list>
</wd:field>
```

- Selection lists can be external and dynamic

```
<wd:selection-list src="cocoon:/build-list.xml">
```


The form definition file

The <wd:repeater> widget

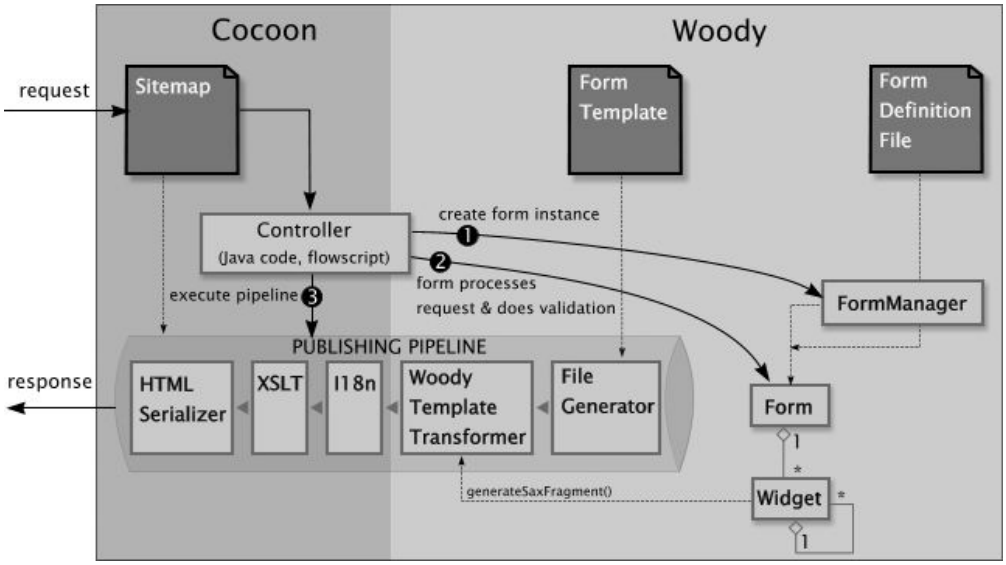
- Repeats a number of child widgets
  → Used to manage collections, tables, etc.

```
<wd:repeater id="contacts">

  <wd:field id="firstname">
    <wd:label>Firstname</wd:label>
    <wd:datatype base="string"/>
  </wd:field>

  <wd:field id="lastname">
    <wd:label>Lastname</wd:label>
    <wd:datatype base="string"/>
  </wd:field>

</wd:repeater>
```

- Specialized <repeater-action> widgets
  → Automatic row addition/deletion

The form template

The big picture (again)



The form template

Role of the WoodyTransformer

```
<html xmlns:wt="http://apache.org/cocoon/woody/template/1.0">
  <head>
    <title>Registration form</title>
  </head>
  <body>
    <h1>Registration</h1>
    <wt:form-template action="registration"
      <wt:widget-label id="name"/>
      <wt:widget id="name"/>
      <br/>
      <wt:widget-label id="email"/>
      <wt:widget id="email"/>
      <br/>
      …/…
      <input type="submit"/>
    </wt:form-template>
  </body>
</html>
```

```
<html xmlns:wt="http://apache.org/cocoon/woody/instance/1.0">
  <head>
    <title>Registration form</title>
  </head>
  <body>
    <h1>Registration</h1>
    <wi:form-template action="registration" method="POST">
      Name:
      <wi:field id="name">
        <wi:label>Name:</wi:label>
        <wi:value>Cocoon</wi:value>
      </wi:field>
      <br/>
      Email address:
      <wi:widget id="email">
        <wi:label>Email address:</wi:label>
        <wi:value>foo</wi:value>
        <wi:validation-message>
          Invalid email address
        </wi:validation-message>
      </wi:widget>
      <br/>
      …/…
      <input type="submit"/>
    </wi:form-template>
  </body>
</html>
```

Expanded widgets

Validation failed

The form template

Role of the WoodyTransformer

- Expand all "wt" elements in their "wi" counterpart
  - → "wt" = Woody template
  - → "wi" = Woody instance
- Output of the transformer goes to styling
  - Provided : HTML styling
  - Other stylings are possible (e.g. WML)
  - → Woody does not hardcode the presentation !

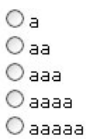
The form template

The <wt:widget> element

- Produces the corresponding widget instance
  - Markup depends on the actual widget
  - For fields : <wi:label>, <wi:value>, <wi:selection-list>
- <wt:widget> can contain styling information
  - Drives the styling stylesheet
  - → Contents of <wi:styling> depends on the stylesheet !

```
<wt:widget id="fourchars">
 <wi:styling list-type="listbox"
             listbox-size="4"/>
</wt:widget>
```


```
<wt:widget id="fourchars">
 <wi:styling list-type="radio"/>
</wt:widget>
```



The form template

The <wt:repeater-widget> element

- Iterates on the contents of a <wd:repeater>

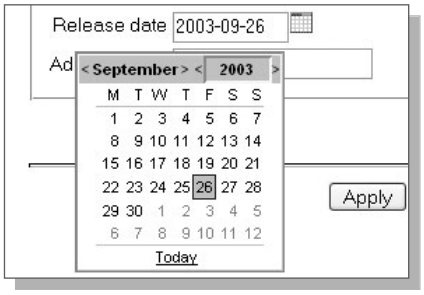
```
<table>
  <tr>
    <th>
      <wt:repeater-widget-label
          id="contacts" widget-id="firstname"/>
    </th>
    <th>
      <wt:repeater-widget-label
          id="contacts" widget-id="email"/>
    </th>
  </tr>
  <wt:repeater-widget id="contacts">
    <tr>
      <td>
        <wt:widget id="firstname"/>
      </td>
      <td>
        <wt:widget id="email"/>
      </td>
    </tr>
  </wt:repeater-widget>
</table>
```

```
<table>
  <tr>
    <th>Name</th>
    <th>Email address</th>
  </tr>
  <tr>
    <td>
      <wi:field id="contacts.0.firstname">
        <wi:label>Name</wi:label>
        <wi:value>Harry</wi:value>
      </wi:field>
    </td>
    <td>
      <wi:field id="contacts.0.email">
        <wi:label>Email address</wi:label>
        <wi:value>harry@potter.com</wi:value>
      </wi:field>
    </td>
  </tr>
  <tr>
    <td>
      <wi:field id="contacts.1.firstname">
        <wi:label>Name</wi:label>
        <wi:value>Anakin</wi:value>
      </wi:field>
    </td>
    <td>
      <wi:field id="contacts.1.email">
        <wi:label>Email address</wi:label>
        <wi:value>anakin@skywalker.com</wi:value>
      </wi:field>
    </td>
  </tr>
</table>
```


Built in HTML styling

Field styling

- Basic styling: html input
- <wi:styling type="...">
  - "password", "hidden", "textarea", "date"
  - "listbox", "radio" for selection-lists

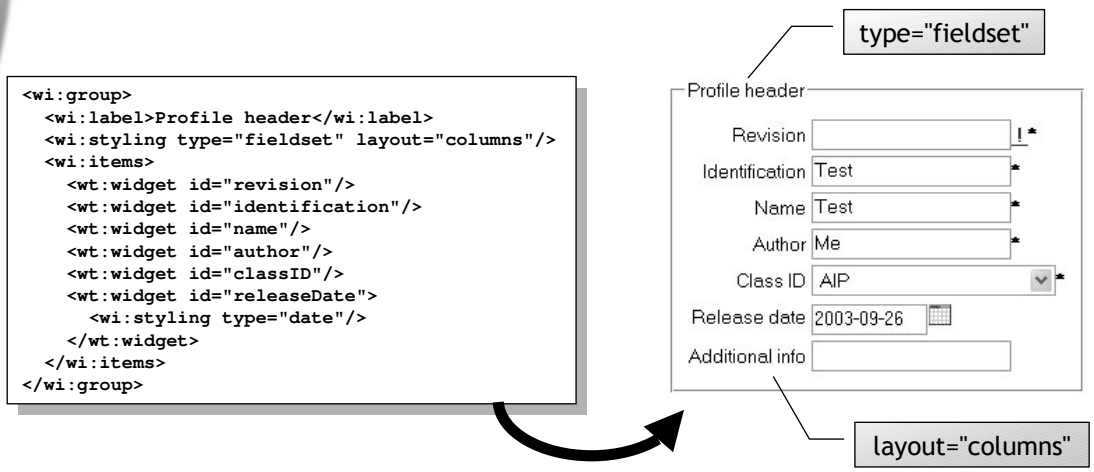


Built in HTML styling

<wi:group> styling

- Instance-only widget providing high-level styling
  → No corresponding <wd:> nor <wt:>

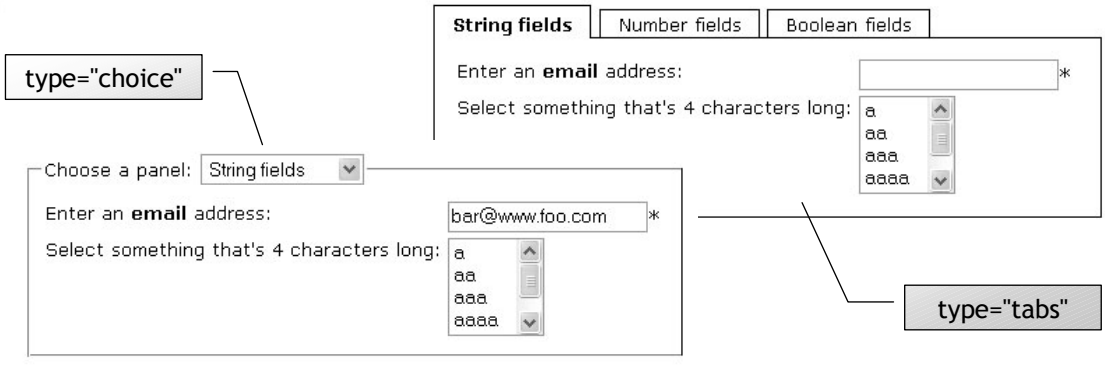
```
<wi:group>
  <wi:label>Profile header</wi:label>
  <wi:styling type="fieldset" layout="columns"/>
  <wi:items>
    <wt:widget id="revision"/>
    <wt:widget id="identification"/>
    <wt:widget id="name"/>
    <wt:widget id="author"/>
    <wt:widget id="classID"/>
    <wt:widget id="releaseDate">
      <wi:styling type="date"/>
    </wt:widget>
  </wi:items>
</wi:group>
```



Built in HTML styling

<wi:group> styling

- Container rendering
  → "type" attribute : "fieldset", "tabs", "choice"
  → Tabs defined with CSS

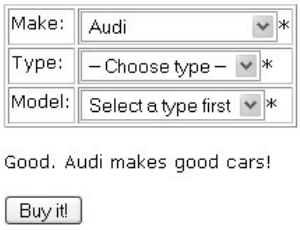


Interactive forms

Server-side event handler, client-side trigger

```
<wd:field id="make" required="true">
  <wd:label>Make:</wd:label>
  <wd:datatype base="string"/>
  <wd:selection-list src="cocoon:/cars" dynamic="true"/>
  <wd:on-value-changed>
    <javascript>
      var value = event.newValue;
      var type =
        event.source.parent.getWidget("type");
      if (value == null) {
        type.setSelectionList(new
          EmptySelectionList("Select a maker first"));
      } else {
        type.setSelectionList("cocoon:/cars/"+value);
      }
      typewidget.setValue(null)
    </javascript>
  </wd:on-value-changed>
</wd:field>
```

```
<wt:widget id="make">
  <wi:styling submit-on-change="true"/>
</wt:widget>
```



Change the type selection list

Linking forms to application data

An additional binding definition file

- Associates widget names to XPath expressions on the data model

  Example : binding to an XML document

Set the context of included paths

Associates a widget to a path

```
<wb:context
  xmlns:wb="http://apache.org/cocoon/woody/binding/1.0"
  xmlns:wd="http://apache.org/cocoon/woody/definition/1.0"
  path="user" >

  <wb:value id="email" path="email" readonly="true"/>

  <wb:value id="number" path="number/@value">
    <wd:convertor datatype="long"/>
  </wb:value>

  <wb:value id="choose" path="choose/@value">
    <wd:convertor datatype="boolean"/>
  </wb:value>
</wb:context>
```

Read-only widget

Binding convertor (XML is text)

Putting it all together

The woody.js library

- Provides a Form class
  - Constructor takes a form definition file
- Method Form.showForm() to display the form
  - Returns when validation ok or non-validating submit
    - → Internal loop on sendPageAndWait()

```
function edit_header() {
  var data = Application.getData();
  var form = new Form("forms/profile-header-def.xml");

  form.createBinding("forms/profile-header-binding.xml");
  form.load(data);

  form.showForm("view-profile-header.html", {foo: bar});

  if (form.submitId == "ok") {
    form.save(data);
    sendDialog("Thanks a lot");
  } else {
    sendDialog("Bye bye");
  }
}
```

Load app. data

Save app. data

Show form and wait

Test submit button

Putting it all together

The sitemap

```
<map:match pattern="edit-*.html">
  <map:select type="method">
    <!-- GET : start the flow for this screen -->
    <map:when test="GET">
      <map:call function="editor_{1}"/>
    </map:when>
    <!-- POST (form submission) : continue the flow -->
    <map:when test="POST">
      <map:call continuation="{request-param:continuation-id}"/>
    </map:when>
  </map:select>
</map:match>

<map:match pattern="view-*.html">
  <map:generate type="jxtemplate" src="forms/{1}-tmpl.xml"/>
  <map:transform type="woody"/>
  <map:transform type="i18n"/>
  <map:transform type="xslt" src="resources/editor-styling.xsl"/>
  <map:serialize type="html"/>
</map:match>
```

Selection by http method: form's action is "" (same URL)

"editor_" prefix restricts access to flowscript functions

JXTemplate to use showForm's context data

Conclusion

- A powerful form framework
  - Rich datatypes and validation rules
  - Easy extension to specific needs
  - Event handling for sophisticated interaction
  - Fancy builtin stylesheets
  - Easy to use with flowscript
- A community development
  - Initiated by Outerthought
  - Welcomes additions and contributions

→ Woody will be *the* form framework for Cocoon

See also http://wiki.cocoondev.org/Wiki.jsp?page=Woody

Questions ?
Answers !

Pipelined Fugues

David Casal

CMS with Cocoon: Lenya

Michael Wechner

Lenya - a Cocoon based CMF/S

Michael Wechner

michi@apache.org

Short Bio of michi

- Math. Physics at Swiss Federal Institute of Technology (Zürich)
- Co-Founder Wyona Ltd.
- Co-Founder OSCOM
- Original Creator of Lenya: A Cocoon based CMS (subproject under incubation)

Sample Publications

- Lenya's Weblog
- Default
- Project Site and Documentation
- OSCOM CMF/S Matrix
- Unipublic

Areas

- Admin
- Site
- Authoring
- (Staging)
- "Live"

Architecture

- Lenya Core
- Lenya Publications

Core Features

- Workflow
- Access Controlling
- Versioning
- Editor Interfaces
- Content Creator/Deleter API
- Publisher API
- Scheduler
- Search Engine

Publications

- Information Architecture
- Functionality

Ant Tasks

- Content Creator/Deleter
- Workflow
- Publisher
- ...

Flow

- User Management
- Group Management
- IP/Subnet Management

Roadmap

- Consolidation of Core Features
- Consolidation of Sample Publications
- WebDAV (AtomAPI), JSR-170
- Multimedia/Data Repository (Lookups)
- Event Calendar Publication

Community

- Installation
- Sample Publications
- Releases
- Documentation
- Leverage

Conclusion

- A lot of work
- Many fights
- ... but nothing to regret :-)
- thanks

Lightweight tools for successful projects: the Open Source Way

Bertrand Delacretaz

LIGHTWEIGHT TOOLS FOR SUCCESSFUL PROJECTS

The Open Source Way

author: Bertrand Delacrétaz, www.codeconsult.ch

event: Cocoon GetTogether 2003, www.orixo.com/events/gt2003/
date: October 7th, 2003

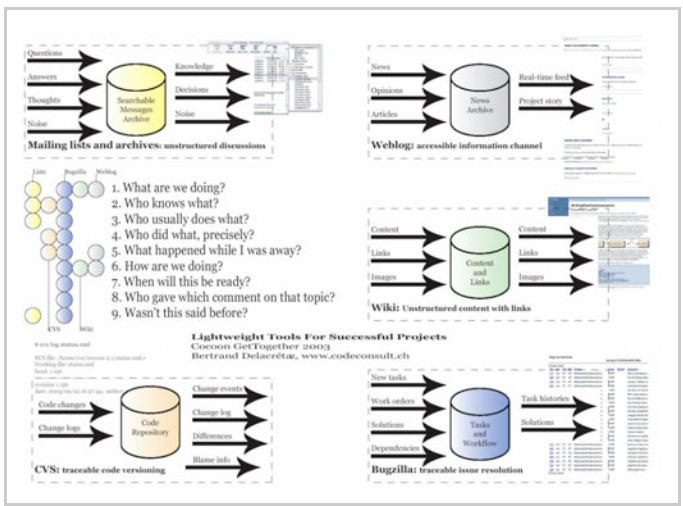
How can we apply the tools used by the Apache Cocoon project team to other projects?
How can we improve the usage of these tools in the Cocoon project?

1. INTRO: WHAT ARE WE TALKING ABOUT?

Project (not coding or build) tools used by the Cocoon team.

What do they bring?

Can we do better?



2. SUCCESSFUL PROJECTS

Communication is the key?

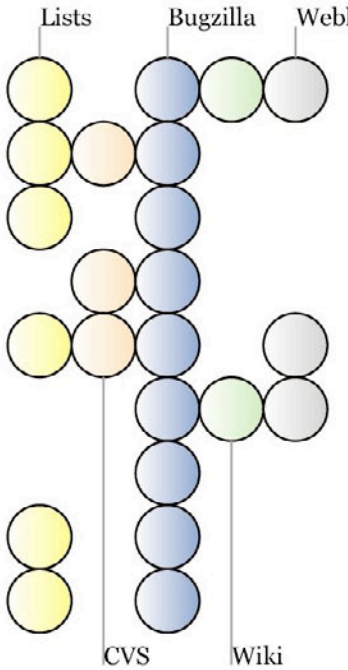
Need Stories!

Email? Post-it notes? Documentation?

Coming into an existing team or project?

Need self-service information tools!

3. PROJECT QUESTIONS



1. What are we doing?
2. Who knows what?
3. Who usually does what?
4. Who did what, precisely?
5. What happened while I was away?
6. How are we doing?
7. When will this be ready?
8. Who gave which comment on that topic?
9. Wasn't this said before?

4. PART ONE: TOOLS

A quick refresher about CVS, Bugzilla, Wiki and Weblogs.

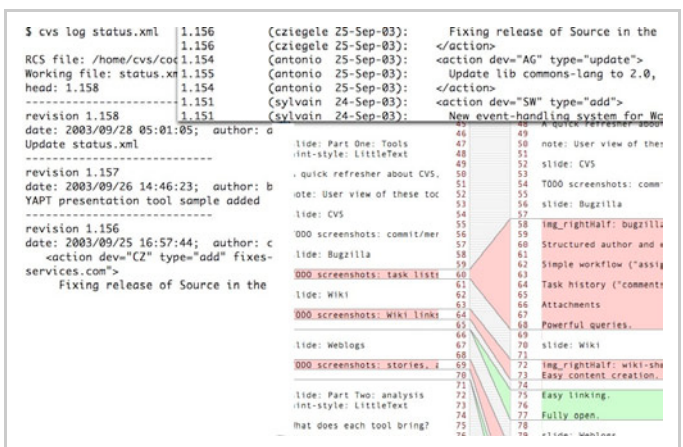
User view of these tools.

5. CVS

Commit/merge (+log messages).

Differences between versions.

Code traceability ("annotate", "blame").



6. BUGZILLA

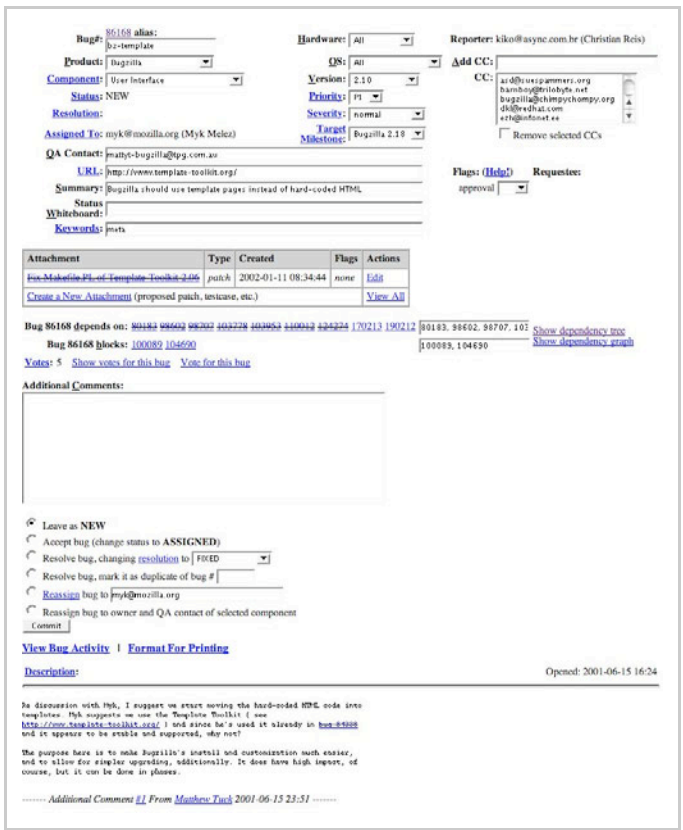
Structured author and component info.

Simple workflow: "assigned to".

Task history: "comments".

Attachments.

Powerful queries.

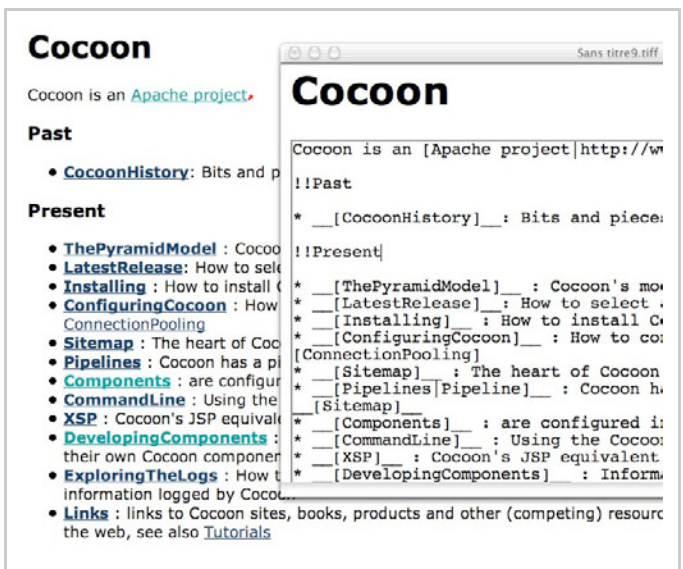


7. WIKI

Easy content creation.

Easy linking.

Fully open.



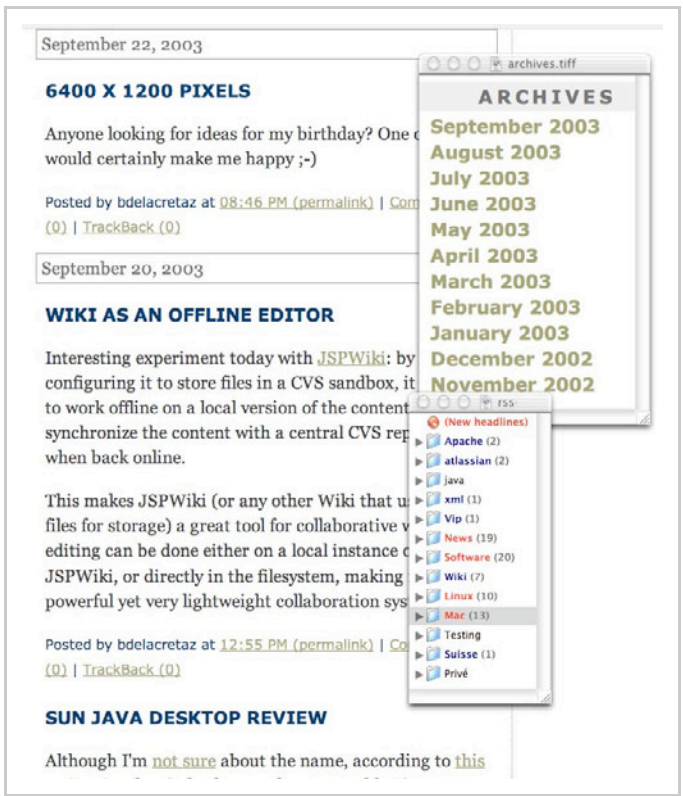
8. WEBLOGS

Stories in chronological order.

Time-based archive.

Full-text search.

RSS feed, multiple subscriptions.



9. PART TWO: ANALYSIS

What does each tool bring?

10. MAILING LISTS + ARCHIVES

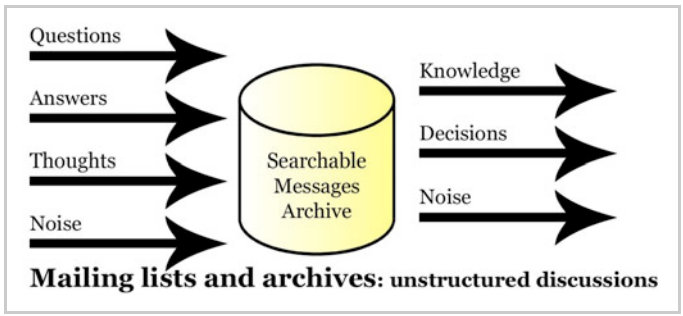
Threaded broadcasts.

Archives have little structure.

Need discipline!

Should that be "more or less" searchable?

Self-service help for 1, 2, 3, 5, 8, 9.

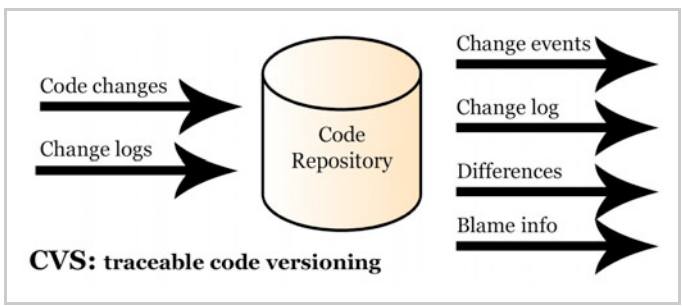


11. CVS (DESERT ISLAND TOOL)

Shared files, strict control, descriptions of changes.

Tells the story of the code.

Self-service help for 4, 5, maybe 2.

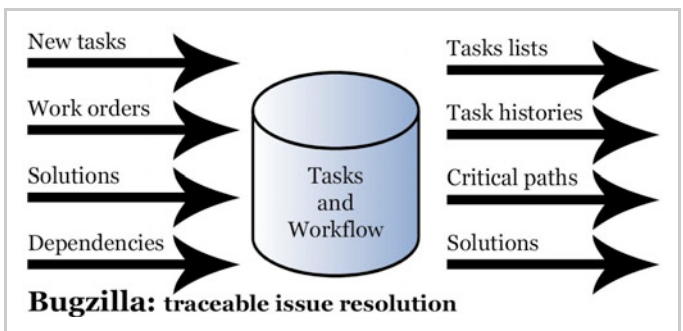


12. BUGZILLA (DESERT ISLAND TOOL)

Dynamic list of tasks, queries, workflow.

Traceable actions (with effort).

Could tell the story of the project in detail!



13. BUZGILLA DEPENDENCIES

Using dependencies between tasks to help answer 6 and 7.

Tasks depend on others.

Dynamic status reports.

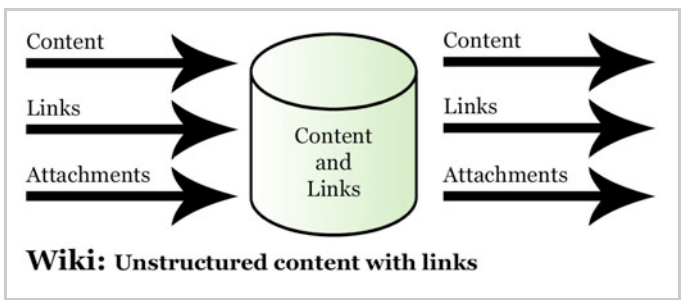
14. WIKI

Unstructured shared documentation, open access and editing.

Traceable edits.

Empowers users!

Self-service help depends on content.



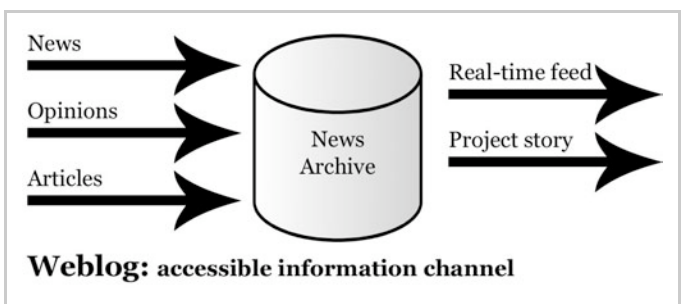
15. PROJECT WEBLOG

Condensed news and public information.

Tells the story!

Notification via RSS feeds.

Self-service help for 1, 5, 6.



16. Desert Island Tools

CVS
Cannot drive nails without a hammer.

Bugzilla
Can replace lists and archives in smaller teams.

Mailing lists
Require lots of discipline.

17. Part Three: Discussion

How are we doing?

Possible improvements?

18. How is Cocoon doing? (the good)

1. What are we doing?

2. Who knows what?

3. Who usually does what?

4. Who did what, precisely?

5. What happened while I was away?

8. Who gave which comment on that topic?

9. Wasn't this said before?

19. How is Cocoon doing? (the..hum...)

Hard questions:

6. How are we doing?

7. When will this be ready?

Hard to answer: no committment on hours worked!

...more bugzilla?


List: xml-cocoon-dev
Subject: [ANN] Apache Cocoon 2.1 Released
From: "Carsten Ziegeler" <cziegeler () s-und-n ! de>
Date: 2003-08-13 9:24:34
[Download message RAW]

Apache Cocoon 2.1 Released

The Apache Cocoon Community is proud to announce the new release of Apache Cocoon.

20. Improvements?

Monday morning staff meeting? (just kidding...)

Shared tests scenarios and reports?

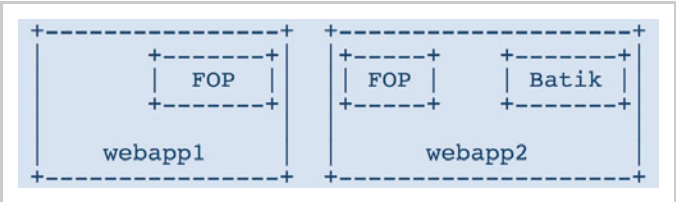
Project weblog?

Bugzilla dependencies?

Instant messaging?

21. THE [RT] EFFECT

Random Thoughts - "what if?" list messages.



Wild ideas floating around.

ASCII art rulez ;-)

Getting to know each other.

Improve community, avoid one-man shows.

Keep it up!

22. WRITTEN COMMUNICATION

5% goes through - be nice ;-)

Assume you're wrong.

Assume you didn't understand.

Ask - clarify.

Be patient..

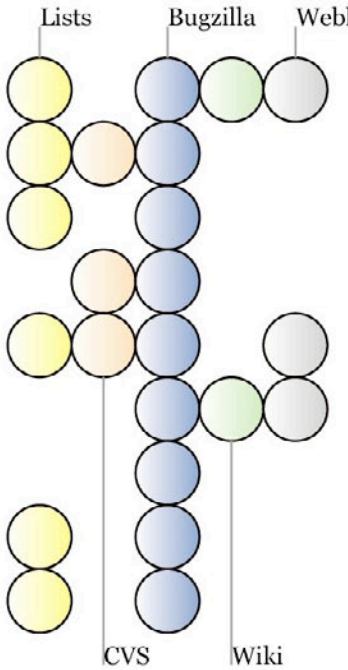
23. CONCLUSIONS

Great tools! Tell your friends about them..

Cocoon Team: Currently hard to answer the "when" questions, and no traceability of tests.

Need more bugzilla?

24. Remember - The Questions



1. What are we doing?
2. Who knows what?
3. Who usually does what?
4. Who did what, precisely?
5. What happened while I was away?
6. How are we doing?
7. When will this be ready?
8. Who gave which comment on that topic?
9. Wasn't this said before?

25. Coda

Thanks!

To Marc Portier for fruitful collaboration on content.

To Antonio Gallardo, Carsten Ziegeler, Geoff Howard and Stefano Mazzocchi for reviewing this.

To the GT team for having us here!

Ants picture: www.imageafter.com

26. TOOLS REFERENCE

www.cvshome.org

www.bugzilla.org

www.jspwiki.org

www.movabletype.org

And many other similar tools!

Cocoon & WebDAV

Gianugo Rabellino
Matthew Langham

Cocoon & WebDAV

Gianugo Rabellino, Matthew Langham

Cocoon GetTogether 2003

Agenda

- Introduction to WebDAV
- WebDAV in Cocoon
- Application Scenarios
- Q/A

WebDAV – Memory Lane

- Tim-Berners Lee's original vision of the Web was that of a collaborative readable and writable medium
- In 1990 a prototype Web editor/browser was introduced on the Next platform
  - "WorldWideWeb" (later "Nexus")
  - Could edit documents in the "file:" space
- But with the advent of NCSA Mosaic – "Publish/Browse" became the dominant model for the Web
- 1995/1996 Netscape Navigator Gold
  - Allowed editing and publishing pages to the Web
- 1996/1997
  - Microsoft Word 97, Lotus WordPro 97 etc. offer HTML editing and remote publishing

(pro-netics)

WebDAV – History

- An ad-hoc collection of people met at the WWW4 conference in 1995 and then at America Online in 1996
  - Extensions to the HTTP Protocol were needed
  - Now known as the WEBDAV working group
- March 20, 1997 – Internet Engineering Steering Group and IETF approve charter of the WebDAV working group
- RFC 2518 – HTTP Extensions for Distributed Authoring – WebDAV
  - Note: No "Versioning" in the rfc
- Separate working groups for DASL, DELTA-V, Access Control

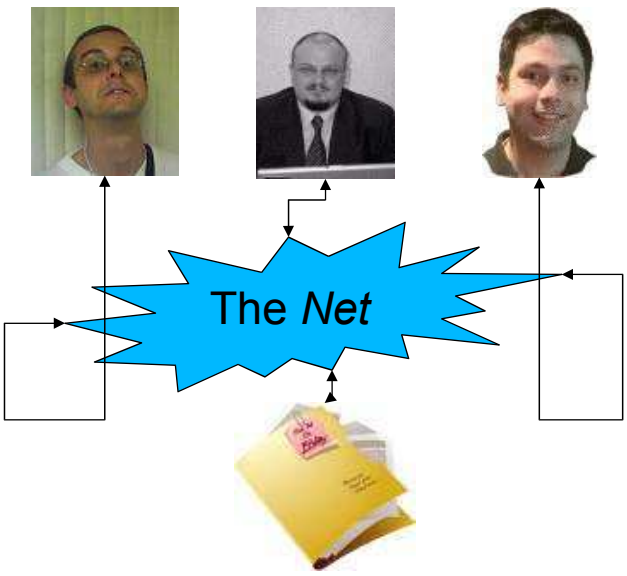
OrixO the xml business alliance

WebDAV – Mission

- "The World is a Folder"
- Allow collaborative authoring of all document types on the Web
- Metadata repository infrastructure
- A Web-based network file system
- A replacement protocol that can handle email, calendaring, directory lookup and more
  - e.g. Apple's iCal supports WebDAV publishing

(pro-netics)
S&N netBank solutions

WebDAV – Mission



Working together on the same document, wherever you are and whatever you use

OrixO the xml business alliance
the xml business alliance

Technical Benefits

- It's Simple !
- It's Extensible !
  - e.g. Using and extending document properties
- Ubiquitous HTTP infrastructure can be used
  - Authentication
  - Encryption
  - Firewall / Proxy navigation
- Allows pluggable data storage (stores)
  - RDBMS
  - XML Database
  - File-System
- Deployment in Internet or Intranet
- Tools available
- Large Know-How pool

(pro-netics)
S&N netBank solutions

Business Benefits

- Technical Infrastructure exists already
- Adding WebDAV support to products is economically possible
  - Components and Toolkits are available
- Build distributed infrastructures quickly and cost-efficiently
- Use as a base for CMS, Project collaboration, Document management etc.
- WebDAV Server provided as add-ons to existing RDBMS or XML Databases
- Investment protection
  - Easily change WebDAV server or data storage (in theory)
- Large number of servers and tools available

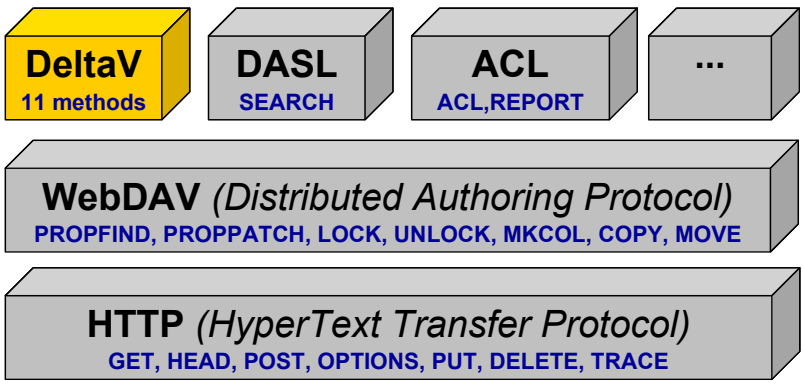
OrixO the xml business alliance
the xml business alliance

Functionality

- WebDAV Basic Functions
  - Locking
  - Metadata Management
  - Namespace Operations
- WebDAV DeltaV
  - Auto-Versioning
  - Checkout/Checkin
  - Version History
- WebDAV ACL
  - Access Control Management
- WebDAV DASL
  - Server-side searching
- A fast progressing standard

(pro-netics)

Functionality - Overview



OrixO the xml business alliance

WebDAV Servers

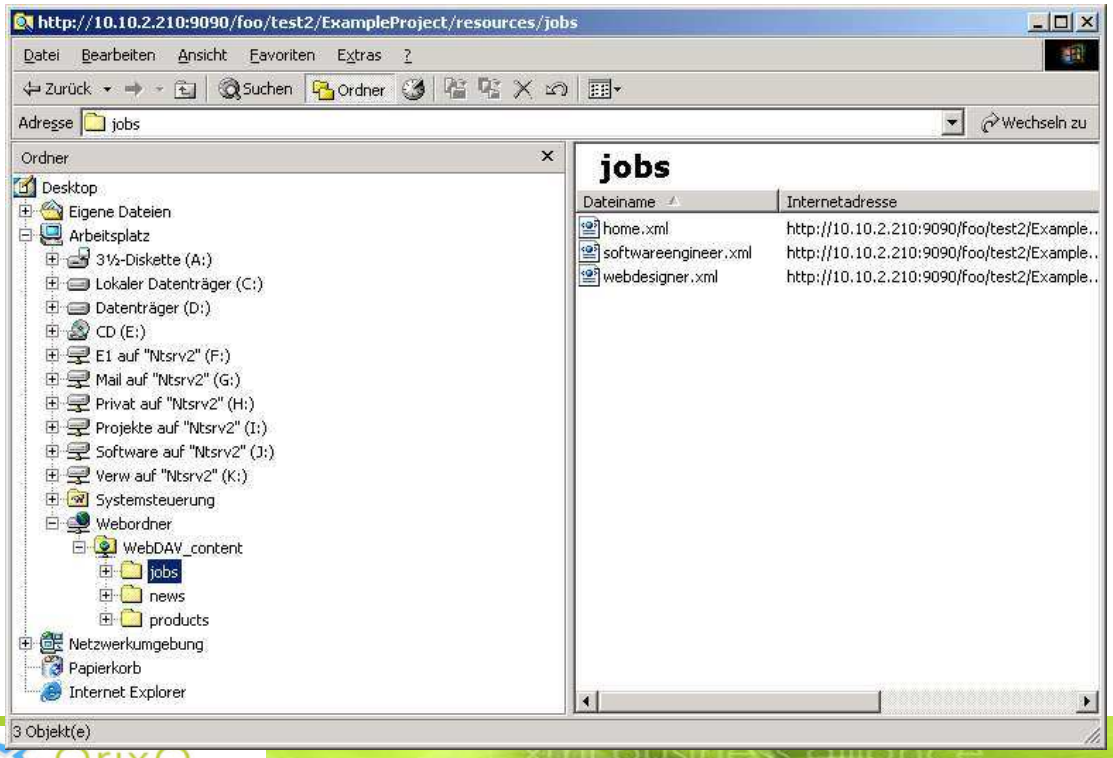
- Apache 2.0
- Catacomb
- Subversion
- Slide
- Tamino WebDAV Server
- Oracle Internet File System
- Microsoft
  - Internet Information Server
  - Exchange Server
  - Sharepoint Portal Server
- Xythos WebFile Server

(pro-netics)

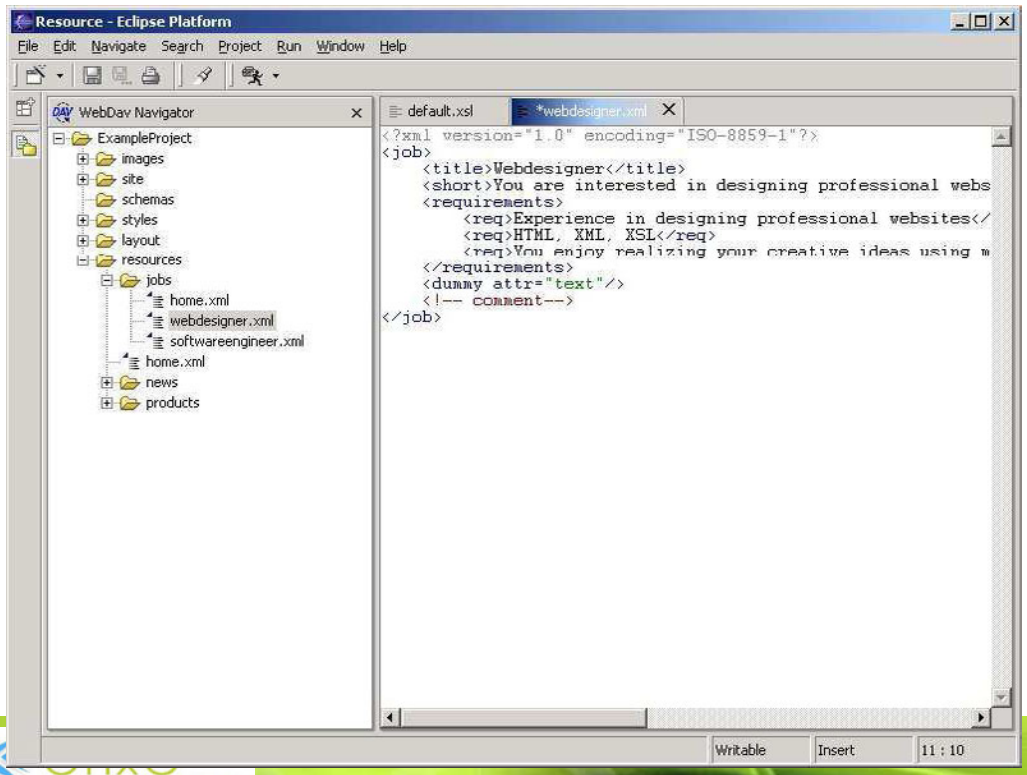
WebDAV Clients

- XML Spy
- XMetal
- Microsoft
  - Office
  - Windows Explorer
- Adobe
  - GoLive
  - Photoshop
- Macromedia Dreamweaver
- WebDrive
- WebDAVfs
- sunDance

OrixO the xml business alliance



(pro-netics)



OrixO the xml business alliance

Cocoon and WebDAV

(yet another) Dynamic Duo?

- (Networked) filesystem + metadata metaphore
- (Extensible) metadata expressed and exposed as XML
- (Revamped) HTTP + XML based protocol

(pro-netics)

WebDAV strong points

S&η netBank solutions

- HTTP based transport, plain HTTP can be intermixed (GET still works)
- Rich semantics for metadata
- Easy to use for easy tasks
- Ubiquitous and cross platform

OrixO the xml business alliance

the xml business alliance

(pro-netics)

WebDAV weak points

S&η netBank solutions

- HTTP is not used a pure transport
  - WebDAV directives are both in HTTP headers and payload (e.g. COPY method)
- No real support for real XML metadata (implementation issue)
- Difficult to use for difficult tasks
- Specs somehow unclear, too many extensions in draft phase (DeltaV, DASL, DAV ACL…)

OrixO the xml business alliance

the xml business alliance

WebDAV in Cocoon

- Support is included in the *webdav* block
- Can be used as
  - client (stable)
  - server (needs work)
  - proxy (unstable but promising)

PROPFIND example response

Dead property

```
<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response xmlns:ns3=http://cocoon.apache.org/cms/state/1.0>
    <D:href>/catacomb/test.xml</D:href>
    <D:propstat>
      <D:prop>
        <ns3:status>publish</ns3:status>
        <D:creationdate>2003-07-15T09:05:42Z</D:creationdate>
        <D:resourcetype/>
        <D:getcontentlength>26067</D:getcontentlength>
        <D:getcontenttype>text/xml</D:getcontenttype>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

WebDAV client components

- `WebDAVSource`: Avalon Source implementing all the Source subinterfaces (Traversable, Modifiable, Inspectable…)
- `SourcePropsWritingTransformer`: enables write access to resource properties
- `DASLTransformer`: performs DASL queries (think SQLTransformer)

WebDAV server components

- None needed, sitemap is enough! (well, almost…)
- The webdav block contains a dir2propfind.xsl to ease property handling
- Needs rework of Cocoon core to be effective (more on this later…)

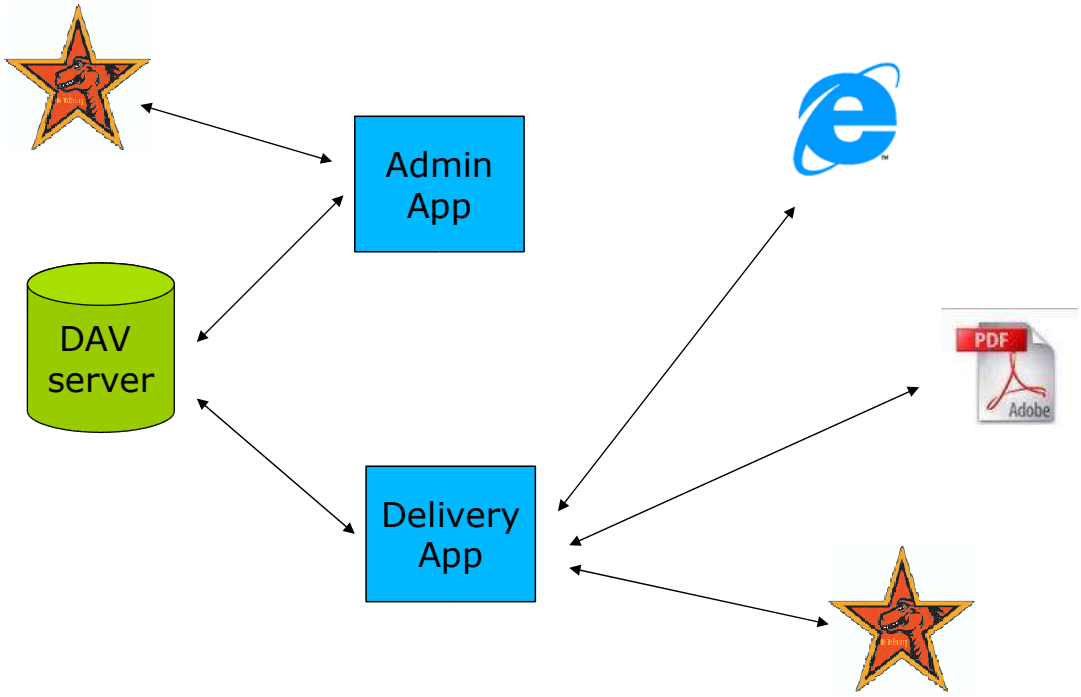
WebDAV proxy components

- Could be used to implement missing features on backend WebDAV servers (e.g. DASL)
- The proxy block contains a generic forwarding proxy, the rest is done via the sitemap
- Proxying is done at the Servlet API level (cloning the Request/Response objects)
- Very promising, but again needs core rework

Why a WebDAV proxy?

- Add virtual resources (e.g. a PDF view or resized images)
- Perform tasks upon WebDAV events (send an email when a file is changed)
- Provide easy, pluggable and effective authentication
- Mangle properties using simple Explorer-like file managers

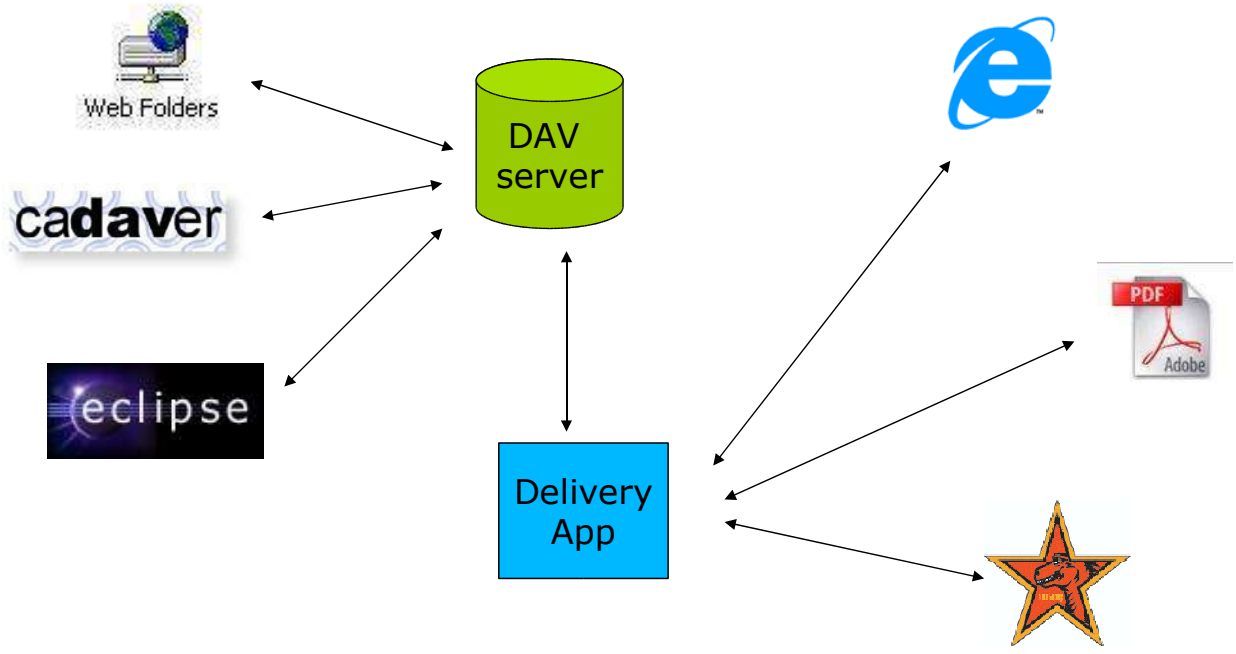
Cocoon/WebDAV architectures



(pro-netics)

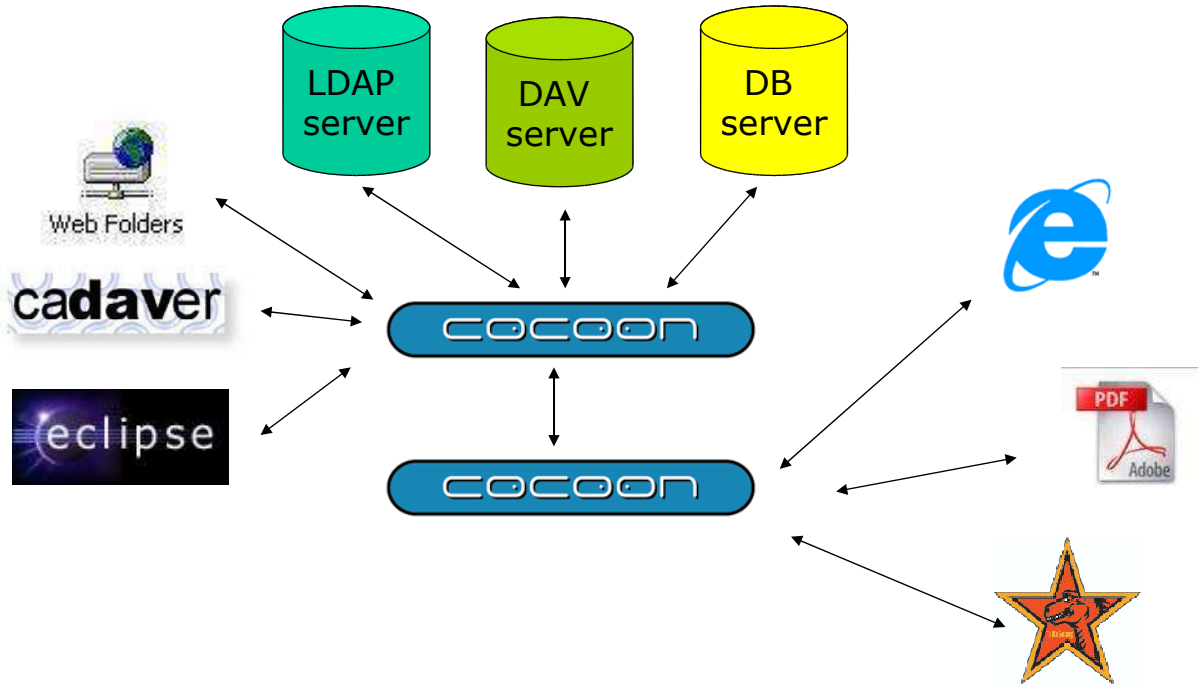
Cocoon/WebDAV architectures

S&N netBank solutions



OrixO the xml business alliance
the xml business alliance

Cocoon/WebDAV architectures



WebDAV/Cocoon dream list

- Cocoon needs much easier access to the request body (e.g. matchers, selectors, flow…)
- Is a different Environment enough?
- WebDAV proxying needs transaction support (e.g. for DASL)

Shameless plug

- Pro-netics is working on an enhanced Catacomb version supporting ODBC as the backend
- The final result (of course) will be Open Source

(pro-netics)

Links

- IETF WebDAV Working Group
  - http://ftp.ics.uci.edu/pub/ietf/webdav/
- RFC 2518 – HTTP Extensions for Distributed Authoring – WebDAV
  - http://ftp.ics.uci.edu/pub/ietf/webdav/protocol/rfc2518.txt

OrixO the xml business alliance

Apache Cocoon GetTogether

Evaluation Sheet

Your Name (opt.):.....

Speakers

	Bad				Good
Introduction Stefano Mazzocchi	1	2	3	4	5
Integrating database with Cocoon Christian Haul	1	2	3	4	5
Flow and woody Sylvain wallez	1	2	3	4	5
Pipelined Fugues David Casal	1	2	3	4	5
CMS with Cocoon: Lenya Michael Wechner	1	2	3	4	5
Lightweight tools for successful projects: the Open Source Way Bertrand Delacretaz	1	2	3	4	5
Cocoon & WebDAV Gianugo Rabellino & Matthew Langham	1	2	3	4	5

Organisation

	Bad				Good
Website	1	2	3	4	5
Marketing & communication	1	2	3	4	5
Registration & follow-up	1	2	3	4	5
Hotel & travel info	1	2	3	4	5
Evening events	1	2	3	4	5
Venue Getting there	1	2	3	4	5
Venue Facilities	1	2	3	4	5
Registration desk	1	2	3	4	5
Exhibition	1	2	3	4	5
Catering	1	2	3	4	5

Next time

Would you attend next year?	Yes	No
Would you recommend this to others?	Yes	No
What country would you prefer?	.....	
Any remarks? (please do!)		

Thanks!