Tools and Services

Search Service, RWiki, Portal Customization and Developing Tools

Dr Ian Boston
CTO,
CARET, University of Cambridge

Good morning, my name is Ian Boston, I am CTO for Caret, University of Cambridge. Over the next 40 minutes I hope to give you a technical introduction to Sakai, ending with a quick look a the structure of some tools.

Technical Presentation

- For a Developer: What is Sakai ?
    - Why Sakai.
    - The Stack
    - The Portal
    - Examples of a Service
    - Examples of a Tool
- For a User: "Introduction to Sakai", Copland Lecture Theatre, now.

This is a technical presentation, if you are not a technical type, then you are very welcome to stay, and I hope I don't bore or confuse you too much.

Peter Knoop is doing a more user oriented presentation in parallel to this one.

So I am going to start with why take and interest in Sakai, look at its conceptual architecture or stack, have a look a the portal, and then look at some examples of bits of the stack.

Why Sakai ?

- Why Open Source ?
- Why Sakai ?

"Open source offers the opportunity to benefit from the work of others for marginal cost. Each community has an equilibrium state between 'free-loaders' and 'workers' driven largely by its modularity and the value of each module. High value open source requires that that equilibrium is maintained. Sakai is a modular environment that supports the participation of a large number of developers"

Why Sakai, Why Open source ?

There are many reasons to use an Open Source product.
Open source communities consist of 2 groups, freeloaders and workers.

Freeloaders, use the open source output with no problems, find it fits their use cases exactly and never need to contribute to the community......... which is fantastic.

It shows the community has got a lot right.
On the other hand workers, of all forms, contribute something back to the community.

Equilibrium....
For the community to prosper it needs an equilibrium of freeloaders and workers, driven and balanced by the modularity and value of the software.

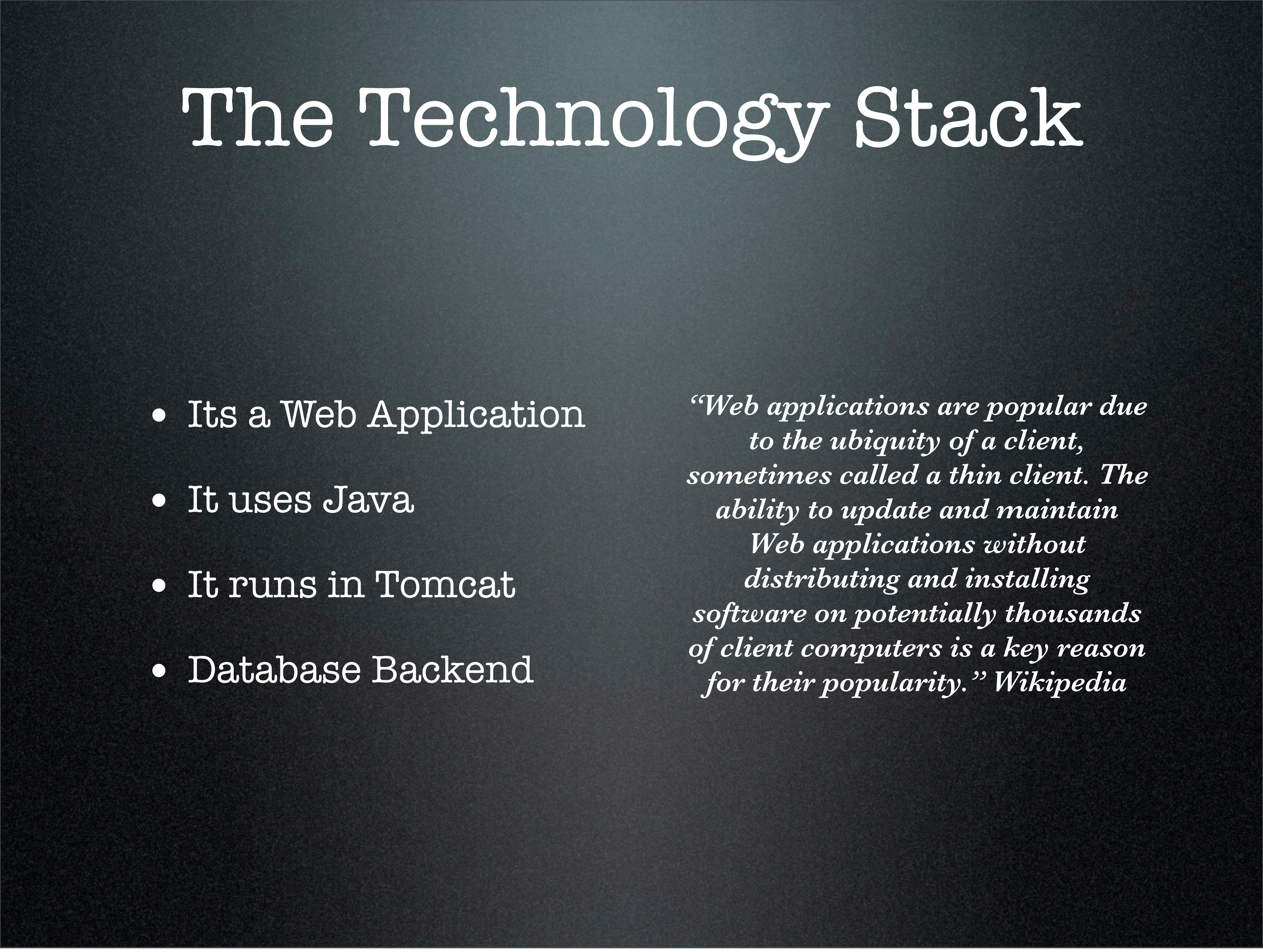
Modular Architecture.....
Sakai has a modular architecture that continues to support contributions, of all sorts, from many institutions world wide. Its equilibrium if firmly balanced towards many parallel contributions, of all types from many institutions.

The Stack

Most applications has some sort of architectural stack.

Some will express this as a massively complex construction of all sorts of entity boxes, interconnected with every standard under the sun.

Try to keep things simple...
With Sakai, we think of the stack as a simple stack of concepts built on-top of other standards. We do get horribly complex at times, but in general there are still big boxes into which we can place the complexity. You might call it separation of concerns.



The Technology Stack

- Its a Web Application
- It uses Java
- It runs in Tomcat
- Database Backend

"Web applications are popular due to the ubiquity of a client, sometimes called a thin client. The ability to update and maintain Web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity." Wikipedia

Sakai is a web application.
Pure and simple, that uses Java. Its not .NET Perl, PHP, Python.... although it can interoperate.

Orriginal reasons for choosing java
Java was originally chose since it gave those in the early project comfort that it would scale to the size they needed.
Even now, mentioning J2EE containers, even the simpler webapp containers gives CIO's a feeling of security. To some extent this is true, although a solid underlaying architecture in any language can be made to scale.

For Sakai the choice of Java made the choice of the J2EE web application model an obvious one. Sakai is targeted at Apache tomcat, although the bindings are relatively weak. Since we QA on Tomcat, and its free over 90% of institutions run on tomcat. For those that want a commercial option, IBM are supporting IBM WebSphere.

Scalability is achieved by deploying multiple tomcat instances, each running a full copy of Sakai. Sakai was written to operate in this mode and most institutions deploy in this mode.

Behind the tomcat instances there is a single database providing a transactional store of information. This database can be MySQL or Oracle in production and we also support HSQLDB for demo and developer modes. There are ports of Sakai to SQLServer, and there is some interest in PostgresQL.

The full stack

- Portal
- Tool
- Service API
- Component

Opening that stack up a bit, we find that a request passes through a portal living in a webapp, is responded by a tool living in its webapp. That tool uses services that are implemented as components. The components live inside a component manager. All communicate via shared service API's.

Core services...

The Sakai Framework provides a core set of services.... so you don't have to. It also provides the scaffolding around which this stack is constructed.

The framework gives the tool writer access to services like UserDirectory, Session, Portal, Tool Configuration, ToolPlacement, Notification, Email, Events .... and so the list goes on.

Portal/Aggregator

- Aggregates and Decorates content
- Dispatches to tools
  - Portlets JSR-168
  - Sakai Tools normal webapps

The portal's job is put some UI scaffolding around the tools.

The portal in Sakai is not a full portal, it doesn't support user level editing of the portlet windows, as you see in uPortal or iGoogle, but it enables tools to be placed within some simple UI scaffolding that supports navigation within the structure of the VLE.

There is nothing to say that the current portal structure is correct and we positively support customization and modification of the portal.

The scaffolding that the portal produces is relevant to the context of the tool or portlets being presented to the user. Once this scaffolding is in place, the portal code dispatches to the tools. Either directly via JSR-168 and a Pluto 1.1. container embedded into the portal, or indirectly via an Iframe.

In performing this dispatch the portal establishes and environment containing the sakai framework, so that tools can use the Service API's to invoke the framework services.

Sakai Tools

- A standard Java Web Application ....
  - with a Sakai Request Filter
    - to connect a Component Manager

"In Apple Macintosh computer programming, Component Manager was one of many approaches to sharing code that originated on the pre-PowerPC Macintosh. It was originally introduced as part of QuickTime, which remained the part of Mac OS that used it most heavily." Wikipedia

Describe the stack webapp, request filter component manager
A Sakai tool, like the portal, is a standard web application.

It has a number of configuration features that enable the portal to communicate with it.

It will have a named tool servlet, know to the portal that accepts named dispatches. Bound to this it will have a Sakai Request filter that injects a reference to the Component Manager. It is the component manager gives the tool access to implementations of the Service API's that it contains, and using a Servlet Request Filter ensures that the component manager and the remainder of the sakai framework is available regardless of if the request came via the portal or as a result of a direct request to the tool servlet..... either of which are valid.

Service API

- Tools use Serivce API's
    - So tools don't have to re-invent the wheel.
- Pure Java interfaces
    - eg UserDirectoryService

"OASIS defines service as "a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description."" Wikipedia

Service API's are the communications contracts between the separated parts of Sakai.

They are nearly always pure java API's and they are stored in the shared classloader that everything can see.

For example, a UserDirectoryService can tell you who the current user is. The provision of these services ensures that tools can build on the framework of services without having to know or understand the details of the implementation.

Reimplementation

It also allows the teams responsible for the underlying services to improve and even re-implement those services without impacting the tool writers.

Evolution of APIs and Services

The service apis in Sakai are still evolving, but they are the most stable area of sakai. Tools that bind to them only need to evolve at the same slow pace regardless of the rate of change of the underlying implementation.

Components

- Provide Implementations of Service API's
- Managed by a Component Manager
  - eg BaseUserDirectoryService

"A component is an object written to a specification. It does not matter what the specification is: COM, Java Beans, etc., as long as the object adheres to the specification. It is only by adhering to the specification that the object becomes a component and gains features like reusability and so forth." Wikipedia

The Components in sakai most of the heavy lifting for core services.

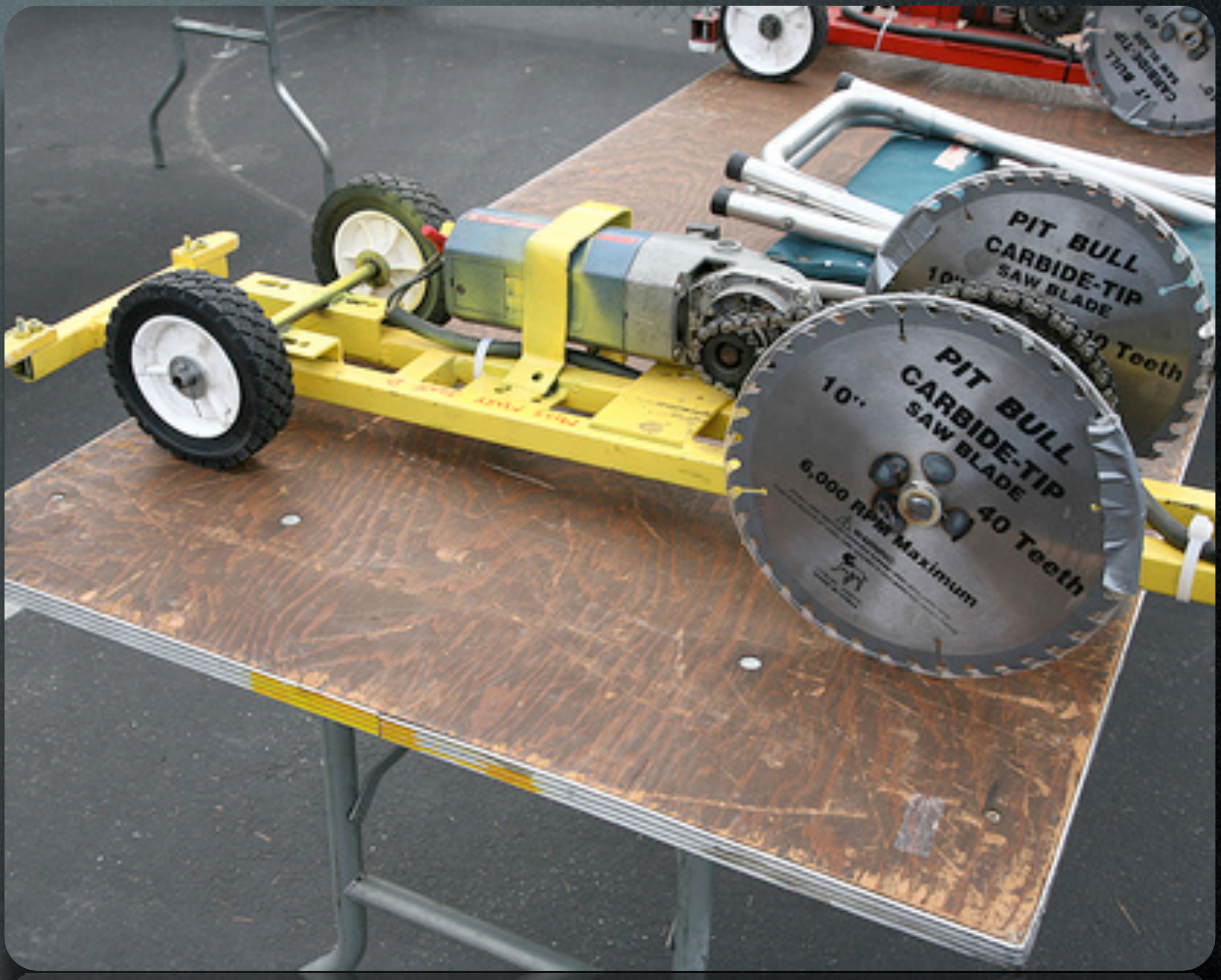
Each component group or pack, is loaded by the component manager into an isolated class-loader and exports its implementations to the component manager.

We tend to re-implement parts of the sakai component space at regular intervals

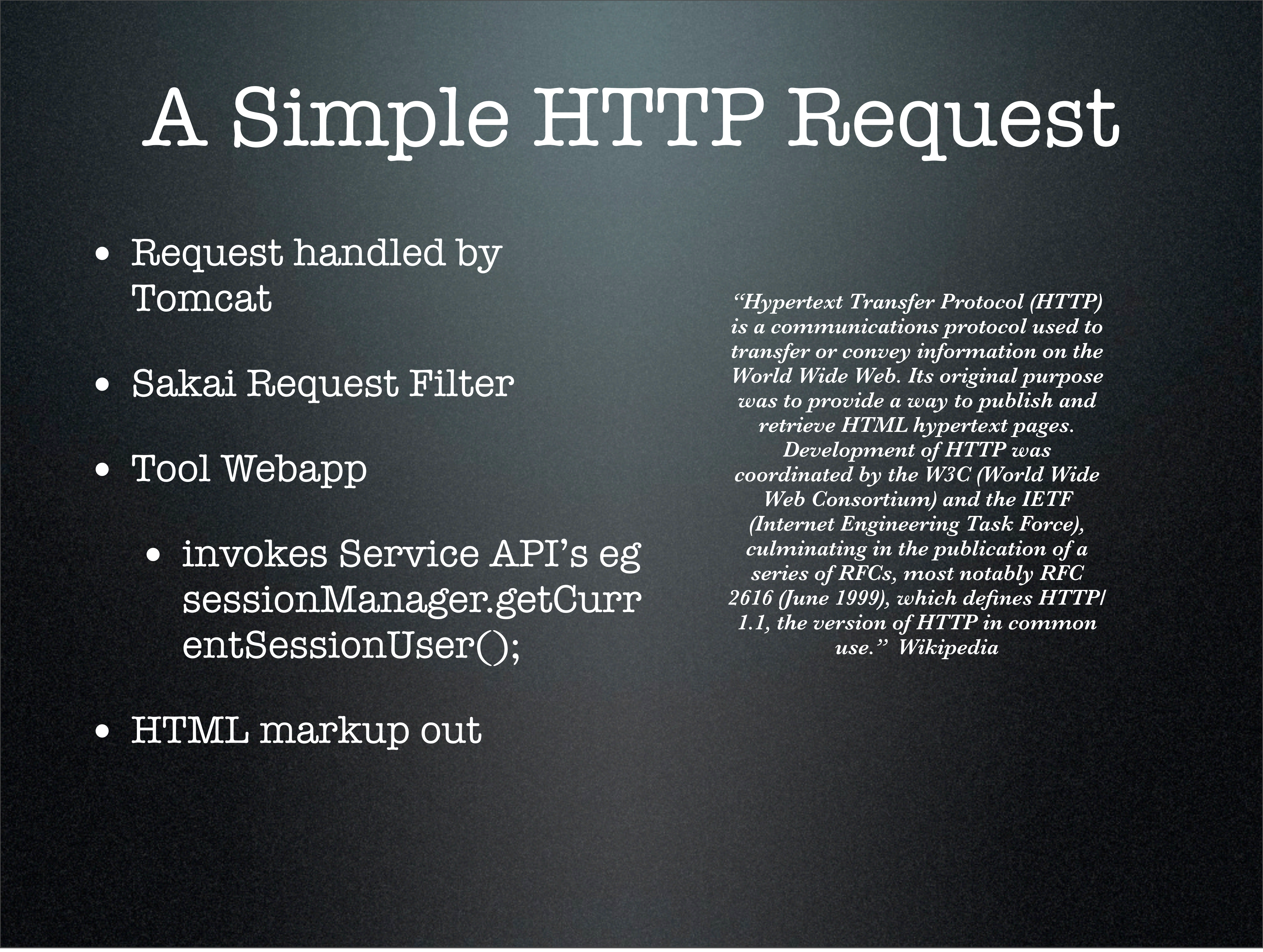
. There is a continuous programme of improvement and optimization of code in this space. If we do this properly, and so far, in most cases we have, the tool writer should not need to do any additional work.

For example in 2.5 we changed the storage implementation of Content Hosting Service, but the resources tool didnt have to change any lines of code as a result.

A Simple Tool



Lets take a look a simple tool. Not the details of how to write a webapp, since there are plenty of books on how to do that, but the details of how to make a webapp work in sakai.



A Simple HTTP Request

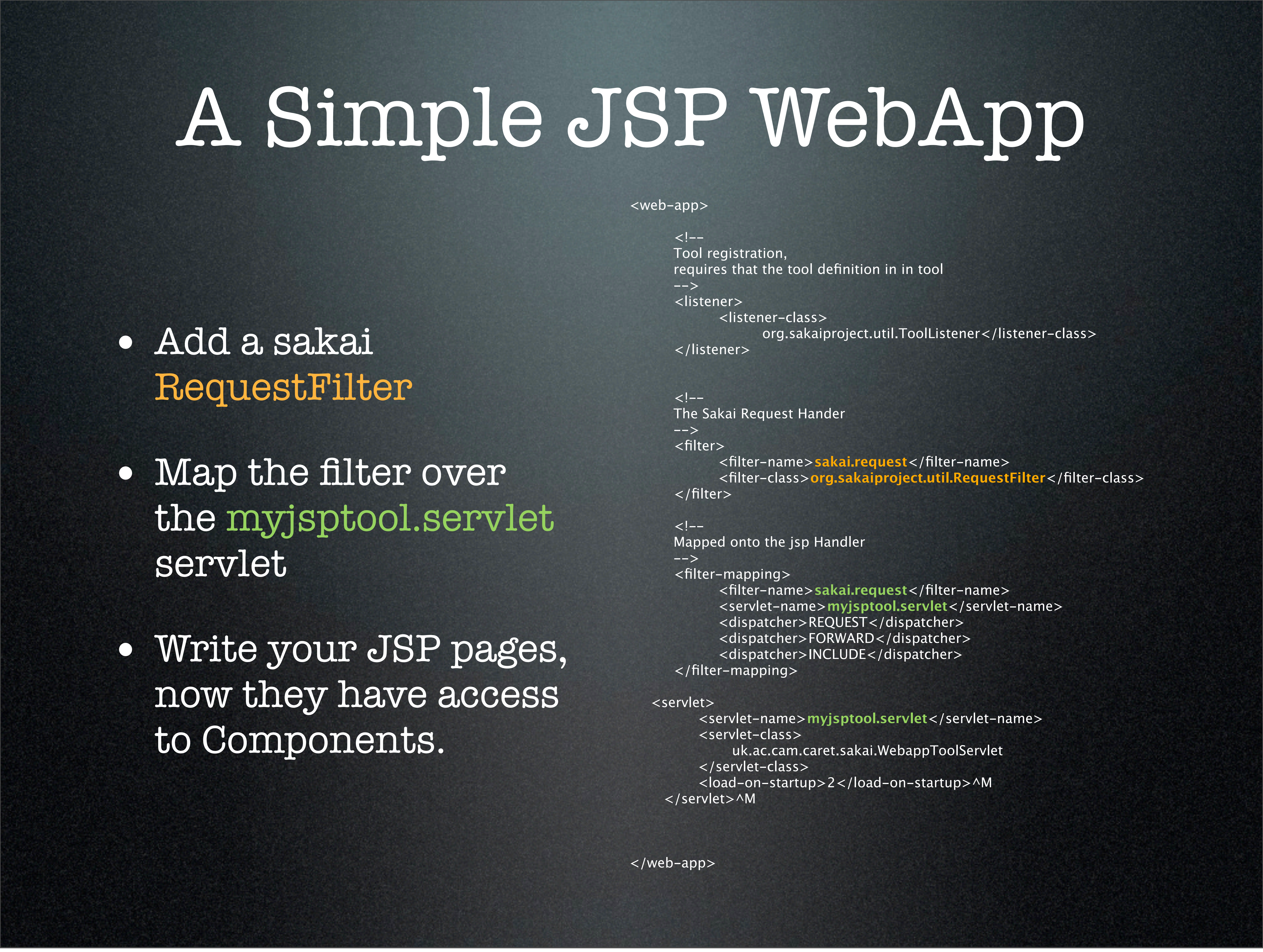
- Request handled by Tomcat
- Sakai Request Filter
- Tool Webapp
  - invokes Service API’s eg sessionManager.getCurrentSessionUser();
- HTML markup out

“Hypertext Transfer Protocol (HTTP) is a communications protocol used to transfer or convey information on the World Wide Web. Its original purpose was to provide a way to publish and retrieve HTML hypertext pages. Development of HTTP was coordinated by the W3C (World Wide Web Consortium) and the IETF (Internet Engineering Task Force), culminating in the publication of a series of RFCs, most notably RFC 2616 (June 1999), which defines HTTP/1.1, the version of HTTP in common use.” Wikipedia

So a HTTP request comes in. Its handled by Tomcat. Next we need to setup the Sakai Framework environment. For this to happen we attache a Sakai Request Filter to both the URL and the servlet by name handling the request.

The Sakai portal dispatches to tools based on their name, but direct URL access is valid so we need to make certain that the sakai framework is present regardless of the path.

The Request Filter injects the Component manager and does some other setup, and then control goes to the tool.
The tool processes the request, perhaps invoking services retrieved from the component manager, and then it sends markup out as a response.... hopefully.



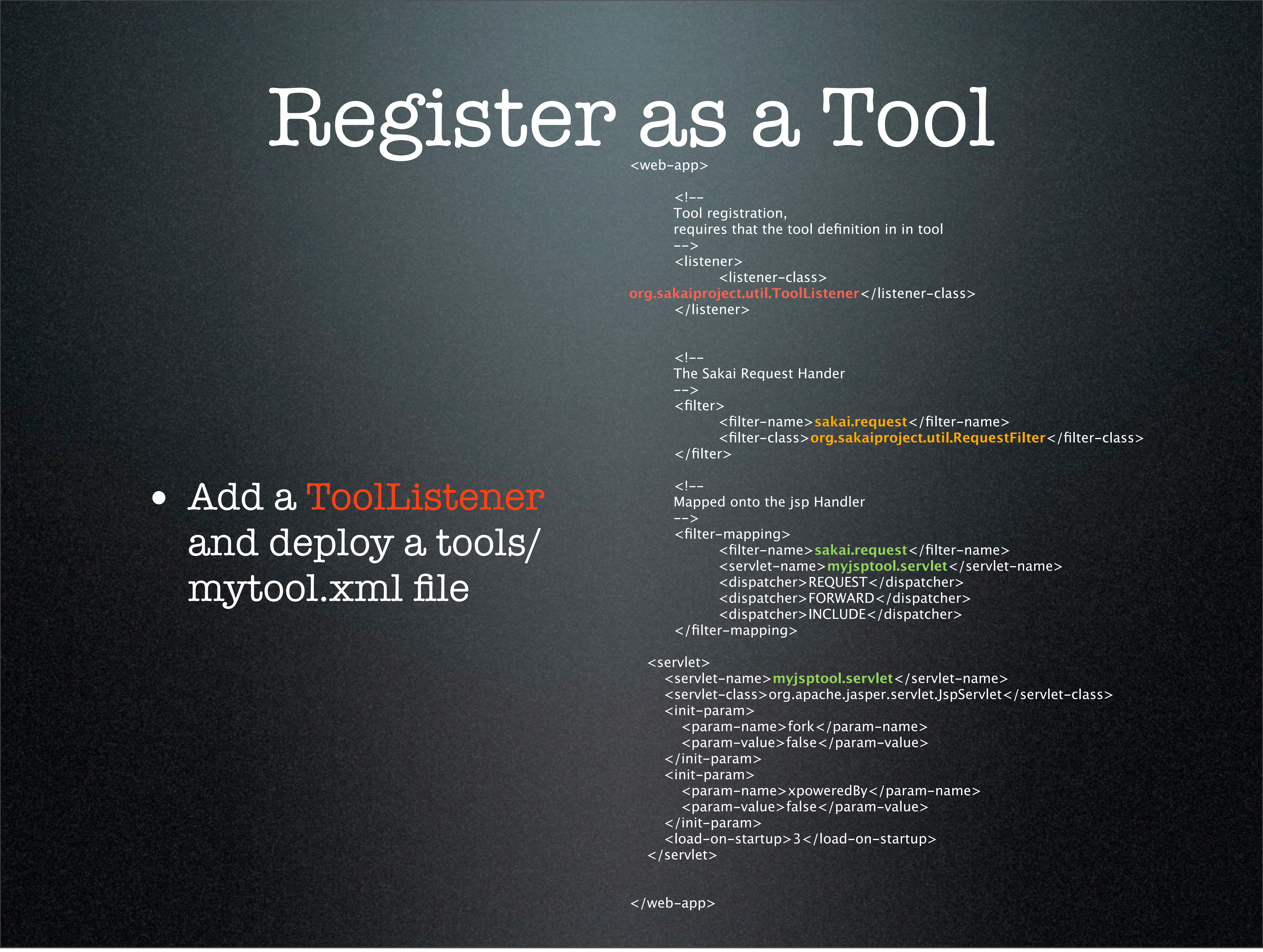
A Simple JSP WebApp

- Add a sakai RequestFilter
- Map the filter over the myjsptool.servlet servlet
- Write your JSP pages, now they have access to Components.

```
<web-app>

        <!--
        Tool registration,
        requires that the tool definition in in tool
        -->
        <listener>
                <listener-class>
                        org.sakaiproject.util.ToolListener</listener-class>
        </listener>


        <!--
        The Sakai Request Hander
        -->
        <filter>
                <filter-name>sakai.request</filter-name>
                <filter-class>org.sakaiproject.util.RequestFilter</filter-class>
        </filter>

        <!--
        Mapped onto the jsp Handler
        -->
        <filter-mapping>
                <filter-name>sakai.request</filter-name>
                <servlet-name>myjsptool.servlet</servlet-name>
                <dispatcher>REQUEST</dispatcher>
                <dispatcher>FORWARD</dispatcher>
                <dispatcher>INCLUDE</dispatcher>
        </filter-mapping>

    <servlet>
        <servlet-name>myjsptool.servlet</servlet-name>
        <servlet-class>
              uk.ac.cam.caret.sakai.WebappToolServlet
        </servlet-class>
        <load-on-startup>2</load-on-startup>^M
    </servlet>^M


</web-app>
```

So If you have written a java tomcat web app, you know all of that except the request filter bit. This is a web.xml file built with the App Builder Plugin for Eclipse. We have added a filter named sakai.request, and we map that to the myjsptool.servlet so that requests, forward dispatches and include dispatches all go through the filter.

The servlet, is a JSP dispatcher servlet that will just perform a second dispatch to the JSP servlet..... so you can write JSP pages to create your tool.



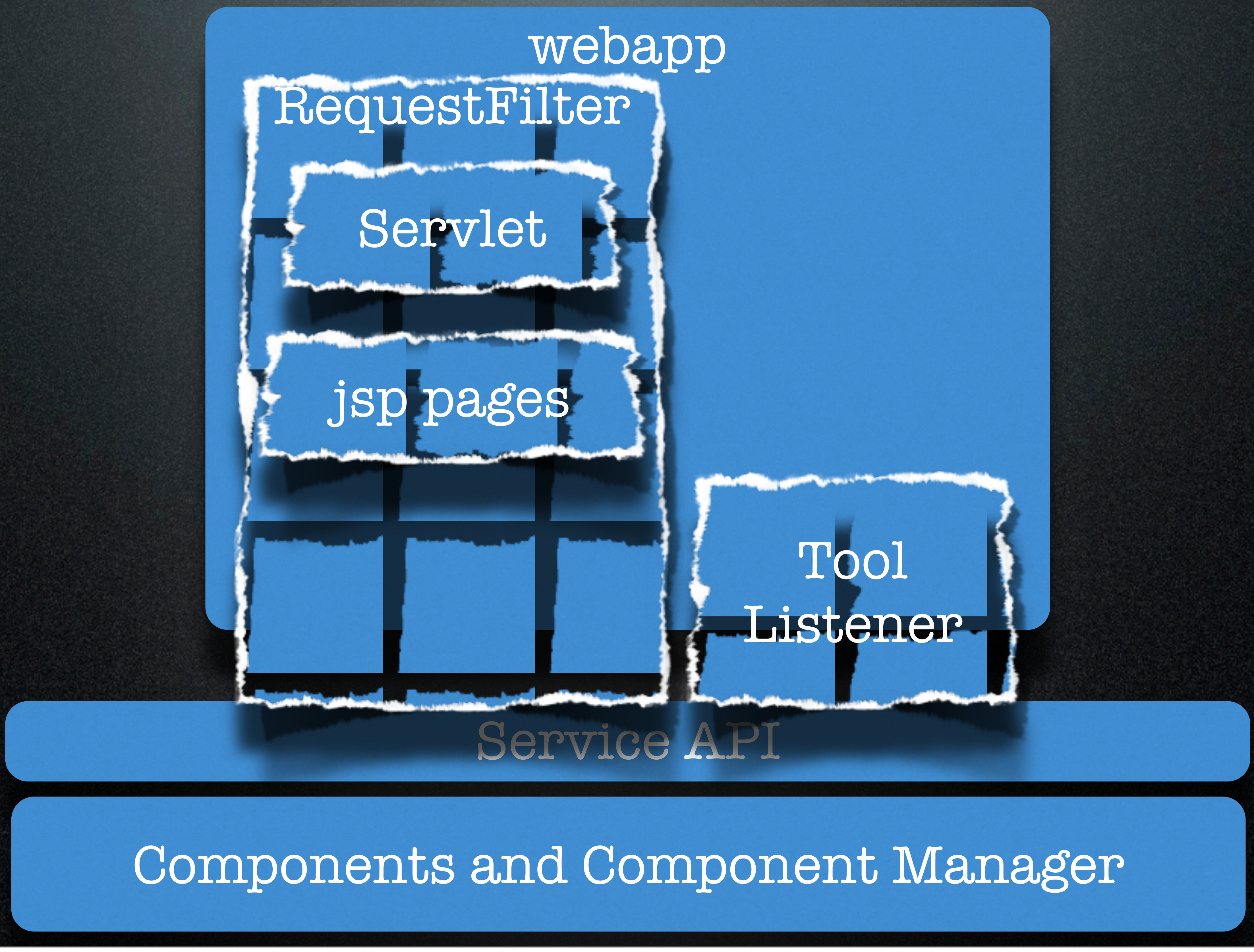
Register as a Tool

- Add a ToolListener and deploy a tools/ mytool.xml file

```
<web-app>

        <!--
        Tool registration,
        requires that the tool definition in in tool
        -->
        <listener>
                <listener-class>
org.sakaiproject.util.ToolListener</listener-class>
        </listener>


        <!--
        The Sakai Request Hander
        -->
        <filter>
                <filter-name>sakai.request</filter-name>
                <filter-class>org.sakaiproject.util.RequestFilter</filter-class>
        </filter>

        <!--
        Mapped onto the jsp Handler
        -->
        <filter-mapping>
                <filter-name>sakai.request</filter-name>
                <servlet-name>myjsptool.servlet</servlet-name>
                <dispatcher>REQUEST</dispatcher>
                <dispatcher>FORWARD</dispatcher>
                <dispatcher>INCLUDE</dispatcher>
        </filter-mapping>

    <servlet>
       <servlet-name>myjsptool.servlet</servlet-name>
       <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
       <init-param>
          <param-name>fork</param-name>
          <param-value>false</param-value>
       </init-param>
       <init-param>
          <param-name>xpoweredBy</param-name>
          <param-value>false</param-value>
       </init-param>
       <load-on-startup>3</load-on-startup>
    </servlet>


</web-app>
```

To make this a tool, we need to tell sakai about it. So we register a ToolListener. This reads all the xml files under webapp/tools/*.xml and injects the tool definition into Sakai for the portal to use. This registration includes the name of the servlet, so the portal can dispatch to it.

The Sakai Stack



So this is our stack.

The request filter enables the servlet and jsp pages to communicate with sakai's service apis.
A tool listener registers the tool with sakai.

View Technologies

- Anything you like....
  - JSP
  - JSF
  - RSF
  - Wicket
  - Velocity

Don't like JSP.
Some people say its ugly and unsustainable.
There are alternatives. JSF.... well thats just hard JSP's

RSF, developed at Cambridge, intended to be a rewrite of JSF to make it much easier to develop tools.
Wicket from Apache. Similar to RSF, Apache top level project.
Velocity, both the heavyweight full life-cycle velocity as used by most of the older tools. Or the light weight tempting only velocity.

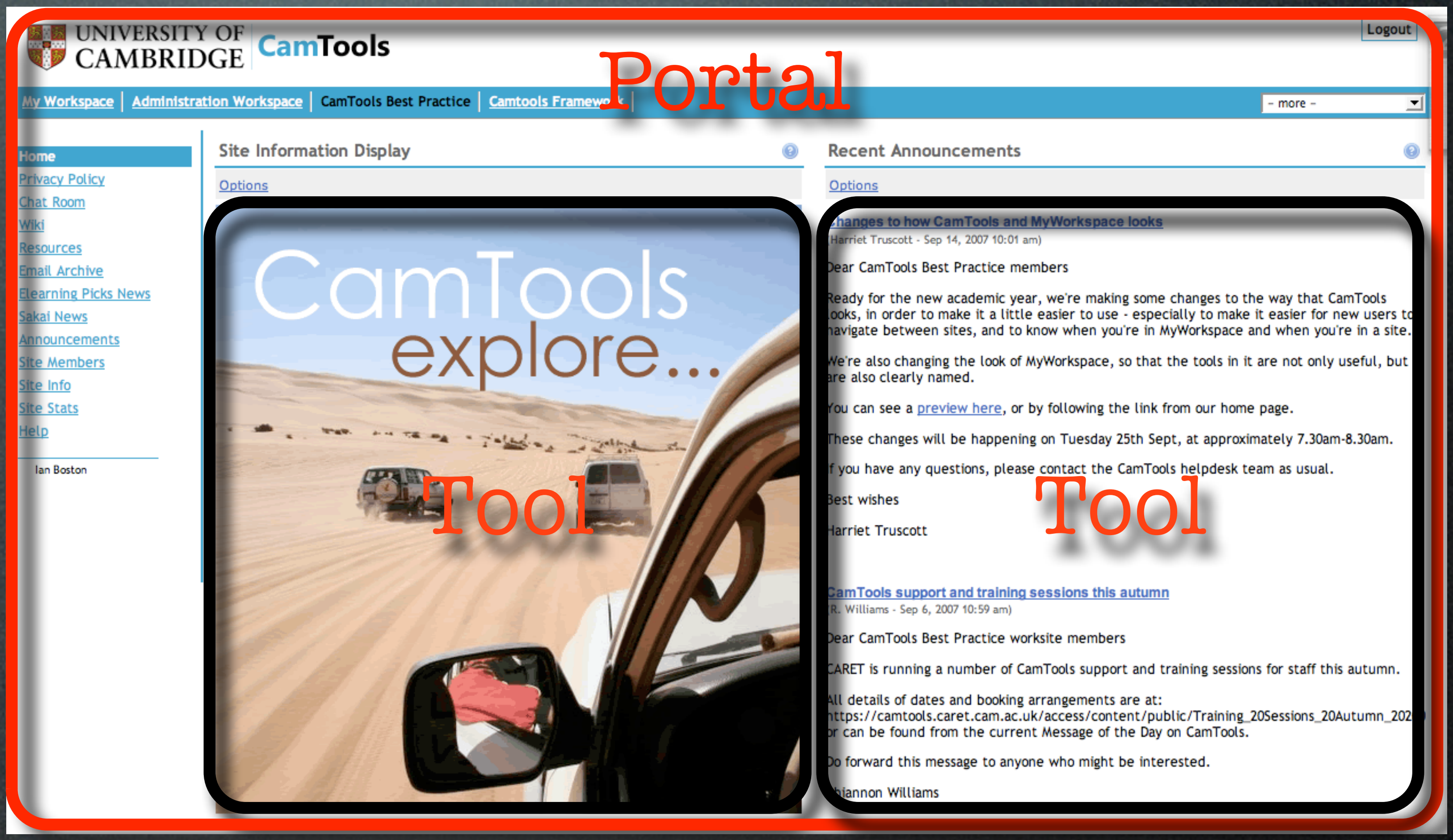
You can choose your favorite,
But if you want to integrate well and give a uniform User Experience, then you should emit uniform markup as the rest of sakai.

The Portal Aggregator

Tools were relatively simple. They don't set the scene for the user as the portal owns most of the screen decoration. The portals job is to provide the user with suitable navigation appropriate for the context and provide a container which the tools can work within.

Screen Space

- Tool renders tool space.
- Portal renders container.
- Portal dispatches to tool



hanges to how CamTools and MyWorkspace looks
(Harriet Truscott - Sep 14, 2007 10:01 am)

ear CamTools Best Practice members

Ready for the new academic year, we're making some changes to the way that CamTools ooks, in order to make it a little easier to use - especially to make it easier for new users to avigate between sites, and to know when you're in MyWorkspace and when you're in a site.

We're also changing the look of MyWorkspace, so that the tools in it are not only useful, but are also clearly named.

You can see a preview here, or by following the link from our home page.

These changes will be happening on Tuesday 25th Sept, at approximately 7.30am-8.30am.

f you have any questions, please contact the CamTools helpdesk team as usual.

Best wishes

Harriet Truscott

CamTools support and training sessions this autumn
(R. Williams - Sep 6, 2007 10:59 am)

Dear CamTools Best Practice worksite members

CARET is running a number of CamTools support and training sessions for staff this autumn.

All details of dates and booking arrangements are at:
https://camtools.caret.cam.ac.uk/access/content/public/Training_20Sessions_20Autumn_202
or can be found from the current Message of the Day on CamTools.

Do forward this message to anyone who might be interested.

hiannon Williams

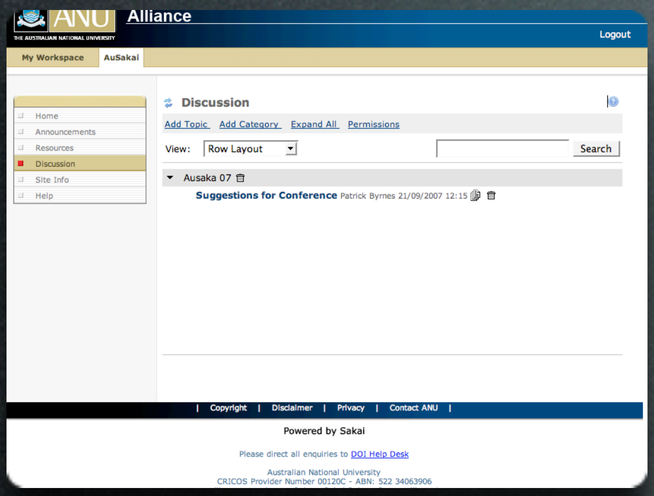
So the portal renders the header, site list for the user, and tool list for the site. It creates 2 containers in this instance, and then dispatches to the tool for the tool content.

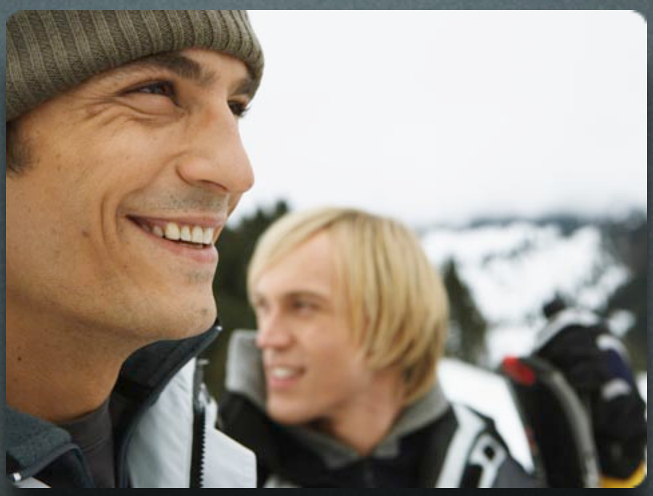
Tool Containers Types

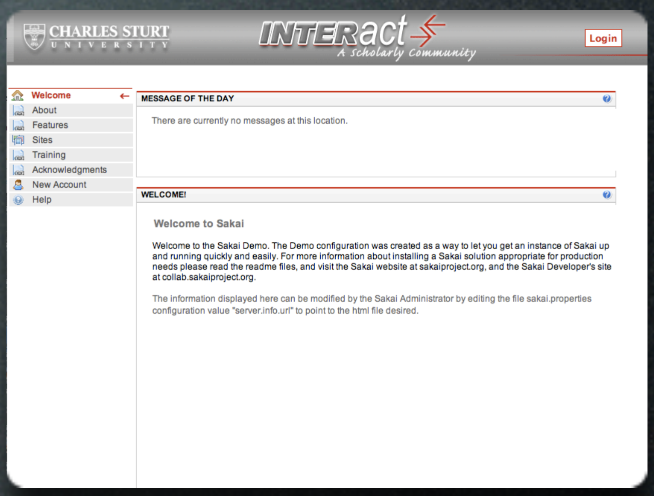
- IFrame
- JSR-168 Portlet
- Frameless

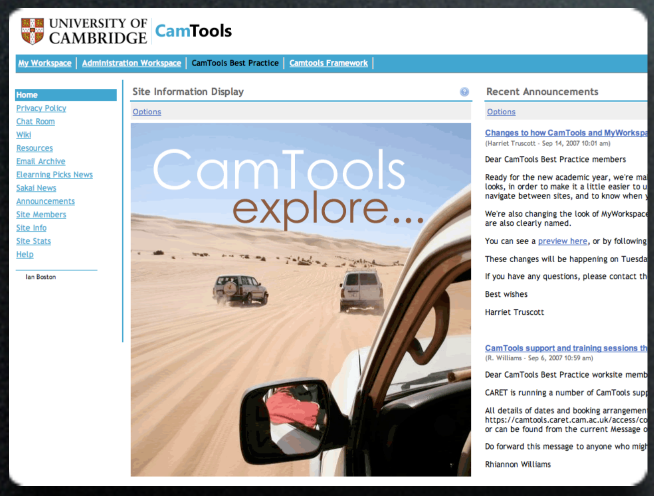
The portal provides containers for the tools. It has three types, the dreaded iframe that we have used for the past 4 years. JSR-168 portlets that render on a flat page and some frameless tools that can work as fragment producers and co-exist with other tools.

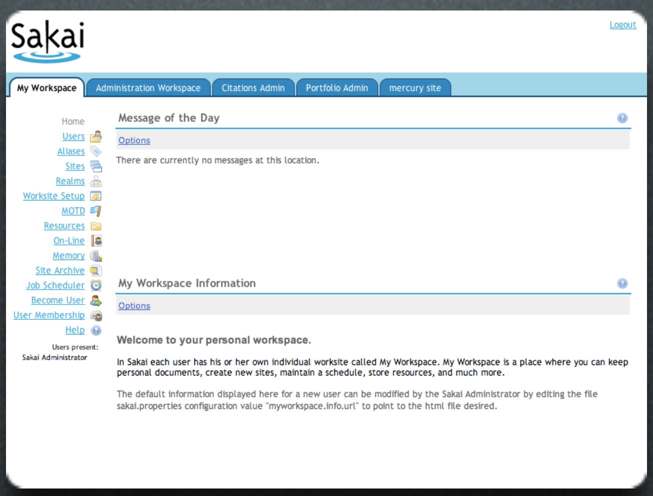
Iframes are not ideal from an accessibility point of view, and generate some tensions between the normal web users experience and sakai, however, there are many users who find tool state persistence extremely useful as the switch between tools.

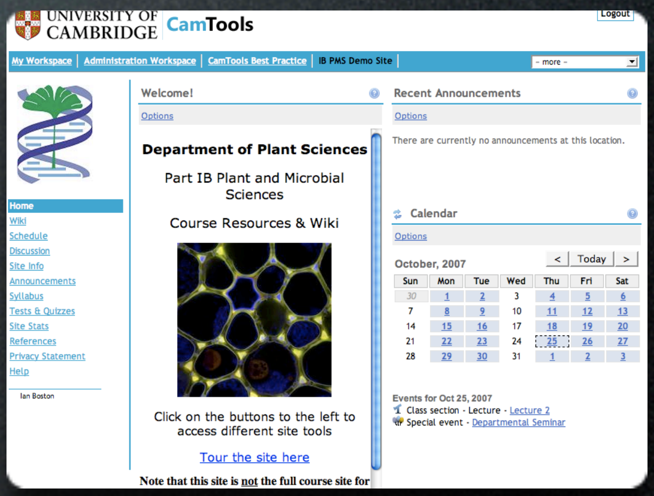











UNIVERSITY OF CAMBRIDGE CamTools
Welcome!
Department of Plant Sciences
Part IB Plant and Microbial Sciences
Course Resources & Wiki
Recent Announcements
Calendar

Customization

local css, per-site css,
templates, new render engines

Monday, 5 November 2007

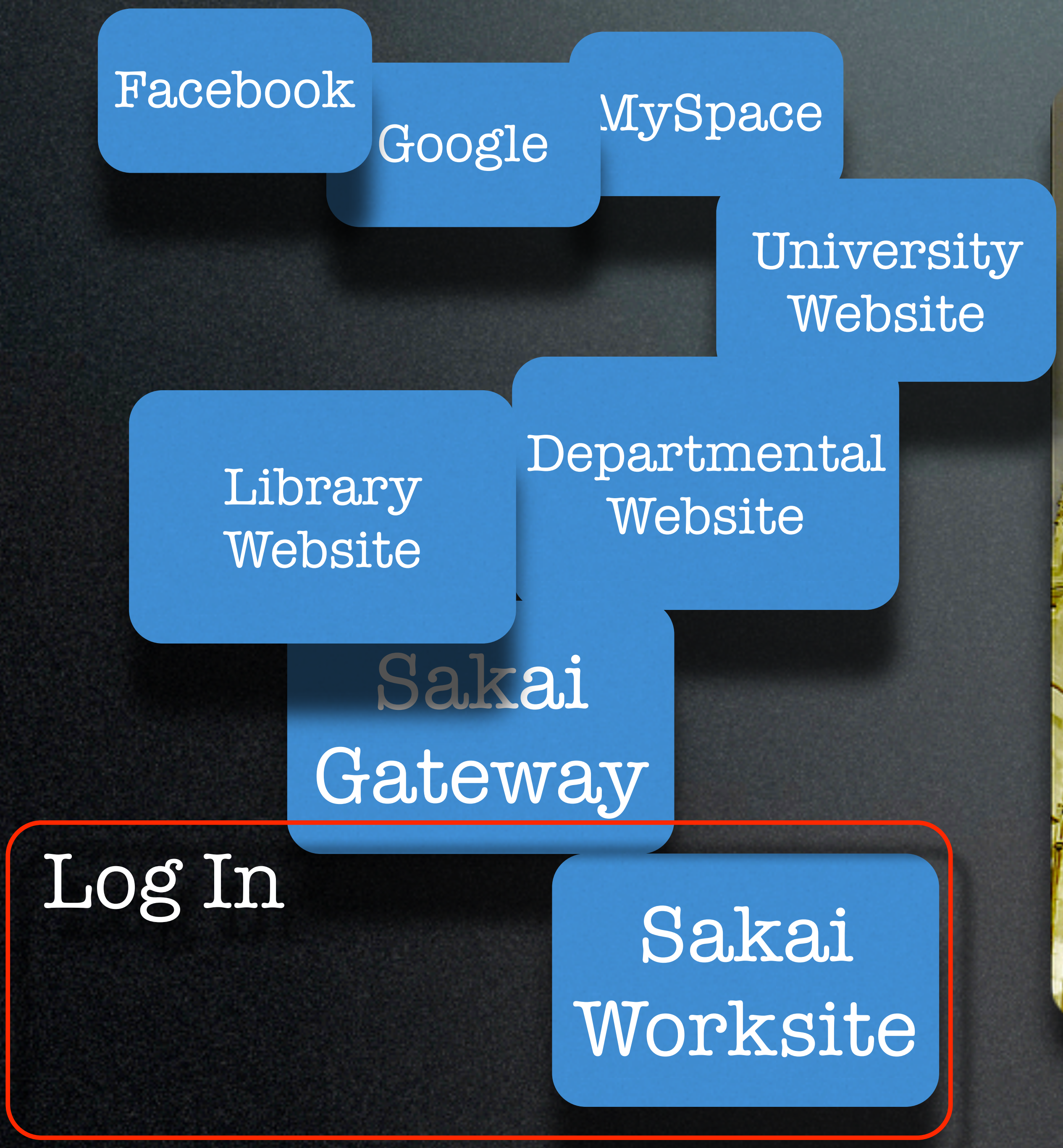
Sakai is open source and we have to support about 160 installations world wide at the moment. We cant dictate look and feel like commercial vendors can, so Sakai portal is totally reskinnable.

Most sites use CSS to change the look and feel, the the ones here are CSS based. You can change the css on a site by site basis. You can go much further and replace the velocity templates to where all the portal markup is held and completely re-configure the look and feel.

If you don’t like velocity templates, you can re-write the render engine to use something else. There is an RSF render engine in contrib.

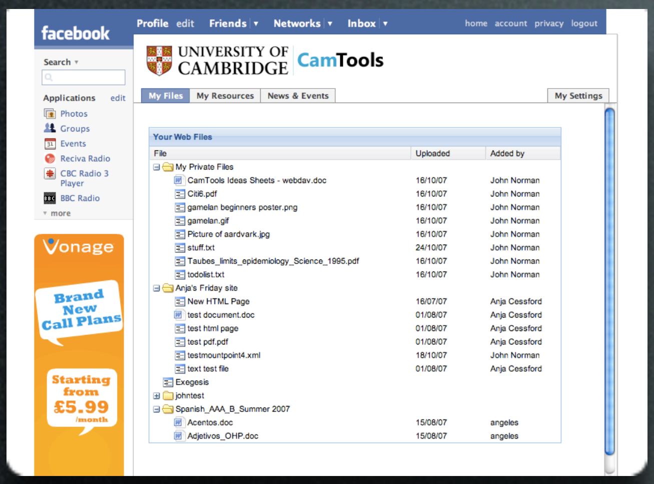
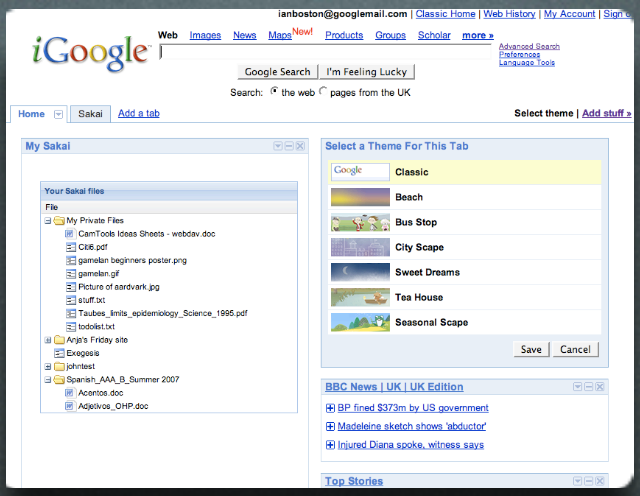
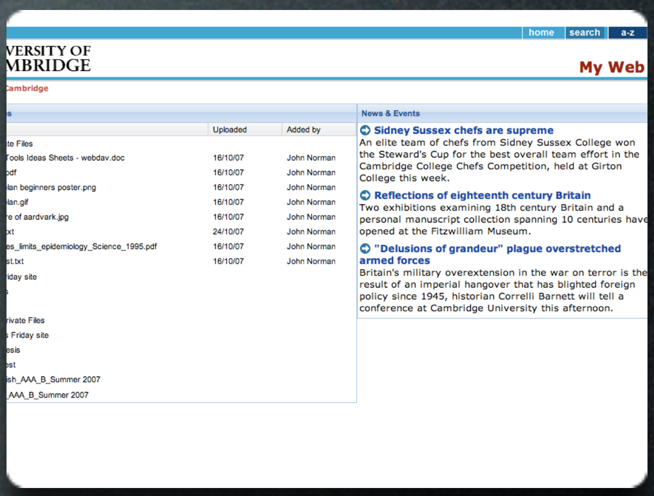
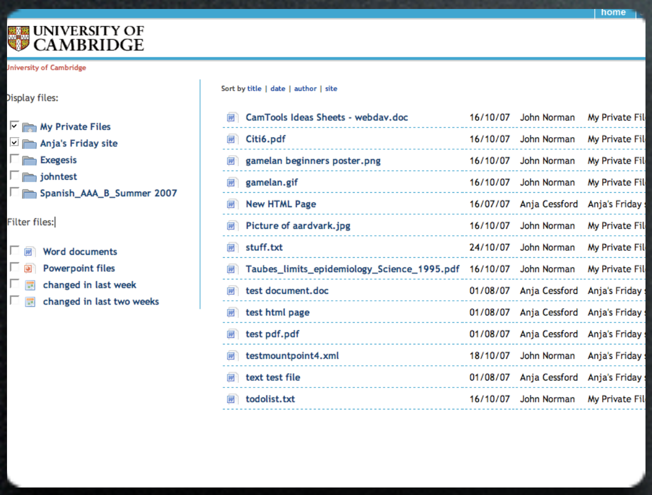
The new osp portal in 2.5 uses an XSLT render engine.

An academic day



But Sakai doesn’t exist in isolation. For new students arriving at a university, its a bewildering experience, especially at Cambridge. The Student may have a Facebook, igoogle, myspace home page. They might visit administrative university websites.

Confused they arrive at the Sakai landing page or gateway, they might log in. And if they are lucky find their courses. But where they want to be is back in Facebook, with their friends.

External Applications

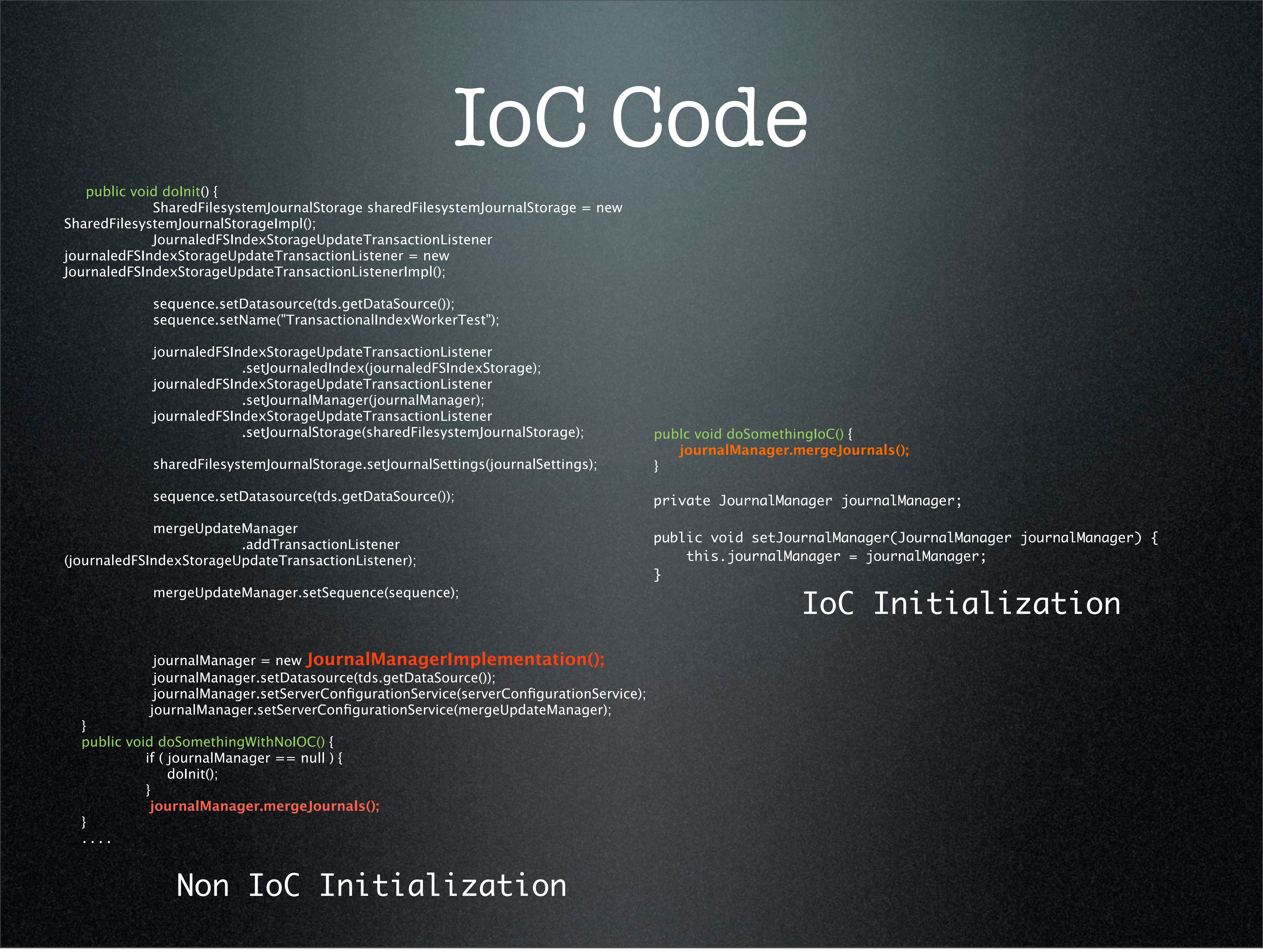
Facebook, iGoogle, PHP webapps, WIP

So we are doing work to make Sakai accessible from within the students natural home page. We have written a prototype Facebook Application, and iGoogle widget and some more friendly pages.

We have done this all in PHP running on a different server from Sakai, but talking to Sakai over REST based web services. Facebook users can see new content in sites they are members of, we want to push the information that Sakai contains out into other spaces and applications. This is work under development, but as with many PHP apps, its moving very fast.

Spring and Components

The core of sakai is its component manager that contains components. We use Spring Framework to deliver this functionality as is a capable Inversion of Control container.



IoC Code

```
    public void doInit() {
            SharedFilesystemJournalStorage sharedFilesystemJournalStorage = new
SharedFilesystemJournalStorageImpl();
            JournaledFSIndexStorageUpdateTransactionListener
journaledFSIndexStorageUpdateTransactionListener = new
JournaledFSIndexStorageUpdateTransactionListenerImpl();

            sequence.setDatasource(tds.getDataSource());
            sequence.setName("TransactionalIndexWorkerTest");

            journaledFSIndexStorageUpdateTransactionListener
                    .setJournaledIndex(journaledFSIndexStorage);
            journaledFSIndexStorageUpdateTransactionListener
                    .setJournalManager(journalManager);
            journaledFSIndexStorageUpdateTransactionListener
                    .setJournalStorage(sharedFilesystemJournalStorage);

            sharedFilesystemJournalStorage.setJournalSettings(journalSettings);

            sequence.setDatasource(tds.getDataSource());

            mergeUpdateManager
                    .addTransactionListener
(journaledFSIndexStorageUpdateTransactionListener);

            mergeUpdateManager.setSequence(sequence);


            journalManager = new JournalManagerImplementation();
            journalManager.setDatasource(tds.getDataSource());
            journalManager.setServerConfigurationService(serverConfigurationService);
            journalManager.setServerConfigurationService(mergeUpdateManager);
    }
    public void doSomethingWithNoIOC() {
          if ( journalManager == null ) {
              doInit();
          }
          journalManager.mergeJournals();
    }
    ....
```

Non IoC Initialization

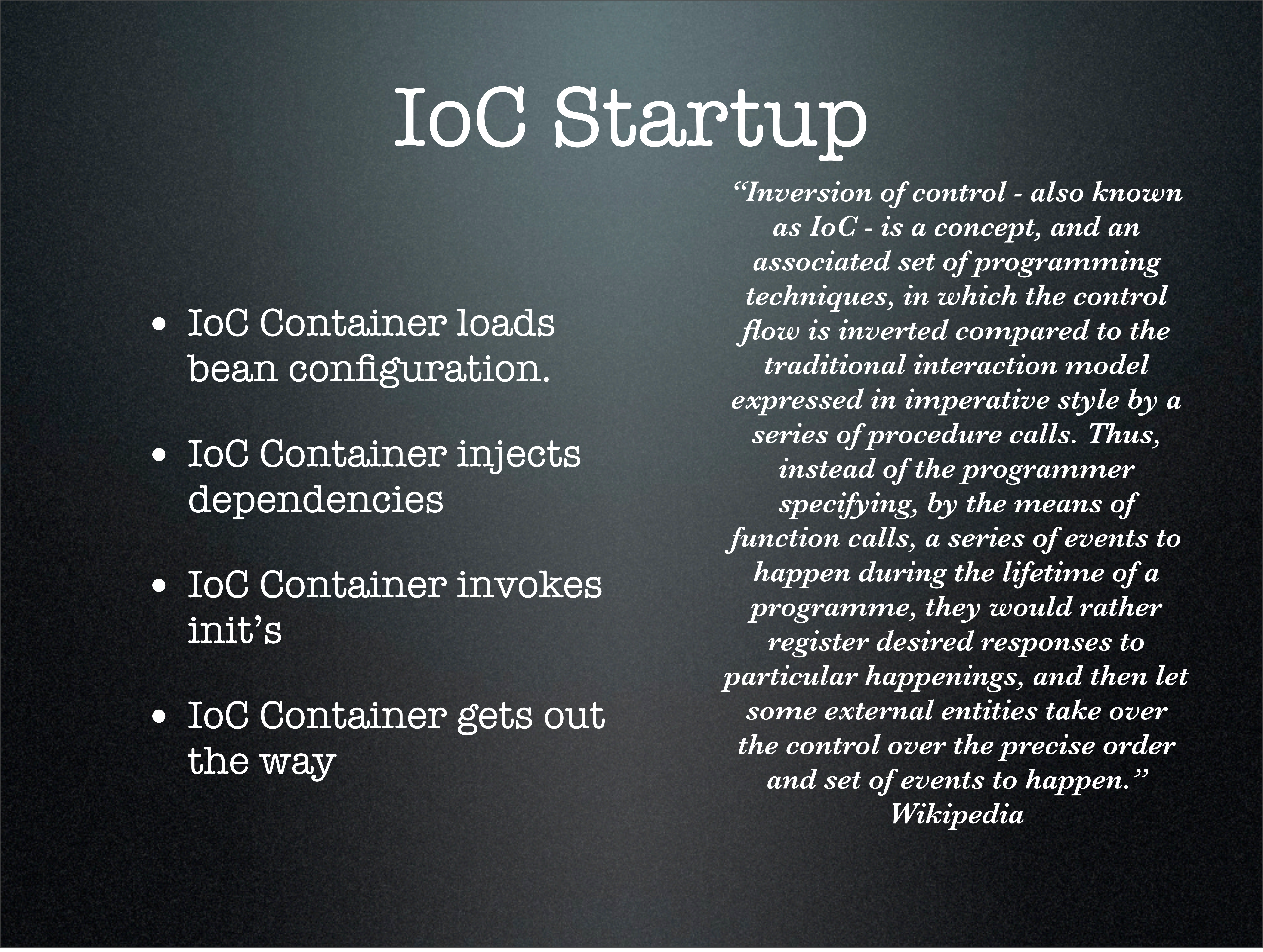
```
publc void doSomethingIoC() {
    journalManager.mergeJournals();
}

private JournalManager journalManager;

public void setJournalManager(JournalManager journalManager) {
    this.journalManager = journalManager;
}
```

IoC Initialization

Why IOC.

On the left is not IoC code. we have to manually inject everything to construct the code. We also have to bind directly to Implementations, so it hard to change implementations. We could use static factories, but that still means we have to do the injection under the covers.

So we use an IoC container, that according to a configuration file, injects the dependencies we need. No binding to implementations. This enables us to separate the services into independent components, leading to a component based architecture rather than an option architecture.



IoC Startup

- IoC Container loads bean configuration.
- IoC Container injects dependencies
- IoC Container invokes init’s
- IoC Container gets out the way

“Inversion of control - also known as IoC - is a concept, and an associated set of programming techniques, in which the control flow is inverted compared to the traditional interaction model expressed in imperative style by a series of procedure calls. Thus, instead of the programmer specifying, by the means of function calls, a series of events to happen during the lifetime of a programme, they would rather register desired responses to particular happenings, and then let some external entities take over the control over the precise order and set of events to happen.” Wikipedia

On Startup the IoC container loads the beans as configured, performs the injection of dependencies and then invokes a number of post creation methods.

Once the architecture is wired up, the IoC container gets out of the way. Spring does bring more than just pure IoC. This can be a blessing and a curse. It makes doing aspect based interception much easier, so we can apply caching and transactions without having to change the code... but it also complicates configuration, there is a balance to be maintained.

Spring Service Injection

- Configuration in XML files
- Components implement Service API's
- Consumers have dependency Service API's injected by Spring.
- Binding is to API's not Implementations

"Informally, it is expressed by the Hollywood Principle - "Don't call us, we'll call you"." Wikipedia

Some IoC containers use Java 5 annotations. Spring uses XML. In Sakai each component is defined in a pack, which contains its own configuration setup. Each pack exports service APIs to the component manager and each pack works within it own isolated class-loader. It can see its own classloader, and the shared classloader where the api's are.

But it cant see the implementations of other classes.... so one component cant bind via an implementation. This guarantees separation of concerns in the code base, but not always in the design.

Persistence

We have constructed a framework of components, busy serving users, but if the framework cant store anything, then its not much use for collaboration.

Database

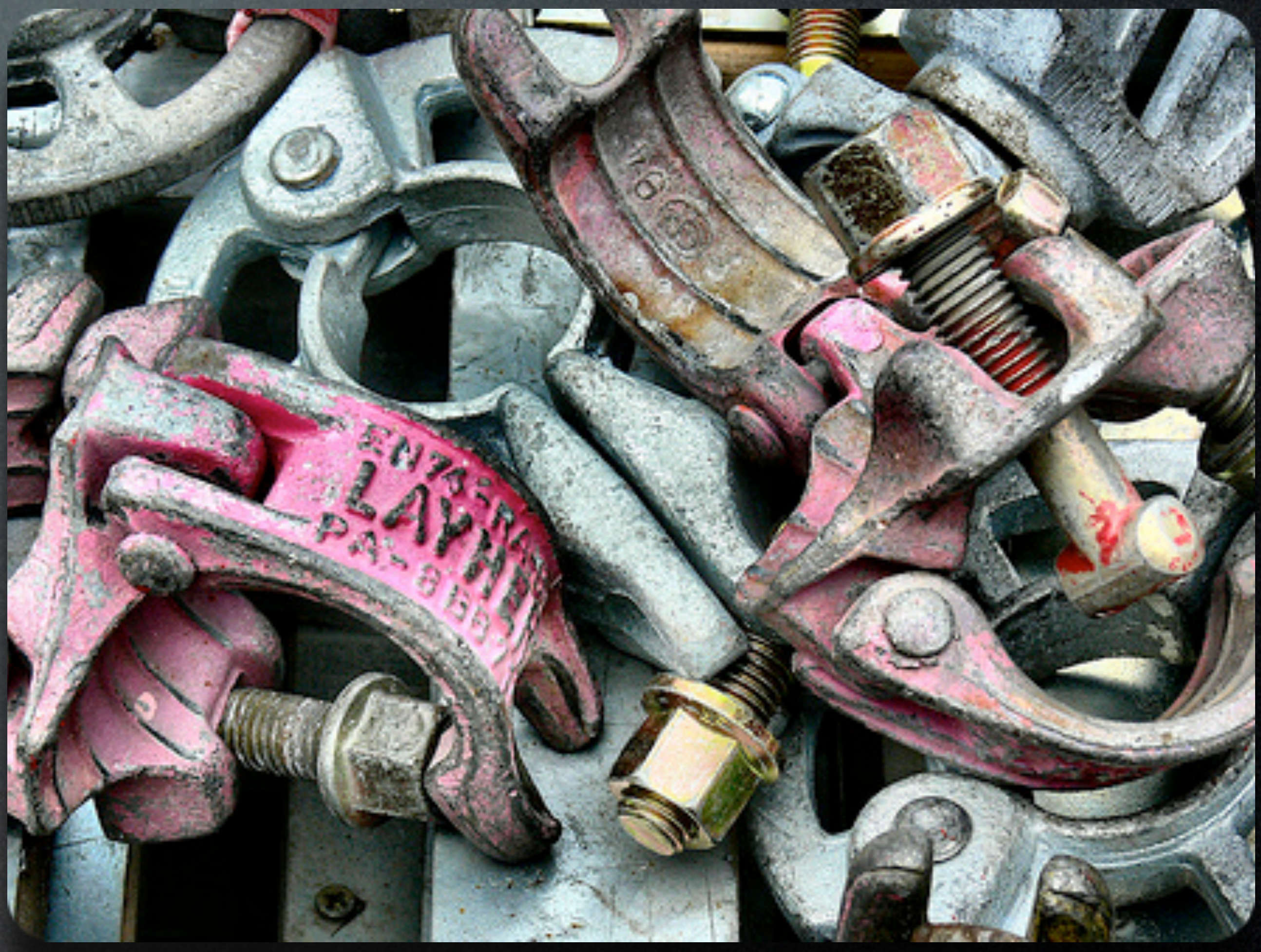
- Oracle
- MySQL
- HSQLDB
- Others

Sakai uses a database for persistence. We use 3, MySQL and Oracle for production and HSQL for demos.

The code base has recently been re-factored to allow other databases and there is a SQLServer port, however we do not currently QA databases other than the three code vendors.

ORM/JDBC

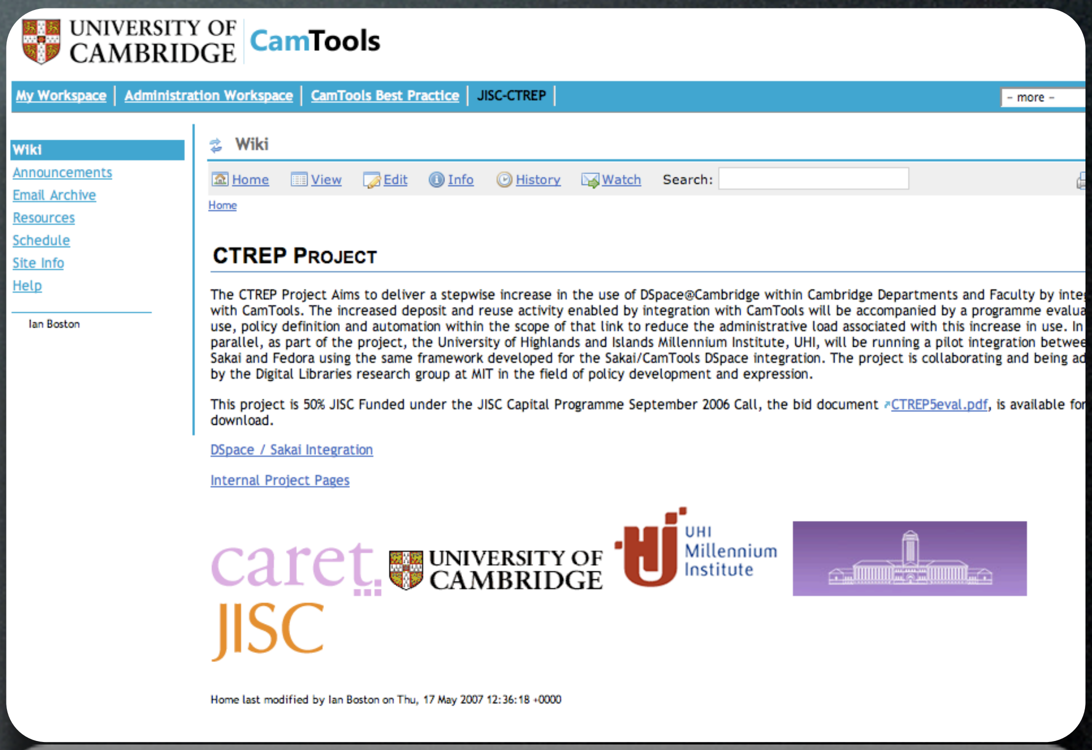
- Hibernate
- SpringJDBC
- Direct JDBC (SqlService)
- others
  - Apache Cayenne



To access those databases you can use an Object Relational Mapping framework or go direct to the SQL. Hibernate support is part of sakai and many tools use it. Spring has some JDBC support that makes it easier to manage transactions and connections, or you can use the original SqlService implementation that deals with the database directly via JDBC. You can even use the JDBC datasource directly if you want to do everything yourself.

Since the datasource is available other ORM frameworks can be used, like the excellent Apache Cayenne.. personal favorite.

RWiki: A Sakai tool


UNIVERSITY OF CAMBRIDGE
CamTools
My Workspace | Administration Workspace | CamTools Best Practice | JISC-CTREP
- more -
Wiki
Announcements
Email Archive
Resources
Schedule
Site Info
Help
Ian Boston
Wiki
Home View Edit Info History Watch Search:
Home

CTREP Project

The CTREP Project Aims to deliver a stepwise increase in the use of DSpace@Cambridge within Cambridge Departments and Faculty by inte
with CamTools. The increased deposit and reuse activity enabled by integration with CamTools will be accompanied by a programme evalua
use, policy definition and automation within the scope of that link to reduce the administrative load associated with this increase in use. In
parallel, as part of the project, the University of Highlands and Islands Millennium Institute, UHI, will be running a pilot integration betwee
Sakai and Fedora using the same framework developed for the Sakai/CamTools DSpace integration. The project is collaborating and being ad
by the Digital Libraries research group at MIT in the field of policy development and expression.

This project is 50% JISC Funded under the JISC Capital Programme September 2006 Call, the bid document CTREP5eval.pdf, is available fo
download.

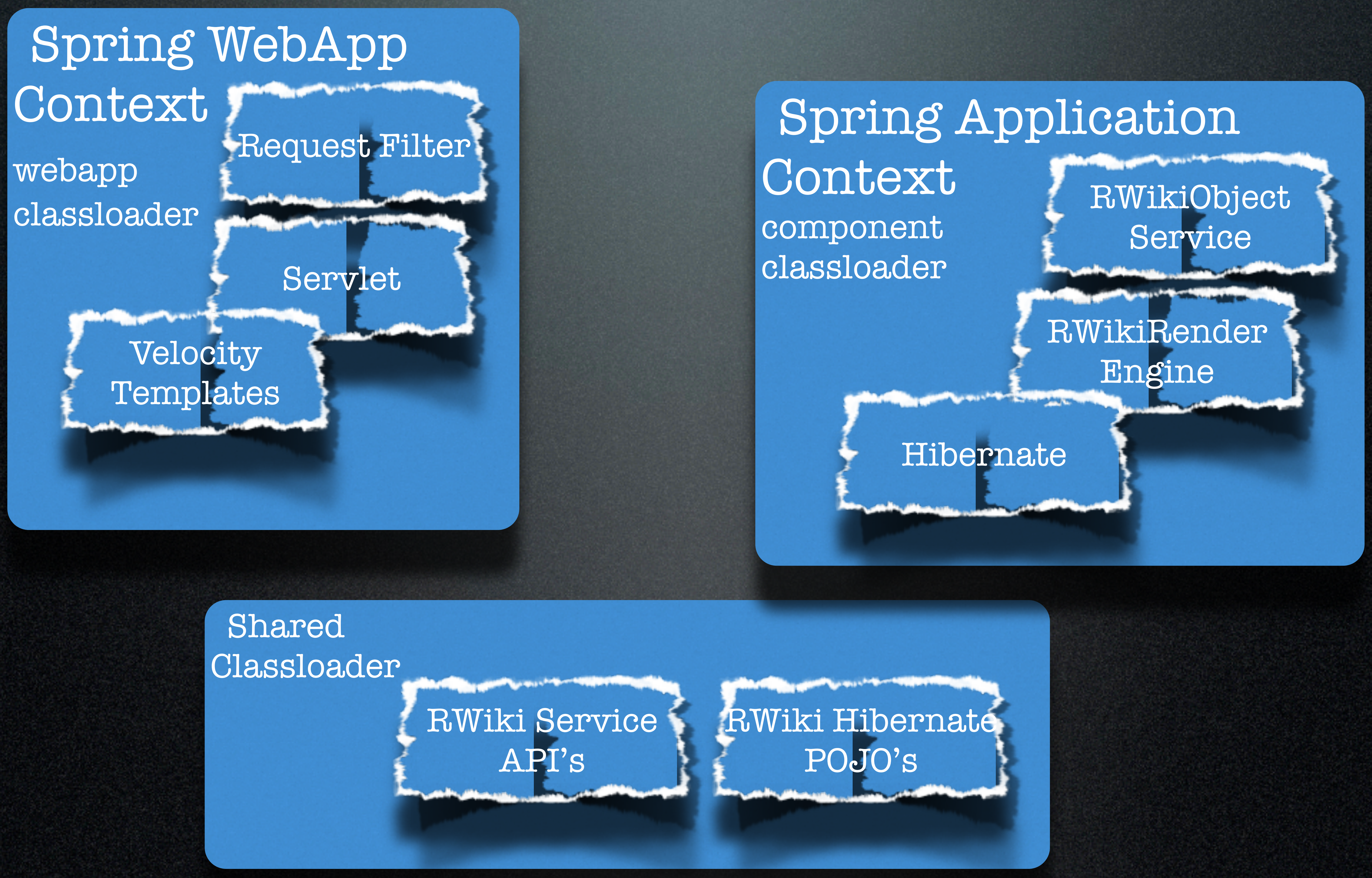
DSpace / Sakai Integration

Internal Project Pages

Home last modified by Ian Boston on Thu, 17 May 2007 12:36:18 +0000

Thats the structure of Sakai, now a real tool. RWiki. Developed for collaborative content writing amongst social science researchers in the field. It uses the Radeox wiki engine that isn SnipSnap and confluence.

RWiki: Structure



It has all the parts seen in a full sakai application. The Tool is housed in a webapp with a request filter. Content markup is provided by Velocity templates, but the Servlet is a pure request scope servlet that does no use any of the heavy semantics of the full velocity servlet. The tool has its own Spring context since it is configured by Spring.

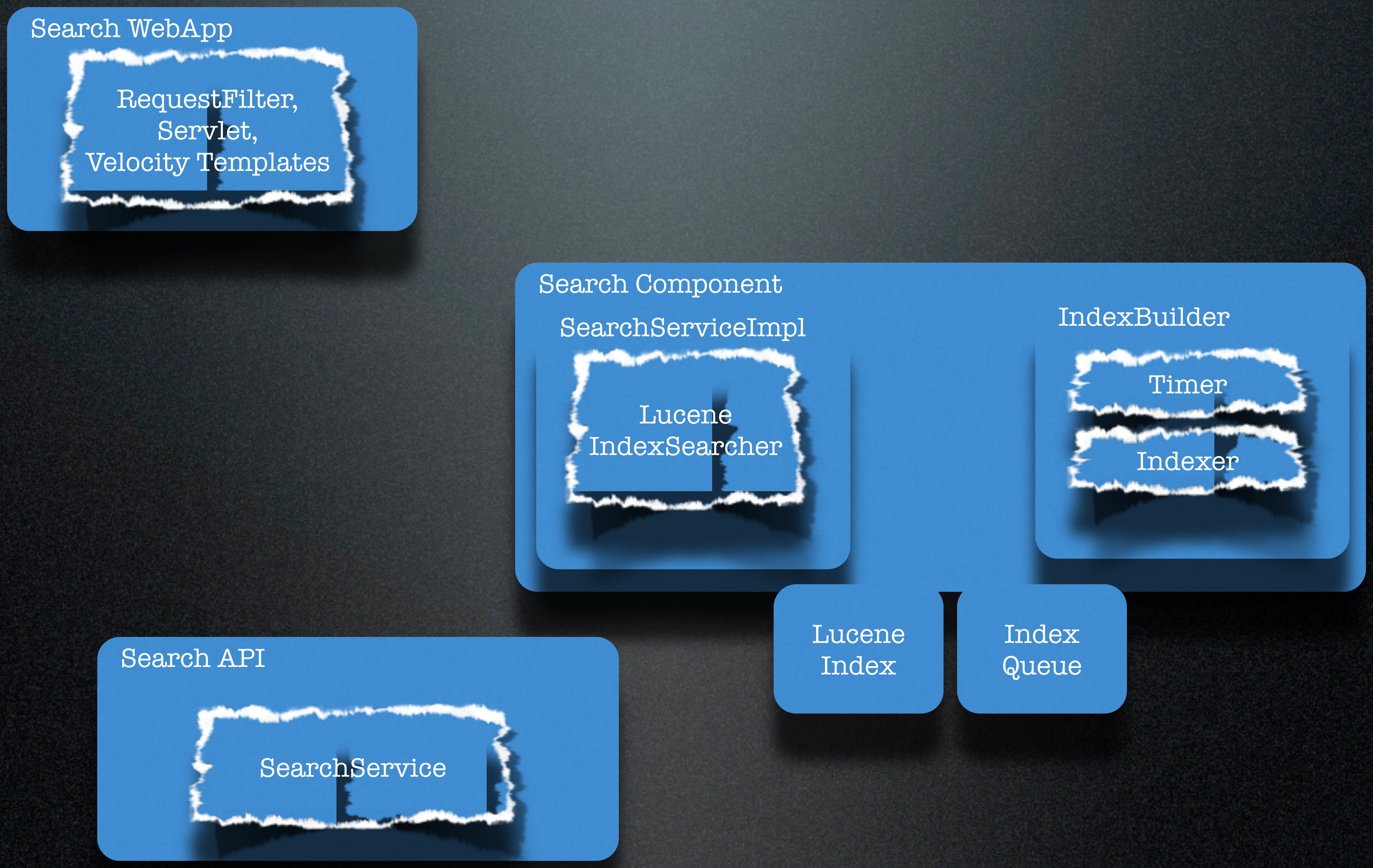
The tool uses and RwikiService api to manage both rendered and raw wiki content.

In the component there are 2 primary implementations, the Rwiki Object Service and the Render Engine. Data is persisted by hibernate and the Hibernate POJO's are in shared. Other tools wanting wiki markup could use the Render Engine via its API.

Search Sakai Component

A more complex sakai tool is search. Search is designed to contain a single search index for all the content in sakai over all users and all sites. The index contains enough authorization information to make it possible for a user to be given only the content that they are allowed to see.

Search Structure



There is a search tool, similar in structure to rwiki. A search API in shared and then an search component. The search component is more complex as it has a request component and a back-end indexer that build journaled log of search index additions.

Search Component

- About 40 beans
- Spring Configured
- Indexer free threaded
- Builds a Journaled Lucene Index

```
<beans>

    <bean id="org.sakaiproject.search.api.JournalSettings"
            class="org.sakaiproject.search.journal.impl.JournalSettings"
            >
            <property name="localIndexBase" ><value>${sakai.home}indexwork</value></property>
            <property name="sharedJournalBase" ><value>${sakai.home}searchjournal</value></property>
            <property name="minimumOptimizeSavePoints" ><value>10</value></property>
            <property name="optimizMergeSize" ><value>5</value></property>
            <property name="soakTest" ><value>false</value></property>
    </bean>


<bean id="org.sakaiproject.search.api.SearchService"
  class="org.sakaiproject.search.component.service.impl.ConcurrentSearchServiceImpl"
  init-method="init" >

    <property name="notificationService"><ref bean="org.sakaiproject.event.api.NotificationService" /></property>
    <property name="eventTrackingService"><ref bean="org.sakaiproject.event.api.EventTrackingService" /></property>
    <property name="userDirectoryService"><ref bean="org.sakaiproject.user.api.UserDirectoryService" /></property>
    <property name="sessionManager"><ref bean="org.sakaiproject.tool.api.SessionManager" /></property>

    <property name="searchIndexBuilder"><ref bean="org.sakaiproject.search.api.SearchIndexBuilder" /></property>
  <property name="indexStorage"><ref bean="org.sakaiproject.search.index.IndexStorage" /></property>


  <property name="autoDdl"><value>${auto.ddl}</value></property>
  <property name="filter"><ref bean="searchSecurityFilterImpl" /></property>
  <property name="defaultSorter"><value>none</value></property>
  <!--
   If you want to make this search instance use a remote search server
   set the search server URL
   http://searchserver/sakai-search/searchservice
   This is probably best done in sakai.properties eg
   searchServerUrl@org.sakaiproject.search.api.SearchService=http://localhost:8080/sakai-search-tool/xmlsearch/
  -->
  <!--
  <property name="searchServerUrl" ><value></value></property>
  -->
  <!--
  For added security a shared key may be added, it must be the same on all nodes
  -->
  <!--
  <property name="sharedKey" ><value></value></property>
  -->
  <property name="luceneSorters">
    <map>
      <entry key="dateRelevanceSort"><ref bean="dateRelevanceSort"/></entry>
    </map>
  </property>
</bean>
  ....
```

Like I said the component is a bit more complex. 40 or so beans representing 4 XA transaction managers fro various parts of the index life cycle, but there is no implementation bindings between the transaction managers and its all configured using Spring. Since it needs close control over the database, it uses JDBC directly for the management of its inbound queue.

Technical Presentation

- That was Sakai:
  - The Stack
  - The Portal
  - Examples of a Service
  - Examples of a Tool
- For a User: “Introduction to Sakai”, Copland Lecture Theatre, probably just finished.

So thats Sakai, its internals. Throughout this conference there will be pointers to information and to parts of the developer community.

If you are a user, congratulations for sitting though that, I hope it was interesting. If you have teaching or research use cases, I would encourage you to contribute them to the community.

If you are a developer or designer, have questions or want to contribute or are just stuck, please just ask the community, they will respond.

And if you are going to be responsible for running sakai, and hit a problem, ask the community will help. Most of those out there, love a challenge and know what it feels like be left exposed by unresponsive support.

Traffic on the sakai lists is 24x7 by almost 365.

Thank you and Questions

Finally, thank you, and are there any questions.