Technical Report
Sakai Project

Sakai Java Framework

June 26, 2005
Version 2.0

Craig Counterman
Glenn Golden
Mark Norton
Charles Severance
Lance Speelmon

www.sakaiproject.org

1 Introduction

This document describes the Sakai Java Framework that is one possible implementation of the Sakai Abstract Architecture. This document describes particular technologies chosen for the layers identified in the Sakai Architecture that leads to the Sakai Tool Portability Profile. The Tool Portability Profile (TPP) is more detailed guidance with respect to the development of tools that will operate within the Sakai Java Framework.

It is important to note that this document is only intended to reflect the best thinking and information for a particular release of Sakai. The version of this document (see title page) is tied to a particular Sakai release. These documents should be expected to change in some possibly significant ways during the early Sakai releases. Because the early phases of the Sakai project are using legacy capabilities coming from the CHEF environment, development and tool deployment will be a mix of legacy and new Sakai capabilities throughout through Sakai releases. In each successive release, the focus will increasingly be on the non-legacy aspects of Sakai.

Users adopting early releases of Sakai should be prepared to track Sakai evolution through the public mailing lists at collab.sakaiproject.org.

1.1 Purpose of this Document

This document describes a design based on the Abstract Sakai Architecture for a Java-based framework that allows tools and services to leverage the powerful support provided by existing web technologies. In particular, it describes a suite of technologies that allow Sakai Services and Tools to be created in a manner that promotes interoperability and code portability.

1.2 Definition of Terms and Acronyms

OKI, Open Knowledge Initiative
OSID, Open Service Interface Definition
OBA, Out of Band Agreement
JSF, JavaServer Faces
JSP, JavaServer Pages
Servlets
Portlets
Web Application
JISC, Joint Information Service Committees
IMS, The IMS Global Learning Consortium
IEEE, The Institute of Electrical and Electronic Engineers
OSPI, Open Source Portfolio Initiative

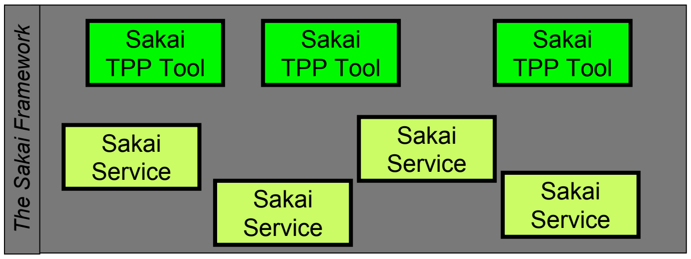
1.3 Intended Audience

The Sakai Tool Portability Profile is intended for people interested in understanding how Sakai is implemented and deployed in a Java environment. It refines the Sakai architecture by providing an example of how that architecture be realized using existing, standard Java technologies.

2 The Sakai Java Framework

The Sakai Java Framework is a particular implementation of the Abstract Sakai Environment focused on support for Sakai TPP tools written in Java operating in a web-browser environment. Many other frameworks can be built to support Sakai TPP applications and different design decisions might be made as those frameworks are developed.

The ultimate goals of the Sakai Tool Portability Profile and the Sakai Java Framework is to provide an environment where tools and the services to support those tools can be dropped in as "units of expansion" or "building blocks" as to allow an organization to assemble the componentized units of functionality together to solve their particular application problem.



The Sakai Java Framework can be thought of as a suitable "container" for Sakai TPP tools and associated services.

In addition to supporting Sakai TPP tools, the framework also supports a number of other "real-world" capabilities such as portal integration, non-TPP application integration, and others. Both the Sakai TPP environment and these real-world issues are described in this document.

There are a number of different approaches to integrating application functionality into the Sakai Java Framework.

- The Sakai TPP Tools and Services are an approach to building units of extension for Sakai that are well coordinated with the other Sakai Tools. The Sakai TTP environment provides a specific set of JSF widgets to insure consistency between these tools.

- For existing tools written in Java, or tools that must operate both within Sakai and outside of Sakai, there is a simpler form of Servlet integration where an application can access the Sakai APIs without completely converting to be a Sakai TPP tool. It is still possible, though more difficult, to match the Sakai UI and provide consistency with the other tools.

- The framework provides a "wrapper" to allow CHEF tools to be placed into the environment with only minor modifications. The Sakai environment provides services to these tools as a complete replacement to the Jetspeed portal that was used by CHEF for rendering.

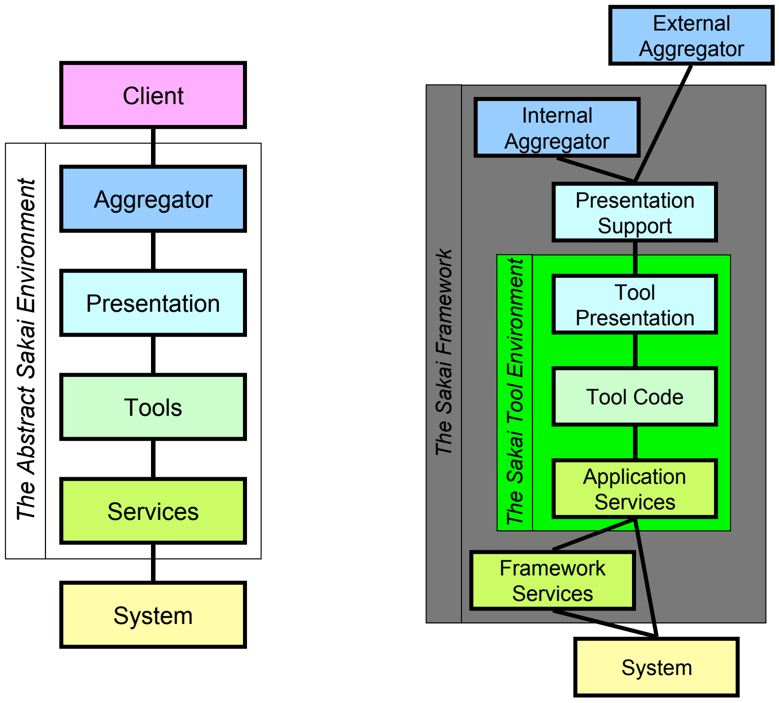
- An application can be integrated into Sakai using the IMS Tool Interoperability (IMS TI) standard. The IMS standard is expected to be final in the Fall 2005 and the feature should be delivered in the Sakai 2.1 release.

- Sakai 2.0, Sakai supports web services using Apache's Axis 1.2 fully integrated into the release. In the 2.0 releases there are no officially supported "Web Service" calls. Several example web services are provided in the release along with a PHP-based test utility. This provides a base for expansion as the community determines the requirements for web services for Sakai.

The rest of this section describes these approaches to integrating application functionality into the Sakai framework.

2.1 Inside the Sakai Java Framework

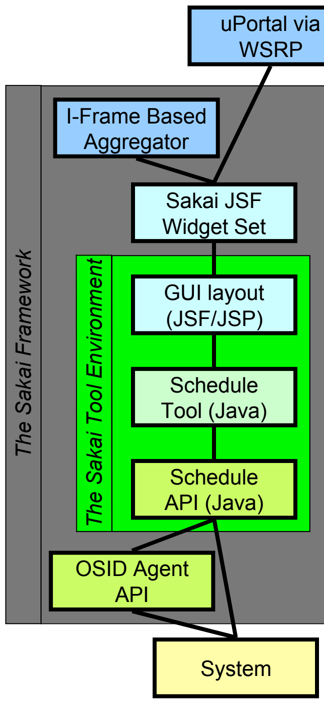
The goal of the Sakai Tool Portability Profile (TPP) is to define the interaction between a Sakai TPP Tool and the Sakai Framework. One goal is to enable portability of a tool between many different local Sakai installations. A Sakai TPP Tool is a well-formed unit of functionality (similar to a plug-in), which can be dropped into a Sakai Framework along with many other Sakai Tools to provide the overall application functionality with a consistent look and feel between tools. Sakai TPP Tools are by their nature more restrictive than general Java web applications described in the next section - particularly because presentation aspects are constrained in a Sakai TPP tool to use the presentation support provided by the framework.



A key element of the Sakai TPP contract is that it completely specifies all of the interactions that a tool will make including the interfaces to a presentation layer, Application Services, and framework services. The Sakai TPP forms a complete "perimeter" around a tool so that it truly can be "dropped" into any environment that complies with the Sakai TPP. It is important to note that the definition of the Sakai TPP is evolving throughout the Sakai project.

Within the Sakai TPP tool environment, tools are broken into three basic layers: (1) Presentation logic, (2) Tool logic, and (3) Application Services. The framework provides a set of services that can be used either by the tools or Application Services to interact with the framework as necessary. The presentation of the tool is broken into two layers. Within the tool, there is an "abstract" expression of what the GUI should look like. The rendering of the GUI is up to the framework. This paves the way for support for many presentation elements without the need to change the tool's description of the presentation layout.

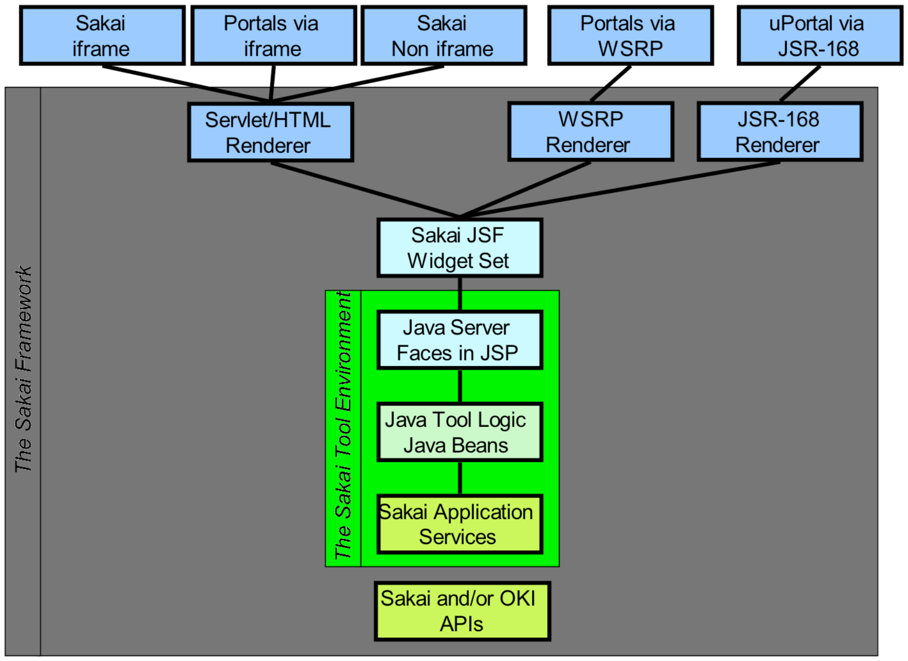
The following identifies some sample technologies for each of the abstract elements of the tool and framework environments in the Sakai Java Framework.



Within the tool environment, tool code is written in Java performing the necessary tool logic, and interacting with the presentation layer using JavaBeans. The presentation is expressed in JavaServer Faces that expresses the presentation elements using the Sakai GUI Components in the Sakai Widget Set.

Each tool will typically have one or more APIs that provide direct support for the tool when interacting with the system and framework APIs. These tool-facing APIs are designed to provide tool writers a convenient and easy-to-use interface. Often these APIs evolve as tool needs and requirements evolve. It is an important design pattern to move as much functionality as possible from the tool logic to the Application API to maximize the opportunity for the reuse of that functionality. It is also important to note that while many of the tool-facing APIs will be used primarily by one tool, there are many examples where the APIs will be used by many tools. A good example of this would be where a grade book tool would be the primary user of the grade book API but an assessment engine that needed to record grades from assessments would also use the grade book API.

As mentioned above, by separating the presentation into two layers where one layer is within the tool and the other layer is in the framework, Sakai tools can be repurposed to support multiple presentation approaches as shown below.



The goal of having the presentation express its GUI in a relatively restricted subset of JavaServer Faces is to allow for a multitude of ultimate presentation capabilities all handled transparently within the framework.

The above diagram can be thought of as a rough roadmap for Sakai presentation efforts. The initial versions of the framework will support the stand-alone aggregation and iframe integration within portals. As there are many issues with iframes in terms of web design, alternative non-iframe renders are being investigated. WSRP producer is expected in the Sakai 2.1 release and JSR-168 integration into uPortal is currently under design investigation. Beyond WSRP and JSR-168 there has been talk of a Swing-based renderer or even a Flash based renderer.

The essence of the Sakai TPP is to define how to build and deploy tools. It is naturally a constraining environment that is designed to maximize portability of the Sakai tools. Some may feel that it is too constraining and may choose to develop and integrate applications into Sakai that use mechanisms other than TPP compliance.

3 Building Tools and Services in Sakai

3.1 Integrating Web Applications into Sakai

There are many cases where the Sakai TPP is not an appropriate approach to integrating functionality into Sakai including:

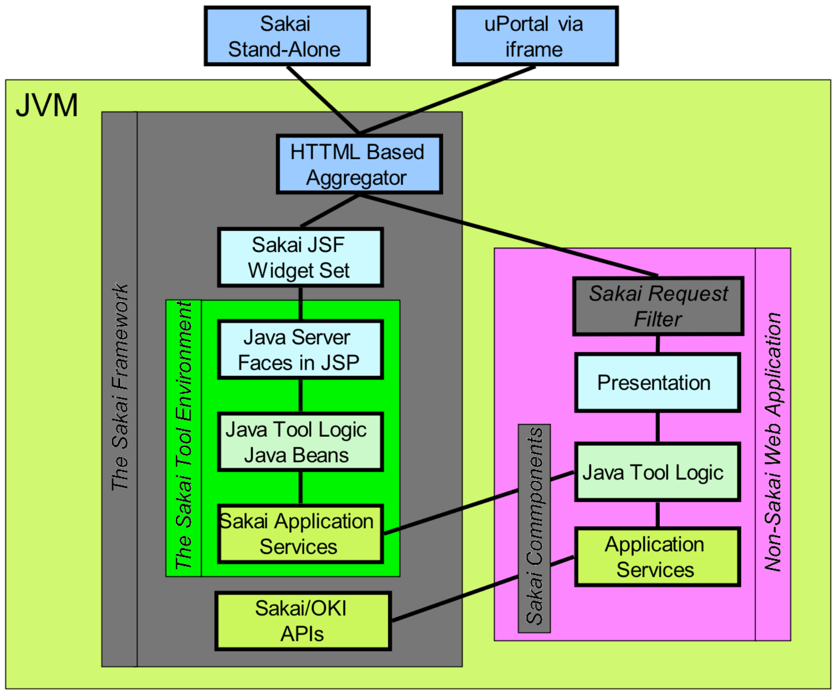
- An application that needs to operate both within Sakai and independent of Sakai
- A large application using presentation technology other than JSF, or is using JSF in ways which are not compatible.

Sakai provides a method to integrate these applications into Sakai without requiring the rewrite of the presentation aspects of the tools. Tools that are integrated in this way have full access to the Sakai Framework and Application APIs.

With the proper code structure, it can be quite natural to maintain both a Sakai and stand-alone version of a tool.

The primary problem which must be solved when bring a Servlet into Sakai is the necessary setup to insure that the Sakai APIs have access to information stored in the Servlet thread. Once the Servlet thread is properly set-up, the Java application can simply make calls to the Sakai APIs using service location or service injection to locate the proper implementations for the Sakai APIs (see later section describing service location).

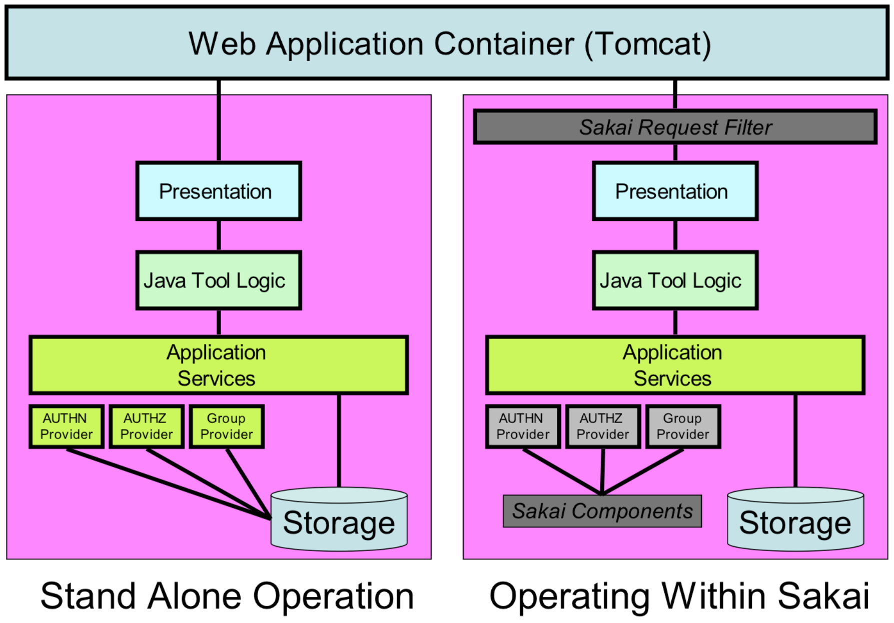
This necessary initialization has been placed in a Request filter that does the necessary work to initialize the Sakai APIs within the web application for each incoming request. Because this is done with a filter, there is no need to modify the application - the change is to add a **filter** entry to the **web.xml** for the Servlet.



Given that the presentation elements are "within" the tool, the application is completely responsible for its look and feel and any compliance to the Sakai Style Guide, conformance with accessibility rules, and other presentation elements which a Sakai TPP tool delegates to the framework.

If the application is written in JSF, it may be able to make use of the Sakai Tag Library so as to more naturally align with the Sakai Style Guide.

Often the application needs to maintain both a "stand-alone" version and a version that works within Sakai. A useful design pattern to solve this problem is for the application to decompose its internal functionality into a set of "providers" which can be plugged into their application services.



When the application is operating as a stand-alone web application, it uses one set of providers for those services that simply stores the appropriate data in the application's local storage. When the application is moved into Sakai it is configured to use a different set of providers for those internal interfaces which make calls to the Sakai services as appropriate.

This way any Sakai-specific code is isolated into these providers rather than being sprinkled throughout the application services and tool logic.

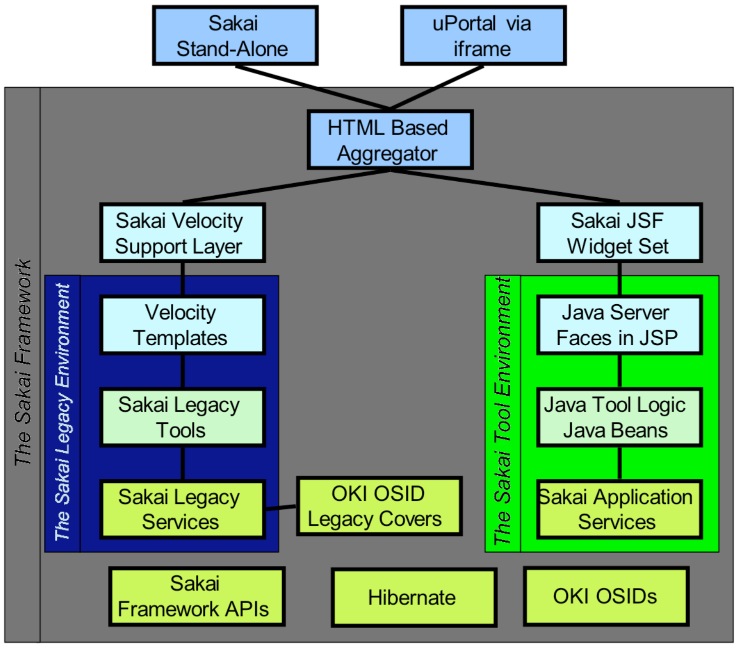
Another approach to providing a standalone version of a Sakai TPP tool is to provide a completely reduced version of the Sakai framework and portal that supports a single tool. This way a tool can be written to take advantage of the Sakai framework, but at the same time run stand-alone on a separate server with no other tools. This effort is under active consideration for a future version of Sakai.

3.2 Legacy Tool Support in Sakai

Much of the collaborative functionality of Sakai Collaborative and Learning Environment is provided by tools that were adapted from CHEF. These tools were originally written to operate in the Jetspeed portal and used the Velocity template engine. These tools were brought into Sakai by producing a layer that provided basic implementations of needed Velocity and Jetspeed APIs but which interacted with the Sakai framework rather than Jetspeed.

To ease the porting of these tools, a legacy framework and set of services was developed and is supported within Sakai in addition to the TPP compliant tools. These two frameworks operate together in the same environment. The aggregator can display Sakai legacy tools and Sakai TPP tools at the same time.

In addition, the framework and application services are usable across both the legacy and non-legacy environments. This allows an evolutionary approach to tool and service development. The legacy services are quite complete in version while the new-generation of Sakai APIs and OKI OSIDs will be developed and used in parallel with the legacy capabilities in Sakai version 2.0 and beyond.



The Sakai legacy environment does not include the Jetspeed portal - the Sakai legacy environment was created by producing a set of proxy implementations for much of the Jetspeed APIs needed by the legacy tools. These Jetspeed APIs are implemented so as to talk to the Sakai Framework.

During development of new tools and capabilities there may be a need to use a combination of Sakai legacy framework capabilities and the new Sakai APIs to accomplish the needed tasks.

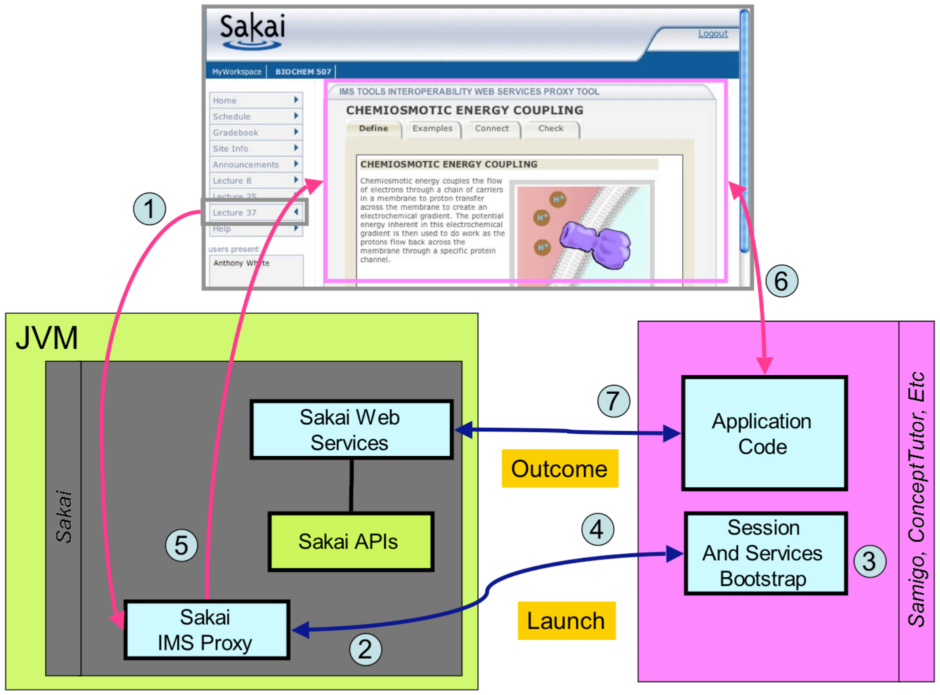
3.3 IMS Tool Interoperability (IMS TI) Support

The IMS Tool Interoperability Specification (IMS TI) can be used to integrate a wide variety of applications into Sakai. These tools are hosted externally from Sakai and communication with these tools is done using web services as specified in the IMS TI specification. These externally hosted applications can be written in any language that supports SOAP-style web services. As of the Sakai 2.0 release, the IMS TI specification is still in draft form although its capabilities have been demonstrated to work with Sakai, Blackboard, WebCT, and Moodle based on a draft of the specification.

The IMS TI uses a blend of web services and iframes in browsers to perform tool integration. The IMS TI consists of two web-services requests:

- The IMS TI Launch Request is a message sent from Sakai to the externally hosted application to establish the identity of the user, the current course/site, role within that site, and session information.
- The IMS TI Outcome Request is sent back from the externally hosted application to Sakai to produce a grade, or other form of outcome to be stored inside of Sakai as appropriate. The Outcome Request is not required

The following diagram describes the steps that take place to orchestrate the operation of the externally hosted application.



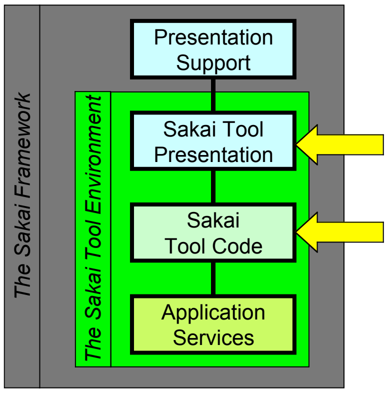
There are a number of steps in the process between the point when the user launches the tool and when the tool interaction is complete.

1. First the user selects the externally hosted tool by pressing a button.
2. The Sakai IMS TI Proxy Tool intercepts the incoming request and sends an IMS TI Launch Request the external application via web services. The IMS TI Launch Request control communicates the user identity, session information, contextual information, and a web-services handle to get back to the Sakai web service gateway.
3. The external web application sets up any session information for the user, records any necessary information about the user and the web services handle.
4. The external application returns an IMS Launch Response that includes a launch URL determined by the external application.
5. The Sakai IMS TI Proxy Tool starts the application by placing the URL returned in the IMS TI Launch Response in a frame.
6. The user then interacts with the application within the iframe using their browser.
7. When the external application needs access to information such as user roles, user information, or other information, Web-Services are used to retrieve or store information from Sakai. Currently the only web services defined for this is the IMS TI Outcome Request which is a way to notify the CLE that some activity has been completed or perhaps a score for the activity.

The IMS TI specification is still in draft form as of Sakai 2.0 so there may be some changes in this approach as the specification is finalized.

3.4 Building a Sakai-Specific Tool

This section describes how the presentation is separated from the tool logic within the Sakai TPP Tool environment.



The GUI layout is described in JavaServer Faces using a set of Sakai-provided UI Components.

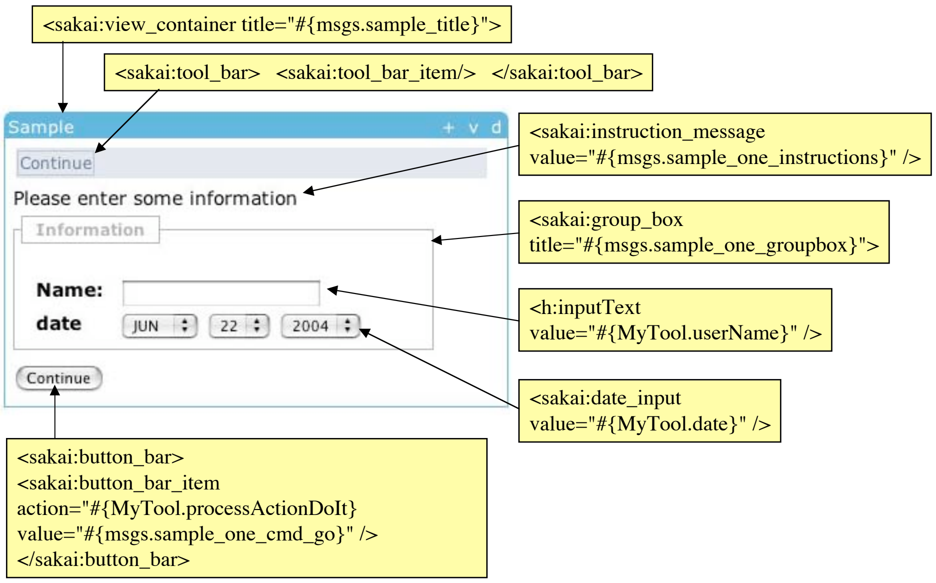
An additional, very important aspect of the Sakai UI Components is to provide for the ability to render the Sakai interface in a number of accessible formats. The Sakai Presentation Support will be informed by the user's accessibility profile and transparently render the Sakai UI components appropriately based on the user's profile. By delegating this to the Sakai Presentation Support we can insure that all Sakai TPP compliant tools are uniformly accessible without placing accessibility oriented code separately into every tool.

The following Sakai GUI Elements are initially defined for the Sakai 1.0 release as a starting point:

- ButtonBar
- ButtonBarItem
- Comment
- DateInput
- DateOutput
- DocProperties
- DocSection
- DocSectionTitle
- FlatList
- GroupBox
- InstructionMessage
- PanelEdit
- Toolbar
- ToolbarItem
- ToolbarMessage
- ToolbarSpacer
- ViewContainer
- ViewContent

This set of widgets was developed for the transition from Sakai Legacy Tools to JSF. As part of the 2.0 development effort, these widgets will be revised and extended so as to implement the functionality put forth in the style guide and required as new tools are developed for the 2.0 Sakai release.

These UI components are assembled together in a JSP description of the interface layout.



```
<sakai:view_container title="#{msgs.sample_title}">
<sakai:tool_bar>  <sakai:tool_bar_item/>  </sakai:tool_bar>
<sakai:instruction_message
value="#{msgs.sample_one_instructions}" />
<sakai:group_box
title="#{msgs.sample_one_groupbox}">
<h:inputText
value="#{MyTool.userName}" />
<sakai:date_input
value="#{MyTool.date}" />
<sakai:button_bar>
<sakai:button_bar_item
action="#{MyTool.processActionDoIt}
value="#{msgs.sample_one_cmd_go}" />
</sakai:button_bar>
```

The values that are used to populate the UI Components come from JavaBeans that are associated with the tool session.

For areas of the user interface where there is some action to be taken, the GUI layout specifies methods to be called in the Tool Java code when a particular action is indicated. In the example below the **processActionDoIt** is a JavaBean method which is called when the "Continue" button is pressed.

In the 2.0 release of Sakai, the Sakai Widget set is known to be incomplete and will need to be evolved as part of the later Sakai releases. The pattern of using the widgets to present GUI elements requires new widgets to be developed to support new capabilities. While many developers are used to doing this more directly in the tools, taking the approach of extending the widgets insures consistency between tools and insures that accessibility can be addressed within the widgets rather than being spread out into the tools.

4 The Sakai Kernel

The Sakai 2.0 release features a completely rewritten kernel. The kernel is a relatively small amount of code that isolates the areas of Sakai that may need modification when moving between different servlet environments. The Kernel establishes a contract through interfaces with the tools, services, renderers and other elements that make up the system. The Sakai kernel does no go above the "servlet" level - the kernel interacts with each servlet in the same whether a servlet is running JSF, Sakai/JSF, Struts, or doing direct Response.write operations. This allows the kernel to be used with a far wider range of application types.

The Sakai kernel addresses the following areas:

- Components - interaction with Spring to register/retrieve the Sakai API implementations with class-loader isolation between components
- Session - Provides consistent sessions for Sakai tools that work properly in a cross-webapp dispatch environment. Sakai supports several different session capabilities including: httpSession - shared Sakai-wide for user/login, sakaiSession - shared Sakai-wide for user/login, and sakaiToolSession - scoped by user/login/placement.
- Tool registry allowing each tool to be registered and discovered within the Kernel - including support for "helpers"
- Identity of current logged in user
- Various utilities including thread local support

The Sakai kernel is designed so as to place as few restrictions on a generic servlet as possible. There are a number of documents describing how to interact with the Sakai Kernel at the Sakai web site collab.sakaiproject.org in the Sakai Development site including:

- Sakai Tool Model
- Sakai Sessions
- Sakai Request Flow
- Sakai Mercury Portal
- Sakai Use of Maven
- Sakai Configuration
- Sakai Charon Portal
- Sakai Component Model
- Sakai Authentication

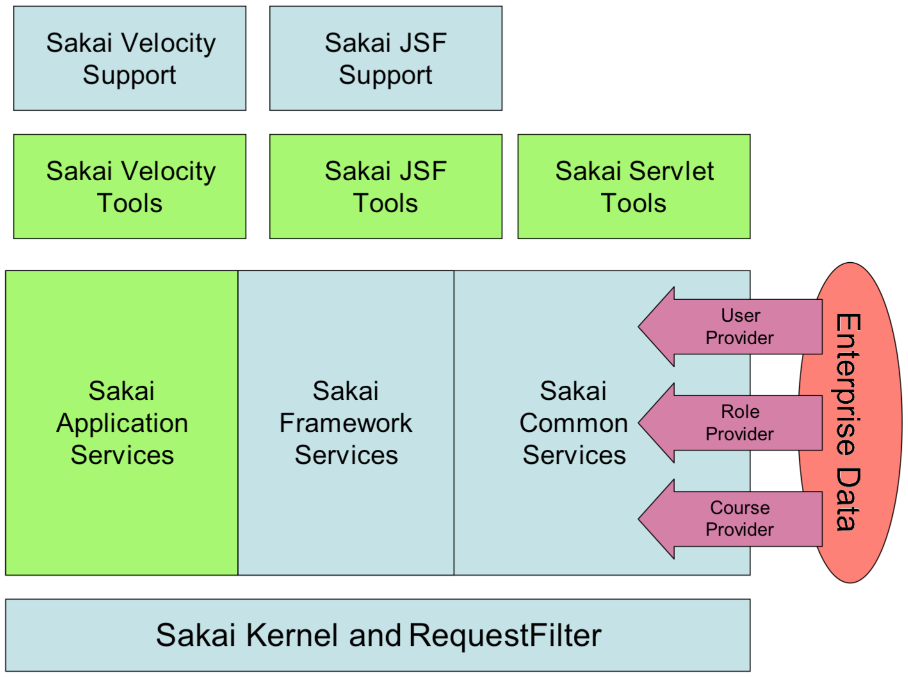
These documents provide more detail than this document.

The discussions around the evolution of the Sakai Kernel will be done in the context of the Sakai Framework Working Group.

5 Security and Authentication in Sakai

The information in this section is being reviewed for possible change beyond the 2.0 release. This section describes the state of security and authentication in Sakai for the 2.0 release.

Sakai provides a number of capabilities that each site can use to configure Sakai to their Security environment. There are basically thee APIs that Sakai can be configured to consult to provide Enterprise data to Sakai as appropriate.



The three APIs include

- **UserDirectoryProvider** - This provides Sakai with identity, authentication, and basic user directory information.

- **CourseProvider** - This provides the list of Courses (Sites) that each user is a member of.

- **RoleProvider** - This provider Sakai with which role each use has within each site.

The UserDirectoryProvider is a well-developed interface and has been in Sakai since the 1.0 release to the point where a number of implementations have been built by the community. Some of these implementations are in the Sakai 2.0 release under the **providers** directory in the source three.

The currently available providers include:

- OpenLDAP
- JLDAP
- Kerberos

Activating these providers is relatively simple - each comes with a sample configuration file and simple instructions for activation.

An additional provider is provided in the distribution based on the IMS Enterprise specification. This provider simply looks in a set of pre-defined tables or views for the information needed by Sakai. This is not yet mature and development will continue on this provider with community involvement.

The Realm and Course providers were added in the 1.5 release and have not been as heavily used as the UserDirectoryProvider. The community is working on developing versions of these providers as well as part of their system integration efforts.

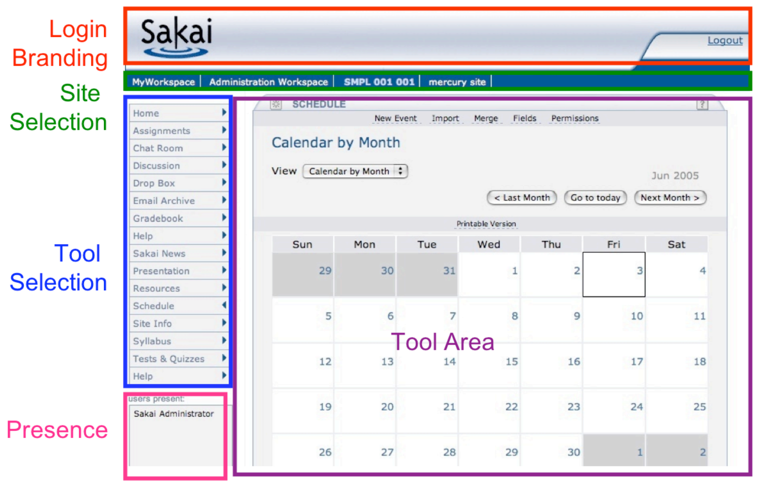
In addition to using these providers for information, Sakai can be configured to trust container authorization. In this scenario, Sakai is protected using some WebISO (Initial Sign On) technology such as CAS and Sakai is configured to accept the identity form CAS.

Even in a WebISO environment, it may be necessary to use a UsrDirectoryProvider to provide basic directory information or to resolve ID/PW pairs needed to support WebDAV and some web-service applications which are not capable of delegating authentication to WebISO capabilities.

Much of the emerging design, requirements, implementation, and evolution in this area will be done within the Sakai Enterprise Integration Working Group.

6 Sakai Internal Aggregator

When Sakai is operating as a stand-alone web application, it uses an internal aggregator to render the user view of their pages. The internal aggregator is simply a template that assembles the different elements together for each user request. Sakai has a pluggable architecture to render the ultimate end-user screen.



The internal aggregator template makes use of the following pieces of information:

- Whether or not the user is logged in - this is used to render the login and branding area.
- The list of subscribed sites for the user - this is used to populate the site selection area.
- Which site is currently selected - this is used to highlight the site in the site selection area and determines the tools that are listed in the tool selection area.
- Which tool is selected - highlights the selected and determines what to place in the tool area.
- Presence - indicates which users are in the site at the same time - this is an optional feature which can be enabled, depending on the desires and needs of the site.

The tool area is actually a panel that can contain a number of different tools. In the example shown above, there is a single tool. In the "Home" tool, you will often see a number of tools (perhaps even two columns of tools) displayed in synoptic views.

The template that renders the portal can be edited locally to change look and feel. In successive releases (post 1.0) the rendering for the internal aggregator may change in significant ways. Users who customize this aspect of Sakai should expect that any look-and-feel changes would require some porting effort as new releases come out.

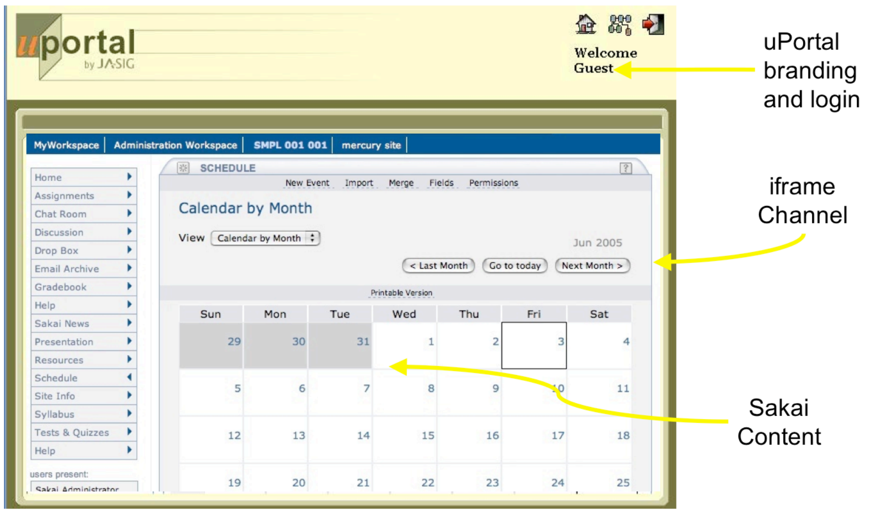
6.1 Integrating Sakai into other Environments

There are a number of approaches in progress with respect to integrating Sakai into a variety of portals including uPortal. WSRP and JSR-168 are not yet supported in Sakai 2.0. Since Sakai 1.5 there has been support for using Sakai within a portal using a simple URL mapping scheme. This approach will work with virtually any portal.

Sakai can produce any "rectangular subset" of the Sakai page.

- The Sakai internal aggregator will produce a reduced portal page that does not include the top segment (branding, login, and logout).
- The portal administrator adds a reference to the portal using an channel capable of describing a URL.
- Optionally, Sakai and the portal use the same external single-on mechanism such as CAS (http://www.yale.edu/tp/auth/cas20.html)

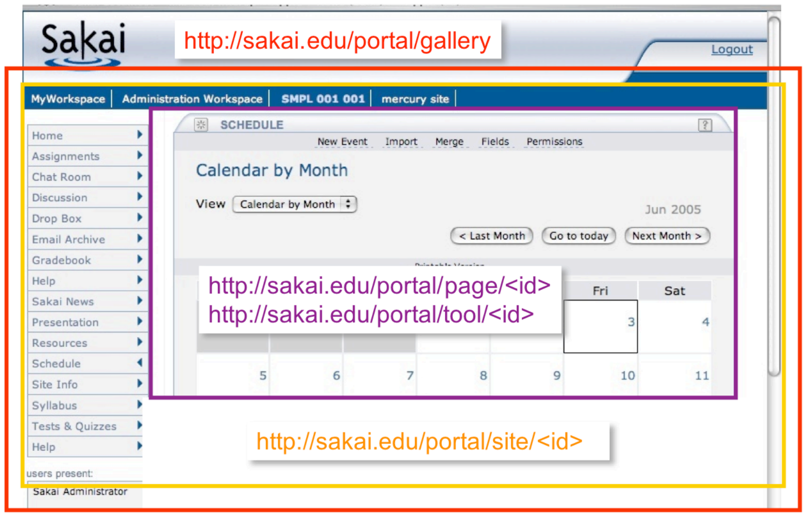
This allows a seamless integration between a portal and Sakai. The top navigation, branding, log in and log out are all handled by the portal.



The aggregator provides direct URLs to several levels of the site:

- The **/portal** URL provides all of the elements of the Sakai aggregator (top branding, site list, page list within site, and currently selected page/tool)
- The **/portal/gallery** URL provides all the elements of the Sakai aggregator except the top branding.
- The **/portal/site** URL provides a single site with the site's page list, and currently selected page/tool
- The **/portal/page** URL provides a view of a single page with a particular site
- If a page consists of more than one tool, the **/tool** URL can access a particular tool placement within the site.

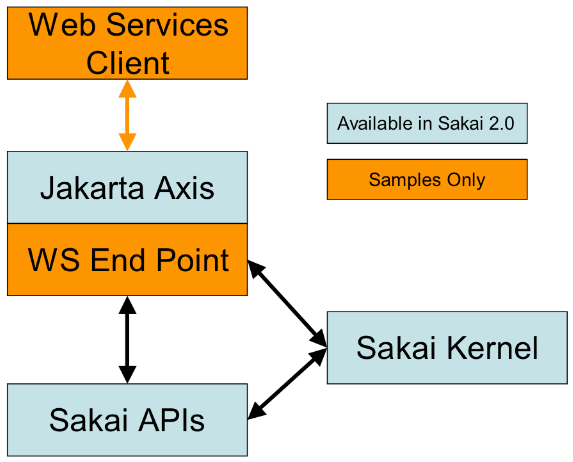
The figure below shows some sample URLs and the subset of the portal presented by the URLs. Once the URLs scope drops below the /portal/gallery URL, a particular, site, page, and/or tool id must be specified for rendering on the URL as there is no navigation presented in these finer-grained views.



For more detail on the URL mapping see the "Charon Portal" document in the Sakai Development site at collab.sakaiproject.org.

7 Sakai Web Services

Sakai version 2.0 has support for web services. There are no web services APIs or protocols defined in the 2.0 release. Axis 1.2 is fully integrated into the Sakai 2.0 release including support for the new Sakai 2.0 kernel.



This web services capability provides the ability to build applications in PHP, Python, and others and still interoperate with Sakai. It is important to note that the development of the web services endpoints and the clients will likely take some significant engineering. All of the Sakai APIs are available to a web service end point. However it is not logical to simply "wrap" those APIs in web-services.

A more fruitful approach will be to establish a more "document" oriented set of Web-Service "APIs" - these Web Service APIs will likely make many calls to the underlying Sakai APIs in a single web service request and return the results of the multiple API operations.

The key goal is to minimize the number of web service exchanges per online transaction so to provide good performance for applications utilizing web services.

A sample web service endpoint to log a user in over web services and establish a session is as follows:

```
public class SakaiLogin {
  public String login(String id,String pw) {
    User user = UserDirectoryService.authenticate(id,pw);
    if ( user != null ) {
      Session s = SessionManager.startSession();
      if (s == null)
      {
         System.out.println("no session established");
         return "Session Null";
      }
      else
      {
         s.setUserId(id);
         s.setUserEid(id);
         System.out.println("session: " + s.getId()
           + " user id: " + s.getUserId()
           + " user enterprise id: " + s.getUserEid());
         return s.getId();
      }
    }
  return "User Null";
  }
}
```

This end point uses the "JWS" approach to add the new web service to the Axis installed in Sakai.

Many clients can communicate with these web services. Samples have already been written in PHP and Flash. Here is a sample code snippet from PHP to communicate with the above web service using SOAP from the Pear Project:

```
require_once('SOAP/Client.php');

if ( ! $_POST['url'] ) $_POST['url'] =
   "http://nightly2.sakaiproject.org/sakai-axis/";

if ( $_POST['login'] ) {
  $site_url = $_POST['url'] . 'SakaiLogin.jws?wsdl';
  echo ("Loggging in to Sakai Web Services at ".$site_url);
  $wsdl=new SOAP_WSDL($site_url);

  // Create an object directly from the proxy code
  $myProxy=$wsdl->getProxy();

  $session=$myProxy->login("admin","admin");

  echo ("Session:");
  print_r ($session );
  $_POST['session'] = $session;
}
```

With the 2.0 a number of groups in the Sakai development community can begin the development of a set of web service capabilities based on the needs and requirements in the community. Much of the new engineering effort in this area will be coordinated through the Sakai Cross-Language Working Group.

8 Sakai APIs and the OKI OSIDs

One of the essential deliverables of the Sakai project is a set of OKI OSID implementations and associated out-of-band agreements for those implementations.

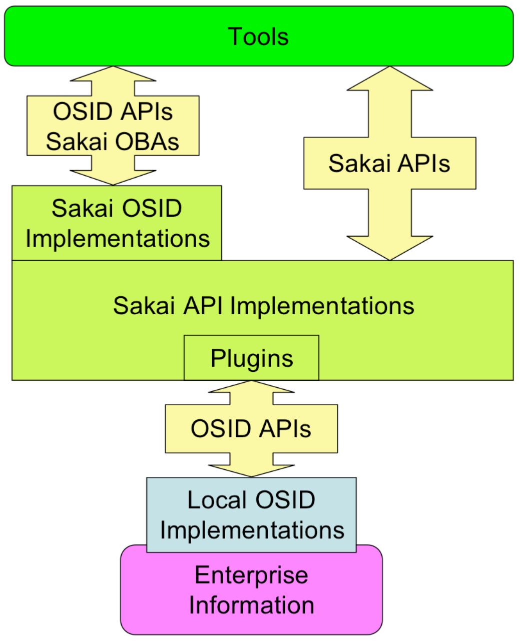
The OKI OSIDs impact the Sakai project in the following ways:

- A set of OKI OSID implementations will be developed that provide access to Sakai API functionality. A tool could be written to use either these OSID implementations or the Sakai APIs (or a mixture of the two) to get the same result. A document describing the out of band agreements for these OSID implementations will also be produced.

- The Sakai APIs that are developed for use by the Sakai tools are being designed based on the principles of the OKI OSIDs and the OKI OSID data model. The APIs are being developed so as to line up precisely with their corresponding OKI OSIDs. For example, Sakai and OKI both have APIs for: Agent, Authorization, Repository, and many others. The APIs will be identically named and method calls will be identically named as well where they overlap.

- Sakai provides a rich plug-in capability to allow local institutions to provide Sakai with enterprise information such as identity, or course enrollment. More plug-ins are being provided with each successive release of Sakai. These plug-ins will be developed so as to accept OSID implantations as the providers of the enterprise data.

A figure of the relationship between the Sakai APIs and the OKI OSIDs is shown below.



9 Installing and Running Sakai

9.1 Hardware/Software Requirements

The Sakai software is intended to work on a wide range of hardware and operating systems that support Java. This section will describe the typical environments used within the Sakai project for both the production and developer environments.

Sakai developers typically use one of the following environments

Operating System: Windows-XP, Macintosh OS/X, Solaris, or Linux
Processor: PowerPC 800Mhz or higher, Pentium 3Ghz or higher
Memory: 512 MB or higher (1GB recommended)
Software: JAVA 1.4 or later and Jakarta Maven build environment
Optional software: Eclipse

The typical production environment:

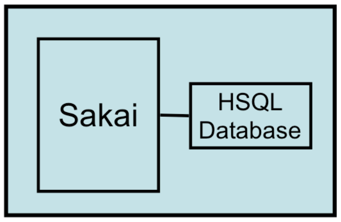
Operating System: Linux or Solaris, or Mac OS/X Server
Processor: Intel-Based
Memory: >= 4GB
Software: Java 1.4

Users are welcome to use other environments, but by sticking with the common environment used by the Sakai developers and the production Sakai systems, it allows users to take advantage of the testing and QA that is done by the developers and other institutions running Sakai in production.

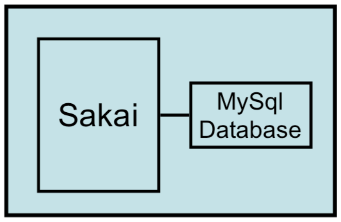
9.2 Sakai Target Environments

Sakai is intended to operate in a number of different environments ranging from a developer's desktop through a scalable production environment.

The developer environment is designed to work without a network connection and is configured to store all data in Hypersonic SQL. Hypersonic SQL is included in the release distribution to simplify startup. Hypersonic can either be configured to store data in-memory or in a file. When HSQL is storing data in memory, each Tomcat restart results in a fresh database. When HSQL is storing data to disk, changes are persisted across Tomcat restarts.



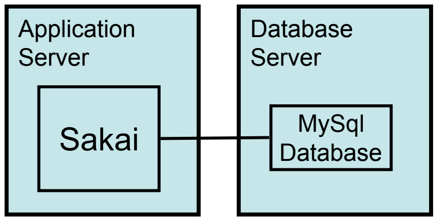
Developer Environment



Small Production Environment

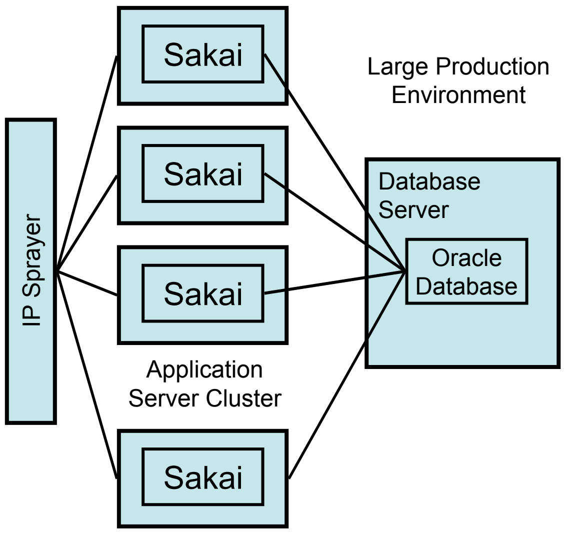
It is a short step to move from the developer environment to a small production environment. The small production environment is reconfigured so as to use a MySql database by changing Sakai properties values and installing the MySQL JDBC jar. This environment is intended for a relatively small population of users (<200) but allows a small server to be set up using a single piece of hardware - perhaps a commodity single-CPU system with 4GB of RAM. While backup and security are important issues, it is conceivable that a clever instructor could set up their own Sakai system to support a single class or few research groups.

For a medium sized institution with < 2000 users, a configuration with a single multi-processor application server and separate MySql (or Oracle) database server is the expected configuration. This would likely be run in a professionally managed central computing facility as an enterprise service.



Medium Sized Production Environment

For a large production environment, it becomes necessary to cluster multiple application servers to provide the necessary performance. The common deployment in a clustered environment is to use Oracle. Clustering is supported using MySql, but because there are no large production environments (as of the writing of this document) that use MySql, there is less testing and QA of the MySQL clustered version.



To use the clustered environment, the servers must be accessed through some type of IP sprayer that maintains "sticky" sessions. When a user first associates with a particular application server their incoming requests must always be routed back to the same service so that session information is maintained. Several options exist to solve this problem ranging from dedicated hardware sprayers to specifically configured Apache servers using the mod_jk connector.

Perhaps the most important aspect of running Sakai in production is to be part of the Sakai Development list at collab.sakaiproject.org. This list is the quickest way to keep up-to-date on bugs and issues as well as getting answers to your questions.

10 Conclusion

The Sakai Java framework provides capabilities to deploy tools and services in a collaborative learning environment. There are a number of different levels of integration between a tool and the Sakai Java Framework. The Sakai Tool Portability Profile describes a Sakai-specific unit of expansion that constrains developers but produces tools with a very uniform look and feel and flexibility in rendering technologies. Sakai also provides a mechanism to integrate tools that already exist without major re-write. In this integration, a tool adds a Request filter and can then use the Sakai APIs to access Sakai information such as Agents, Authorizations, or other information.

By allowing multiple approaches developers can choose how to integrate their particular application into Sakai. So far the pattern has been to build new small tools as TPP compliant tools (Sakai Syllabus Tool or Sakai Profile Tool) while larger separately developed applications (The SAMigo assessment engine, Open Source Portfolio, and the Berkeley Gradebook) have chosen to integrate as Java applications.

There is a clear understanding that the TPP capabilities present in any release will need to evolve as the tools are developed and new requirements are identified.

The Sakai Java Framework provides both a production environment and developer environment and is a blend of old technologies with new and evolving technologies. Between releases 1.0, 2.0 and 3.0 there will be a shift in emphasis from the legacy capabilities to the newer elements of the framework. The framework has been designed to allow smooth transitions for a wide variety of applications into Sakai.

11 List of Contributors

The following individuals contributed to the development of this document:

Craig Counterman	Massachusetts Institute of Technology
Glenn Golden	University of Michigan
Rachel Golub	Stanford University
Mark Norton	Sakai
Charles Severance	University of Michigan
Lance Speelmon	Indiana University

12 Revision History

Version No.	**Release Date**	**Comments**
1.0	October 14, 2004	Developed from earlier documents for the Sakai 1.0 release.
1.0	October 24, 2004	Added diagram to the "Non-Java" section. Typos.
1.1	December 2004	Modifications as the framework transitions from 1.0 to 1.5.
1.5	March 2005	Modifications to reflect the Sakai 1.5 release.
2.0	June 2005	Modifications to reflect the Sakai 2.0 release.