# Technical Report
# Sakai Project

## The Sakai Technology Portability Profile

## March 30, 2004

Craig Counterman
Glenn Golden
Rachel Gollub
Mark Norton
Charles Severance
Lance Speelmon

Comments to: csev@umich.edu
www.sakaiproject.org

## Introduction

The Sakai project is producing an extensible open-source learning management system that provides and uses a complete implementation of the OKI OSID standard interfaces for LMS portability. In addition Sakai will deploy the components of its learning management system using the uPortal enterprise portal technology.

It is important to understand the distinct components that the Sakai project will produce. While they are related, there are important distinctions between the components:

- The Sakai Tool and Service Portability Profile describes standards, techniques, and technologies that will allow developers to create tools and services that can be deployed within any Sakai compliant framework.

- The Sakai Reference Framework is a software environment based on uPortal that implements the Sakai Tool and Service Portability Profile and provides an environment that supports the deployment of Sakai TSPP compliant tools and services.

- The Sakai Collaborative/Learning System - which is a set of TSPP-Compliant tools that can be used in various combinations to deploy a learning management system, enterprise portal, small group collaboration environment, or collaborative problem-solving environment.

In addition, because there is a wide range of existing enterprise portal-oriented tools that are already available for the uPortal technology, the Sakai Learning Management System can be deployed within the context of an enterprise portal. For organizations that do not want to deploy an enterprise portal, Sakai will install out of the box as a stand-alone learning management system.

Our Vision:

> *Sakai will create an open-source learning management system, but at the same time create a framework, market, clearinghouse, cadre of skilled programmers, and set of documentation necessary to enable many organizations to focus their energy on developing capabilities/tools which advance the pedagogy and effectiveness of technology-enhanced teaching, learning, and collaboration rather than just building or deploying another variant on a threaded discussion tool as an LMS.*

This document describes the Sakai Tool and Service Portability Profile and how to write Sakai compliant tools and services.

The elements of the TSPP are selected to ensure the long-term value, portability, and interoperability of the tools and services that are developed using the TSPP.

One of the goals is to keep the TSPP relatively simple, and to limit the explicit dependencies of the tools and services to as few as possible with the idea that every additional explicit dependency is a possible constraint on portability.

## Standards

One of the essential elements of a profile is to select a set of standards, and add guidance, select options and define best practices around those services sufficient to insure that portable and interoperable code can be developed using the profile.

The standards that form the foundation of the Sakai Tool and Service Portability Profile include:

**JavaServer Faces** - In the early releases of Sakai, these will be stored in JSP files. It is important to note that the specification for a Sakai view is the XML representation of the JSF layout with no other JSP in the file - no HTML, no Java. In the future, Sakai may convert to using XML files, instead of JSP files, to declare the views. To be TSPP compliant, the JSF document must only use the approved JSF HTML, JSF core, and Sakai tag sets.

**OKI OSIDs** - These APIs provide an integration layer that ensures portability of the tools and services across any environment that provides implementations for the OKI OSIDs. In addition, the OKI OSIDs provide interfaces where local implementations of the OSIDs can be developed to integrate OKI compliant tools and services into the local environment.

The Sakai Project will define its own internal standards that are part of the Portability Profile - these standards will build upon and add detail to the OKI and JSF Standards to define their use within the Sakai Framework.

**Sakai GUI Elements** - Sakai will define additional JSF tags based on the Sakai Tool Development Style Guide. By using these elements, tool developers will automatically comply with the Sakai Tool Style Guide. These elements also insure a uniform look and feel across Sakai tools developed by different developers. It is also important to note that the Sakai mechanisms are in place to ensure that tools have a consistent look and feel that is under the control of the deploying institutions. The goal of the Sakai GUI elements is to provide for nearly all of the GUI requirements of the Sakai tools. The GUI elements will be extended as new tool requirements are identified. The Sakai GUI elements will be built to support display in multiple languages and multiple formats including

ADA/508 complaint formats. By using the Sakai GUI elements, tools take advantage of these capabilities implicity.

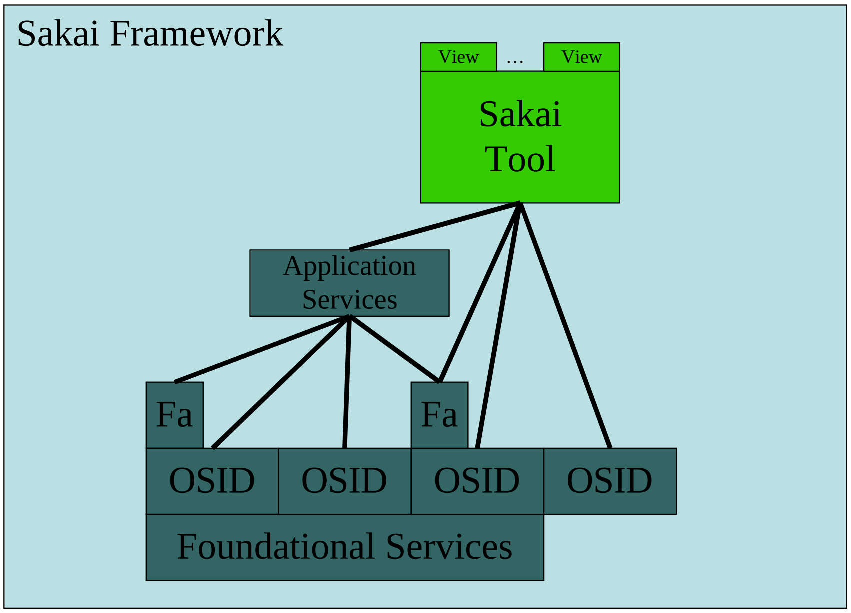
**Sakai/OKI Façade Services** - These will be new interfaces and implementations developed by the Sakai Project primarily to add a semantic layer on top of the OKI OSID interfaces and add schema-specific semantics to the OKI OSIDs which are designed to support a wide range of schemas. The purpose of the façade services is to provide a mechanism to insure interoperability in terms of the data used by Sakai tools using the OKI OSIDs to store and retrieve data. These Façade services will act as a layer on top of the OKI OSIDs - in order to move the Sakai tools to a different OKI compliant environment, one must also bring the façade services as well. The intent for the façade services is to be relatively "thin" - their purpose is not to capture business logic but instead to add a schema and semantic model and explicit typing to abstract data objects stored using the OKI OSIDs. The application services described below are the proper place to capture business logic.

In addition, another set of services will be developed which fall outside the Portability Profile as they are more related to the Sakai-produced tools and services which will be developed for deployment within the framework rather than being part of the framework:

**Sakai Application Services** - These will be new interfaces developed by the Sakai Project as part of the development of the tools that make up the Sakai Collaborative Learning Toolkit - The application classes will evolve as the needs of tools expand. The goal is to limit the amount of code that is place in the tools with respect to interacting with the storage services. The application Services are a convenient way to implement functionality once and use it across multiple tools.

**Sakai Foundational Services** - This a set of interfaces developed for the specific purpose of supporting the storage needs of the OSID implementations. The goal is to provide an abstract general-purpose storage layer which handles object-to-relational mapping, multi-system caching, scaling, and high-performance access. Not all OSID implementations will use the foundational services - sites can develop their own OSID implementations that replace the Sakai OSID implementations for particular services. These locally developed services may completely ignore the Sakai foundation services.

As the figure below shows, while there is a logical layering in terms of the general purpose of each of the types of interfaces, much of the layering is neither required nor strict:



A tool can talk to any of the interfaces from the application directly to the OKI OSIDs. As a matter of fact, it is quite reasonable for some tools not to use application or façade classes at all and to instead go directly to the OKI OSIDs. The Sakai framework must support this direct access to OSIDs to accommodate tools that are developed to comply with the OSIDs but not initially developed in the Sakai environment.

Similarly an application service can use OSIDs directly or work through the façade classes.

Façade classes only communicate with a single OSID and have a one-to-one mapping with their underlying OSIDs.

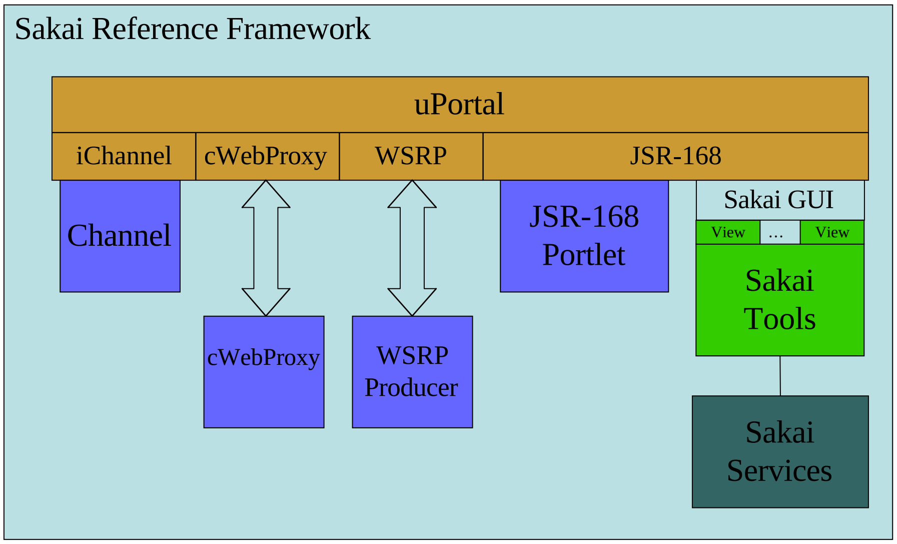
Foundational services should only be used by the high-performance enterprise implementations of the Sakai OSIDs. Tools and application services should use the OSIDs for their storage needs rather than calling the foundation services. If developers begin to develop or modify an enterprise OSID implementation, then they will interact with the foundation services.

## The Sakai Reference Framework

The Sakai Project is producing a reference implementation of a Sakai TSPP compliant framework that will support the deployment of Sakai compliant tools and services. The reference framework will target the web browser/portal environment. It is important that we separate the definition of the TSPP from the reference framework that is delivered that implements the TSPP. Just because a piece of software is used as part of the framework (say a particular web-services implementation) it does not imply that by including that software component, the

TSPP is automatically extended. The TSPP documents are the definition of the TSPP contract, not the Sakai Reference Framework.

The following figure shows the implementation of the Sakai reference framework:



By using uPortal as the foundation of the framework, the Sakai framework will support all of the popular portal standards in the market today. A collaborative system can make use of the many sources of existing and in-development portlets implemented using the JSR-168, iChannel, cWebProxy, or Web Services for Remote Portals (WSRP) to produce an integrated enterprise portal and collaborative environment. To further aid in producing a unified look and feel between TSPP tools and these other ways to implement tools, CSS class definitions will be aligned between JSR-168, uPortal, and the Sakai TSPP, adopting the JSR-168 CSS elements wherever possible.

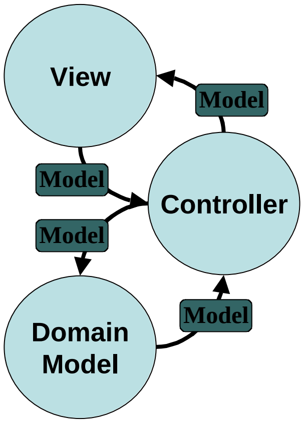
In addition, because the Sakai Tools are ultimately rendered through JSR-168, the framework provides a path forward to making use of Sakai tools in JSR-168 compliant environments other than uPortal. However, in the short term, there will be close integration between the Sakai service implementations and uPortal in the areas of authentication, layout, and configuration, leading to a smooth end-user experience.

When a developer is considering the development of a new tool, any of the options described above can be chosen. However if the tool is intended to interoperate closely with the rest of the tools being developed by the Sakai project then the tool should be developed using the Sakai TSPP standards.

## Design Patterns

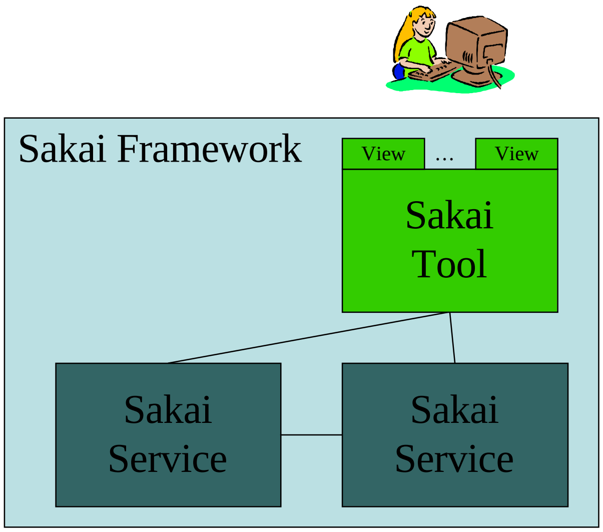
The Sakai TSPP embraces and extends a number of basic design patterns. Much of the TSPP design is built around these design patterns as founding principles.

**Model-View-Controller** - In Sakai parlance, the Service is the persistent aspect of the system (often called the Domain Model). The tool presentation views expressed in JavaServer Faces make up the View, and the Controller is the tool logic. Strictly speaking the Model (as compared to the Domain Model) is the data that is handed from the controller to the View. In Sakai this is a Java bean that contains the information retrieved from the Service (Domain Model) plus any needed decoration elements.



**Programming to interfaces rather than implementations** - It is up to the framework to provide the proper implementation for each interface required by a tool or service. This is basic Java programming best practices. In order to allow for pluggable implementations of interfaces, the tools are kept completely unaware of the particular class which implements the desired interface. Tools and services simply express their desire in terms of the interfaces needed.

**Separation of graphic view and rendering from tool logic** - Sakai tools are decomposed into tool logic (written in Java) and tool presentation view (written in JavaServer Faces). This approach insures that these domains are kept separate so that it is possible to find and debug these elements separately. Furthermore it allows the separate development of the graphical look and feel of a tool from the basic logic and operation of the tool. It is clear that the design of JSF and the Sakai conventions on the use of JSF will enable graphical layout tools (such as Dreamweaver) to be developed which automatically generate the JSF markup from a tool used by the interface designer. Once this happens, then the interface design and mockup effort can be done directly in JSF, leading to quicker design-build-cycles.



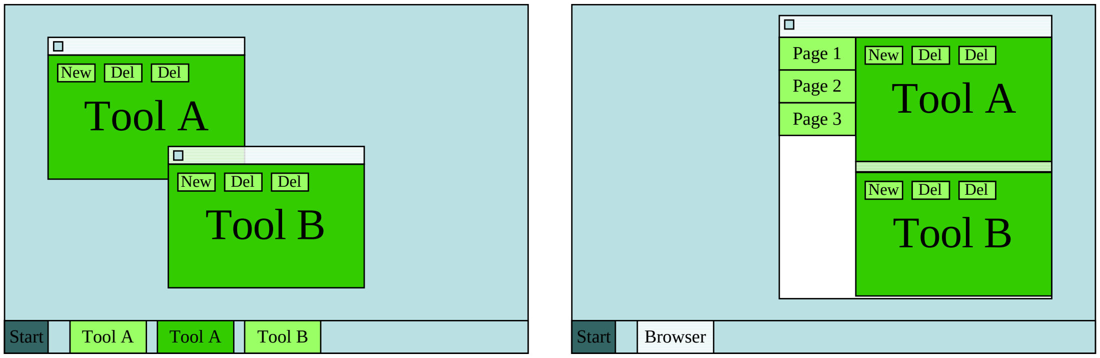
**Separation of persistence/storage from tool logic** - Often the persistence layer in a system (Database, File system, etc.) causes code to be non-portable because it depends on the availability of a particular storage technology. Because of this impact on portability Sakai demands that all storage activity be performed in services which tool accesses via an interface. This way, any non-portable code is hidden behind a clean interface.

### User Interface Elements Controlled by the Framework

The design of the portability profile is intended to maximize the portability of the components built to comply with the profile. Because the ultimate intent is to deploy TSPP compliant tools beyond the web browser environment, it is important that some aspects of the user interface are left to the framework implementation, and are not considered part of the tool itself.

- Presentation Rendering - While the initial target of the Sakai TSPP is web/browser environments, it has been designed to allow tools to operate in a desktop environment with the development of the proper desktop framework environment.
- Tool Arrangement - The user / desktop / framework controls the arrangement of tools - tools can be moved, maximized, minimized, opened, or closed under control of the framework.
- Tool Selection and Inter-Tool Navigation - Tools are launched using mechanisms provided by the framework - the user moves between tools using the framework as well.

Each tool should behave as though it is an independent window that can be composed and arranged by the user and/or the framework. A tool should make sense if it is displayed in a paneled environment with other tools such as a web portal or as a separate window on the user's desktop. The tool does not need to concern itself with providing a mechanism for "closing" the tool - the framework will control the launching and closing of the tools.



Each tool can be in use in multiple times by the same user. The tool does not need to keep track of separate instances of itself - the framework will insure that each instance of the tool will include a distinct set of data values that are kept separate from the other instances of the tool and the other instances of other tools.

The best way to keep this element in mind is to look at the duality between a portal and desktop environment. Given that uPortal has the capability to "tear off" windows from the portal, this paradigm is not so far-fetched as one might thing. The more far-fetched (but not impossible) notion is that these tools may actually be separate Java applications running on the desktop.

**Service Framework**

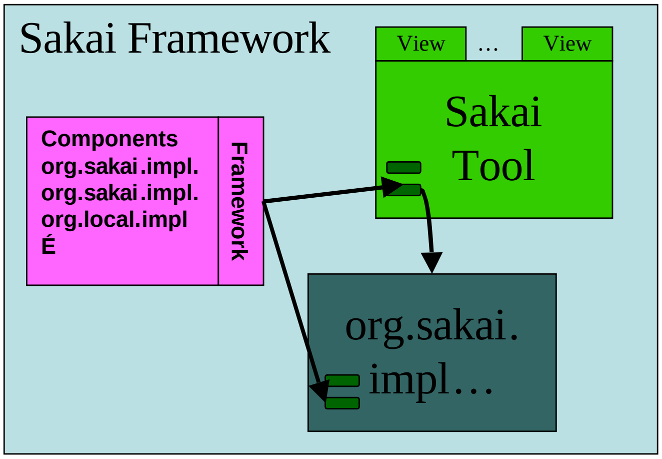
Usually in a system where there is a dynamic mapping of implementation classes to interfaces it is necessary to specify a service framework that can be used so that tools can locate services, and so that services can locate other services. Traditionally, the pattern used to solve this problem is for the service framework to provide an API that can be used to "look-up" the framework configured implementation for a particular interface.

This is called the "service locator" pattern and there are a number of examples of the use of this pattern:

- Jakarta Turbine
- Jakarta Avalon
- OKI OSID Loader
- IBM WebSphere

The problem with this pattern is that it introduces a dependency on the particular chosen framework because each framework invents its own method syntax, types, and parameters even though the underlying functionality is nearly always the same. As a result the tool or service must import packages specific to a particular framework. This explicit dependency on the framework is a few lines of code that is used in nearly every tool and service.

A new generation of frameworks typified by Spring and Pico removes this explicit dependency by allowing the tool or service express its dependencies either using a Java Bean setter method or constructor. When the tool or service object is being created, the framework examines it for the external dependencies and "injects" the implementation classes in via the constructor or bean-setter methods, making the necessary linkups. As the object begins execution, all of its dependencies are already satisfied and stored in local variables.



This approach makes the tool/service dependency resolution an implicit dependency rather than an explicit dependency. So while the bean-setter pattern was initially made popular using the Spring framework, it could be performed by some completely different framework such as a web-services based framework.

The Sakai Portability Profile includes bean-setting dependency as its best practice approach to dependency resolution. The bean-setting pattern is far more flexible when using containers (such as JSR-168 or JSF) that perform the actual construction of the object. To implement constructor-style injection in a JSR-168 or JSF environment you would have to revise JSF or JSR-168 so that it properly performs the service injection at constructor time. This is not practical in situations where you either do not have source to a component (such as JSF) or do not want to make modifications to a component (such as JSR-168) so as not to end up with a forked-source tree.

The bean-setter pattern allows dependencies to be injected after the constructor has been completed but before the object has been activated. Using this approach allows the opportunity to make clever use of existing features (such as the fact that every JSF tool is also a managed bean) to satisfy the necessary dependencies using standard mechanisms.

The Sakai framework easily supports constructor style and service locator mechanisms in addition to the bean-setter method, because there are situations where these are preferred, but the bean-setter method is the recommended best practice.

**Degrees of Portability**

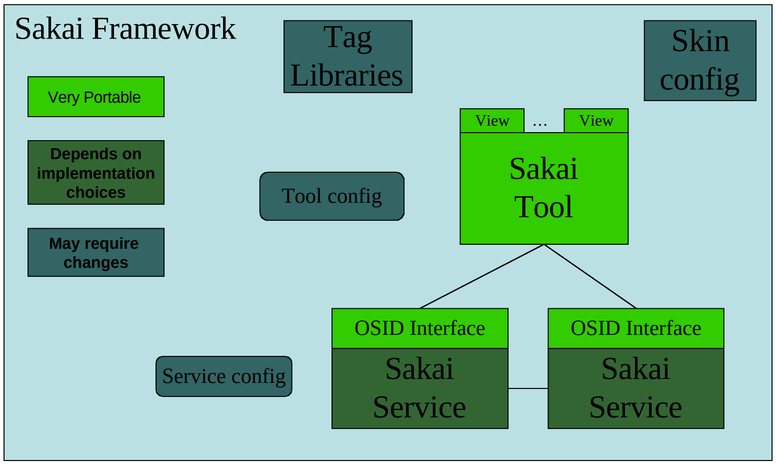
There are varying levels of portability of the various elements within the Sakai Technology portability profile. The goal of the profile is to ensure the maximum portability of the following elements:

- The Java code within the tools
- The View templates expressed in JavaServer Faces
- The service interfaces, APIs or OSIDs, for all the various levels of services
- The Java code within services which is used to communicate with tools and other services

We expect that these elements will be portable across a wide range of current and imagined frameworks. While the Sakai project is focused on delivering a browser-based JSR-168 portlet framework, other frameworks that have been imagined include:

- A desktop Swing-based Sakai framework
- A secure web services based Sakai framework
- A Grid Services based Sakai framework

There are no current plans to implement these alternative frameworks - some of them require new technologies to be developed or current technologies to mature before they are practical. While these frameworks may be a way off, it is instructive to examine how they might affect portability. The "maximum portability" elements have been designed so as to be portable across all of these frameworks with no changes to the code.



There are other elements that are part of developing Sakai components that may require modification as they are moved between different frameworks. These are some illustrative examples:

- If a developer produces a custom JSF tag that generates HTML or JavaScript, work will have to be done when operating in a non-HTML Sakai Framework such as Swing.

- If a developer places hard-coded Oracle calls in a particular service implementation, then that service implementation will only be deployable on systems that support Oracle. For a particular implementation of a service OSID, it may be an absolute requirement to use Oracle directly to ensure performance – however, this then limits portability. This is one reason that persistence activity is confined to services - to ensure that non-portable code does not sneak into the tool code.

- The XML configuration files that describe the tools and services to the framework may need to be changed to provide particular information that the framework needs. For example, a Grid based framework may need configuration information that describes where to find a particular service within the Grid.

- If a developer chooses to develop a tool using iChannel (uPortal), JSR-168, or WSRP (Web Services for Remote Portals) rather than using the Sakai GUI mechanism, then the tool will only be portable to environments that support those interfaces.

For the time being, these are academic distinctions because there is only one Sakai framework reference implementation that supports Sakai TSPP, JSR-168, WSRP, JSR-168, and many other standards. However as you look at the Portability Profile, and the reference implementation of the Framework, it is important to keep track of those elements that are "maximally portable" and other elements that are framework-dependent and are only present because of the particular framework that is in use.

Great care has gone into keeping the framework-dependent elements as small and simple as possible, so that most of the complex development is in the portable elements of the Sakai framework. In many ways, the TSPP is developed as part of a quest to find and define maximally portable elements.

## Other Documents

This overview document summarizes a number of other documents for the Sakai project:

- The Sakai Architecture - Describes the language independent terminology and structure of the Sakai tools, services, and frameworks. Much of this information is also included in this document.
- The Sakai Tool and Service Portability Profile for Java - describes the standards used by Sakai and describes how tools and services are developed in Java.
- The Sakai GUI Elements - Describes the display elements which are available to Sakai tools
- The Sakai Reference Implementation for Java in a Web Server - Describes the implementation of Sakai which will be delivers as part of the Sakai project
- The Sakai Collaborative Tools - Describes the tools, which will be provided as part of the Sakai project.

These documents will be released as they are completed. Some of these documents will be revised over time. In addition to these formal documents, a number of whitepapers will be produced addressing issues ranging from PHP and web-services to relationships with other projects such as JISC.

## Summary

This document provides a basic introduction to the architecture, technologies, and approaches, which will guide the Sakai project. All of these areas will be refined further in upcoming Sakai documents.

Sakai Tool and Service Portability Profile
This is a live process and many of the details of the Sakai project will not be known until portions of the development process have been completed. If you have any comments on this material, please send E-Mail to csev@umich.edu.

## Acknowledgements

This document is the result of the Sakai Architecture Research Group including: Craig Counterman, MIT, Rachel Gollub, Stanford, Mark Norton, Sakai Educational Partnership Program, Charles Severance, University of Michigan, Lance Speelmon, Indiana University, Curt Seiffert, Indiana University, Brian McGough, Indiana University, Glenn Golden, University of Michigan, Ken Weiner, uPortal, and many others from the core Sakai institutions.

In addition, a number of groups and individuals helped significantly in early review of this material including teams from the NMI Grid Portals project team, Cambridge University and Berkeley University as well as productive discussions with Benjamin Maillistrat who is the project architect of the eXo Portal project and Jason Novotny who is the architect of the GridSphere project.

## References

Tomcat Servlet Container (jakarta.apache.org)
JavaServer Pages and standard tags library (java.sun.org/jsp)
Enterprise JavaBeans (java.sun.com/products/ejb)
Spring Framework (www.springframework.org)
OKI Open Service Interface Definitions (www.sourceforge.net/okiproject)
JavaServer Faces (java.sun.com/j2ee/javaserverfaces)