The Sakai Project Final Report

14 October 2006

A report to the Andrew W. Mellon Foundation

By

University of Michigan

Principle Investigator:

Joseph Hardin
Principal Investigator

2161 Media Union
2281 Bonisteel Blvd
Ann Arbor, MI 48109
734-763-3266
hardin@umich.edu

1. Executive Summary

The Sakai Project was proposed as a partnership between four universities Indiana Univeristy, University of Michigan, Stanford University, and Massechusits Institute of Technology. The partnership also included two well established higher education projects in the Open Knowledge Iniative and the uPortal activity. The project had a two year timeline to "align the clocks" at these four schools and produce software for the four schools that met the needs of the four schools. The schedule included very early deliverables (low hanging fruit) which were made possible by working from the University of Michigan CHEF code base followed by a rewrite of the framework and rewrite of the tools using a new presentation technology (Java Server Faces) to produce the final product (Sakai 2.1) of the project in November of 2005. Knowing the interest the Sakai would generate outside of the core institutions, the Hewlitt Foundation funded the Sakai Educational Partners Program (SEPP) separately to allow the core schools to focus on this project and its deliverables while the SEPP staff maintained a liason to the Educational Partners.

> *[Sakai] will yield 1) A framework that builds on the recently ratified JSR 168 portlet standard and the OKI open services interface definitions to create a services-based, enterprise portal for tool delivery, 2) a re-factored set of educational software tools that blends the best of features from the participants' disparate software, and 3) a synchronization of the institutional clocks of these schools in developing, adopting and using a common set of open source software.*

The entire project plan and management structure was set up based on two-years of focused and uninterriupted development effort by a staff of 20 dedicated developers producing the software needed by the four partner schools.

A number of factors forced us to depart from the plans laid out in the proposal. These were forces demanding profound changes in the technical direction of the project as a condition of their continued participation in the effort. At each juncture we *could* have ignored the forces and demands coming from inside and outside the project, but in each case, we kept in mind the long-term goal of building a community that would sustain the product. If in each case there was disagreement in terms of project direction, we had simply held fast to the original project plan and ignoring the needs of the other stakeholders, we would have resulted in a far inferior product and likely a very small community of adopters (probably three schools in November 2005) and effectively no community to take the product forward. By listening to these new broader requirements and changing our plans as we went forward we have a very broad community and wide adoption around the world and an effective long-term sustainability model.

> *This effort will demonstrate the compelling economics of "software code mobility" for higher education, and it will provide a clear roadmap for others to follow.*

These changes in overall project direction often happenned at the expense of the founding partners priorities - each time precious resources were invested in new directions demanded by new stakeholders, it meant that less resources were applied to the needs and objectives of the core institutions. These choices have resulted in some lingering animosity amongst the core institutions and has left a number of incomplete technical requirements hanging at the end of the project.

As in any technology effort, some of the technical decisions made in the project can turn out to make the path much more difficult and require far more investment of resources than was anticipated. In the remainder of this report we describe the achievements of the Sakai project and the shortcomings and give background and rationale for both.

2. Project Motivations and Contributions

Overcoming Barriers to Application Sharing

The overall goal of Sakai was to define the Tool Portability Profile (TPP). The idea was to fully define a "unit of expansion" for Sakai to enable any set of developers anywhere in the world to build a "modular unit" for Sakai and have that unit able to be "dropped into" a running Sakai system and have it work. The TPP needs to fully describe the entire environment of the new module including (a) how the tool will interact at the presentation layer so as to be consistent with the other tools, (b) how the tool will call the rest of the Sakai Application Programming Interfaces (APIs), and (c) how the new tool can add its own APIs to Sakai and make them available to other tools.

The technical barriers can now be overcome by distilling the accumulated architectural knowledge and programming experience gained in building these systems into a Tool Portability Profile (TPP) that provides four essential elements for code mobility.

Even though there is not a single document in Sakai at this time called the "TPP", all of these areas are well documented and well defined and many developers around the world are now in the process of developing their own tools that expand Sakai's capabilities. This work has been so successful that the majority of the Sakai tools that are now produced come from well outside the core institutions. There are many tools in many stages of development around the world. Part of the effort of the Sakai Foundation going forward is to use Foundation resources to find and integrate these tools into the Sakai releases.

The TPP has two major components:

- The presentation layer attempts to provide a clean abstraction that tools can use to produce their user interface.
- The services layer organizes the Sakai Services - these services provide all of the persistence capabilities of Sakai.

By building strong abstractions for both of these aspects we achieved component-based expansion and seamless code mobility across Sakai systems.

TPP: Presentation Layer

One of the goals of the Sakai project at the presentation layer is to produce an abstraction that would allow the Sakai tools to work in non-HTML environments such as desktop applications. At the same time, we knew that the Sakai software would be deployed in browsers for many years so it was important to produce a good web experience. We evaluated a number of technologies and found them wanting:

- **Velocity** (http://jakarta.apache.org/velocity/) - Velocity is very HTML based - it has a good abstraction on the outbound markup generation but uses purely HTML mechanisms for incoming parameters. CHEF used Velocity so we had a lot of experience in it and felt that it was not ideal for the "future proofed" presentation layer we had hoped for in Sakai.

- **Struts** (http://struts.apache.org/) - Struts is a much more popular approach than Velocity - it is the de-facto choice for most top-end web applications today. Like Velocity - it is not a pure abstraction as it is HTML-only. Java Server Faces was the heir-apparent for the Struts community.

- **XUL** (http://www.mozilla.org/projects/xul/) - XUL looked very promising and it is the markup used in the popular Mozilla (now Firefox) browser. The problem with XUL was that there was

little or no supported effort to move XUL from being a technology that ran in the browser to becoming technology that ran in the browser.

- **XML/XSLT** - This is a generic approach which has tools produce each screen as XML to which an XSLT is applied to produce the final markup. This is also commonly uses in many applications including used heavily in uPortal 2.x. The problem with this technology is again on the inbound request - which is HTML based only.

After this analysis we settled on Java Server Faces. At the time of this decision (November 2003) JSF was only at an Alpha release level but it had a number of things going for it which led to its selection as the presentation technology for Sakai.

- JSF was the heir apparent to the very popular Struts technology - any of the leading thinkers of the Struts community had moved to work on JSF.

- JSF supported asymmetric interaction between the presentation layer (widgets and layout) and the Tool logic writing in Java. There was no need for the tool logic to be aware of HTML at all. This provided hope that the Tool logic written in Java might someday be reusable in a new presentation technology.

- JSF included the ability to build reusable components such as calendar or text-editor widgets and reuse those components across many tools.

- SUN provided a reference implementation of JSF and a set of basic widgets. This would jump-start our ability to support JSF so as to be able to use the new presentation technology as quickly as possible. We hoped quickly build a Sakai style guide and a set of widgets to implement the Sakai style guide. This would make it very simple to develop a wide range of tools with a common look and feel.

> *One barrier which has consistently defeated various efforts to pool higher education software development investments unique local technical architectures, including heterogeneous hardware, software interoperability requirements between systems, and diverse user interface requirements, have impeded great software at one institution from being moved to another even when it was available as no-fee, open source software.*

- There was a rumor of a Swing Desktop implementation of JSF - the hope was that this would help deliver a whole new way to deploy Sakai tools at some point in the future. This has sadly never materialized.

- There was a hope that we could write an implementation of JSF to bridge to the Portlet API (JSR-168) so that all JSF tools could be used in JSR-168 environments.

This work resulted in several outputs for the Sakai TPP: (1) the Sakai style guide and (2) the Sakai JSF Tag library.

TPP: Presentation Layer - Sakai Style Guide

This is an excerpt from the Sakai Style Guide:

> This document is intended as a guide to inform Sakai User Interface (UI) design. More practically, those developing the JSF renderings and/or HTML renderings of this application should use this guide to direct their efforts. Its foci are the principles of accessible and usable interface design, and the elements that are used in it – starting with the more common complex/composite sorts of screens, and from them deriving a discrete common set of elements

as well as rules governing the syntax used to combine them. There is also a general guideline for language.

Each type of screen/interaction contains the following sections: 1) a definition and purpose, 2) a listing of required and optional elements that compose it, and 3) an illustration. Relevant Accessibility considerations are contained in Accessibility Notes throughout the document. Adaptive Technology (AT) includes, but is not limited to, screen readers, text magnifiers, voice-activated controls, specialized keyboards and Braille displays.

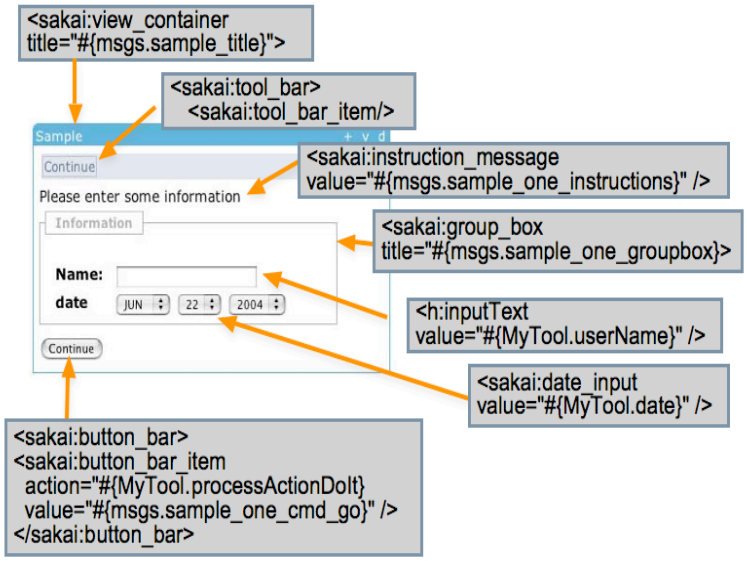
The Sakai style guide is a living document and is revised periodically as new issues or technologies are identified. You can peruse the entire Sakai Style Guide at:

http://cvs.sakaiproject.org/release/1.0.0/docs/Sakai-Style-Guide-1.0.0.pdf

The Sakai Style Guide is a live document that is revised as new technologies or issues are identified and need solutions.

TPP: Presentation Layer - Java Server Faces Widgets

An important aspect of the Sakai TPP presentation approach is the development and use of Sakai JSF widgets. The Sakai widgets are combined into a tool's layout as follows:



These widgets do not include any HTML - by using this widget pattern we isolate the code for elements such as date pickers in a single location allowing customization or localization of the calendar picker.

A number of widgets were produced and used in Sakai tools:

sakai:alphaIndex	sakai:multiColumn
sakai:anchorReference	sakai:output_date
sakai:applet	sakai:pager
sakai:button_bar	sakai:panel_edit
sakai:button_bar_item	sakai:panel_titled
sakai:courier	sakai:peer_refresh
sakai:dataLine	sakai:popup
sakai:debug	sakai:progressBar
sakai:doc_properties	sakai:rich_text_area
sakai:doc_section	sakai:script
sakai:doc_section_title	sakai:selectCommand
sakai:dynaTable	sakai:stylesheet
sakai:flat_list	sakai:timerBar
sakai:flowState	sakai:tool_bar
sakai:group_box	sakai:tool_bar_item
sakai:hideDivision	sakai:tool_bar_message
sakai:input_date	sakai:tool_bar_spacer
sakai:inputColor	sakai:view
sakai:inputFileUpload	sakai:view_container
sakai:inputRichText	sakai:view_content
sakai:instruction_message	sakai:view_title
sakai:messages	

The following is the documentation for the Sakai JSF widgets:

http://source.sakaiproject.org/release/2.2.0/taglibdoc/

TPP: Local Customization and Internationalization

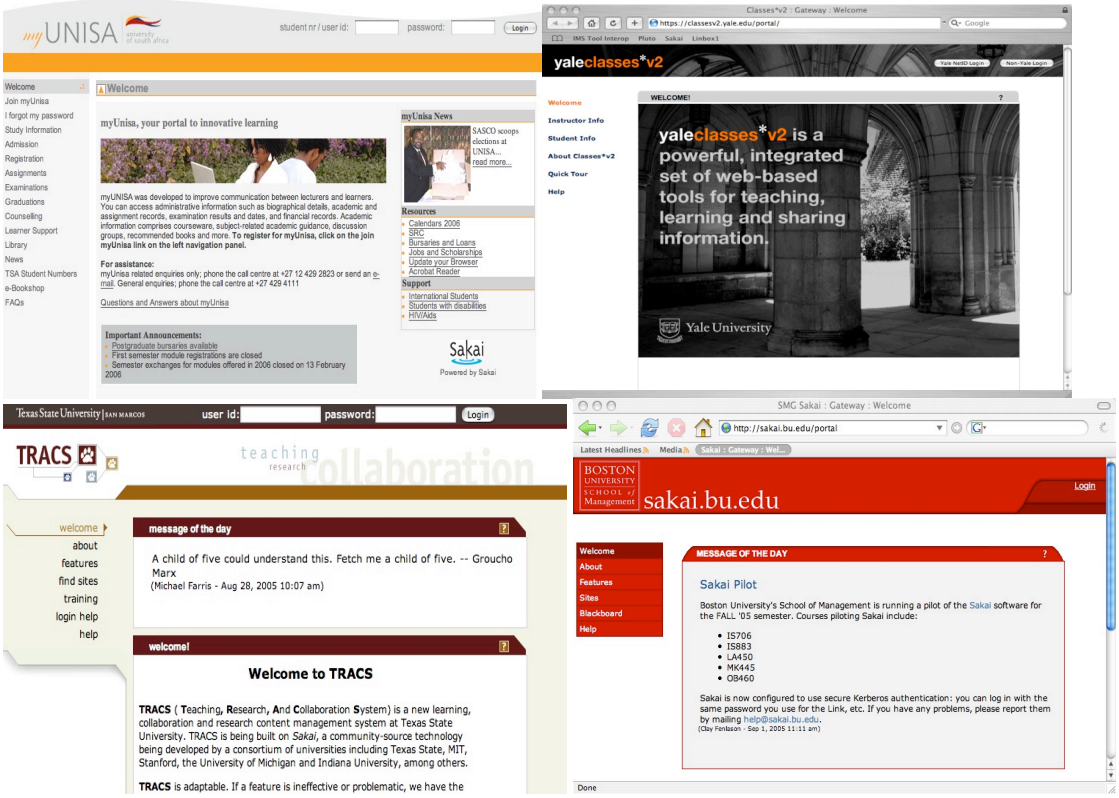
The Sakai TPP has developed and evolved a sophisticated ability to customize the Sakai Application in terms of components, look, feel, and provides a large number of options which allow sites to produce their own particular application of Sakai.

This work falls into a number of important categories:

- **Local configuration of functionality** - Different organizations and different user bases often have very different needs in terms of how the Sakai application and tools are to operate. Sometimes we could reach a strong consensus about a particular way the software should function. Other times we simply realized that there is more than one right way to do things. This has led to the need to build a general configuration capability that allows each site to make their choices and configure their Sakai installation to suit their needs. By providing these alternatives, we avoid forking the source code at each institution because they have a different point of view as to how the software should operate. This configuration capability in Sakai is described in a 69 page document available at this URL: https://source.sakaiproject.org/svn/reference/trunk/docs/architecture/sakai_properties.doc

Institutional adoption by faculty and students often relies on user interfaces that match other familiar systems. A user interface includes colors, fonts, logos, and navigation aids that can be localized as needed without disturbing the underlying functionality of the software.

- **Skinning and visual appearance** - All of the look and feel of a Sakai site is delegated to a set of Cascading Style Sheets (CSS) that allow local customization of the look and feel and even interaction patterns of a Sakai system. Here are several examples of the different skins from adopting Sakai institutions:



The procedures to develop a skin for Sakai is a 26 page document available at the following URL: https://source.sakaiproject.org/svn/reference/trunk/docs/architecture/sakai_skin.doc

- **Internationalization and Localization** - While the original proposal was to produce software for use at four US-based schools, as interest developed internationally in Sakai it became increasingly important for Sakai to support multiple languages. Since this was not part of the Mellon funded work plan, additional resources were identified and used to add support for multiple languages to Sakai 2.0. These additional resources were obtained from the Universitat de Lledia , University of Michigan, Nagoya University, and many others. As of the Sakai 2.1 release the following languages are completely or partially supported by Sakai: Chinese (China), Korean, Japanese, Dutch, Danish, Brazilian Portuguese, Portuguese, Slovakian, Catalan, Spanish, Chinese (Taiwan), Spanish (Spain), Spanish (Mexico), Spanish (Chile), Spanish (Argentina), Russian, Swedish, German, and Turkish.

The ability to make such extensive local configuration changes without requiring modification to the base code greatly improves the reliability of each of the installed Sakai systems as they are less likely to fork the code to make a local customization. Allowing each site to have maximum flexibility in configuring their Sakai installation leads to increased adoption and increased end-user satisfaction.

TPP: Services and Components

Another important area of the Tool Portability Profile is the way that Sakai supports Services. The CHEF product form the University of Michigan used a services framework from Apache called Turbine (http://jakarta.apache.org/turbine/). As of the beginning of the Sakai project it was felt that the Turbine technology was a bit old so we looked for alternative options for implementing the Service components needed by Sakai.

During the first few months of 2004, the following alternatives were evaluated:

- **Enterprise Java Beans** (http://java.sun.com/products/ejb/) - EJB technology was used in the Eden project at Indiana University. We did an evaluation of EJB as a replacement for Turbine and concluded that EJB was simply too heavyweight for what we wanted to accomplish with Sakai.

- **Apache Avalon** (http://avalon.apache.org/) - Apache Avalon was a lightweight service system that was a peer to Turbine - but Avalon was clearly a "previous generation technology" with less market uptake than Turbine so it was not considered seriously. If you visit the Avalon home page you will find that the project is now shut down.

- **Pico Container** (http://www.picocontainer.org/) - The Pico container had an advantage in that Pico would "wire up" services together using Java Constructors. Pico was ultimately seen as too simple and brittle for use with Sakai.

- **Spring Framework** (http://www.springframework.org/) - The Spring framework was becoming increasingly popular as it supported an "Inversion of Control" pattern that allowed services to be wired together without placing any container-dependent code in the Sakai services. Spring also had excellent support for database transactions without requiring changes to the service code. Spring was also the simplest technology that we could integrate with Java Server Faces (JSF).

Our analysis and decision was heavily influenced by a paper written by Martin Fowler that was titled "Inversion of Control Containers and the Dependency Injection Pattern" (http://www.martinfowler.com/articles/injection.html). Martin evaluated many of the different patterns for services finding the other services which they were dependent on (dependency analysis). Martin's work concluded that the Inversion of Control pattern best supported by Spring was the ideal approach and we agreed with his analysis.

The education community will benefit greatly from a Tool Portability Profile that provides an open, non-proprietary, and fully articulated specification for interoperable software. Any institution or commercial entity can build to this Profile, thus helping all institutions integrate software from multiple sources as their timing may require.

Sakai had two major implementations of the Sakai TPP - The Sakai Framework I was used in Sakai versions 1.0 and 1.5 and the Sakai Framework II was used for Sakai version 2.0 and later.

Sakai Framework I placed services in web applications and then used Spring to wire up the services. This approach had some technical limitations so in Sakai Framework II we moved services out of web applications and into their own component area.

You can review some of the documentation regarding the Sakai TPP Services at these URLs:

https://source.sakaiproject.org/svn/reference/tags/sakai_2-2-0/docs/architecture/sakai_component.doc

TPP: Tools Developed using the TPP

The TPP was a dynamically developed profile that evolved over the life of the Sakai project. There were a number of new tools that were developed using the full Sakai TPP Specification during the duration of the Sakai Project that saw use amongst Sakai adopters:

Profile	Developed by Indiana University (*)
Roster	Developed by Indiana University (*)
Gradebook	Developed by University of California, Berkeley
Samigo Assessment	Developed by Stanford University (*)
Melete Module System	Developed by the Etudes Consortium at Foothill College. (**)
Section Management	Developed by University of California, Berkeley (*)
Syllabus	Developed by Indiana University (*)
OSP: Forms Tool	Developed by Indiana University and rSmart, Inc. (***)
OSP: Evaluations Tool	Developed by Indiana University and rSmart, Inc. (***)
OSP: Glossary Tool	Developed by Indiana University and rSmart, Inc. (***)
OSP: Matrices Tool	Developed by Indiana University and rSmart, Inc. (***)
OSP: Templates	Developed by Indiana University and rSmart, Inc. (***)
OSP: Wizards	Developed by Indiana University and rSmart, Inc. (***)
Presentation Tool	Developed by University of Michigan (*)
Wiki Tool	Developed by Cambridge University
IMS Tool Interoperability	Developed by University of Michigan (*)

Development of TPP Compliant tools continued after the Sakai Project with the following tools being under development and/or in use by Sakai adopters:

Postem	Developed by Indiana University (a simple grading system)
Blog	Developed by Lancaster University (UK)
Search	Developed by Cambridge University
Site Statistics	Developed by University Fernando Pessoa
Goal Aware Tool	Developed by Syracuse University
Calendar Tool	Developed by University Fernando Pessoa
WSRP Consumer	Developed by Daresbury Laboratory, UK
Shared Whiteboard	Under Development at Lancaster University, UK.
Audio/Video Multicast	Under Development at Lancaster University, UK.
Shared Desktop	Under Development at Lancaster University, UK.
Citation Tool	Developed by Indiana University and University of Michigan (***)
Mail Tool	Developed by Northwestern University and Boston University
MessageCenter	Developed by Indiana University (*)
SCORM 1.2	Developed by University of California, Davis
SCORM 2004	Developed by the IBM Corporation
iTunes University Tool	Developed by the University of Michigan
Paper Submission and Review Tool	Under Development by Daresbury Labs, UK.

(*) Funded by the Mellon Foundation as part of the Sakai Project
(**) Funded by the Hewlitt Foundation
(***) Funded by the Mellon Foundation

There are many more tools under development - typically we only learn of a tool once the developers announce it to the community as available for sharing.

These large-scale deployments of common applications based on the TPP at four complex institutions will clearly demonstrate the real viability of open source code mobility for higher education.

You can track activity around existing and emerging tools for Sakai at the following URL:

http://bugs.sakaiproject.org/confluence/display/MGT/Home

TPP: Tools Which Partially Comply with the TPP

The Sakai Framework does not demand that all tools precisely comply with the Tool Portability Profile. With Sakai 2.0 and later, nearly any presentation technology can be supported including: Velocity, Spring MVC, Struts, XML/XSLT, and many others. While these other technologies cannot take advantage of the shared JSF components in the TPP, tools written using these other presentation technologies can make use of all of the Sakai services. Having flexibility to support applications beyond Java Server Faces in the Sakai TPP has greatly improved the ability to quickly add capabilities to Sakai.

A number of tools that came from the CHEF system used the Velocity presentation technology. These tools and their respective services were all re-written to use TPP compliant technology for their services (Spring, etc). A layer was written to bridge between Sakai and the Velocity layer and so these tools are seen as native tools within Sakai without requiring a rewrite of their presentation layer. These tools were all written by the University of Michigan: Announcements, Drop Box, Email Archive, Resources, Chat Room, Message Of The Day, News/RSS, Threaded Discussion, Preferences, Web Content, Worksite Setup, Assignments, Schedule an all of the Sakai Administrator tools. These tools form the core of the Sakai collaborative capabilities.

All Sakai tools will be both modular and also pre-integrated to work with each other. The software will be made available to the world at the same time via an open source license.

Another partially compliant TPP tool is JForum (http://www.jforum.net/). The Etudes Consortium at Foothill College ported JForum into Sakai without altering its internal structure or presentation layer.

The University of South Africa ported nine locally developed Struts tools using a Struts adapter to the TPP.

Cross-Community Coordination

The primary technical deliverables of the project beyond the Sakai Product itself and the Sakai Tool Portability Profile (TPP) were the interaction with two existing Open Source Communities working in Higher Education. The Sakai Project proposal made relatively broad promises about the ability to coordinate activities and bring three communities together to function as one. The Sakai Project proposal also promised that each of the technologies would be deeply integrated into Sakai.

While the Sakai Project significant progress was made supporting and working with these communities, the ultimate deliveries on the technical promises made in the Sakai Proposal were not achieved during the life of the project phase. Collaboration continues between Sakai and these communities and we expect to achieve the desired results in time.

Over the life of the project, we were regularly faced with a very basic decision in terms of our relationship between Sakai and these communities. We had two choices:

- Use the Melon funding to take control of the leadership of these existing communities and direct them to evolve in a way to become deeply integrated into the Sakai project and potential by the end of the project be simply a part of Sakai.

- Simply pass the Melon funding through to the management of these projects and cooperate with the projects as peer efforts, allowing the other projects to make their own internal decisions while maintaining coordination between Sakai and the other communities to maximize potential overlap and re-use and produce increasingly common solutions in the long run.

The maturing of the OKI OSIDS, recent demonstration of a working tool interoperability framework at U. Michigan, and industry ratification of the JSR-168 portlet specification make the timing perfect for developing a full Tool Portability Profile for higher education.

A combination of the two approaches was used in the interaction between the communities. However when there was a disagreement in direction, the Sakai leadership always respected the wishes of the community that had long-term responsibility for their products.

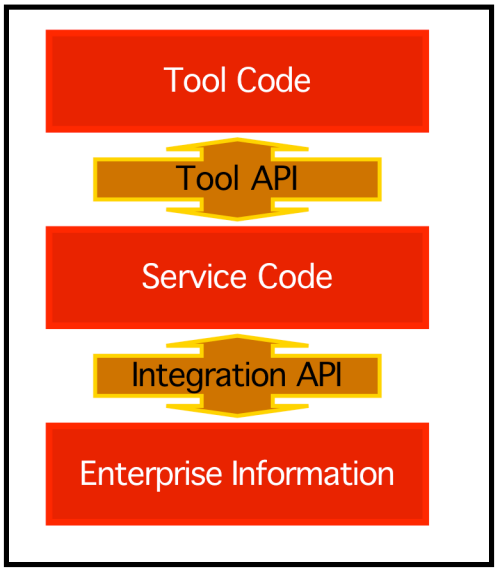
This limited Sakai's ability to *force* these communities to change direction. We felt that if we stepped in and started pushing the communities in our particular direction against their will - that the communities volunteer members would simply leave the community. Since Sakai was initially attracted to these communities because of their existing leadership and communities around the software and specifications, we felt that harming the existing communities would be a mistake.

In this section we describe the outcomes of these interactions, and the factors that limited our success in achieving the stated deliverables.

After the completion of the Sakai Project we continue to work with these communities in an attempt to align our products. There is some hope for improved alignment that respects the needs and directions of each community. The after project interactions and some roadmaps to ultimately align the projects are presented in the "Looking Forward" section later in this document.

OKI Open Services Interface Definitions (OSIDs) - MIT

The OKI project was four years old when Sakai became involved in OKI - OKI laid much of the groundwork for the Architectural concepts of the Sakai TPP. OKI posited that we should adopt a service-oriented approach within applications and use APIs to define and standardize the interactions within the elements that comprised Sakai. This OKI influence led to a simple architectural approach used across Sakai.



This approach led us to work to define highly detailed Application Programming Interfaces between the Tool and Service code and between the Service code and Enterprise Information. We insisted that both these API layers were very clean and that no tool or service violated this architecture.

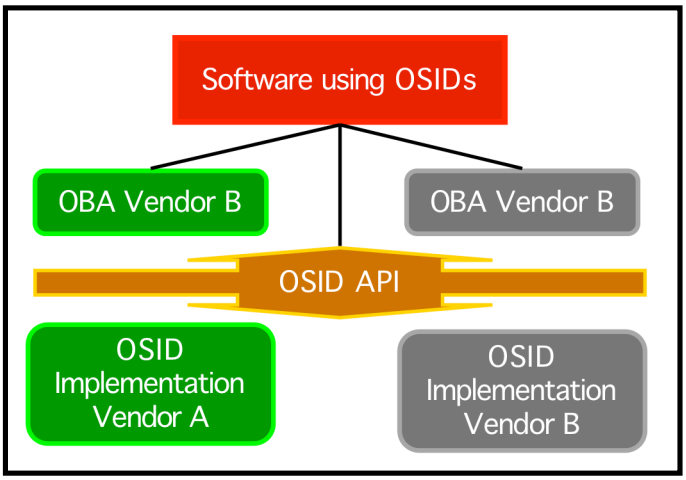
The OKI project released the OSID specification Version 1 Release Candidate 6 prior to the start of the Sakai Project and used Sakai Mellon funds to work on and release the version 2.0 OSID Specification. Sakai used both specifications.

The OKI OSIDs provided two sets of APIs related to teaching and learning. The first set of APIs were relatively generic and with a broad scope sufficient for use across many applications many applications called the Common Services: Authentication, Authorization, SQL, Logging, Filing, Dictionary, Hierarchy, Agent, ID, User Messaging, Scheduling, and Workflow. The second set of APIs was targeted toward teaching and learning environments: Repository, Course Management, Grading, and Assessment.

The OKI OSIDs have a relatively unique approach to defining semantic detail of an API. Most API specifications completely encode the semantics of an API within the API and its specification ((http://java.sun.com/j2se/1.3/docs/api/java/lang/String.html) - developers can work directly from the API specification and write code directly using the specification as programmer documentation.

The Open Knowledge Initiative Open Services Interface Definitions (OSIDs) have provided an essential first contribution to solve the technical challenge. Local implementations of the OSIDs at an institution integrate heterogeneous local architectures (e.g., an authentication system or directory service) by using common connectors that enable code mobility for OKI-based application software

OKI OSIDs are generic APIs that can be used across many applications and implementations. An example of an API with this flexibility is the Java Database API (http://en.wikipedia.org/wiki/JDBC). The JDBC API supports many different databases. Each database uses JDBC is a slightly different manner and the Application developer using the JDBC API is responsible for know which database implementation they are interacting with and use the JDBC API in the proper way according to how that particular database vendor describes how to use the JDBC API.



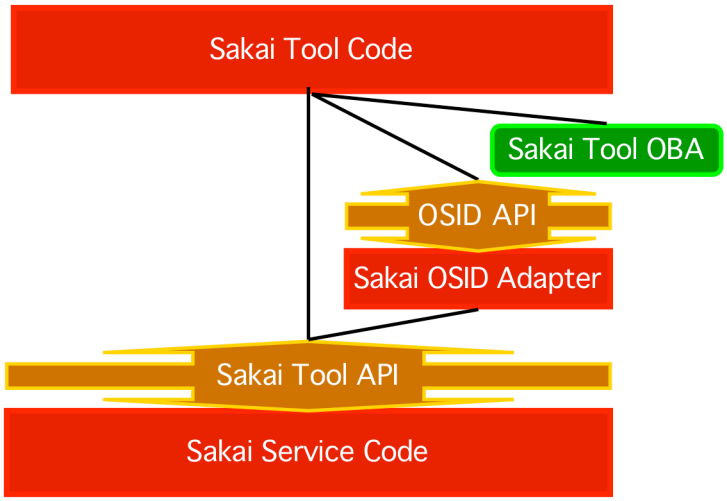
This flexibility implicit in the OKI OSIDs is called "out of band agreements" or OBAs for short. To use an OKI OSID, a developer must both consult the OKI OSID documentation as well as the OBA documentation for the particular OSID implementation. The application developer had to add code to their application to detect which OBA their OSID was using and includes code in their application to support the superset of the OBAs that were needed.

This implicit flexibility in OKI OSIDs is essential to allowing OSIDs to be used in many different applications and across multiple programming languages. This need to support multiple programming languages also leads to a bit of inefficiencies when using the OKI OSIDs in any one language.

OSIDs at the Sakai Tool Interface

The goal of Sakai was to create a very rich ecology of tools with powerful and easy to understand APIs. The Sakai APIs were designed to hide implementation detail from the tools and allow transparent substitution of implementations without any awareness on the part of the tools. Given these goals, it was decided that the OKI OSID API plus out-of-band approach was not suitable as the exclusive

representation of either the Sakai Tool or Integration APIs. The agreed to goal of OSIDs at the Tool API was to develop a set of Sakai Specific Out of Band Agreements to allow tools to be developed make use of any combination of the Sakai APIs and OSIDs augmented by the Sakai out of band agreements as shown in the following figure.



One key in making this figure function as well as possible was to evolve the underlying Sakai APIs so as to be as similar to the OKI OSIDs as possible. This was done and many of the Sakai Tool APIs follow the OKI API naming conventions.

This approach was taken and several of the OKI OSIDs have been implemented with defined Sakai out of band agreements and are part of the Sakai releases and are used in a number of tools using the above pattern:

- Id OSID (org.sakaiproject.component.osid.id)
- OSID Loader (org.sakaiproject.component.osid.loader)
- Logging OSID (org.sakaiproject.component.osid.logging)
- Repository OSID (org.sakaiproject.component.osid.registry)
- Repository Registry OSID (org.sakaiproject.component.osid.repository.registry)

The Repository Registry OSID is a new OSID developed to help federate across using multiple Repository OSIDs. The choice of which OSIDs to implement and support in Sakai was based on the requirements of the community using OSIDs in Sakai. Most of the use of OSIDs was at the Integration API level (below) so the OSIDs implemented at the tool level were primarily the utility APIs that supported the other OSID APIs within Sakai.

You can peruse the documentation and out of band agreements for these OSIDs at http://source.sakaiproject.org/release/2.2.2/javadoc/ and searching for the string "osid".

OSIDS at the Sakai Integration Interface

A similar approach was taken to enable the use of the OSIDs at the layer between the Sakai services and enterprise data. Sakai defined a semantically complete Integration API and code could be developed to map OSID implementations and their out-of-band agreements to the Sakai Integration APIs.

Like the relationship between the Sakai Tool APIs and the OKI OSID APIs, the Sakai Integration APIs

have been heavily influenced by the OSIDs so as to make integrating enterprise information into Sakai as direct as possible.

During the lifetime of the Sakai project, the OKI project increasingly focused on the enterprise data integration use case and placed a great deal of energy and effort on the Repository Integration use case for Sakai, VUE, and many other application. Sakai encouraged and supported this approach, as repository integration is a critical use case for the Sakai product. Sakai effectively was able to "out-source" much of the work of exploring repository integration into Sakai to the OKI team. The abstraction layer provided by the OKI Repository OSID allowed the Sakai and OKI teams to work in parallel throughout the Sakai project.

A number of projects were undertaken with the Repository OSID at their core:

- **TwinPeaks** was an effort initially started by Indiana University and then work continued with the involvement of the OKI team. Twin Peaks provided a general way to search across multiple repositories using the OKI OSID and was integrated into the Sakai WSYWIG editor.

- **SakaiBrary** is an effort to go beyond TwinPeaks to tightly integrate premium content and to add the rich support for federated searching of many repositories providing librarians the ability to create virtual collections for each class. SakaiBrary is also producing a Citation tool that allows teachers and students to assemble lists of references to external material and share those references within Sakai.

- Apple's **iTunesU** effort has been integrated into Sakai's Content Hosting system using OKI OSIDs as well. This was demonstrated at the 2006 Alt-I-Lab meting and the 2006 Educause meeting.

An extensive document describing the necessary out of band agreements to support the SakaiBrary use of the OKI OSID is available at:

http://bugs.sakaiproject.org/confluence/display/SLIB/Phase+1+Consumer+Guide
http://bugs.sakaiproject.org/confluence/display/SLIB/Search+Management
http://bugs.sakaiproject.org/confluence/display/SLIB/Repository+Types
http://bugs.sakaiproject.org/confluence/display/SLIB/Session+Management

Going forward, the OKI Specification development has been moved to the IMS (www.imsglobal.org) specification organization. Members of Sakai and the OKI project are working together in the IMS forum to further evolve the OKI specifications.

uPortal - University of Delaware

The Sakai Project Proposal included a number of primary deliverables with respect to uPortal and the Portlet API (http://jcp.org/en/jsr/detail?id=168). Here is an excerpt from the proposal:

> *Advanced CMS's are based on portals that aggregate class information and services and allow the user to personalize and customize their views of these classes, services and information. At the same time, university-wide services are migrating from independent web-based interfaces that accessed siloed systems (e.g., Bursar, Library, Registrar, CMS, etc.) to enterprise-wide portals that integrate a personalized view of the full range of the university's services and information. The uPortal effort has brought forth a powerful portal environment that has commanded broad adoption, but it currently lacks the recently ratified JSR 168 portlet specification needed for tool interoperability. A standards-based portal that can be used as both the academic portal for the CMS as well as for delivering other university services via the JSR-168 portlet standard is a core building block of the TPP.*

This can be parsed into several separable deliverables:

- Using JSR-168 as a core standard in the Sakai TPP. It should be possible to write JSR-168 tools and use them in Sakai. In addition, JSR-168 should be the "recommended" way to write applications for the TPP.

- Sakai was to be both an academic (university-wide) portal and a course management system. At its strongest, this statement effectively implies that the Sakai and uPortal products are merged together as one product capable of satisfying the needs of the academic portal and course management

These deliverables were conceived with a genuine lack of understanding of the following issues:

- The requirements for a course management system and a campus portal are actually quite different. This is true in terms of how the systems are managed, deployed, configured and supported.

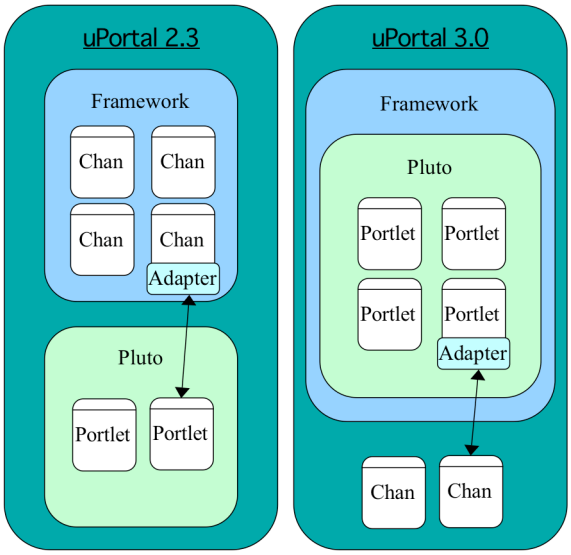
- The JSR-168 Portlet API and WSRP 1.0 (http://www.oasis-open.org/committees/wsrp/) were the initial versions in both areas and both relatively immature. Worse still, the open source reference implementations for JSR-168 (Pluto 1.0 - http://portals.apache.org/pluto/) and WSRP 1.0 (WSRP4J - http://portals.apache.org/wsrp4j/) were also very immature and were very challenging to make use of in a flexible way.

We encountered these challenges along the way and as we learned of each issue, we adjusted our expectations and changed our course to reflect the new information. At each decision point we tried to stay as true as possible to the intent of original deliverables while still making good technical decisions along the way and maximizing the value to the Sakai and uPortal communities over the life of the project.

Looking back at the project these were the efforts that we *did* accomplish during the project:

- Using a forked version of Pluto 1.0, the uPortal team quickly developed a **JSR-168 to iChannel adapter** for uPortal 2.x. The idea of this deliverable was to provide a solid framework to experiment with JSR-168 portlets. The uPortal team delivered JSR-168 support within a month

of the start of the project. This adapter has been used successfully in production at uPortal sites running 2.x over the past 2.5 years and has allowed the uPortal community to smoothly make a successful transition from programming to the uPortal-propretary iChannel interface to almost exclusively building JSR-168 portlets.



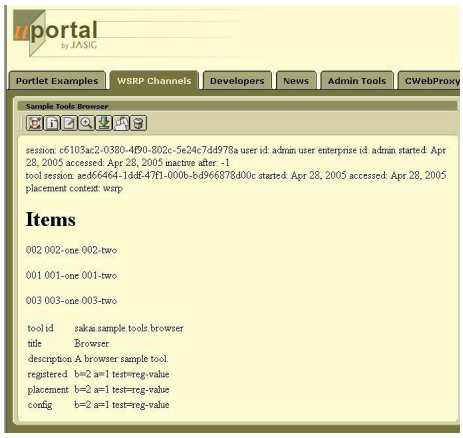
- Produce **uPortal 3.0** which would use JSR-168 as its primary interface and provide an adapter for the existing uPortal-proprietary iChannel portlets. uPortal 3.0 was also supposed to provide strong support for WSRP 1.0 Consumer and WSRP 1.0 producer. In addition, uPortal 3.0 was supposed to provide a flexible rendering pipeline to allow uPortal to support the Sakai navigational use cases without requiring modifications to uPortal's core code. The uPortal 3.0 work was a ground-up rewrite and done by a small dedicated team funded by Sakai / Mellon funds. The uPortal 3.0 effort is described later in its own section.

- Build a version of Sakai, which used uPortal 2.x as its presentation framework. This work was completed in Summer 2004. The plan was to ship the modified version of uPortal as part of the Sakai 1.0 release but ultimately removed from that release. Because Sakai's navigation and contextualization requirements were more sophisticated than uPortal 2.x could handle at the time, it was necessary to make some dramatic changes to uPortal internal structure to support Sakai's needs. When we discussed the approach which we had taken with the uPortal leadership it was determined that these modifications would be a dangerous fork to the uPortal code base and if it were released it had the potential of splitting the uPortal developer community. The decision was made not to release the forked uPortal with Sakai 1.0 and instead to focus on using standards to interact between the systems rather than customizing uPortal with Sakai-specific features. In August 2004, we decided to move from using JSR-168 to integrate Sakai into uPortal to using WSRP 1.0 to do the integration.

- Add features to the Sakai to support "**gallery mode**" where the login and branding were removed from the Sakai user interface. This allowed Sakai to be included in *any* portal using an iFrame and single sign on. This capability was provided at the end of 2004 and has been used successfully in a number of production sites that use both Sakai and uPortal as a interim solution while better integration between Sakai and uPortal was being developed.

- Build a reusable iChannel adapter in uPortal 2.x to support **WSRP 1.0 Consumer**. This was to be used as a testing harness to help with the development of the expected WSRP 1.0 Producer for Sakai that would provide a ready tool to test the Sakai capabilities. The uPortal WSRP Consumer

portlet has been used by many organizations to experiment with WSRP development and run some production uses of WSRP.



- Build an implementation of **WSRP Producer 1.0 for Sakai**. Since this work was outside of the scope of the original proposal and all of the Mellon-funded resources were working on uPortal 2.0, this work was undertaken by additional resources provided by SunGard Higher Education. This was delivered in June of 2005 but not deemed suitable for production use because of incompleteness of the WSRP 1.0 specification. . It was not until late in the project when a team from Daresbury Labs in the UK did a thorough interoperability test between multiple WSRP implementations that found that there was effectively zero interoperability between the WSRP implementations of different portals (both commercial and open source). Effectively this rendered the WSRP effort nearly useless in the real world.

- Build a set of standard **JSR-168 portlets** that acted as a proxy for Sakai in any JSR-168 compliant portal. These portlets were delivered late in 2005 and continue to evolve as their usage increases. These portlets use web services and iFrames to provide a much better user experience than the simple gallery mode iFrame integration provided by Sakai 1.5.

The timeline and choices that led to these deliverables are explored in some more detail in the Project Execution section below. The uPortal 3.0 deliverable consumed much of the Sakai-provided Mellon funding so it is separately discussed here.

uPortal 3.0

The uPortal technical team was extremely efficient and capable in delivering solutions to the initial deliverables as described above:

- JSR-168 support for uPortal 2.x
- WSRP Consumer support for uPortal 2.x

These deliverables were completed in the early months of 2004 and so the uPortal team funded by Sakai/Mellon turned its sole attention to developing the native JSR-168 uPortal 3.0.

Probably the largest tactical error that we made in setting up the uPortal 3.0 project was to keep it somewhat separate (in a sandbox) from the uPortal 2.x efforts. This slowly created a rift in the community as it removed any dedicated centrally funded resources (Unicon staff primarily) from the uPortal 2.x growth and maintenance. This put the entire management burden of uPortal 2.x onto the community while Unicon was directed to produce uPortal 3.0 as quickly as possible.

If uPortal 3.0 had arrived by the end of 2004, this strategy would have paid off as the community would be re-united with the centrally funded resources and could have moved forward together under the uPortal 3.0 banner.

Unfortunately, uPortal 3.0 too far longer than anticipated. While we hoped that uPortal 3.0 would have been out early in 2005, in truth the first release of uPortal 3.0 is expected to come out late in 2006 - nearly two years after it was expected. This long period caused significant readjustment in all of our plans around uPortal 3.0 and caused increasing friction between the community running uPortal 2.x in production and the Sakai-funded team. It seemed as though uPortal 3.0 would never arrive and all the while it was sapping precious resources from being invested in needed improvements to uPortal 2.x.

There are a number of reasons that the uPortal 3.0 delivery was (and continues to be) so delayed:

- Because Pluto 1.0 was designed as a reference implementation of JSR-168 it was not factored to be used as the basis for a new portal development. Significant effort was invested in the uPortal 3.0 team to clean up the Pluto 1.0 code to use it as the basis for uPortal 3.0.

- The uPortal 2.x community had effectively forked the Layout Manager functionality of uPortal 2.x - there were a number of features in uPortal 2.x that each had their own sub-communities: Simple Layout Manager, Aggregated Layout Manager, and Dynamic Layout Manager. In addition there were layout features such as two-layers of tabs that were also in forked versions of uPortal 2.x. In order for uPortal 3.0 to be successful it not only had to meet the layout needs of Sakai but also meet the needs of the increasingly fragmented layout approaches used across the uPortal 2.x community.

- The Apache WSRP4J reference implementation of the WSRP 1.0 specification was extremely weak and poorly factored for reuse. Throughout the Sakai project attempts were made to work with the Apache WSRP4J project to make improvements both from the Sakai community and uPortal - in general the perception was that the WSRP4J project had no resources and very little leadership.

- While the notion of emulating the iChannel interface using a JSR-168 portlet sounded simple on paper and looked elegant in PowerPoint presentations, it was effectively impossible in reality. As the work progressed on the iChannel adapter, it became increasingly apparent that iChannel based portlets had taken advantage of being in the same web application as the other portlets resulting in a number of very complex dependencies between portlets. Since JSR-168 naturally places all portlets in different webapps, many of these portlet-to-portlet dependencies simply broke with no hope of being solved. This remains unsolved - but in the time it has taken to deliver uPortal 3.0 and with strong support for JSR-168 in uPortal 2.x for the past two years, the uPortal development community has effectively converted to exclusive use of JSR-168 when developing new portlets. This leaves a few remaining important legacy iChannel portlets. At this time it is much easier simply to convert these portlets to use JSR-168 that it is to but full iChannnel support into uPortal 3.0. The sad effect is that the hard work invested in trying to make uPortal 3.0 have perfect support for iChannels was likely wasted effort.

Historically uPortal has responded to expressed community needs and contributed functionality. The work under this proposal differs from prior work because of the coordination needed by others to achieve interoperability. The work consists of four parts. One is the use of OKI OSIDs in uPortal. Another is the development and implementation of the JSR 168 portlet standard, which may or may not be an implementation of the Pluto reference code from the Apache Software Foundation. Some extensions of JSR 168 may be required to provide capabilities now being made available to channel (portlet) developers. Some examples are channel-level authorization, roles, device use, language preferences and a "shim" that would permit current channels to operate, without modification, as JSR 168 compliant-portlets.

- In January 2005, Ken Wiener left Unicon and the uPortal technical team. Ken was the initial developer of uPortal 1.0 and had been the project manager and public face of uPortal leadership for over five years. This began a leadership transition that is still underway in October 2006 (nearly two years later). There are a number of talented technical individuals who continued the development of uPortal 3.0 but there was little overall leadership that represented the entire uPortal 2.x and 3.0 communities. This loss of leadership coupled with Carl Jacobson playing a decreasing role in the project throughout 2005 made it difficult for the community to find a clear new leader and difficult to make focused progress on the broad goals of bringing the uPortal community together from its many 2.x branches and the 3.0 development. In 2006, the JA-Sig board has taken on that leadership role themselves.

The highly oversimplified summary is that uPortal 3.0 was not delivered in time to be integrated into Sakai in any way during the life of the Sakai project (ending December 2005). The Sakai leadership can and should be held partially accountable for this delivery failure.

The good news is that the JS-Sig board is becoming increasingly involved in the strategic directions for the uPortal community and is taking steps to work towards a unified uPortal community which is well aligned with Sakai and understands and addresses the differing needs and use cases of the Enterprise portal and course management system.

Project Contributions - Synchronizing Efforts Across Institutions

One important deliverable in the Sakai proposal was as follows:

> *The third contribution overcomes the barrier of institutional timing by synchronizing the development and implementation clocks of four complex institutions: Michigan, Indiana, MIT, and Stanford. All institutions are committing to an initial implementation of the Sakai tools, as a campus-wide CMS and/or campus-wide enterprise Portal by Fall of 05 when the tools are fully released. Synchronized clocks will greatly facilitate further shared developments in the vears bevond the Sakai Proiect.*

The original goal of the Sakai project was to work together for a two-year period and synchronize the clocks at *four* institutions and get those institutions into production with Sakai.

When the Sakai project completed in December 2005, there were thirty institutions in full production around the world and sixty-seven institutions in some for of early pilot of evaluative use of Sakai.

You can explore the map of Sakai institutions at the following URL:

http://sakaiproject.org/sakai-map/

Sakai developers come from all over the world and each Sakai QA and release effort has significant participation from around the world.

While the level of Sakai distribution is important, it is also important to report on the progress made by the four founding institutions with respect to their deployment of Sakai:

- **University of Michigan** - Uses Sakai as its exclusive CMS system with over 50,000 users. Michigan completely retired its legacy Lotus-Notes based system in the May of 2005.

- **Indiana University** - Attempted to completely migrate all eight of their state-wide campuses in the Fall of 2005 to Sakai. This effort was scaled back because of significant problems with the Samigo testing system in terms of Scalability and reliability. As of Fall 2006, there are 120,000 users in the Indiana Sakai system with 70 percent of classes being taught using Sakai. Indiana is still using the testing system from OnCourse while issues with Samigo are worked out. It is hoped to use Samigo at Indiana in the Fall of 2007 and then commence the decommissioning of the legacy system.

- **Stanford University** - Stanford University has Sakai in production and is in the midst of a multi-year transition from CourseWork to Sakai. Stanford has some technical requirements for Sakai that are not yet met. The Samigo Assessment engine (developed by Stanford as part of the Sakai/Mellon funding) needs a number of features to replace the assessment in Course Work. Stanford has held off implementing these features, placing higher priority on meeting the needs of the broader Sakai community w.r.t Samigo than meeting its own needs. Stanford also needs a far richer support in Sakai for complex Course Management capabilities. An initial implementation

of Course Management will be part of the Sakai 2.3 Release (End of 2006) that should begin to address Stanford's issues. Also it is hoped that Sakai 2.4 (mid 2007) will support Hierarchy - another important Stanford requirement. Assuming these developments come to pass, Stanford can begin a plan to decommission CourseWork in Fall of 2007.

- **MIT** - MIT has had a number of technical reservations about the Sakai Framework - the largest being iFrames and the use of the browser's back button. In order to move forward, MIT has included Sakai tools into Stellar in such a way that meets MIT's usability requirements. This was placed in production in the Fall of 2006. Going forward Sakai is working to meet all of MIT's technical and usability requirements to allow all of Sakai to go into production at MIT.

One interesting pattern among the four core schools in terms of their production deployments of Sakai is that all four schools quickly realized the importance of the much larger community of Sakai adopters and During the second year of the Sakai project (2005) many decisions were made to meet the needs of the broad and growing non-core Sakai community at the expense of the needs of the schools which founded Sakai.

The hope is that by investing in the broader community that in the long run the four schools will receive far greater benefit form the world-wide investment in Sakai than if we simply kept to ourselves for the entire duration of the Sakai Project.

3. Project Execution

The Sakai Project started as a very focused activity that was to collect common resources and produce software for four schools. The schedule was made possible because of a very clear technical approach that was well underway even before the project was funded. This aggressive schedule was possible because the CHEF project provided a flexible configurable base system from which to operate.

The Sakai Project is organized to achieve rapid work products on an aggressive timeline. The means to achieve this is through a large, experienced, and committed team from the Sakai Core institutions. Over the last few years a number of independent groups have been working on foundational open standards and open source software systems for educational domains.

In a sense development of Sakai actually started in October 2003 and the architecture was well defined and the first Alpha release of Sakai was produced in February 2004 and available to developers at the kick off meeting two months into the project.

The basic outline was to take the CHEF product, add support for Java Server Faces (JSF) and the Spring Framework, rework the CHEF APIs, add the OSID API layer described above, then rewrite all of the tools in Java Server Faces. Given that the JSF and Spring work was complete at a prototype level even before the project started, this seemed like a reasonable schedule.

There were several unanticipated challenges that impacted the schedule. The project reacted to these challenges and produced a far superior product than if the original technical plan had been followed exactly as written. Surprisingly, even with the twists and turns with a large and exciting project such as Sakai, a surprising amount of the deliverables were done as described in the project plan.

- The original intention was to *only* support the Sakai version of JSF and limit developers to using the Sakai JSF widgets. The initial goal of the architecture was to dramatically limit the options of a developer at a presentation layer so as to force uniformity across Sakai tools. This inflexibility turned out to be unrealistic and Sakai 2.0 allowed developers to use many different technologies.

- We had not done an analysis of the features of the CHEF toolset before the start of the project relative to CourseWork, Stellar, and OnCourse. We assumed that the CHEF toolset would be sufficient to "get us started". Shortly after the start of the project analysis was done that found many features lacking in the CHEF toolset. This began a process we called "Gaps" - where we listed the features which needed to be added to CHEF so as to be suitable as a replacement for the other three systems. By April 2004, we had become distracted from improving the architecture or rewriting the tools and focused on a plan for addressing the gaps. While many of the gaps were addressed during the project - many still remain unfinished at the end of the project. The remaining gaps have been moved into the Sakai Foundation Requirements process for tracking going forward.

The starting point for functionality in the Sakai 1.0 CMS will be the University of Michigan's CHEF 1.1 CourseTools.NG, which are in production use at UM this Fall with over 5,000 users. This constitutes the rock-bottom, guaranteed CMS feature set for Sakai 1.0, Fall 2004.

- CHEF had not been tested at high levels of scalability. CHEF made significant use of memory based structures, which performed well until the system reached about 10,000 users. It was necessary to re-write the database interaction code to reach our current performance level where we support over 150,000 users. This performance tuning effort lasted from July 2004 through February 2005.

- There was disagreement on the design of the Sakai APIs and the relationship between the Sakai APIs and how the OKI OSIDs should be used in the Sakai Architecture. This discussion started in May 2004 and lasted through February 2005. The discussion slowed progress on reworking the APIs and moving forward.

- The original project design assumed that the separately funded Sakai Educational Partners Program (SEPP) would interact with the partners during the two-year period allowing the core to focus on delivering on the grant proposal. The partner schools did not want to wait until January 2006 to fully participate in the Sakai project - they expected that like any Open Source project that they could fully participate immediately. This new and continuously growing set of stakeholders produced even more requirements. The staff hired for the SEPP were originally intended to "run interference" for the project staff - but they quickly changed their role to becoming strong advocates for deeper involvement of the SEPP and addressing the requirements identified as important to the SEPP members. In retrospect involving the SEPP members early in the project resulted in an improved product and far quicker uptake of the Sakai product - but it did distract resources from the deliverables in the project proposal.

  The SEPP will create an organization, staff, and a set of services to quickly build a broad and sustainable institutional community for Sakai Tools. The SEPP is critical to coordinate communication and cultivate institutional readiness while allowing the Sakai Project Core team to focus on Sakai 1.x and 2.x deliverables.

- The Sakai Project effectively expanded its "core" with the early addition of the Etudes Consortium at Foothill College and the University of California Berkeley to the Sakai Project board. These organizations initiated the Melete and GradeBook projects that effectively became core tasks, creating architectural demands and spreading a thin core staff even thinner. Again, great benefit came from these additions as Melete fills a very critical functional gap within Sakai.

- When the Sakai 1.0 beta was released in June 2004, it underwent a technical review by the uPortal team and the MIT team. This review was pretty negative in terms of its assessment of the "elegance" of the Sakai 1.0 architecture. This review led to a meeting in August 2005 where there was significant discussion about whether to invest our time and energy in improving or rewriting the framework or addressing the functionality gaps in the product. This resulted in the Sakai 2.0 architecture and a full rewrite of the Sakai framework. This rewrite happened in two phases - the first was from January to June 2005 resulting in the Sakai 2.0 release - the work was completed in a second phase from January to June 2006 resulting in the Sakai 2.2 release.

- Early in 2005, the Open Source Portfolio project decided to merge its code base into Sakai as a long-term sustainability strategy and to better synchronize releases between the projects. OSP required a number of architectural improvements to be fully integrated into Sakai. The Integration of the OSP software lasted from May 2005, through June 2006.

It is important to realize that while each of these challenges took resources and time, they all resulted in a far superior product and a far more adoptable product. Responding to these challenges is what made Sakai the product and community it is today.

Deliverables, Tasks, and Timeline

The following table shows the planned schedule and deliverables and the actual deliverables for the project. As you can see from comparing the entries, the technical direction and even scope of the deliverables was altered as we faced and resolved challenges. But through it all, we followed the deadlines and intent of each of the releases.

Software Release Schedule:

Planned Schedule	Actual Schedule
Sakai Alpha Release: December 15, 2003 (Core) • *to start to play with developing Sakai applications; not ready for serious development.* • *basic support for tools (JSR 168 portlet), views (jsp, velocity) and services (OSID implementations and Avalon Framework services).* • *test implementations of some CHEF and OKI OSIDS. .*	Various source releases from December 15, 2003 onwards.
Sakai Beta Release: (Feb 15, 2004 Core; April 30, 2004 SEPP)) • *to start integration of applications under development; not ready for serious deployment.* • *fleshed out tool, view and service support.* • *full test implementations of CHEF and OKI OSIDS, some production implementations available.* • *portal engine availability, limited configurability and integration.*	Sakai Technology Preview: February 15, 2004 • Based on CHEF • Complete removal of Jetspeed and replacement with emulation layer and simple iFrame based aggregation • Support from Java Server Faces 1.0 • Replace Turbine with Spring
	Sakai Alpha 1.0: June 2, 2004 • Improved packaging and installation • uPortal 2.x sample integration • Sample tools using JSF and Hibernate • Quick Start
Sakai 1.0 release: July 1, 2004 (Public) • *to start pilot Sakai service installations.* • *full support for the Sakai model for tools, views and services.* • *full production implementations of CHEF and OKI OSIDS.* • *fully integrated and configurable JSR 168 Portal.* • *CMS, Research Tools, Assessment Tools, Calendar, Collaboration Tools*	Sakai 1.0 Release: October 31, 2005 • Based on software running in production at Indiana and University of Michigan • Sakai JSF Widget set • Some Gaps Addressed • uPortal removed from release • Internal iFrame based aggregator (Sedna) • Generic WorkSite setup • Improved performance - Tomcat 5 • Cross-webapp Spring service injection
	Sakai 1.5 Release: March 25, 2005

	• Performance fixes • Group Provider • Realm Provider • Samigo released separately • Rewritten internal aggregator (Varuna) supporting gallery mode of Sakai • Syllabus, Help, and Profile Tools • Emergent new Sakai APIs.
	Sakai 1.5.1 Release: May 27, 2005 • Samigo integrated into the release • Performance improvements
Sakai 2.0 release: May 1, 2005 (Public) • *(Note…interim development releases to Core and SEPP through 04-05 academic year)* • *uPortal 3.1 - refined* • *refined Tool Portability Profile with full production implementations of CHEF and OKI OSIDS.* • *refined CMS, Research Tools, Assessment Tools, Calendar, Collaboration Tools, Workflow*	Sakai 2.0 Release: June 15, 2005 • Tools 80% style guide compliant • New internal aggregator - Charon • Improved Skin • Completely re-written framework 2.0 • Internationalization • Gradebook • Web Services Support • Architecture design documents • Development overlapped between 1.5 and 2.0 • Coordinated Community QA • Improved install documentation
Sakai 2.1 release: November 1, 2005 (Public) • *refinements and bug fixes*	Sakai 2.1 release: December 2, 2005 • WSRP Producer • Community Driven Release and QA • Resource tool with meta Obj • Group and Section Support • Course Site Template / Student Role • Chinese / Korean / Dutch / Japanese • In Porgress: Danish / Hebrew / Portugese / Slovikian /Catalon / French / Spanish • MySql Performance Improvement • Improved Providers • SakaiScript • Become User Tool • Roster Tool • Wiki • Repository OSID / TwinPeaks

The above table shows that the project had some periods that we faced challenges and other periods where we had resolved significant issues and made excellent progress. This is a quarter-by-quarter summary of the project status on technical matters:

- **1Q04** - Project was running smoothly - because much of the work had started in October 2003, we had a head start on making the necessary technical choices to move away from Velocity, Turbine, and Jetspeed - we were able to produce a working prototype release which had some support for the new approaches in February 15, 2004 - at the project kick off meeting.

- **2Q04** - Significant effort was spent in looking at the functionality gaps in the CHEF tools and trying to work on some of the gaps. We hoped to fix many of the gaps over the summer. There was significant discussion about the Sakai APIs and whether they would be derived form the CHEF APIs, be the OKI OSIDs, or be re-engineered from scratch. With our first SEPP Meeting in Denver with 180 attendees we released the Sakai Alpha 1 release four weeks late. We had held up architecture development for about six weeks while having architecture discussions so the Alpha release was less mature than we had hoped.

- **3Q04** - Over the summer, we continued working on the gaps and attempting to make the release suitable for general use. By mid-summer we realized that our approach to use uPortal as the presentation framework would be too damaging to uPortal so we scrapped that work and shifted direction to WSRP for portal integration. In August we realized that the system would not scale much above 5000 users because of the memory-based structures. We also had a retreat in Indianapolis that included the Board, Architecture and Tools team where the primary question was whether to continue to improve functionality or to invest significant effort in the framework. The agreement was to spend Fall 2004 on functionality and to begin the framework after January 2005.

- **4Q04** - With Michigan and Indiana in production, we focused on issues or performance and other issues that arose. The Sakai 1.0 final release came out at the end of October - this was a very solid release as it had the patches form six weeks of Indiana and Michigan production. Several sites ran Sakai 1.0 for several years before upgrading. Toward the end of the year we focused on the 1.5 release which would have all of the performance fixes and the Samigo Assessment engine.

- **1Q05** - The Sakai 2.0 rewrite started promptly in January as QA and other issues were worked out for 1.5. The 1.5 release came out with Samigo as an add-on as we simply ran out of time. The 1.5.1 release came out later with Samigo integrated in a single release. Sakai 1.5.1 was the last release where we released *after* the software was in production. A number of sites took Sakai 1.5.1 and put it in production - OSP 2.0 was based on this Sakai release.

- **2Q05** - As Sakai 2.0 came together - the framework looked very nice - there had been extensive design documentation written as Sakai 2.0 was developed. In April and May we moved the applications from the Sakai 1.5 framework to the Sakai 2.0 framework. This was the first Sakai integration Week May 16-20, 2005 - where the Sakai teams assembled and performed the final integration of the tools into the 2.0 release. Sakai 2.0 was released June 15, 2006 - right on schedule. Sakai 2.0 was the first Sakai release that (a) happened on schedule and (b) happened before it was put into production. Many schools installed Sakai 2.0.

- **3Q05** - The Sakai 2.0 release began a transition in technical approach and governance as Sakai prepared for the transition to the "post-project" phase. Increasingly developers and QA folks from outside the core team were becoming involved in the development and release of the product. With a stable framework for the first time, the whole project was quite productive. A decision had been made to merge the source trees of the Open Source Portfolio and Sakai projects so as to have a sustainability model for the OSP code after the

Over the two years of the Sakai Project, the participation of the partners will progressively increase, as the initial Sakai software is released and the conditions for increased community participation are put in place. As this happens, the board will evolve into an institution more representative of the community by bringing selected partners onto the board.

OSP grant finished. Over the summer, Indiana University produced a version of Sakai 2.0 that also included OSP 2.0.

- **4Q05** - A large number of sites came up for the North American Fall semester. Indiana University tried to shift nearly their whole population into Sakai. This led to some performance tuning "opportunities" and showed that the Samigo product has significant weaknesses and issues with scaling.

Overall, the most challenging phase of the project was between May 2004 and May 2005. During this period we had to work out the architecture, governance, relationship with the partners, and build and rewrite an entire framework. The delivery of the Sakai 2.0 release was a powerful turning point in the project where things had become clearly aligned and the software was quite solid and has allowed the community to open up and grow around the world.

Navigo Project Assessment Tool

The original intent was to simply port the Navigo Assessment tool into Sakai by the summer of 2004. Navigo was a project that pre-dated Sakai and was well underway when Sakai started. Navigo was written in Struts. One of the first goals was to merge the efforts of Navigo and the Stanford Assessment Manager (SAM) and port the combined product to Java Server Faces (JSF). The combined product was jokingly given the name "Samigo" in one project status report in the summer of 2004 and the name has stuck.

Beyond the CMS, the Navigo Project's Assessment Tool will be conformed to the Tool Portability Profile for Summer 04.

This turned out to be far more daunting task than originally imagined. Initially the work was done by a combined Indiana / Stanford team and later the project was handed to Stanford for completion. Because the Sakai Framework was still evolving Samigo was initially developed for most of 2004 as a standalone web application. Toward the end of 2004 it was ported into the Sakai framework. The integration of Samigo into the Sakai 1.5 release was less than elegant - in Sakai 2.0 the integration was much better.

Stanford continued to work on Samigo throughout the entire project and continues to support Samigo after the project has been completed.

4. Project Budget

The Mellon Foundation has been provided with a complete budget report from the University of Michigan.

5. Post-Project Sustainability Plan

Sakai Commercial Affiliates
Second, by establishing the Sakai Commercial Affiliates as an organization composed of commercial partners interested in providing support for the Sakai software, as a group of "Sakai Redhats", we have helped to insure the existence of a wide range of commercial offerings, from consulting to complete ASP services, for those wishing them. This is a critical need to get Sakai software available to smaller schools and those not wishing to run the software themselves. An example of the success of this is shown in the adoption of Sakai by Appalachian College through the efforts of the Longsight consulting firm. It also gets some larger institutions over the hump of adoption of open source software, giving them someone to call late at night if they encounter problems. In short, the SCA lowers the adoption threshold for a number of schools and organizations and increases the likelihood of wide scale adoption. SCA members have also emerged as prime contributors to the development of the Sakai code base and community, with contributions from many SCA partners and sites like the Unicon "TestDrive Sakai" site providing valuable exposure for the Sakai software (https://www.academusopencampus.com/registration/register/index.php).

Working with Industry Organizations
Third, by helping to initiate the IMS Working Groups on Tool Portability and Common Cartridge, and working on a charter that is targeted at collaboration between providers of course management software, proprietary and open source, we have engaged this community effectively in the areas of mutual interest that will be of most benefit to the larger educational community, without getting bogged down in unproductive debates surrounding definitions of open standards implementations and standards compliance. By working with, among others, like publishers, WebCT/Blackboard, in developing actual examples of tool portability between all our systems, we are advancing the practice, as well as the theory, of tool portability, to the benefit of adopters of any of our systems.

All this of course points to a whole dimension of effort in the Sakai project, and especially the SPP development. We are not only developing software and methods as we go along. We are also developing, experimenting with, discarding some and choosing between, practices that will result in a sustainable model for Community Source.

The board must engage in efforts to build a community of adopters that will install the software at their institutions and will pass the knowledge and practices gained in doing so back into the community knowledge base.

Sustainability – Building Community Resources
The first years of the Sakai Partners Program have seen the steady and measured growth of an increasingly dedicated community of software designers, software developers, IT administrators,

pedagogical technologists, User Interface and Usability experts, and support staff. Active and effective Discussion and Work Groups have arisen, and more will as we move along. Workgroups dedicated to adding functionality and reviewing existing software and solutions have been formed, and the strengths of the community are increasingly tapped and coordinated through the various Sakai practices.

The Sakai Foundation Board strongly believes that this is the way to build an enduring community of support within the higher education community: from the ground up, including staff and leadership from all levels in every stage of development. This rich mixture of commitment, experience and expertise from all levels of the individual academic enterprise and the academic community as a whole is necessary to move beyond traditional models of isolated software development, outsourced development, or mandated institutional collaborations, that are generally not picked up by a community of support, and toward a set of methods that increase the probability of sustainable success of open source efforts. This is not achieved by the application of resources alone, but by mobilizing the interest and commitment of staff and leadership throughout the academic organization. The open source model we are pursuing is founded on a mix of dedicated resources and volunteerism based on both enlightened self-interest and a knowledge that contributed work can never be taken away, that the source remains open.

A community needs trained developers who can contribute to the open source base with fixes and extensions to existing tools or contribute new tools that use Sakai's Tool Portability Profile. Support for this effort has been proposed through the Sakai Educational Partners Program.

The success of these first years in developing a resource base for the Sakai Foundation and in involving and exciting a diverse, international community of developers and users, from large research institutions to small liberal arts colleges to community colleges, leads us to feel confident that the future will see even more progress. Add to this the continuing commitment of core schools and the deepening commitments of new institutions and it is clear that we are well on our way to meeting our goals of having a vibrant and self-sustaining community-based organization to see this community into the future.

6. Self Assessment

In this section, we give ourselves a "grade" and some commentary on the major deliverables and thrusts of the project.

Deliverable	**Assessment / Comments**
Community Development	A+ In many situations Sakai made choices that would expand the community as a top priority. Instead of waiting 18 months to being to "open" Sakai to a broader community, the broader involvement started within six months of the start of the project. While this distracted us from many of our core deliverables it resulted in a very solid sustainability plan under the Sakai Foundation.
Code Mobility Across Institutions (TPP)	B- The Sakai software does provide a rich and powerful ecology for tool and service development and interchange. The recent innovation coming from

	around the world is strong evidence of the success of this approach. The selection of JSF as the presentation layer for the TPP has made the road much more challenging for the TPP. The proposal said that JSR-168 would be a founding part of the TPP - this is not yet the case. Going forward there are plans to evolve the TPP forward beyond JSF and to JSR-168 and ultimately deliver on that promise.
uPortal	C+ This project was the catalyst fro the uPortal community to move from iChannel to JSR-168. The project funded a number of investments into uPortal 2.x that worked very well and have been quite useful to the uPortal community. The uPortal 3.0 effort was far more challenging than anticipated and has yet to bear fruit. Work on Portal 3.0 continues and there is a roadmap forward that will hopefully deliver uPortal 3.0 after the project phases has ended.
OKI	C+ The OKI OSIDs did not end up as the "foundation" for the Sakai APIs due in a large part to the desire not to produce APIs with out of band agreements. The Sakai APIs are highly aligned with the OKI OSIDs. Sakai has integrated a number of the OKI OSIDs and has successfully used the OKI OSIDs for repository integration efforts. As OKI moves forward with the Repository and other OSIDs, Sakai remains an excellent platform to make use of OSIDs as they are developed.
Delivery to the four schools	B- Sakai has always been "playing catchup" when it comes to matching the capabilities of OnCourse, Stellar, and CourseWork. There was a continuous choice that we faced in investing in framework or features. We probably lost 50% of the project time due to working through framework issues including the entire framework being thrown out and rewritten once. This will ultimately be a success - but with an extended timeline for some schools.

7. Going Forward

The Sakai Project provided the starting impetus that now has evolved into the Sakai Community and Sakai Foundation. During the project period, often choices were made to maximize the long-term viability of Sakai rather than the short-term features / deliverables. This leaves us with a solid foundation for the future. This section will allude to some hoped for post-project efforts and deliverables.

- uPortal 3.0 should be delivered in December 2006 - this will include solid support for JSR-168 and WSRP. With this product delivered the effort can begin to transition the community from uPortal 2.x to 3.x. Sakai can revisit the integration of Sakai into uPortal.

- A great deal has been learned in the two years of the Sakai project about the relationship between a course management system and a portal. Our initial instinct was to simply put all of the functionality of the CMS into the portal - instead most enterprise portal deployers would by far prefer that Sakai export its data ins standard formats like RSS and iCal so as to be presented this information in the portal. If the user wants to use Sakai - the portal can simply redirect them to Sakai. Sakai is working on providing these feeds to enterprise portals as a high priority task.

- Sakai is adding JSR-168 support using Pluto 1.1 - Pluto 1.1 has been significantly refactored making it far easier to add to an existing application. JSR-168 support should be available as a provisional feature in Sakai release 2.4 - this allows general portability of JSR-168 portlets between any standards compliant and Sakai. This is expected to encourage developers to write simple Sakai tools using JSR-168 rather than JSF so as to allow those tools to be deployed either in Sakai or in any JSR-168 compliant portal.

- Sakai is involved in the follow on standard to JSR-168. The JSR-286 standard is an extension of JSR-168 and addresses many of the concerns that we had with JSR-168 that prevented its use in Sakai. Since JSR-286 will be part of Pluto 2.0 - it is likely that Sakai will be able to support JSR-286 by the end of 2007.

- Sakai was instrumental in creating the IMS Tool Interoperability standard and will be participating in the evolution of this standard going forward - this provides the promise of creating a TPP-like standard which will enable a tool to be written that is portable across many systems including both commercial and non-commercial. Support for IMS Tool Interoperability will be in an upcoming release of Sakai.

- Sakai was instrumental in creating the IMS Common Cartridge standard that allows for publishers to produce course material for import into Sakai in a standard format that works across both commercial and non-commercial systems. Sakai will provide export support for Common Cartridge which will enable far more natural exchange of course materials amongst the faculty and greatly enhancing learning object repositories such as Merlot and Open Course Ware (OCW). IMS Common Cartridge import will appear in Sakai 2.3 and IMS Common Cartridge export is planned for Sakai 2.4.

- Sakai is committed to improving the usability and accessibility of Sakai. Sakai 2.3 made great strides in improving accessibility and this is expected to continue.

Overall, Sakai is well positioned to achieve nearly all of the outcomes in the original Sakai proposal - it will simply take somewhat longer than originally anticipated.

8. Conclusion

The Sakai Project intended to act as a catalyst to bring universities together to create an open source course management system. The idea was to take two years, and develop a product with a dedicated team from four universities and two open source projects and then build a community around that product.

The idea of an open source course management system was so compelling that almost no one wanted to wait to create that community. This led to Sakai having to "build the bike" while we were riding. Given the amount of upheaval during the project including one major rewrite in the middle of the project, we far exceeded the expected impact of the project.