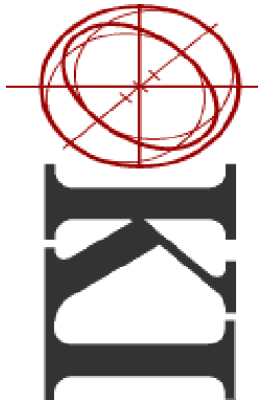


OKI Architecture Overview

by Scott Thorne, *OKI Lead Architect, MIT,*
Chuck Shubert, *OKI Developer, MIT,*
and Jeff Merriman, *OKI Team Leader, MIT*

March 22, 2002

[Note: This is still a draft document – while it has been reviewed for accuracy, it is not yet complete and may undergo structural and graphical changes in the future. Please stay tuned for versioned updates.]

The goal of OKI is to define an architectural framework that facilitates the development and delivery of educational software applications. This architecture is being generally described by two service layers defined by implementation independent Application Programming Interfaces (APIs). One, Common Services, provides hooks to institutional infrastructure and other fundamental services that provide a backbone for higher level services and applications. The second layer, Educational, will bundle functionality that is of particular usefulness to the developers of various kinds of educational software applications.

By defining Application Programming Interfaces (APIs) that are not bound to any one implementation of a particular service, OKI intends to provide a boundary layer that buffers educational software from infrastructure that is localized or that might go through changes that would otherwise require major re-writes of code at an application level. The services themselves are also intended to be modular, bound together loosely through shared objects and interfaces.

The goal of all this is to facilitate development and sharing of educational software applications, which OKI commonly calls "Tools." Development is facilitated through providing a rich set of services allowing developers to concentrate on the real pedagogical aspects of design and not about basics like how to authenticate a user or where to store documents and metadata. Sharing is facilitated through the abstraction provided by APIs, allowing an application built at one institution, using a particular collection of infrastructure services, to be easily transported to another.

Overview

The Open Knowledge Initiative is defining an architecture to facilitate the construction and use of educational applications for the next decade. The implications of this goal are enormous.

Educational applications touch a wide variety of user interfaces. These range from simple maintenance and delivery of content to tele-presence control of lab equipment to immersive simulation environments. These various user interfaces may perform simple administrative tasks, such as managing class membership, or more advanced tasks like time-based annotation of streaming video. Improvements in connectivity and computer performance will continue the

evolution of user interfaces. This evolving front means that an architecture too closely bonded to a particular user interface will ultimately fail to meet the future needs of courseware developers as technology evolves and pedagogical models become more sophisticated.

Current educational courseware uses traditional institutional services such as course registration, posting of grades, authentication and authorization, etc. It is a small step from these more traditional services to more advanced ones, such as integrating library services to protect intellectual property rights and control distribution of course materials. The real challenge in this domain is to create an architecture that allows courseware applications from one institution to run at another institution through a converging set of service definitions, while allowing dissimilar implementations of those services. In the future, institutions are not likely to adopt courseware technology that is inconsistent with growing investments in technological infrastructure.

The OKI architecture must accommodate diverse environments and growth in both technology and pedagogy. It must isolate the courseware tools from infrastructure services. It must also isolate one courseware component from changes in another courseware component. It must scale to handle the demands of large institutions with many students, faculty, and courses on multiple campuses, or nonstandard courses and participants.

OKI has all the noble design goals that are common among systems development projects: Open, scalable, secure, reliable, flexible, and extensible. In addition, OKI foresees the creation of a stable, sustainable, and viable platform on which to extend new educational applications and learning systems. Several years in the future we won't know what kind of computers and other devices will be prevalent. Our goal is to keep the OKI design as technology independent as possible. As hardware platforms, operating systems, and other technologies evolve, OKI should adapt with them.

Design Philosophy

The OKI architecture will meet the challenges presented by the constant evolution of technology by using an innovative adaptation of a traditional software development technique. The technique essentially involves collecting bits of functionality into a bundle and defining a limited number of ways to access that bundle. The adaptation of this traditional technique focuses on how the functionality is implemented. Traditionally, access to a function has been tightly bound to the actual implementation of that function. OKI's architecture will loosely bind access to a function with its implementation. In fact, it will be possible to change the implementation of a function that an educational application is executing. This dynamic binding of function access to implementation is at the heart of the OKI architecture. We call the definition of the access to a bundle of functionality an OKI Application Programming Interface (API)

Standards are also key elements of permanent solutions, unfortunately creating new standards is slow work, and it sometimes takes years before this approach yields working implementations. However, just knowing or predicting that there will be standards in an area and that they will continue to evolve is enough to guide OKI. By anticipating an area where common services or standards will emerge OKI can structure interfaces in these areas, so that over time these emerging standards can be introduced without negatively impacting the other areas of OKI.

Implementation independent APIs are a major feature of the OKI architecture. They provide a way to manage change, adapt to new standards by providing a predictable boundary to the system. Generic interfaces are hard to get right, and we expect that they will need to evolve for some time. [Many of the current standards are specifying data interfaces only. Although this is necessary, it is not adequate to provide real time integration between systems.] Currently efforts exist that are specifying data interfaces only in support of educational technology. The IMS Global Learning Consortium has been

OKI API's

An API is an application programming interface. It gives a precise definition of a particular service within an application, so that a programmer can use that functionality without having to know how it's implemented.

There are several benefits that flow from an API approach. The most important benefit is that the

work of building an application can take place independently of the services it will require from the API; the programmer doesn't need to know how the API-accessed service is implemented, including what kinds of network transport mechanisms are being used or even whether network connectivity exists at all. Another important benefit is that more than one implementation of a service is possible without requiring the application program to change. So long as an implementation of a service maintains the API, implementations can vary without requiring any changes in an application using the API.

Some examples of services that are familiar to most applications and suitable for an API, include database access, file access, authorization, authentication, messaging, logging, workflow, registration and so on. Examples of additional API services that might exist in OKI are publishing, collaboration, and assessment.

How does an API work?

An API functions through the definition of the service it provides. To define an API it is necessary to define the objects that make up the service. For example, an Authorization API would contain objects such as a Person, a Function to be performed, a Qualifier that gives the context for performance of the Function, and the Authorization itself. Defining the objects involves not only naming them, but also determining what information they contain. To complete the definition of an object it is necessary to define how the information in an object can be accessed.

Definitions of objects associated with an API go a long way toward defining the rest of the API. For any collection of objects there is set of things that can be done with the objects. Examples of things that can be done with an Authorization API include: Determining who can perform which functions in a qualifier context, or determining who has authorization to perform a function.

Most of this work can be done outside the constraints of a programming language, but for a programmer to be able to use the API, it must be cast into a programming language. We will do this casting into Java; the API object definitions will become Java interfaces. The advantage of a Java interface is that it is well defined and the programmer can use it when writing the application, but it does not imply a specific implementation.

Functions performed with API objects are collected into a Java class that is called a Factory. The Factory collects Functions and binds them to an implementation of the API. The API objects are represented with Java interfaces. The Factory provides a way for the application to get instances of these API object interfaces. After an API is created, an application may access any implementation of the API through the Factory.

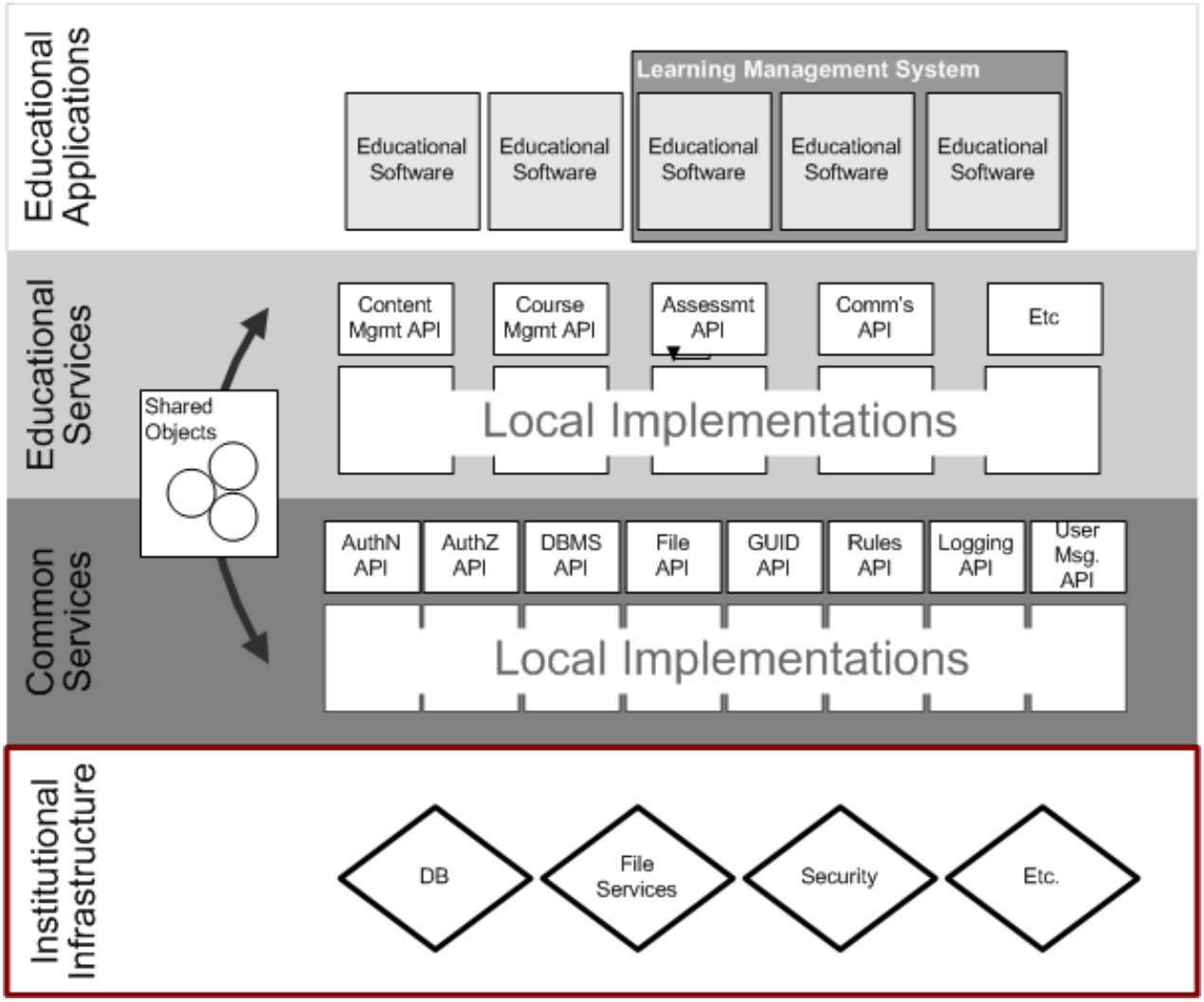
Architecture

The OKI architecture is founded on a number of features that will enhance its adaptability, and ensure a broad variety of uses for many different pedagogical systems.

Modular

By creating several distinct modules of functionality with well-defined interfaces, OKI gains several advantages. It makes it possible to more easily upgrade a specific functional area without impacting the entire system. Also entirely new modules of functionality can be added without negative impact on the system. The difficulty in this approach is to cleanly partition functionality.

OKI Architecture



Common Service API's

Many application sectors need the same sets of functionality. For instance, there is a common need for authentication in most applications. But while there is a common need for authentication, there are many ways to provide it and several standards already exist in this area. In addition in an area such as authentication, many enterprises have enterprise wide systems that should be leveraged against enterprise applications.

It is through this Common Services API layer that OKI will allow for integration with enterprise infrastructure and help to assure adaptability to multiple and evolving standards and changing technology at this level. API modularity is designed to allow multiple OKI sites, say various schools at the same institution, to share some common infrastructure, like Authentication/Authorization, while keeping others local, like file management.

Common Services currently being described include, but are not limited to:

Authentication

> The authentication process gathers required credentials from an agent, vouches for their (the credentials') authenticity and introduces a person (known internally as an Agent) to the system. The OKI authentication API abstracts this process allowing the OKI application to determine and manipulate the authentication status of a person without having to manage the details of a particular institution's environment.

Authorization

<short description will go here>

DBC

The database API allows an application to access and update the contents of a database. The intent of this interface is to allow actual connection to the database to exist on a machine other than the machine hosting the application. The intent of this API is to leverage the java.sql package definition of database access functionality. By using Java interfaces and the SQLFactory class an API is created that allows database API objects to migrate across machine boundaries. This ability for database objects to move between machines is lacking in the java.sql package or through use of database drivers alone.

File

The OKI file service provides platform independent storage and handling of files and directories. Files and directories are associated with meta data such as owner, mimetype, quota and versioning in this service. This distinguishes the OKI file service from standard file systems.

Local Unique Identifier Service

This is a simple service to be used for generating IDs to represent different entities in the system, such as a file, log, etc. The Locally Unique Identifier Service is used to generate an id local to a specific OKI system. One or more services in a given OKI system may use the local identifier service internally, but this identifier will not be exported. LUID is the common name given to this identifier

Rules

The Rules Service is a generic service that supports the storage, maintenance and access of tuples of information of various Rule Types. Tuples are in the basic form of subject, verb, predicate. Rules could be of several types such as, Authorization, Workflow, Intellectual Property, Presentation, or User Preferences. The predicates or objects can be linked in a hierarchy to more efficiently maintain them.

Logging

The Logging Service will be used to capture and access usage information. It will track events throughout the system for diagnostic, performance: metrics, tuning, benchmarking, monitoring, security, performance, data collection, usage or trend analysis, forecasting, and statistics.

User Messaging

<short description will go here>

Shared

The Shared service describes a number of interfaces and entities that interact with, are utilized by, and help to tie together the OKI Common Services. Examples of entities include, a person, a requesting service, an external OKI instance, or an OKI tool.

Educational Component API's

This layer of OKI services is where functionality of value to educational applications will reside.

These services will likely be built on the lower level APIs but may also include hooks to other enterprise services, like Student Information Services (SIS) or Digital Libraries. As of the writing of this white paper, the OKI architecture team has begun to design the various shared objects and interfaces that will support these services. However, services relating to Course/Class Management, Content Management, Assessment and Communication are anticipated.

Expect further description of these services and related interfaces as this work progresses.

Java

We plan on using Java 1.3 to define all APIs and as an implementation language. Although the API could be defined in several other ways, OKI will use Java as we stabilize functionality. In the future other language bindings for the API could be created. We are trying to make our APIs callable from various contexts, so that they could be used to deploy servlets, applets or applications.

OKI to OKI Service

Although ultimately any learner might want to interact with any OKI instance, we are going to approach this area with a major simplifying assumption; that users will interact only with an OKI instance, which is in their local "domain". Remote content (content served from an OKI instance in a foreign "domain") will be passed through the local server on the way to the user. This eliminates the need for a common authentication and authorization mechanism across all OKI sites. Later as common authentication & authorization systems are deployed, OKI can take advantage and deal more directly with cross-domain usage. Although this interaction adds a hop to the data flow, it is assumed that cross domain usage will be a small percentage of the initial use, and there are advantages to each site dealing with its own users, such as capacity planning and user support.