

## Guiding Principles for Technical Architecture

This document is a statement of the principles that will guide the technical development of the Kuali Student system. It will serve as a reference throughout the full lifecycle of the project. While these principles are open to amendments as the project progresses and as lessons are learned, every effort should be made to preserve these ten guiding principles. As a means of preserving these guiding principles, the Change Management Process will be used to amend this document and to make exceptions to the principles while building Kuali Student.

The following are the ten guiding principles for Kuali Student.

### Service Oriented Architecture

#### Principle 1: SOA Methodology

The first characteristic of Kuali Student is that it is based on Service Oriented Architecture (SOA). Although many Java Enterprise applications are “service” based in that objects are accessed via service interfaces, the SOA approach is different in several respects:

- There is a greater emphasis on the up-front design of entities and service contracts.
- The artifacts of the design phase are entity models and service definitions.
- Services should be autonomous; they are not controlled or constrained by another service and therefore may run remotely. From a development perspective, there will be a strong presumption in favor of building services that can be deployed remotely. There will however be cases where this is impracticable for performance, security, or other reasons.
- Services should be loosely coupled; they are modeled and exposed through an interface that is separate from its implementation. Through loose coupling, services can be implemented in any environment as long as the implementation fulfills the service contract.
- There is a high degree of emphasis placed on the identification of re-useable services.

An example of the benefit of SOA is the design of the authentication system for Kuali Student. By designing services (via service contracts) that are autonomous and loosely coupled it's possible to plug in any authentication system (e.g., CAS, home-grown, etc.) that implements the system's service contracts.

#### Principle 2: Web Services

The preferred implementation of the SOA is web service technology. Web services have the advantages in their simplicity, their universality, and the fact that they are platform neutral. “Web services” means SOAP (Simple Object Access Protocol) and WSDL (Web Service Definition Language). Apache Axis is a good example of a product that

implements these technologies. XML schema is the primary vehicle for expressing entity models and service contracts. In short, "XML is the platform."

### Principle 3: Standards Based

Kuali Student will follow open standards wherever feasible. The standards observed will be in the following areas (and others where applicable):

1. The W3C Web services framework (SOAP and WSDL)
2. Kuali Student will follow WS-\* standards where they apply (e.g., WS-Security, WS-Transactions, WS-Addressing, etc.).
3. Industry standards such as those supported by PESC-AACRAO
4. Java Community standards such as JSR 168 (Portlet), and JSR 94 (Rules Engine)
5. Internationalization standards

Standards compliance is a key issue in product selection. For example, it is important that the Kuali Student Web-services engine implement the latest SOAP and WSDL standards. A product's adherence to the latest standards has the additional benefit of freeing the Kuali Student technical team from the constraint of being continuously up-to-date on every change in a standard's definition.

It's understood that standards will evolve over the course of the project. While it's generally a good idea to maintain Kuali Student's adherence to the latest standards, the technical team should be prudent in its attempts to keep pace with the latest standards as doing so may have a negative impact on delivery of the product.

### Principle 4: Separate Governance Process for Service Contracts

Service contracts are business assets of an SOA-based system, are the public definition of the system, and must be the most stable part of the system. They are governed separately and differently from that of typical development artifacts in that the governing body has representation from each service domain, the involved business units, and technical subject matter experts. This body can be decentralized where each unit is responsible for their services and those they want to make available to others, or it can be a centralized body that reviews all new, modified, or retired service contracts, or the organization can be somewhere in between. The governance body's organization will be specified in the methodology section of the Kuali Student Charter.

The management of service contracts can be extended to external contracts as there may be cases where Kuali Student consumes stable 3<sup>rd</sup>-party services (such as the validation of addresses by a postal service).

Service contracts created by an institution (i.e., for the purpose of customization or for the consumption of external services, for example) will be maintained by the institution. Service contracts contained in the reference distribution will be maintained by Kuali Student.

## Component Abstraction

### Principle 5: Abstraction of Business Processes and Business Rules

Business rules and business process logic will be abstracted from the code base.

- Rules engines are the preferred vehicles for abstracting business rules
- Workflow and BPEL engines are the preferred vehicles for abstracting business process logic.

The use of rules engines takes the traditional separation of business logic and presentation in an n-tier architecture one step further by externalizing business logic. Changes to business logic can be defined to the system without any programming changes. Routing logic for the workflow engine is also expressed through the rules engine. An important part of the project will involve developing interfaces for the rules engine(s). Rules engine technology also introduces the possibility of artificial intelligence in the system since one of the outputs of a rules engine can be a new set of optimized rules.

The Kuali Student reference distributions will include standard templates for common business practices that may be extended by implementers to meet the particular needs of their institutions. As an example, the template “calculateGPA” could be extended to meet the specific rules of an institution in the calculation of a student’s GPA.

In cases where the business process and business rules vehicles are insufficient there will be clear guidelines on how to abstract business logic in the code (as dictated by the Change Management Process).

### Principle 6: Abstraction of Presentation Layer and Use of an Open Source Portal

Abstraction of the presentation layer allows User Interface (UI) components to be separate from the orchestration layer and the business service layer. To illustrate, a benefit of the abstraction is the ability to deliver the UI to a wide array of devices (such as PCs, cell phones, PDAs) without modification of the presentation layer’s input stream to accommodate the differences in each.

To help facilitate this approach, Kuali Student will be delivered through an existing portal product. Using a portal provides the opportunity to abstract the presentation layer through standards (such as JSR 168 and WSRP where there is a need for remote portlets).

Portal functionality such as provisioning, customization, and personalization will remain within the domain of the portal rather than within the scope of the Kuali Student project.

### Principle 7: Abstraction of the Data Layer

Data abstraction is implemented in three ways:

1. Much of Kuali Student’s data model will be derived from simple abstractions; that is, abstractions representing basic concepts and objects such as time, people, learning

units, and learning results. Implementing these abstractions as identifiable domain objects (real-world objects like courses, sections, students, and instructors) is carried out through a series of configuration templates.

2. Data access will be abstracted in the data layer. The purpose is to provide database independence, allowing any ANSI SQL compliant database to be used while still allowing database-specific calls to be made within the data layer for things like performance enhancements.
3. Data access should be abstracted through an ORM framework and as a rule it will be services that provide data. Consequently, applications that need data do not need to understand the details of database navigation.

## Leveraging Open Source

### Principle 8: System Will Be Built Entirely On An Open Source Software Stack

#### Licensing and IP Management

Kuali Student will be built entirely on an open source software stack compatible with the outbound Educational Community License (ECL). Further, Kuali Student will adhere to the Kuali Foundation's IP management policies for inbound licensing and assessment of 3<sup>rd</sup>-party licenses.

#### Open Source Reference Distribution

Reference distributions of Kuali Student are entirely open source. The reference distributions of Kuali student are entirely open source. That, however, does not preclude implementers from swapping in commercial products for part of the stack. For example, an institution that has a deep investment in Oracle may wish to continue using Oracle for Kuali Student.

### Principle 9: Infrastructure Will Be Composed Of Existing Open Source Products

#### Use of Open Source Infrastructure Components

It is not within the scope of Kuali Student to build infrastructure components, although the technical team may need to develop web service wrappers for existing products. Kuali Student will use existing open source products for BPEL engines, an Enterprise Service Bus, Workflow and Rules Engine Technology and UI frameworks. By using a formal open source software assessment methodology, such as OpenBRR, OSSM, or QSOS, the technical team will evaluate and select software from well-established open source organizations.

#### Involvement in Open Source Projects

Because of the large scope and complex requirements of Kuali Student, and the varying stages of maturity of the various open source solutions, the technical team might become involved with the development of some of the necessary open source components. Involvement with other projects will be evaluated on a case by case basis. To help insure that Kuali Student's involvement with other open source projects doesn't become an unnecessary drain on resources, involvement should be limited to very a specific purpose such as a bug fix or feature enhancement.

For those cases in which Kuali Student necessarily becomes involved in open source projects that pose the risk of becoming costly and ongoing responsibilities, attempts should be made to find external organizations to provide stewardship so that Kuali Student may devolve that responsibility.

## Evaluation of Open Source Infrastructure Components

Infrastructure developed within the Kuali foundation will be evaluated using the same criteria as any other product. Adherence to service orientation is the most important principle. However, within these constraints, Kuali Student will actively seek to leverage Kuali code, concepts, and expertise. Kuali infrastructure products will change and mature over time and therefore this is another area in which there may be a re-evaluation of original choices.

## **Development**

### Principle 10: Java as the Language and Platform of Choice

Code that is written as part of the core Kuali Student product should be written in Java and Java will be the platform of choice for Kuali Student. However, partners and other third parties can develop add-ons written in any language and that use a non-Java platform as long as they can work with the Kuali Student web service framework. Above all, this means they must work with the Kuali Student service contracts.