Installing, Configuring & Customizing KFS

Mike Criswell (Michigan State University)

Warren Liang (University of California, Irvine)

Minimum Requirements

- Java 1.5
- Ant 1.6.5
- Oracle 10.1+ or MySQL 5.0.5+
- Servlet 2.4 container (e.g. Apache Tomcat)

https://test.kuali.org/confluence/x/IoTI

Other Tools

Development & Testing

- IDE, e.g. Eclipse 3.2+
- KFS test framework based on Junit 3.8.1
- Continuous Integration, e.g. Anthill Pro 2.50+

Production

- Application Server, e.g. Tomcat 5.5.16+
- Web Server, e.g. Apache 2.0.55+
- Load Balancer, Zeus ZXTM-lb

Selected KFS Terminology

- Business Objects (BOs)
  - Java classes, instances of which represent a DB row in table
- OJB, for Object Relational Mapping
  - Configured using XML files
- Data Dictionary
  - Configured using XML files
- Spring, for Service and Transactional Management
  - Configured using XML files

Evaluation: Database Setup

- Install appropriate version of a supported database (currently MySQL and Oracle)
- Install JDBC drivers
- Configure database import/export tool via ~/impex-build.properties
- Perform initial setup steps
- Run bootstrap target
- Import demo data set

https://test.kuali.org/confluence/x/YvY

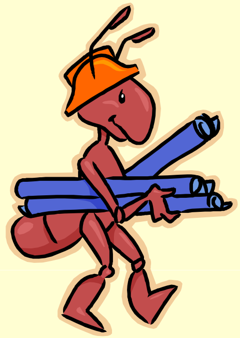
Evaluation: Environment Setup

- Configure build process via kuali-build.properties
- Set up java, ant, tomcat, and optionally Eclipse
- Run ant target “dist-local” (and “make-source”, if no IDE)
- Start tomcat

https://test.kuali.org/confluence/x/X-Y

Build Files

- User properties ("~/kuali-build.properties")
  - Institution shared properties
  - Project build.properties
- build.xml
  - build directory
    - external
    - project



https://test.kuali.org/confluence/x/Y-Y

Configuration & Customization: build.properties

- Deployment
- Database platform
- Batch
- User Maintenance
- Authentication
- Spring Files
- User Interface
- User Messages

Functional Implementation Questions

? Should we use workflow for other applications

? Will our security office frown on the encryption strategy

- The default implementation for KFS is demonstration grade, meaning that it should **not** be used with real sensitive data.

kuali foundation

Functional Implementation Questions

? How will we source institutional user data

? How and when will our batch schedule run

? Should our static content (e.g. images, help pages, etc.) be release independent and who will maintain it

Functional Implementation Questions

? What will our chart and organization hierarchy look like

? Will we use flexible offsets

? Do we have additional attributes that we need to represent and use in KFS

? How can we change workflow to meet our approval process

Configuration & Customization: Spring: Bean Overrides

- User Service
- Authentication Service
- Mail Service
- Encryption Service (should be overridden)
- Modules
- Any other service

Spring: Module Definition

- Provides definitions and directory locations for the following:
  - Users
  - Authorization
  - Data Dictionary
  - Database/OJB mappings
  - Batch
  - DWR (AJAX) configuration files
  - Module metadata

Example Spring Module Definition

```
<bean id="purapModule" class="org.kuali.core.KualiModule">
  <property name="moduleId" value="purap" />
  <property name="moduleName" value="Purchasing/Accounts
    Payable" />
  <property name="moduleCode" value="PA" />
  <property name="initializeDataDictionary" value="true"/>
  <property name="moduleUserService"
    ref="purapUserService" />
```

Represents module metadata, and most of these probably won’t be overridden

Example Spring Module Definition (cont.)

```
<property name="moduleAuthorizer">
  <bean
    class="org.kuali.kfs.authorization.
    KfsModuleAuthorizerBase">
    <property name="packagePrefixes">
      <list>
        <value>org.kuali.module.purap.</value>
      </list>
    </property>
  </bean>
</property>
```

Which java package(s) belong in the module?

Example Spring Module Definition (cont.)

```
<property name="moduleUserRule">
  <null />
</property>
<property name="moduleUserPreRules">
  <null />
</property>
<property name="dataDictionaryPackages">
  <list>
    <value>org/kuali/module/purap/datadictionary</value>
  </list>
</property>
```

Location(s) of data dictionary files

Example Spring Module Definition (cont.)

```
<property name="databaseRepositoryFilePaths">
  <list>
    <value>org/kuali/module/purap/OJB-repository-
      purap.xml</value>
  </list>
</property>
<property name="jobNames">
  <list>
    <value>
      purchasingPreDisbursementImmediatesExtractJob
    </value>
  </list>
</property>
```

Location(s) of OJB files

Batch job(s) associated with the module (name of job Spring beans)

Example Spring Module Definition (cont.)

```
  <property name="triggerNames">
    <value>purchasingPreDisbursementImmediates
      ExtractJobTrigger</value>
  </property>
</bean>
```

Trigger(s) (i.e. invoker of jobs at a given time), if necessary, for batch jobs in module (name of trigger Spring beans)

Configuration & Customization: Spring Module Definition

- To override/add OJB mappings, data dictionary entries, packages, jobs, etc., the module bean must be overridden.

Configuration & Customization: Spring Module Definition

- Create a new Spring bean file
- Copy the original Spring module bean and paste it to the new Spring file
- Make desired changes (e.g. add OJB file)
- Remember that overriding Spring beans (by adding new Spring files to the application) must be done by adding the path/name of the new files to the “spring.source.files” build property

kuali foundation

Data Dictionary

- The data dictionary is one of KFS/Rice's repository for metadata about business objects, attributes, and documents.
- Specified with XML files located within the project structure.

https://test.kuali.org/confluence/x/r4HgAQ

Data Dictionary

- The DD defines the following:
  - Labels
  - Presence & Order of Fields
  - Custom Fields
  - Inquiry Association
  - Authorization
  - Maintainables
  - Business Rules

Configuration & Customization: Data Dictionary

- Copy and modify an existing DD file or create a new one in the module's institutional DD directory under work/src
- Override Spring module to include the new institutional DD directory at end of list.

```
<property name="dataDictionaryPackages">
  <list>
    <value>org/kuali/module/purap/datadictionary</value>
    <value>edu/mySchool/module/purap/datadictionary</value>
  </list>
</property>
```

Extended Attributes

- Institutions will probably need to add additional attributes to the tables shipped with KFS.
- KFS comes with a mechanism to do this in a minimally invasive and standardized way.
- Extended Attributes reduce the pain of upgrading KFS.

https://test.kuali.org/confluence/x/goERAQ

Extended Attributes: Example

- Institutions may decide to add additional attributes for the Account table.
- They create a secondary table to store the extended attributes with the same primary keys as the account table.
- Using an equi-join/natural join, the framework can access both the built-in and extended attributes from the Account Business Object.

Configuration & Customization: Extended Attributes

- Module Spring bean override
- Database object(s) (i.e. DB tables, etc.)
- Extension class descriptor & base descriptor override in OJB
- Extension Business Object & rule override
- Extension BO DD entry, base BO and maintenance document DD override
- Optional control values & AJAX

Extended Attributes: Example

- After an extended attribute is created, we can easily implement lookups and inquiries with the extended attribute(s), as well as create/modify maintenance documents to use those extended attributes.

kuali foundation®

System Parameters

- Allow institutions to customize out of the box business rules based on their own policies
- Controlled by functional users via maintenance documents
- Externalize constants out of the code
- Maintained in System Parameters Tables
- Convenient service and evaluator methods are provided for developers to access and use the constant values in business rules validations

https://test.kuali.org/confluence/x/3IDS

Configuration & Customization: Parameters

- Configuration
  - Exception mailing lists
  - Authorized groups
  - Derivations
- Validation: simple and compound constraints
- Relative help URLs

Configuration & Customization: Parameters

- Review each parameter to determine whether it needs to be changed
- Write a SQL script, or modify the XML database data manually

kuali foundation

Business Rules Class

- Pluggable through Data Dictionary
- Since written in Java, they are much more expressive than just matching
- Has access to all Kuali Spring-based services
- Extensive code reuse through inheritance and services

https://test.kuali.org/confluence/x/poDS

Plugging Business Rules Class

- Specify the fully qualified class name of the business rule class with the <businessRuleClass> tag in the document's data dictionary file.

Plugging Business Rules Class

Example, PaymentRequestDocument.xml:

```
<documentClass>
  org.kuali.module.purap.document.PaymentRequestDocument
</documentClass>
<businessRulesClass>
  org.kuali.module.purap.rules.PaymentRequestDocumentRule
</businessRulesClass>
```

kuali foundation®

Document Authorizer Framework

- Determines who can initiate a document
  - Initiator workgroup in Doc DD XML
- How document fields are rendered
- What buttons are displayed

https://test.kuali.org/confluence/x/goAUAQ

kuali foundation

Plugging Authorizer Class

```
<documentClass>
   org.kuali.module.purap.document.PaymentRequestDocument
</documentClass>
<documentAuthorizerClass>
   org.kuali.module.purap.document.PaymentRequestDocument
   Authorizer
</documentAuthorizerClass>


<authorizations>
  <authorization action="initiate">
    <workgroups>
      <workgroup>
        KUALI_PURAP_ACCOUNTS_PAYABLE
      </workgroup>
    </workgroups>
  </authorization>
</authorizations>
```

Document Authorizer class implementation

Document initiator workgroup

Configuration & Customization: web.xml

- Workflow Servlet Mappings
- CAS Servlet Mappings
- Filters
- Listeners

kuali foundation

Configuration & Customization: Workflow

- Document Types
  - Exception, blanket approval, etc. workgroups
  - Route nodes / path
  - Search configuration
- Rules / Searching
  - Add rule templates / modify existing templates
  - Modify / add XML attribute definitions
  - Extend and customize java attributes

https://test.kuali.org/confluence/x/agAEAQ

Implementation

- Database Setup
  - Import bootstrap data set
  - Review data setup page for delivered data description and dependency information
  - Determine what you will load through UI vs other
  - Use the post-data load encryption process as needed for data not loaded through UI
- Environment Setup
  - Revisit build properties
  - Wrap or replace KFS build

Maintaining Customizations

- Additions should reside outside the org.kuali package
- Do not modify delivered files (data dictionary, OJB, spring) – override them
- When you need to modify delivered files, use the keyword: INSTITUTIONAL CUSTOMIZATION (in comment appropriate for file type)
- Track modifications to delivered document types, parameters, workgroups
- Version control or other comparison tool to assist with merge of modifications to delivered files

Upgrade Process

- Get distribution, point at your prior version, run delivered code upgrade process
- Use delivered process to generate database upgrade script, review & execute
- Branch your current development version, replace custom code with the distribution
- Sync and reapply changes

KFS Installation/Configuration and Customization at MSU

Installation/Configuration and Customization at MSU

- MSU and KFS
- Approach
- Technical environment
- Distribution: What do you get ?
- Goals/Objectives
- MSU load process
- How did it go?
- Next steps
- Recommendations

MSU and KFS

- 3 technical staff, 4 KFS developers, 14 functional staff. Continuing with staff additions
- Four sandbox environments running KFS 2
- Review, learn, and provide input on installation efforts

MSU’s Approach

- Create sandboxes using MSU data
- Become familiar with the application prior to implementation
- For bootstrap version much attention was given to data, setup and configuration
- Prior to installation functional/technical staff reviewed process from KFS release 1 and developed plan for KFS release 2

Technical Environment

	MSU Sandboxes	MSU CVS/Dev/Test
Hardware	**Application Server**: Dell PowerEdge 2850, 2 Dual Processors, 8 Gig RAM, (2) 60 GB Drives, Red Hat Enterprise RHEL 5-32 - Linux (64 bit OS), 4 instances of KFS for various functional groups to play in **Database Server**: Dell Dual Processor, 8 GIG RAM, Red Hat Enterprise RHEL 5-32 - Linux (64 bit OS), 400GB Local RAID5 storage	**Application Server**: Dell PowerEdge 2950 III, 2 Quad Processors, 32 Gig RAM, VMWare 6, CVS, Dev and Test on different VM guests, CVS on Win2003, Red Hat Enterprise RHEL 5-32 - Linux (64 bit OS), SAN-based storage **Database Server**: Dell PowerEdge 2950 III, Quad Processors, 32 Gig RAM, VMWare 6, Dev and Test DBs on different VM guests, Red Hat Enterprise RHEL 5-32 - Linux (64 bit OS), SAN-based storage
Database	Oracle 10g Standard Edition for Linux (64 bit edition)	Oracle 10g Enterprise Edition for Linux (64 bit edition)
Download	Source Code Distribution	Source Code Distribution
Dataset	KULBOOTSTRAP + MSU Data	KULBOOTSTRAP + MSU Data

Distribution: What do you get?

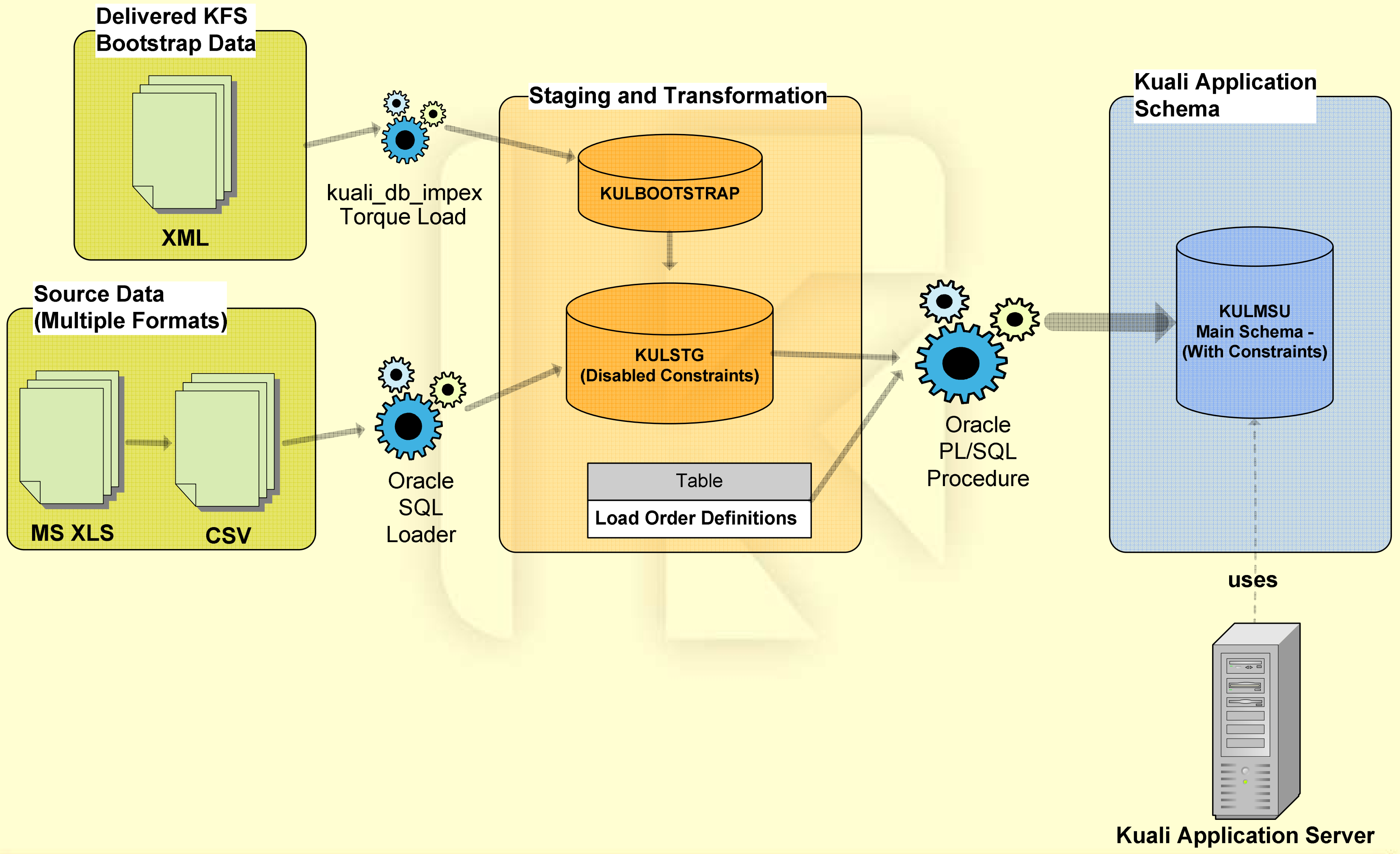
- Kuali Financial System w/embedded Rice
- Database Import/Export Tool
  - Demo Data Set
  - Bootstrap Data Set
- Kuali Rice Source

kuali foundation®

Goals/Objectives (MSU)

- Build the application from source
- Populate system with MSU data
- Test ability to make institutional enhancements
  - Change color scheme
  - Replace CAS Authentication with MSU's home-grown authentication.
  - Test extended attribute feature.
  - Build Lookup and Maintenance eDoc to support added attributes
  - Create a custom business rule.
  - Configure KEW routing rules for certain eDocs
  - Create new KEW Rule Attribute, Rule Template and Rule based on an extended attribute

MSU Load Process



Delivered KFS Bootstrap Data
XML
kuali_db_impex Torque Load
Source Data (Multiple Formats)
MS XLS
CSV
Oracle SQL Loader
Staging and Transformation
KULBOOTSTRAP
KULSTG (Disabled Constraints)
Table
Load Order Definitions
Oracle PL/SQL Procedure
Kuali Application Schema
KULMSU Main Schema - (With Constraints)
uses
Kuali Application Server

How did it go? (MSU)

- Installation went well (some expected challenges)
  - Performance issues initially but addressed by Foundation developers in 2.2 (significant improvement)
- Referential integrity constraints helped ensure proper data load.
  - MSU created Entity Relationship Diagrams (ERD) to help understand data model. Used both DB constraints and reverse-engineered OJB repository files to build ERD
- Documentation continues to improve
- Experience with installing prior releases and improving load process is making upgrades easier.

Next Steps for MSU

- Prepare infrastructure for dev and test environments
- Upgrade to KFS 2.2
- Streamline upgrade process for future releases
- Test KFS with standalone RICE KEW/Server
  - Goal: central workflow server for KFS and KRA
- Continue to identify gaps
- MSU KFS implementation will likely be phased with P1 late summer 2009 and P2 following six months later
- KRA First Release – Create first KRA sandbox

Recommendations

- Start with kualitestdrive site (http://kualitestdrive.org/)
- Begin with demo data and delivered configuration values
- Technical and Functional staff collaboration is key
- Analyze KFS and Rice data model. Focus on Chart/Org/GL modules before trying to build with bootstrap and institutional data
- Create a sandbox environment and begin with minimal set of institutional data

kuali foundation®

Questions?

kualitestdrive@oncourse.iu.edu