# Data Bus Architecture DRAFT DRAFT DRAFT

## Contents

## Reviews and Revisions

| Date | Initials | Reviewer | Notes/Comments |
|---|---|---|---|
| 5/17/2014 | nw | Norm Wright | Initial Draft |
| 5/19/2014 | tb | Tony Baratta | Reviewed |
| 5/20/2014 | nw | Norm Wright | Response |
| 7/30/2015 | nw | Norm Wright | Added Section for Server Requirements |
| 9/16/2015 | nw | Norm Wright | Added related Documents section |

## Purpose

### Goals

Even though the colleges have agreed to the *general architecture* of the Data Bus, a much more detailed description of that proposed architecture is required ensure a full understanding and agreement.

Evolving -- Although many aspects of the architecture are clear, we understand that many other aspects of the architecture can and should evolve as we move forward. Our hope is that this document will facilitate a dialog with the Data bus technical participants at the colleges and career centers so that they may:

1. Understand the proposed architecture
2. More fully vet the proposed architecture
3. Articulate any concerns they may have with the proposal
4. Express any additional technical requirements
5. Make suggestions for improvements
6. Help decide what aspects we can and should do as part of the POC and what aspects can be deferred

### Audience

This is a highly technical document that describes the architecture of the proposed Data Bus from several perspectives:

- Overall
    - how the adapters connect to the bus
    - how consumers connect to the bus's restful api
- Internal
    - How the adapters are internally architected
- Development
    - How the code is structured into projects
- Configuration
    - how the adapter software is organized to facilitate local configuration and control

An understanding of certain architectural and technical topics is critical to reading this document.  Some such topics include::

- SSL, certificates and encryption
- Authentication, Basic Auth, CAS, Shibboleth
- Authorization -- roles and permissions
- Application Servers (tomcat)
- Databases (oracle)
- Source Code Control
- Maven
- Spring Injection
- CXF, Soap and REST
- Caching, logging

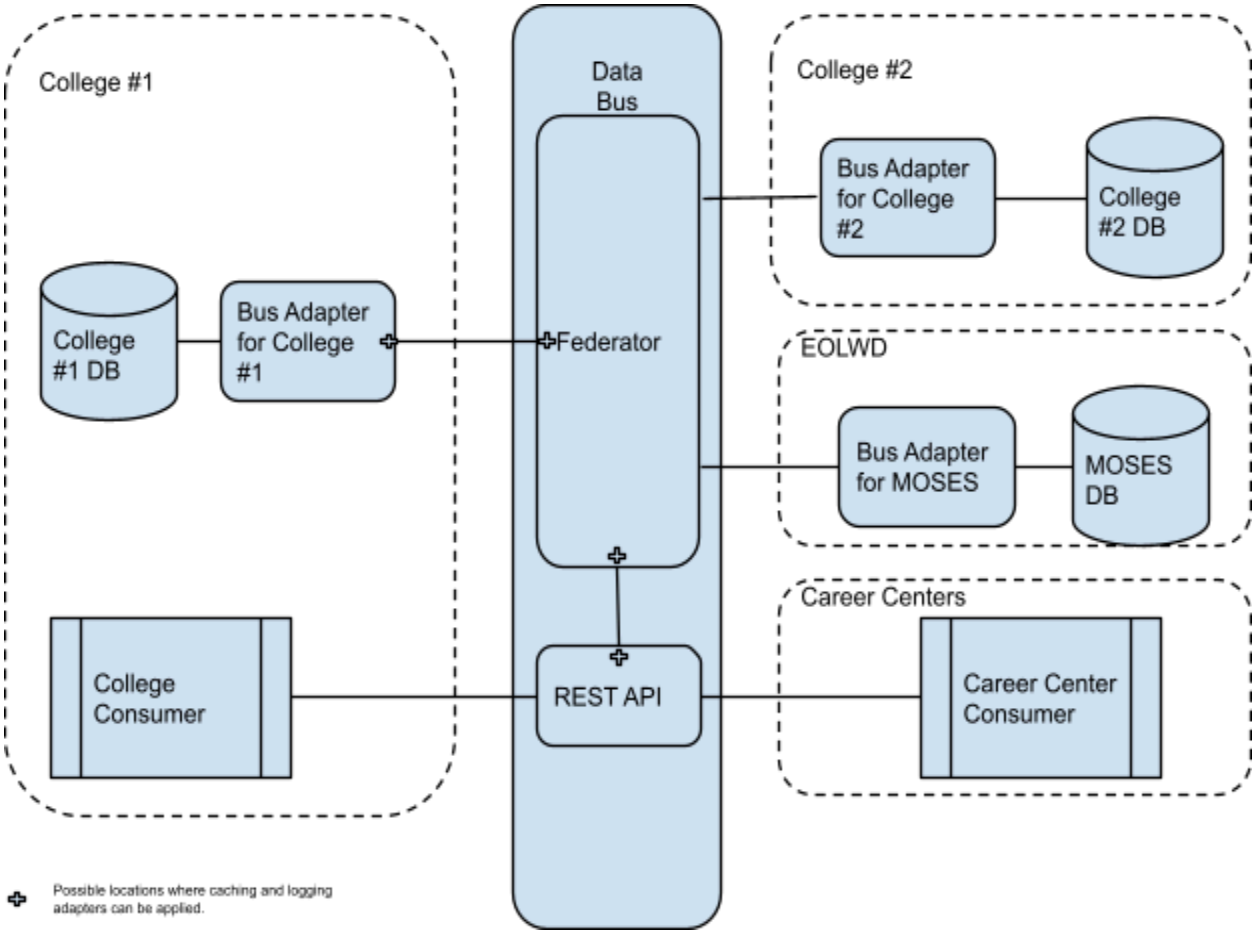
### Related Documents

- How to Setup the Developer Environment For the Databus POC

## Section 1 - Overall Architecture

This section describes the overall architecture, how the adapter connects to the bus and how the bus works.

## Diagram




## Components

### Encryption

1. All connections to and from the bus should be encrypted via SSL
    a. Should we do this for the POC?
        i. If we are only exposing publicly available data then perhaps not?
2. All connections within the bus should also be encrypted via SSL
    a. Initially we envision the REST API and Federator components to live in the same WAR deployment (same JVM) so the communication will be internal but for scalability we envision the REST API module could be deployed separately (see performance below) and if that is done the connections should be via SSL
3. Are there certain data elements that we should consider encrypting the data as well?
    a. Perhaps SSN?
        i. Would need a VERY strong use case for even exposing this!
    b. Other???

## Authentication

1. Rest API authentication to consumers
    a. ANONYMOUS_GUEST access
        i. Allow for in architecture but not implement in POC?
        ii. Would only apply to certain public transactions
    b. Different kinds of consuming applications:
        i. Server based applications that authenticate the user themselves and then:
            1. Have server side code talking to the REST API on behalf of that user
            2. Serve up browser components that need to directly talk to the REST API
        ii. Browser only applications (like plugins) that need to authenticate themselves to the REST API
    c. Options
        i. Suggest we use authentication keys like MIT's MC3's Handcar
        ii. What about OAUTH?
            1. OAUTH does not covers the browser only applications
        iii. What are the limitations of securing server to server connections via IP addresses? Only allow listed IPs or IP Subnets to connect? [AB]
            1. I thought IP addresses could be easily spoofed (NW)
            2. Spoofing only works one way, sending. You can't route the reply info back to yourself unless you "closed the system", and if you could do that you'd have physical access to the server and/or router for that sub-net. At that point you have worse problems. [AB]
        iv. ???
2. Federator to adapter authentication
    a. We could use specially issued certificates to ensure the adapter is really talking to the expected federator and not spoofed into giving up data to unauthorized people?
        i. This is how MIT secures application to application, they call them App Certs.
    b. For POC can we just use Basic AUTH?
        i. Or can SSL be implemented quickly on the Community College side? (AB)
        ii. Yea... I just "hate" installing certificates in Java (NW)
        iii. also, I believe that we can use "server to server" certs too within Apache, just need to verify that. Probably a module we can install that's not default in tomcat. Off load that security to the web service instead of in code.[AB]
3. Proxy authentication needs to be supported

a. Consuming applications would need to authenticate the user and the services would then manage a chain of trust that they actually did that effectively.
    i. What kinds of policies/procedures should we put in place to ensure that.
        1. I.e. we don't want a consuming application to do a lousy job at authentication so that it could be hacked.
b. Note: The services support two principals
    i. The principal that is immediately connecting to and is authenticated to the service
        1. This needs to match the principal obtained from the web service context
        2. This principal needs to be explicitly authorized to be able to serve as a proxy for the actual end user's principal
    ii. The ultimate principal on whose behalf the request is being made
        1. This user must have been explicitly authorized to execute the service method in question
c. For read only servers, could we just authenticate the server / proxy via the Application Certificates? Will the Federator need to do more than read from the Community DBs? (AB)
    i. Eventually they will do more than read (NW)

### Authorization

1. Needs to be enforced in the adapters so that:
    a. It follows the rule that the enforcement point must be close to the resource being guarded.
    b. Each community college has control over those authorizations
2. Authorizations need to be very fine grained
    a. Person A can do X to Y part of student B's record for duration Q for example:
        i. Student Maryellen (B) grants career counselor Mark (A) the right to to view (X) my attendance record (Y) for the next 9 months (Q)
3. Authorizations need to have expirations
4. Services will provide the interface for managing authorizations
    a. During the POC we should be able to just run a hard coded "mocked" authorization as part of the deployed java code
        i. This can be Spring injected to point to any service that implements the same methods

### Performance, Scaling and Caching

1. We do not expect to have to address performance directly in the POC but our goal of the POC is to have an architecture that supports techniques that we know we can leverage should we move past the POC stage
2. A caching adapter can be applied at any of the service interconnection points (See the plus sign, "+" in the above diagram)

   a. On the college side of the college adapter to federator link to reduce the impact/hits on the college's database
   b. On the federator side of the same connection to reduce the calls/hits across the wire to the college adapter
   c. On the federator side of the federator to REST API to reduce the calls/hist on the federators
   d. On the REST API side to to reduce calls/hist across the wire on the federators
3. Caching adapters need to be developed so they take into account the user who is making the request.
   a. This is to ensure a cache does not serve up data to an unauthorized person just because that same data was put in the cache by an authorized person.
4. To enable scaling all of the major components should be able to:
   a. Run independently on their own server
      i. During the POC
         1. The college adapters are expected to run on their own servers
         2. The the federator and REST API will be run on the same server
         3. TBD where the consuming application will live
   b. Have multiple instances spun up and load balanced via something like an F5
      i. As such all of the services are stateless and should support failover on a call by call basis.
5. Propose to leverage Ehcache as the tool to implement the caching scheme

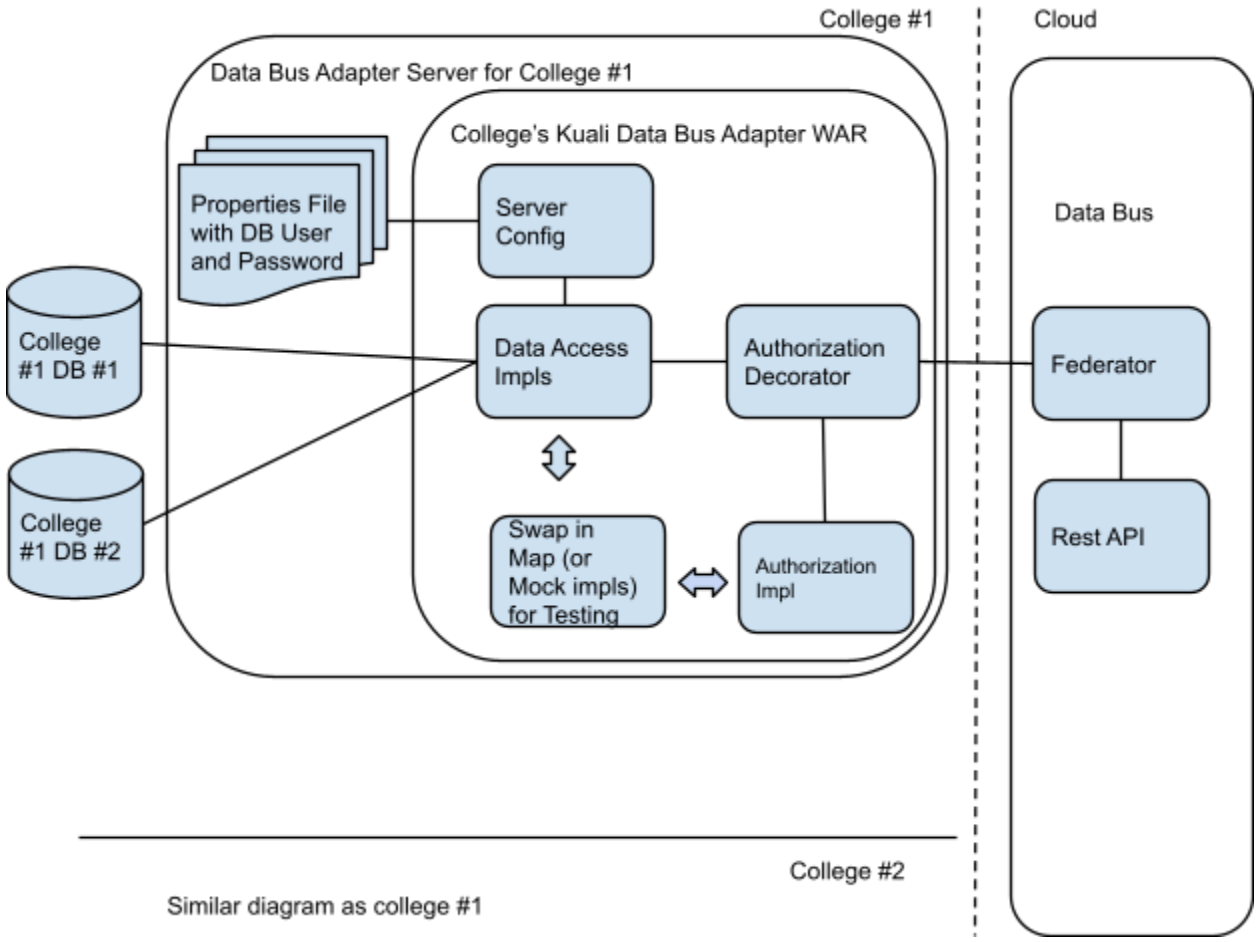
### Logging and Remote Management

1. We do not expect to have to fully address logging and remote management directly in the POC but a goal of the POC is to have an architecture that supports techniques that we can leverage should we move past the POC stage.
   a. Note: We now have instrumented the services for performance monitoring
2. A logging adapter can be applied at any of the same interconnection points as caching adapters to collect statistics for management purposes
3. JMX could like wise be leveraged to provide this remote management of the bus.
4. Logging could also serve as a mechanism to track all requests for audit purposes.
   a. Is this something we need to implement during the POC?
      i. I think the "hooks" should be laid down for logging at minimum.(AB)
         1. Perhaps a simple logging adapter...(NW)
         2. With all the debugging one does to get the software running, the logging could be laid down early for debugging purposes and repurposed later for security logging.(AB)
   b. For business events, such as "this person saw this student's record" we will leverage the logging service.

## Section 2 - Internal Architecture of Adapters

This describes how the adapters are internally structured, showing how authorization is an integral aspect to the adapter architecture giving the colleges complete control over what they choose to share and not share.

### Diagram




### Components

#### Databases, Properties and Server Config

1. In the POC the databases are all Oracle databases running Ellucian's Banner system.
    a. Connections are secured via Oracle's thin JDBC driver
2. Login usernames and passwords are managed via properties files that are located in a secure location on the server machine
    a. Typically there are different properties for each tier of deployment
        i. DEV
        ii. TEST

   iii. QA
   iv. PROD
3. For the POC we are only expecting a single deployment to a TEST or QA tier.
4. The properties files will eventually also hold information about the locations on the server of keystores and truststores for SSL encryption
5. The Server Config component allows colleges to override/specify exactly where these properties are to be located
6. Question: The diagram above looks like multiple College DBs are controlled by the same server config, which I don't think is how you intended to display the info. Not sure what would be a better diagram at this time, but I would recommend showing only one College DB unless we can show that each College DB has it's own configuration control. Maybe multiple stacked images (e.g. several same shapes over laying one another offset by 2-5 pixels right and bottom.) for the Server Configs and Data Access Imps. Also maybe the Server Configs and Data Access Impls need to be the same image or a hybrid. (AB)
   a. Ahhh... no that just shows one college's adapter but I can see how it could be confusing... I added more clarification to the diagram....(NW)

### Data Access Implementations

1. The data access implementations implement the relevant Service Contract.
   a. See Kuali Student Contract Documents
   b. for example:
      i. Atp Service (academic time period) for Terms and Semesters supported by the college
      ii. Course Service for general information about all the courses that are offered at the college
      iii. Course Offering Service for courses and sections offered by the college in a particular term
      iv. Academic Record Service for information about a student's enrollments, registrations and grades and degrees
      v. etc...
2. Each data access impl maps the college's data to the standard service contract definitions
3. Map impls (also called Mock impls) can be swapped in for testing
   a. This is how the "Hello World" war works -- it does not connect to any database but just serves up hardwired test data.

### Authorization Decorator

1. The authorization decorator also implements the relevant Service Contract by "adding (decorating it with) authorization functionality."
2. Every service call must pass an authorization check before the underlying data access implementation is invoked to fetch the data

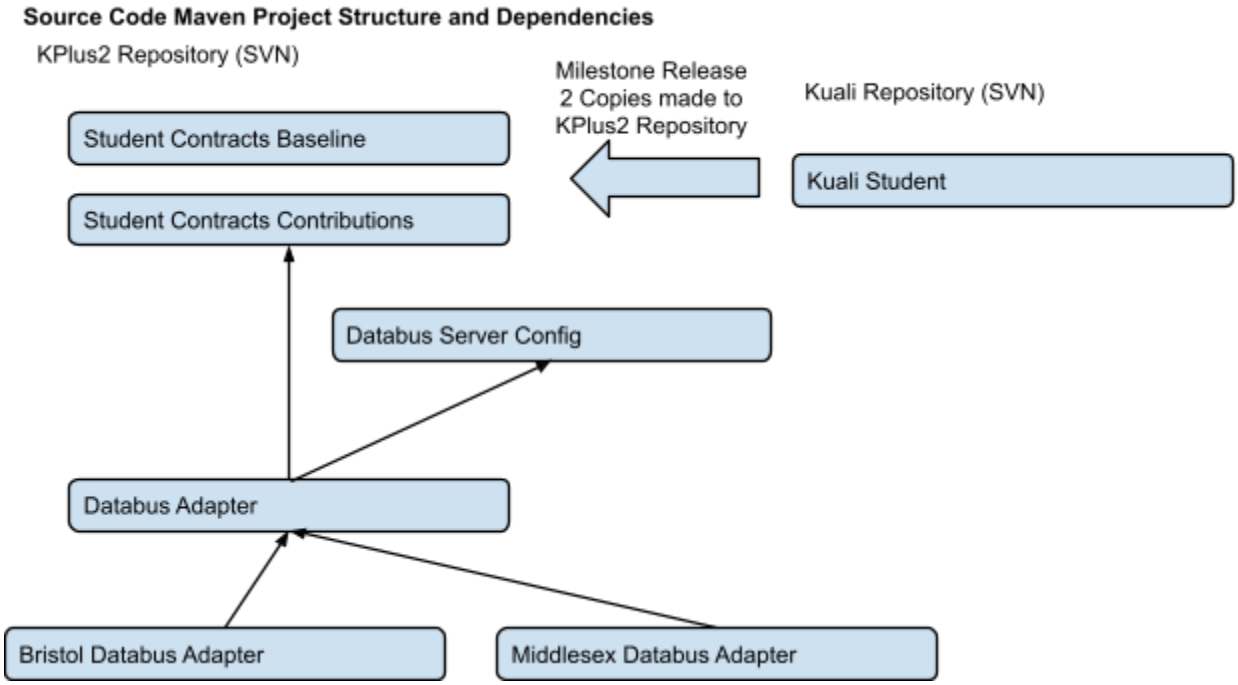
    a. Authorization controls can also be enforced on the way out to block or filter certain records.
3. The authorization decorator is the policy enforcement point for the authorization
4. The authorization decorator calls the role permission implementation to ask if the person who is making the call “isAuthorized” to make that call

### Authorization Implementation

1. An authorization service implementation will be used by default that persists to a database
2. A map (or Mock) implementation can be swapped in and used instead of the persistence Implementation
    a. This technique will be used during the POC to hardwire which users can and cannot call which service method
3. Should this project move past the POC stage colleges have several options to allow for more flexible management authorizations
    a. Install authorization Services
    b. Implement a more lightweight call to the college’s own role and permission system, for example LDAP

# Section 3 - Source Code and Maven Project Structure

## Diagram




## Components

While we are NOT installing Kuali Student software per se we are leveraging pieces of the code base, especially the service contracts for this project.

### Kuali Student

1. Massive source tree holding all of the student specific service definitions, implementations and screen user interfaces
2. See https://svn.kuali.org/repos/student/

### Student Baseline

1. This is a copy of the Kuali Student source code.
2. See https://svn.kuali.org/repos/student/enrollment/aggregate/tags/student-2.0.0-M8/
3. This is to be used as a baseline for comparisons to the contributions branch should that be needed.

### Contributions

1. Location for us to make any “bug fixes” or “enhancements” to the Student code, should that be required.
2. Initially this is a copy of the milestone 8 internal release just like the baseline.

3. Initially this code holds an exact copy of the baseline which is an exact copy of the milestone release.

### Databus Server Config

1. Small utility project which provides a shell layer to accessing the properties file that needs to be specific to that particular server
2. Often used to hold the configuration that is specific to each server tier:
    a. DEV
    b. TEST
    c. QA
    d. PROD
3. For example this provides a mechanism to override the location of the properties that hold the JDBC connection parameters.

### Databus Adapter

1. Base adapter that will be extended and configured by each college.
2. Holds shared code that is needed by all colleges who wish to implement their own adapter.
3. Produces a “Hello World” war file that can be used for testing the application without accessing any real data.
4. Minor Diagram Question: Are the college databus adapters going through the KPlus2 Databus Adapter, or cloned and extended from it? (AB)
    a. This is a source code dependency diagram so it is the 2nd... the college adapters extend the base adapter. (NW)

### Bristol Databus Adapter

1. Bristol’s overrides and configurations to the base adapter.
2. Holds configurations that are specific to Bristol’s implementation of Banner
3. Holds configurations that are specific to Bristol’s desired authorization controls

### Middlesex  Databus Adapter

1. Middlesex’s overrides and configurations to the base adapter.
2. Holds configurations that are specific to Middlesex’s implementation of Banner
3. Holds configurations that are specific to Middlesex’s desired authorization controls

## Section 4 - Configuration By Colleges

### Customizations, Configurations, and Contributions

We define these terms as follows:

1. Customizations -- Changes an institution has to makes to the baseline code
    a. This can be done either via:
        i. class path overriding where the same class name and package are used so that the compiler picks up the institutions version of the particular file.
        ii. Via patches applied to the baseline code

    b. *Strongly discouraged*
    c. Not expected to survive any upgrades
    d. Encouraged to instead to divide the change into two parts:
        i. Institutional specific logic as configurations
        ii. General fixes that may be contributed back to the project
2. Configurations -- Changes made to an institution's own project that overrides or enhances the baseline code to meet the needs of the institution
    a. Expected to survive minor upgrades
    b. Expected to some chance of surviving major upgrades
    c. Typically Leverages Spring Bean Injection to accomplish
3. Contributions -- Changes made to the baseline code that are intended to be contributed back to the project
    a. Two types of Contributions
        i. Major new functionality, for example:
            1. This entire Data Bus will be contributed to the project
        ii. Small fixes, for example:
            1. Bug fixes
            2. Small enhancements that are are general and not specific to the institution
    b. Each type of contribution has its own mechanism to submit the contribution
        i. For major contributions you must fill out a special contribution form and submit it with the source to the project leadership team
        ii. For small fixes, we create Jira's and attach a patch file to be applied by the project team at the next release

## Goals

The goals are:

1. Allow schools to easily configure the application to enhance/override baseline processing to match their own needs
2. Allow those configurations survive all minor and hopefully most major upgrades

## Spring Bean Injection

Standard Spring Bean injection is used to configure all of the adapter components. This way schools may override some or all of the default adapter logic.

### Example ATP Service Injection

In the example below the component that is actually exposed to the bus is the authorization decorator. This authorization decorator is then configured with two items:

1. The underlying atp service that it is guarding
2. The role permission service that is used to ask if the user "isAuthorized" to invoke the method

In both cases these are "Mock" implementations that were designed for the "Hello World" war for infrastructure testing.

```
<bean id="atpServiceMapImpl"
 class="org.kuali.student.kplus2.databus.adapters.AtpServiceMapImpl"></bean>
<bean id="atpServiceMockDataImpl" init-method="init"
   class="org.kuali.student.kplus2.databus.adapters.AtpServiceMockDataImpl">
   <property name="nextDecorator" ref="atpServiceMapImpl"/>
</bean>

<bean id="rolePermissionServiceMapImpl"
class="org.kuali.student.kplus2.databus.adapters.RolePermissionServiceMapImpl"></bean>
<bean id="rolePermissionServiceMockDataImpl" init-method="init"
  class="org.kuali.student.kplus2.databus.adapters.RolePermissionServiceMockDataImpl">
        <property name="nextDecorator" ref="rolePermissionServiceMapImpl"/>
</bean>


<bean id="atpServiceAuthorizationDecorator"
     class="org.kuali.student.kplus2.databus.adapters.AtpServiceAuthorizationDecorator">
        <property name="nextDecorator" ref="atpServiceMockDataImpl"/>
        <property name="rolePermissionService" ref="rolePermissionServiceMockDataImpl"/>
</bean>

<jaxws:endpoint
        id="atpService"
        implementor="#atpServiceAuthorizationDecorator"
        address="/AtpService" />
```

### Example Middlesex Code overrides using Spring Beans

AtpServiceMiddlesexImpl.java

Implements ATP Service impl against Middlesex's Banner Database

```
public AtpServiceMiddlesexImpl extends AtpServiceBannerBaseImpl
... Middlesex specific database configurations
```

RolePermissionServiceMiddlesexImpl.java

Implements the role permission service, perhaps wiring it to it's LDAP server

```
public RolePermissionServiceMiddlesexImpl implements RolePermissionService
... Middlesex specific connections to LDAP
```

AtpServiceAuthorizationServiceDecoratorMiddlesexImpl.java

Implements additional authorization controls not envisioned by the base adapter. For example it may simply throw PermissionDenied exceptions to any service calls it does not wish to authorize.

```
public AtpServiceAuthorizationServiceDecoratorMiddlesexImpl extends
AtpServiceAuthorizationServiceDecorator
... Middlesex overrides to the base line authorization decorator.
```

Spring Bean Wiring

```
<bean id="atpServiceMiddlesexImpl" init-method="init"
   class="org.kuali.student.kplus2.databus.adapters.AtpServiceMiddlesexImpl">
</bean>

<bean id="rolePermissionServiceMiddlesexImpl"
  class="org.kuali.student.kplus2.databus.adapters.RolePermissionServiceMiddlesexImpl">
</bean>


<bean id="atpServiceAuthorizationDecoratorMiddlesexImpl"
class="org.kuali.student.kplus2.databus.adapters.AtpServiceAuthorizationDecoratorMiddlesexImpl
">
        <property name="nextDecorator" ref="atpServiceMiddlesexImpl"/>
        <property name="rolePermissionService" ref="rolePermissionServiceMiddlesexImpl"/>
</bean>

<jaxws:endpoint
        id="atpService"
        implementor="#atpServiceAuthorizationDecoratorMiddlesexImpl"
        address="/AtpService" />
```

## Section 5 -- Internal Architecture of the Data Bus

TODO: this still needs to be written

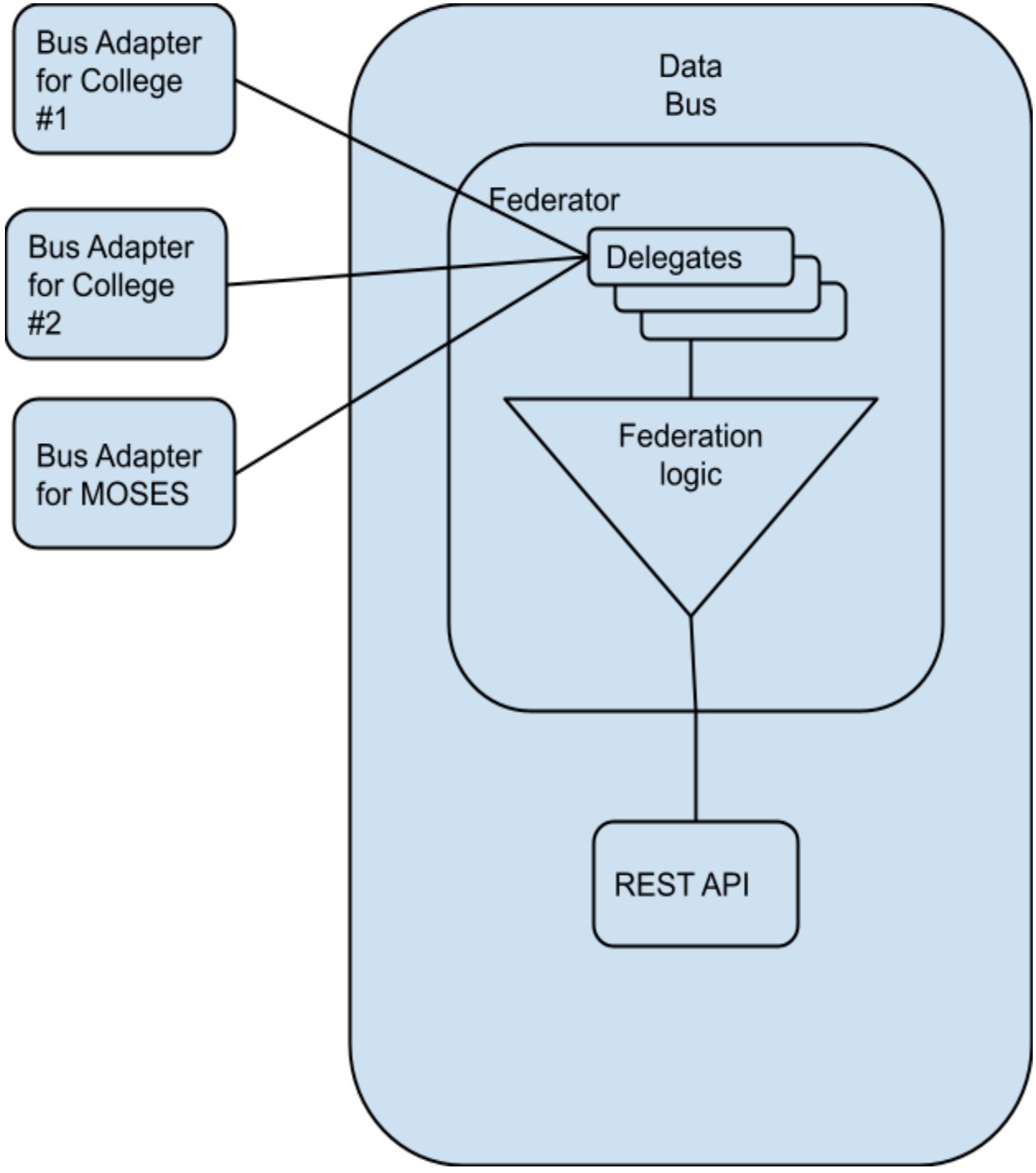
Question: Are there minimum data points that must be exposed in order for this data share to work? Do we need to outline these baseline data points? (ab)

There is a list of “Opportunities” that came out of the analysis phase

See Data Bus Opportunities

We not yet decided on exactly what data will be put on the bus as part of the POC. (nw)

### Diagram




## Components

#### Federator

1. The federator implements the specified contract by making calls to each of its delegates and aggregating the results using federating logic.

2. The results are then available to the REST API

### Delegates

1. Each delegate uses Soap to point to the remote adapter provided by each college or EOLWD.
2. Each delegate is identified by a key to indicate it's source, for example:
    a. "middlesex" for Middlesex College
    b. "bristol' for Bristol College
    c. "moses" for the MOSES system managed by the EOLWD
3. Each delegate could be optionally cached to speed repetitive access to the same information.
4. Minor Diagram Update: The Delegates image should be a "multiple", e.g. several same shapes over laying one another offset by 2-5 pixels right and bottom. (ab)
    a. Thanks. Fixed (nw)

### Federation Logic

1. The federating logic aggregates the data from each of the delegates
2. For example if you ask the Federator for all of the Fall Terms it will return you a list of the Fall Terms defined for Bristol and Middlesex with their respective names, codes and start and end dates.
    a. Bristol - Fall 2013, Fall 2014, Fall 2015
    b. Middlesex -- Fall 2012, Fall 2013, Fall 2014
    c. Federated: Bristol Fall 2013, Bristol Fall 2014, Bristol Fall 2015, Middlesex Fall 2012, Middlesex Fall of 2013, Middlesex Fall 2014
3. The federator will also fetch the detailed information from the appropriate delegate given it's id.
    a. Therefore the federator ID is altered, prefixed with a key to indicate which delegate it came from.
        i. For example a if the ID for the Fall 2014 from Bristol is 12345, the federated id would be "bristol:12345"
        ii. On the way back in the the "bristol:"is detected and removed and the bristol delegate is called to return the appropriate ATP.

### Rest API

1. The Rest API exposes the Student Service Contracts, which are authored as SOAP contracts as RESTful contracts.
2. The URLs are mapped according the the service name and object names, for example:
    a. http://demo.kplus2.com/services/rest/atp/atp would return all atps regardless of source
    b. http://demo.kplus2.com/services/rest/atp/atp/{id} would return the atp identified by the specified ID.
    c. Note: these URLS are not yet operational [Question: Is this going to be Saturn or Jupiter?]
    d. Additional details will be documented elsewhere

3. We consider this REST API an “alternate” expression of the service contract

## Section 6 -- POC User Interface Wireframe

See MIT_Search.pdf

## Section 7 -- College Adapter -- Recommended Configuration

Note: Unlike traditional vendor recommended software configuration the plan is to support a wide range of configuration options because we want the “ownership” of the this college connector to be with the school.

Server

- Hardware (whatever is your standard configuration for servers)
    - 8GB Memory should be enough
    - 10GM hard disk should be enough
        - We do not store data on the client adapter
        - It is primarily used to store server logs (in addition to the configuration software)
- Operating System (your choice) whatever can run apache tomcat.
    - Known to work with
        - Windows Server
        - Linux
- Use of a VM Image is OK
- Application Software
    - Apache Tomcat 7x or 8x
- Database JDBC Driver
    - Banner: Oracle 6 jdbc driver
    - Jenzabar: Informix jdbc driver?
- SSL Certificate
    - Can be self signed
- Java 1.7
- Port 443 or 8443 are the ones that need to be opened to the bus (which runs in the cloud)
- This server has to be connected to two other machines
    - Your System Database(s)
    - The Bus’s Federator that is running in the cloud

Installation Process

- We have a “Hello World” war that tests the basic server connectivity without connecting to any data
- Then we install the college adapter

- Database connection properties