Smarter Balanced System Architecture and Technology Report

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

21 March, 2012

Table of Contents

1. Overview

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

1. Overview

This document provides a comprehensive understanding of the Enterprise Architecture of the Smarter Balanced Assessment System. The document is the culmination of a rigorous three-month effort undertaken to identify the business goals and processes that will make up the Smarter Balanced Assessment System, and the technology components, and their relationships, needed to address these business goals and processes.

The intended audience for this document is the application architects of the teams that will be tasked with designing and developing the eventual software applications that will deliver on the business goals and processes. The document provides a framework to guide these application architects while they make the many design decisions they will need to make while designing the individual systems and applications.

It is important to note that the Enterprise Architecture is a definition of what components and relationships are needed to address the business goals and processes, not how the eventual applications will deliver on these. The how will require much further analysis by the application architects and their respective development teams during later phases of this project. The value of the Enterprise Architecture is to provide the necessary frameworks and context that the application architects will need to build a cohesive software system of applications that efficiently work together to solve the business goals and processes.

The Enterprise Architecture definition in this document contains the following:

Architecture Principles
A set of guiding principles to be considered by all teams, systems, and applications. These principles are a set of musts defined by Smarter Balanced teams early on and should be used to guide all design solutions going forward.

Architecture Constraints
Constraints identified that each resultant system and application must adhere to.

High-Level System Component Diagram
Identifies the individual components and their relationships. Each of these components will result in one or more applications in the final system.

Domain Definition
Looks at the assessment system's domain model, which highlights the scope, attributes and relationships of a domain. This is followed by a series of high-level system component diagrams, which illustrates the various component parts and the functions they perform.

Deployment and Hosting
Presents the deployment and hosting models and their different capabilities in a number of scenarios, demonstrating possible set-ups; how it could function as an independent system as well as how it could function when incorporating a states' or a districts' systems, where applicable.

Data Architecture Definition

Highlights the principles on creation, management and delivery of assessments, by providing best practice recommendations and industry benchmark approaches where appropriate.

Interoperability

Discusses the interoperability between components, identifying where interoperability is required and at what stage, as well as providing suggested standards to use to enable such capability.

Non-Functional Requirement Constraints

Addresses concerns with regards to the assessment system. These include: requirements and recommendations on open licensing, interoperability and standards, system high-availability and scalability, accessibility and technology.

Security

Requirements that exist in component-to-component communication, user authentication and authorization and student data areas. This section highlights the security concerns that must be considered when designing and implementing the components.

Technical Architecture Definition

Covers these topics: server and browser hardware and software requirements, networking requirements, database, data storage and archiving approaches and requirements, middleware and integration software requirements, as well as the security approach and requirements for applications, data and end-user access.

Build vs Buy

Based on a range of features and components, a weighted matrix is provided, along with the application development model. These will help Smarter Balanced prescribe the best application development approaches and practices to build the components, from managing multiple streams of development work with varying time lines, to industry best practices that ensure quality of output.

Application Development Model

Defines suggested development processes that will be needed to ensure ease of integration of the software products of the different teams.

Scenario Mapping to Component Systems

A mapping of the user scenarios identified during the workshops to the components defined in the Enterprise Architecture.

Component Priorities

A suggested sequence of development priorities informed by the Smarter Balanced published schedule and component relationships.

Glossary

Provides a reference to the terminology and concepts that have been discussed throughout this document.

Release Notes

Details the changes that have been made to this document from one version to the next.

1.1. Assessment Lifecycle

The assessment life cycle is designed to include any type of assessment and take into account the iterative nature of assessments. It focuses on the overall assessment processes from conception to post-delivery. Depending on the assessment type, the process can begin and end at any point in the life cycle. The six over arching categories are broken down into subprocesses. Data and information can be exchanged at any point in the process with another assessment, or administrative or instructional application.

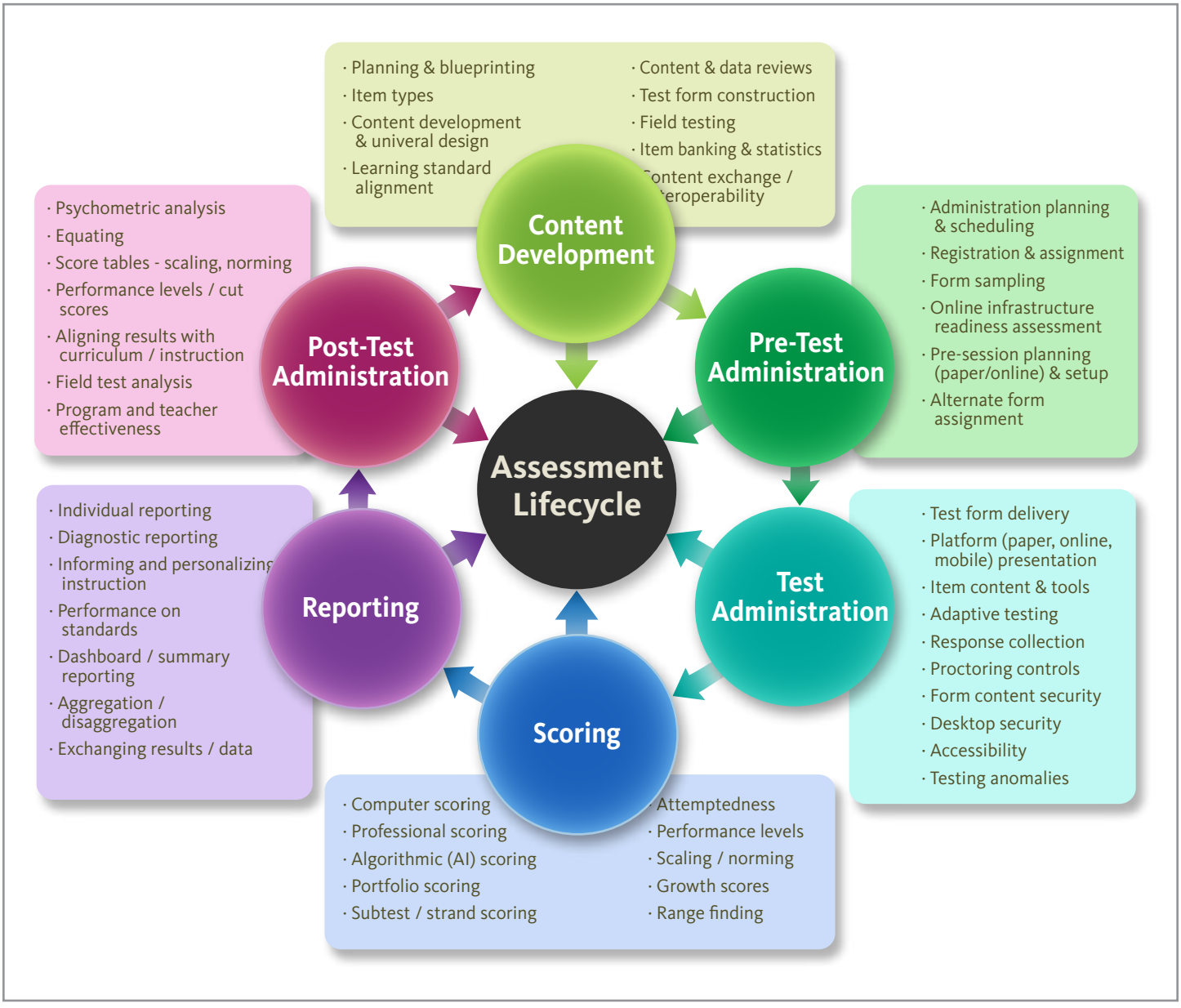

Figure 1.0 Assessment Life cycle

Descriptions

Content Development

This phase of the life cycle includes everything involved in developing the assessment content. This includes item development through asset development, alignment, and learning standards to field tests.

Pre-Test Administration

Pre-test administration includes the processes necessary prior to test administration. This includes form assignment, registration of students and scheduling of the assessment.

Test Administration

This phase of the assessment life cycle includes the actual delivery of the assessment and the subprocesses contained within the phase. This includes proctoring, delivering, and collecting student responses.

Scoring

The scoring phase incorporates not only the actually scoring of student responses as well as any data needed for item statistics and trending.

Reporting

Reporting occurs after the scoring of the assessments. This reporting could be as formal as summative assessments or as informal as formative assessments, providing immediate feedback to teachers and students.

Post-Test Administration

The final phase in the assessment life cycle is post-test administration. This phase includes all processing associated with the finality of the administered assessment, including use of the data and information, equating, and needed psychometrics.

2. Architecture Principles

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

measured progress

2. Architecture Principles

Architecture principles are used during the design process, as guidelines for deciding between options.

Some rules of thumb apply when defining these principles:

1. The number of principles needs be kept to a reasonable number (less than 10 is recommended);
2. They need to be maintained to support decision making. They are especially useful when the options presented are equally viable;
3. They need to be maintained by the Architecture Review Board;
4. They also need to be at a level that aids decision making.

The following are the principles, with some rationale and implications:

2.1 Choose single-responsibility systems

Similar to the Single Responsibility principle class, this system-level design principle encourages creating or acquiring systems that interact with other systems through standard protocols.

Rationale

- Systems must be upgraded or replaced as business needs change.
- Systems that perform multiple business functions are harder to upgrade and replace.

Implications

- Systems utilize standards for inter-system communication.
- Systems can be replaced with minimal disruption to other systems.

2.2 Design for emergent reuse

Emergent reuse is the ability to identify existing systems that can be used in implementing new systems. Utilizing existing systems in a new application is more effective than designing a new application.

Rationale

- Reusing capabilities in different areas reduces the overall system maintenance costs.
- Overly-detailed design when designing for reuse is often expensive and ineffective.
- Identifying and refactoring for reuse is less expensive and more effective.

Implications

- Software designers and architects need to be aware of existing system capabilities.
- Systems that use open standard interfaces that are able to interact with other systems are easier to utilize.

2.3 Develop Homogeneous systems

Developing homogeneous systems is a prerequisite to emergent reuse. Systems that exchange information through common standard protocols are easier to manage and enhance.

Rationale

- Lower maintenance costs.
- Faster on-boarding of new team members.
- Simpler environment configurations.

Implications

- Standards are extensively used.
- Small number of standard operational environments (e.g. Java containers).
- System are built using repeatable patterns.

2.4 Demand Driven Releases

Demand-driven releases indicate that the IT organization is in tune with the business demand. As a result, releases are in step with business changes.

Implications

- Software and system releases are made at a frequency driven by business demand and by the business' capacity to absorb those changes.
- Software updates are available more frequently for UAT.
- Systems are well maintained and can be maintained on an ongoing basis.
- Systems are flexible and amenable to change.

2.5 Business continuity

Software systems being available at all times to support users.

Implications

- Systems economically scale to accommodate increased business demand.
- Systems are tolerant of infrastructure faults.
- Systems support Disaster Recovery scenarios.
- Operational parameters are actively monitored - runtime metrics and dashboards.
- Requirements are traceable to ensure compliance.
- Code is readable and easy to understand.
- Legacy systems are aggressively retired to maintain simplicity of options and lower maintenance costs.

2.6 Low cost for SEA

A principle where design and implementation designs will strive towards a low cost of ownership for member SEA, so that an SEA can implement or adopt the Smarter Balanced system with low up front costs and ongoing operational costs.

Implications

- Design/Implementation decisions that require investments from the SEA must be balanced with benefit and alternatives need to be considered.

3. Architecture Constraint

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

measured progress

3. Architecture Constraint

3.1. Smarter Balanced certified components and interoperability

Smarter Balanced will provide all the components defined in this architecture. Given the high level of interoperability that is supported, SEAs may choose to use their existing systems in place of one or many Smarter Balanced components. This could be in the form of extending the component or it may be a vendor product that the SEA chooses to procure or custom written by the SEA. In such cases, for these components to administer Smarter Balanced related assessments or items, these components must certified by Smarter Balanced as a valid replacement. In period B, the ARB will work with Smarter Balanced in defining guidelines for this certification.

4. High Level System Component Diagram

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

4. High-Level System Component Diagram

4.1. Logical Responsibility Groupings

Introduction

This section contains the high level system component of the Smarter Balanced assessment system. It aims to explain and describe the purpose of each component that makes up the assessment system, and the purpose of each component.

The components fall into one of the following logical groupings from a development point of view. The components in each grouping have similar development needs and technologies. The components within each group have a higher interdependency.

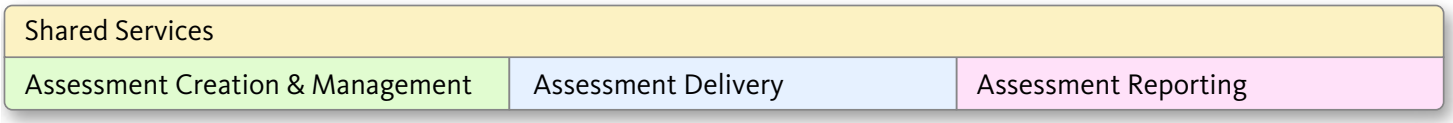

Figure 4.1 Logical Component Responsibility Groupings

Here are the brief descriptions of what each grouping is responsible for:

Assessment Creation and Management
These components manage the process and workflow of the creation and life cycle of Assessment Artifacts.

Assessment Delivery
These components deliver the assessment to the students and gather the data and metadata about the assessment.

Assessment Reporting
These components analyze the assessment results and produce reports intended to improve education and benefit student learning.

Shared Services
These components assist the other components and create a cohesive system for the end users.

Definition: Component

In this document, the definition of component describes a logically separate capability that could (but not necessarily) be separately deployed and managed. This means that any component in the system could be replaceable by other implementations of the same component. A vendor solution may entail multiple components that are tightly coupled to give enhanced functionality; parts of that solution should enable the use of other implementations of these components.

As an example, a vendor has an Item & Test authoring and banking system with tight integration into its own Test Delivery component. This solution should allow a Smarter Balanced member state the capability to use another Test Delivery component.

4.2. Logical Component Diagram

The diagram below depicts the components of the assessment system. This illustration expands on the concept described above in more detail.

As you study the diagram, you will find that the Test Delivery component, under the Assessment Delivery group, is made up of two components within it. This is to denote that they are individually deployed. It is required to have a tighter degree of integration between these two components than the other components in the diagram.

You will also find descriptions of each of the components, accompanying the diagram, below.

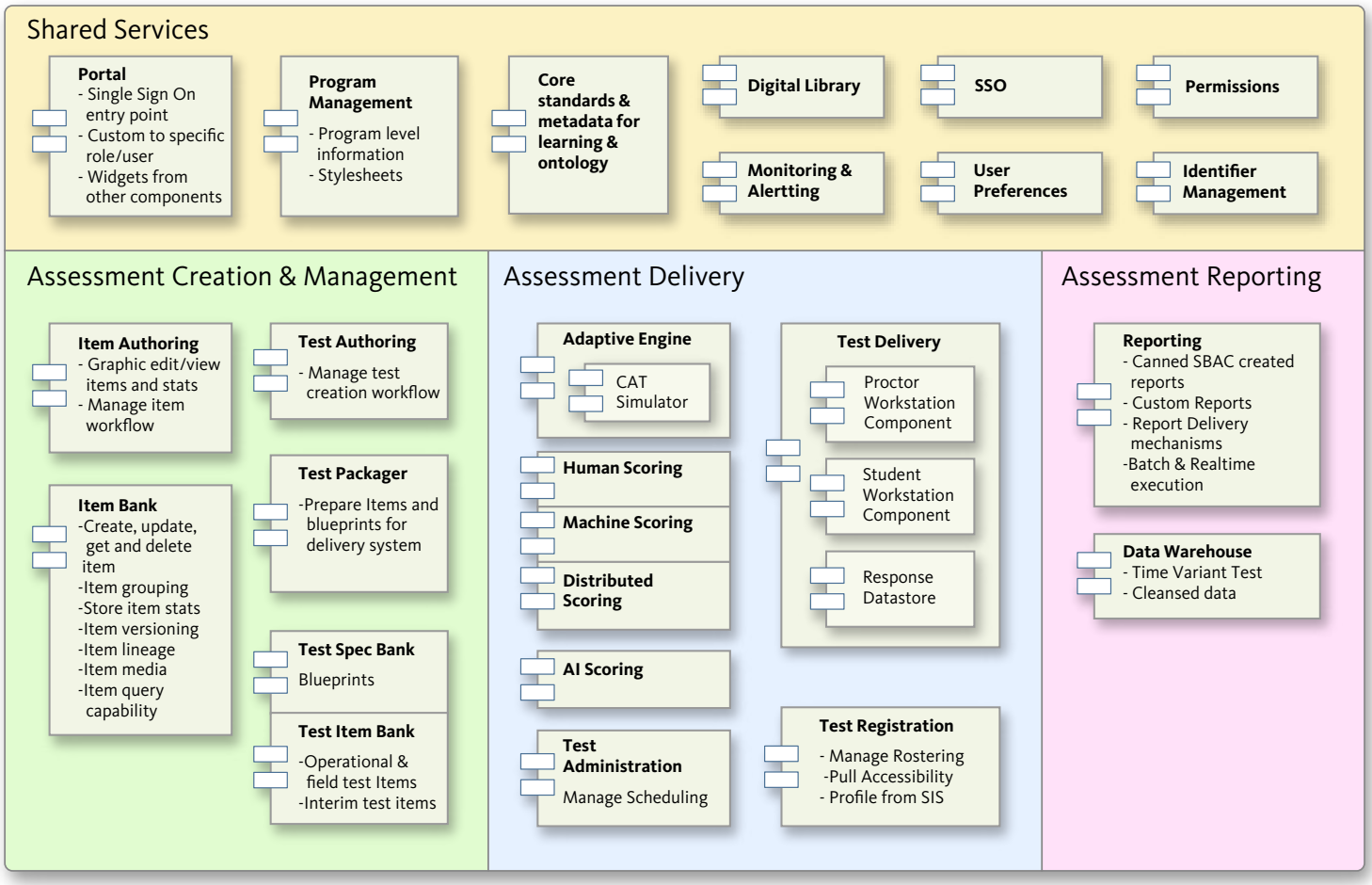

Figure 4.2 Logical Components

Portal

This is the entry point where end-users access the components of the Smarter Balanced system. It handles what components a user has access to. It allows the display of information and dashboard widgets from multiple different components. It is expected that this component be a consumer in the Web Services for Remote Portlets Specification v2.0 [http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html]. Those components that will have interfaces embedded in the portal should offer a producer WSRP interface.

Program Management

This is the master data repository and service. It is a set of services to provide data that crosses component concerns. For instance, style sheets and Common Core Standards, etc.

Digital Library

This component is an interactive teacher professional development tool to monitor professional learning goals. Teachers will primarily use this component to access resources for their own professional development. This will include resources such as documents, videos, guides and sample summative / interim tests and responses, forums. Teachers can customize their content, post their reflections, monitor their progress on implementing new practices. In addition, it also contains work area to guide teachers to identify and use the best resources for their needs; the system may also be able to use the teacher's interaction with the system to suggest additional resources.

Monitoring & Alerting

This a shared set of services that allow components to send alerts in a consistent way. Also those alerts can be monitored and acted upon in a similarly consistent way. It also will allow vendors to develop add on applications and features to use and act on these alerts.

Single Sign On (SSO)

This component is responsible for user authentication for any user interface. The components interface needs to check for an authentication token on each request. If the token does not exist, the component should redirect to the SSO system for authentication. This allows the user to only sign-in once, while still able to use all the components that they are authorized to use.

User Preferences

Similar to permissions, in order to enable a seamless user experience, having a capability to store user preferences that are used by multiple components is necessary. Components may be developed by multiple vendors, but end-users see the system as a single place to do their work. If the user sets up preferences for things that commonly exist in multiple components, that preference will apply to those components.

Core Standards & Metadata for learning data and ontology

This is the component that needs to manage the Common Core State Standards and learning metadata so that other components can reference and use them in the same manor. It is the single version of truth for these standards. When a component needs to reference a core standard, it will use the identifiers and text that have been retrieved from Core Standards component.

Permissions

This is a centralized permissions management for the systems components. It is necessary to require that components share the same permissioning capabilities in order to reduce permissions management complexity, and to allow consistent user experience across multiple components developed by different vendors.

Identifier Management

Because the Smarter Balanced system needs to integrate with other systems, it is difficult to deal with identifier codes from differing systems that refer to the same object. The system may have to deal with multiple representations of the same object (i.e. Student identifiers from SIS system and state systems). Many legacy systems have mistakenly used identifiers such as SSN or reused identifiers; this can cause systems that rely on these identifiers numerous problems. It is the goal of the Identifier Management component to store a mapping between Smarter Balanced and non-Smarter Balanced identifiers, so that the Smarter Balanced components can rely on their own identifiers and be managed in a single, centralized location. When the integration points between the outside system and Smarter Balanced are processing data, the Identifier Management component can be used to help map between said systems.

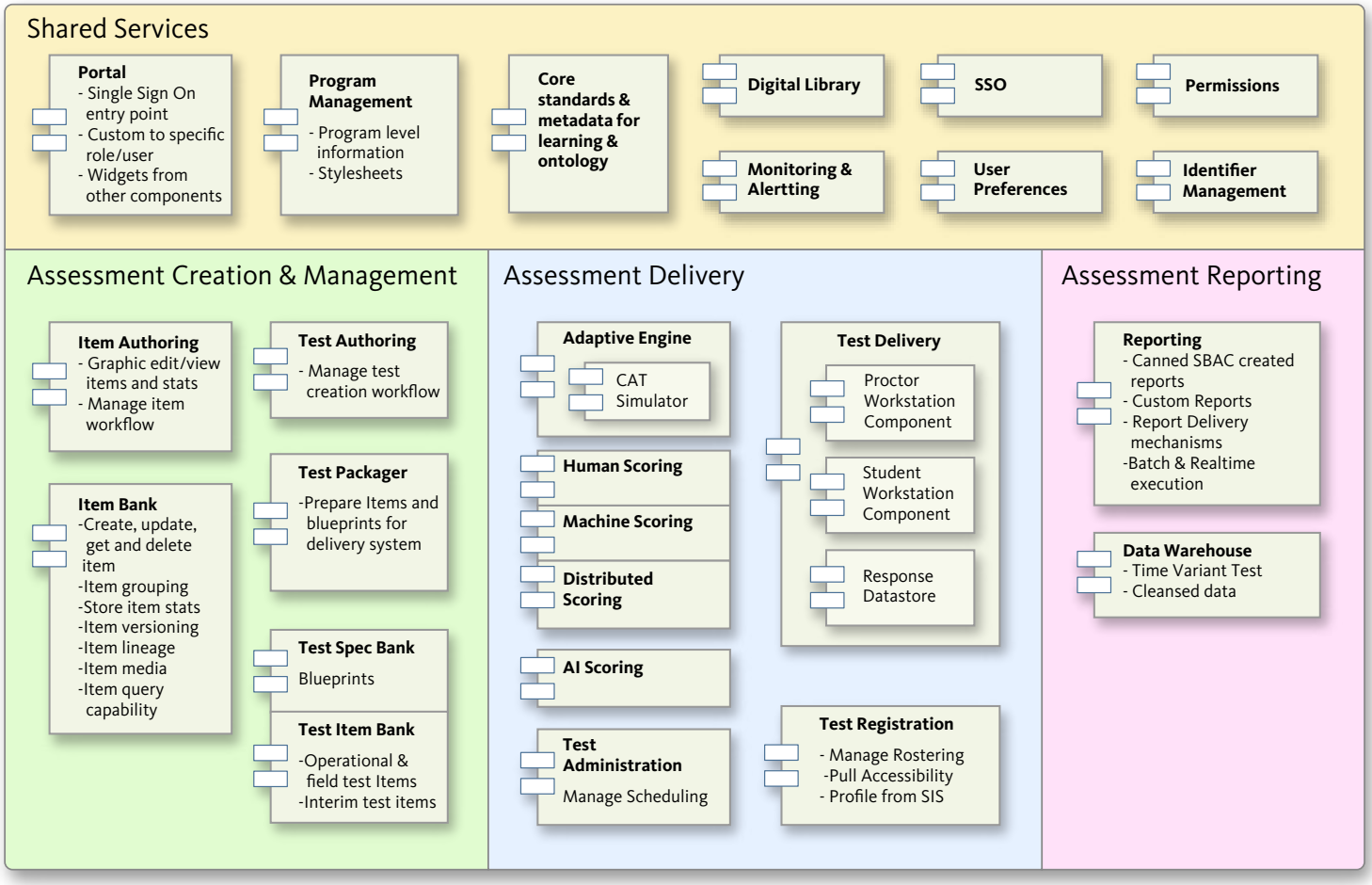

Figure 4.2 Logical Components (repeated)

Assessment Creation & Management

Item Authoring

This is a graphical interface used for the authoring and workflow related to item creation. It interacts with the Item Bank component.

Item Bank

This component is responsible for:

- Storing and retrieving assessment items.
- Storing and retrieving assets and metadata related to the assessment items.
- Tracking item versioning.
- Tracking item lineage. (If an item changes to such an extent that it becomes a new item, the lineage tracks what the item used to be.)
- Providing a robust search and query capability that allows searching on all types of metadata.

The Smarter Balanced instance of the Item Bank will be considered as the system of record for Smarter Balanced items. Items can be moved into other Item Bank and Test Item Bank instances. An item in SBAC's instance will be considered the definitive source of the item.

Test Authoring

This component is a graphical interface used for creating test blueprints and specifications and manage the workflow. It will interact with the Test Item Bank component and the Test Spec Bank component.

Test Packager

This component prepares the test items and the test specifications for use by the test delivery system. Test packager preprocesses assessment assets to make them more efficient for the Test Delivery component. The packager creates the assessment instrument that a Test Delivery system can consume and use to deliver the assessment.

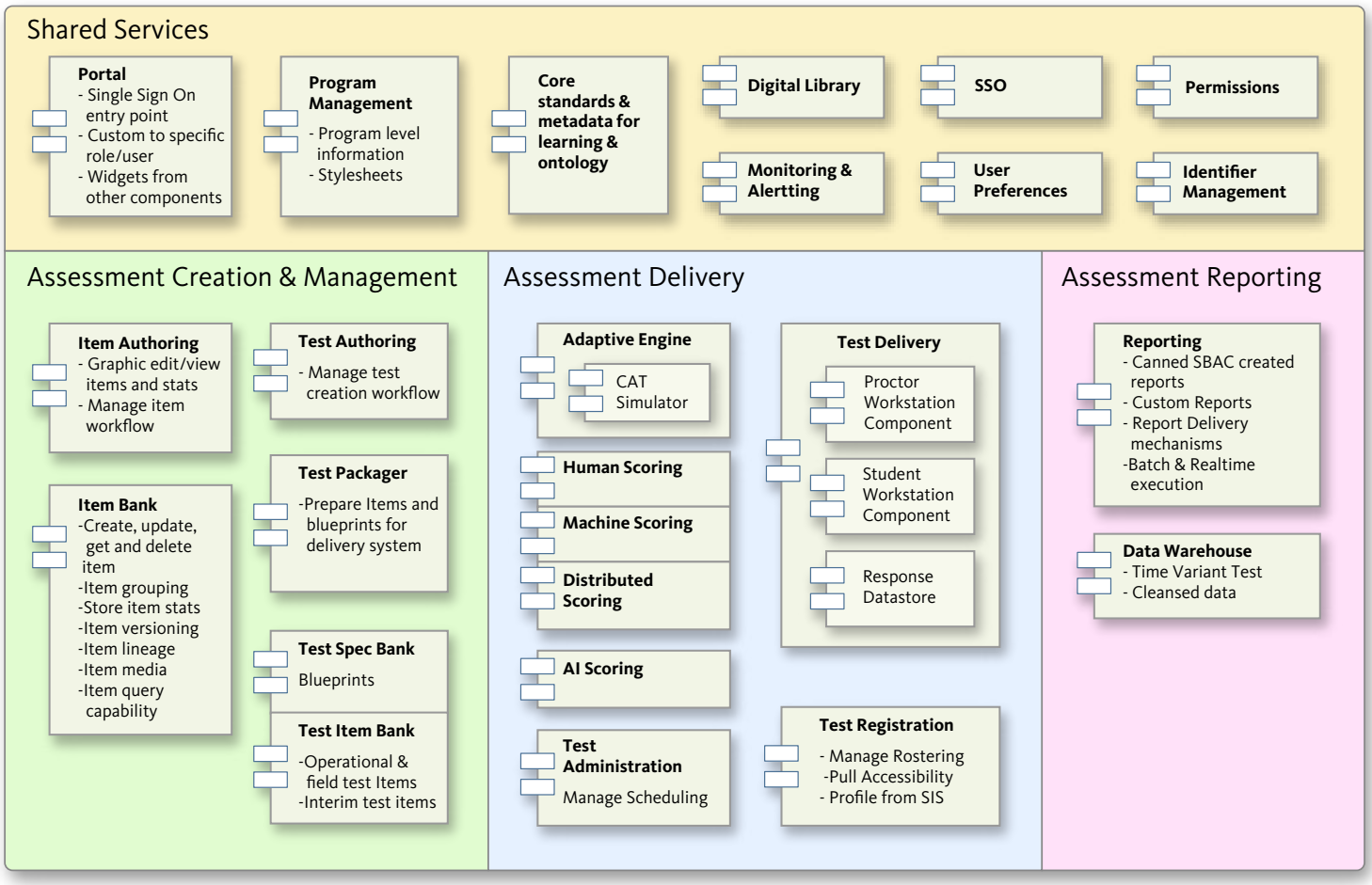

Figure 4.2 Logical Components (repeated)

Test Spec Bank

This component is a repository for test specifications, blueprints and other data about tests such as the adaptive algorithm to be used during the test.

Test Item Bank

Similar to the Item Bank, this component contains items that are in operational, field, or interim tests.

Assessment Delivery

Adaptive Engine

This component is an implementation of an adaptive algorithm. Multiple Adaptive Engines may be developed. Similar to the AI Scoring Engine component, this component will have performance issues if implemented as a network service instead of a Test Delivery component plugin. The Test Spec Bank contains metadata that defines the algorithm or engine to use. The Test Delivery Component is expected to load the correct algorithm or engine when the test is being administered.

Human Scoring, Machine Scoring, Distributed Scoring

This component provides the interface humans use to score the assessment and view rubrics on how to score the items even when the scorers are not centrally located. It also delivers those scores back to the Test Delivery and Data Warehouse components to be stored with the student responses. It also allows for the development of machine scoring capabilities that are used in conjunction with the human scoring capabilities. It should be able to use the AI Scoring engine(s) that are used in adaptive testing by Test Delivery.

AI Scoring Engine

This component programmatically scores an item in real-time while the student is taking the test. This must be a high-performing component: the adaptive engine needs to either decide the next item or select an item a few items ahead. This component must be a plugin to the Test Delivery component as network latency may impede the performance characteristics. The Test Delivery component must initialize the engine with items so the engine can preprocess and cache information to most efficiently score in real-time.

Test Administration

This component manages the capabilities and methods required for assessment scheduling, test windowing, room scheduling, proctor assignment, student assignment, and student identification methods.

Test Delivery

The overall responsibility of this component is to:

- Securely deliver the assessment to the student
- Store the student responses
- Store other information about how the student responded (i.e. time to answer, time to render for the student, etc.)
- Deliver the test items in the proper accessible format that the student needs

Student Workstation	This subcomponent interacts with the student. It delivers items to the student and gathers the responses and response metadata. It also contains the tools the student needs to take the test. (i.e. calculators, tables, accessibility tooling etc.)
Proctor Workstation	This is a subcomponent that the proctor uses to manage the test delivery. It allows the proctor to start, stop, suspend, resume, and help students when they are having issues.

NOTE: This component must be able to provide scalability and allow for deployment of additional servers as required to meet demand. In addition, it must also be designed for the highest level of recoverability, redundancy, and traceability. This component will encompass the largest number of hardware and network differences.

Test Registration

This component registers the student(s) for assessments and interacts with the Student Information System (SIS) to gather the student information and the accessibility profile, used by the Test Administration component. It must also manage staff identification for managing the a assessment event.

NOTE: It is quite understandable if the Test Registration and Test Administration components need to be merged into a single component as the features of the registration component can be seen as administration functions.

Assessment Reporting

Data Warehouse

This component contains information moved from the Test Delivery Component(s). This data should be temporal in nature so that queries against the data can be executed using different points in time. This must be a relational database (i.e. SQL). At the consortium level, the warehouse will house data from different sources. While Data Warehouses at the State level may or may not need that capability, the database schema will support it should the states elect to use it.

Reporting

This component must be able to run Smarter Balanced-created reports against the Data Warehouse, and to deliver those reports in multiple formats to authorized users who need to view them. It also must be able to generate and deliver custom-built reports that each state, LEA, etc. may create. Some of these reports may be scheduled to run at a specific date and time, repeating if need. These reports may be created when a user makes the request (i.e. near real-time).

4.3. Component Interfaces

This diagram shows the connection points between logical components, including where interoperability standards need to be defined and followed.

1. The shared services box are contains components that are required by most of the other components.

2. The dotted arrows indicate connection points between components and are labeled with either an action or an artifact that exists between the components.

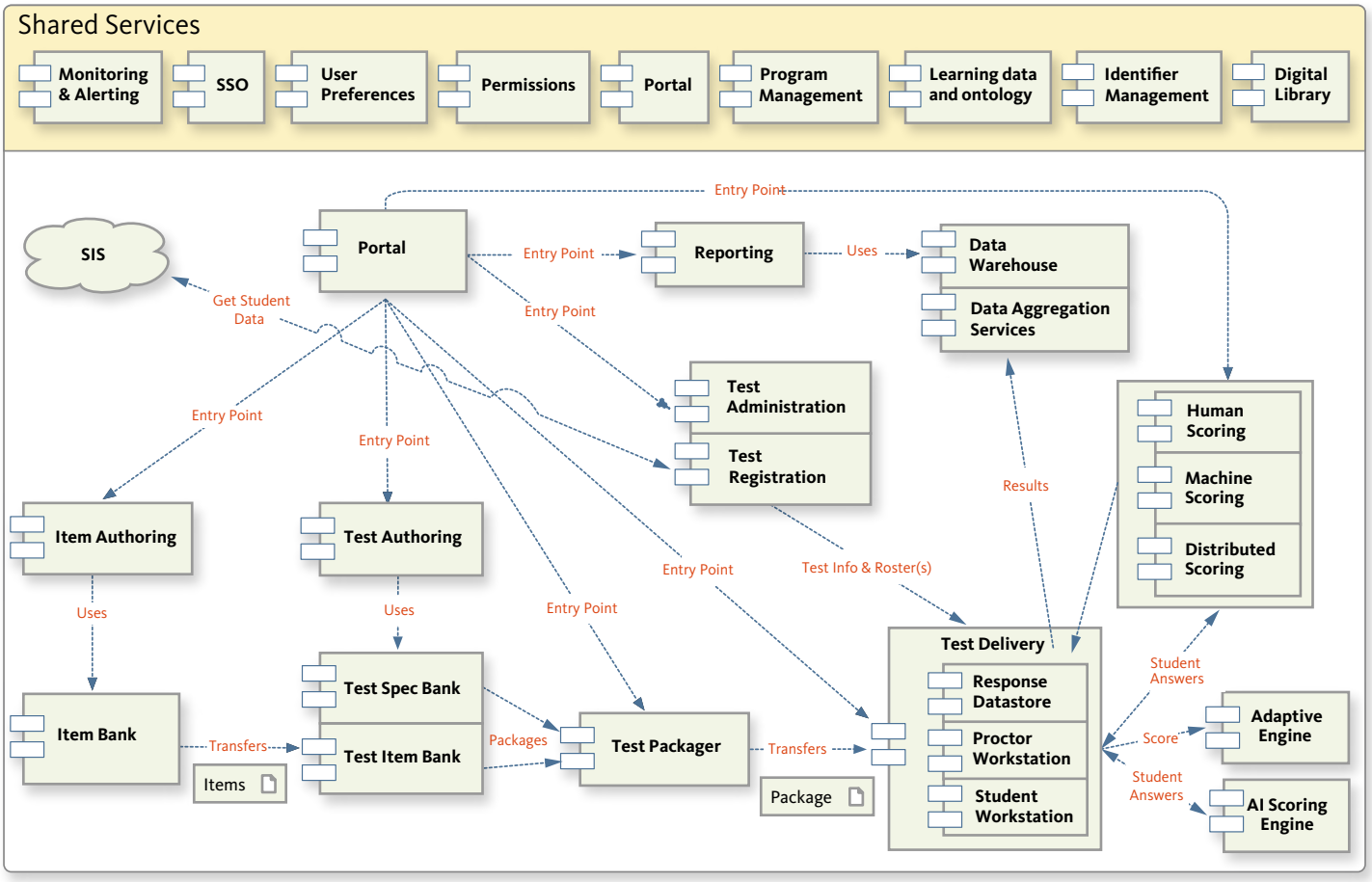

Figure 4.3 Logical Component Interfaces

4.4. Component Transport Path

The component transport path is the path that artifacts will take through the Smarter Balanced components.

The Plugin Binary Transport determines the optimal transport between components. This requires an abstract API to be developed that components can call with a consistent interface. The API uses the data format described by the accepted to standard for that assets domain (i.e. item format, student information, student response, etc.).

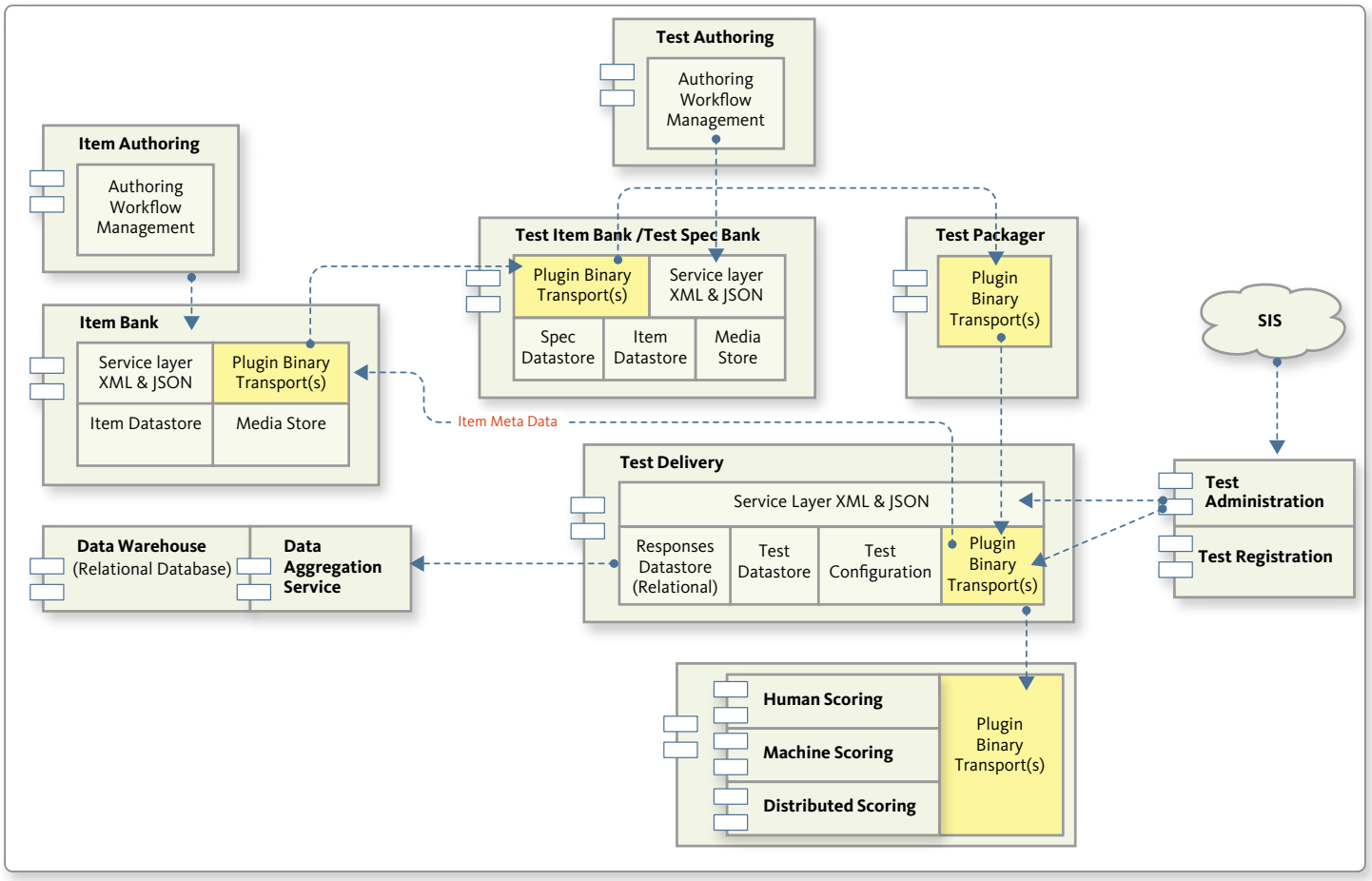

Figure 4.4 Component Transport Path

The following is a Java™ example showing how a component may use a plugin transport:

```
public void sendItems(List<Item> items,SBACTarget target) {
 SBACWriter itemWriter = SBACInterop.getWriter(SBACType.ITEM,target);
 itemWriter.write(items);

}

public List<Item> receiveItems(SBACSource source, long limit) ) {
 SBACReader itemReader = SBACInterop.getReader(SBACType.ITEM, source,
limit);
 return (List<Item>)itemReader.read();

}
public void fooMethod(List<Item> items)
{

 //export items to the filesystem
 sendItems(items,SBACTarget.EXPORT);

 // retrieve items that have been sent to us from
 // the configured source item banks and save to the datastore
 // limit to 500 at a time so as not to run out of memory
 List<Item> retrievedItems = new ArrayList<Item>();
 while((retrievedItems = receiveItems(SBACSource.ITEM_BANK,500)) != null
)

 ItemRepository.save(retrievedItems);

 //send items to the configured target Test Item Bank
 sendItems(items,SBACTarget.TEST_ITEM_BANK);
 ....

}
```

Following is an example plugin configuration file:

```
<plugins>
 <source name="ITEM_BANK" type="ITEM" description="SBAC item bank">
 <interop-standard name="QTI_2.1"
 provider="org.sbac.interop.provider.qti2_1.itembank"/>
 <transport name="SBAC_TEST_ITEM_BANK"

 provider="org.sbac.transport.hadoop">
 <hadoop-config file="/opt/hadoop-0.20.0/conf/hdfs-site.xml"/>
 <hadoop-dir dir="/sbac/test_items"/>

 </transport>
 </source>
 <source name="ITEM_IMPORT" type="ITEM" description="">

 <interop-standard name="QTI_2.1"
 provider="org.sbac.interop.provider.qti2_1.itembank"/>
 <transport name="ITEM_IMPORT_DIR"
 provider="org.sbac.transport.filesystem">
 <dir name="/home/itembank/import"/>

 </transport>
 </source>
 <target name="ITEM_EXPORT" type="ITEM" description="">

 <interop-standard name="QTI_2.1"
 provider="org.sbac.interop.provider.qti2_1.itembank"/>
 <transport name="ITEM_EXPORT_DIR"
 provider="org.sbac.transport.filesystem">
 <dir name="/home/itembank/export"/>
 </transport>
 </target>
</plugins>
```

These examples show that the plug-in binary transport is configured outside of the Java program. The SBACInterop returns an object that understands the object type and interoperability standard. The source / target object must deliver the object to the requested format, and both deliver it to and return it from the configured transport (i.e. file system, http, socket etc.). Components with well-defined sources, targets, and types can be developed independently and can be configured to use different transports and interoperability standards.

There is a case that if using the same platform and language across the components, it would be possible to share the code across these components.

Although most data that needs to be moved can be represented in an XML format, binary assets such as graphics, movies, sound files, etc. also must be moved. These assets can be large and the space to store them grows significantly when multiple instances of these file types are required as part of an assessment. This causes system memory issues and complicates the use of protocols such as HTTP. By creating a plug-able transport capability, the most efficient transport can be used between components. For example, it may be efficient to use an XML REST API to deliver test results from the Test Delivery component to the Data Warehouse component, but the transport among the Item Bank, the Test Item Bank and the Test Delivery component may use an Apache Hadoop™ Distributed File System.

Important

This will also allow multiple vendors to innovate methodologies in order enhance this transport, and incorporate other features at these integration points and plugins.

4.5. Alignment of Logical Components to Assessment Lifecycle

The following maps components to the corresponding lifecycle area.

Content Development

- Item Authoring
- Item Bank
- Test Authoring
- Test Packager
- Test Spec Bank
- Test Item Bank
- Calibration Engine

Pre-Test Administration

- Test Administration
- Test Registration

Test Administration

- Adaptive Engine
- Test Delivery
- Monitoring and Alerting

Scoring

- Scoring
- AI Scoring Engine

Reporting

- Date Warehouse
- Reporting

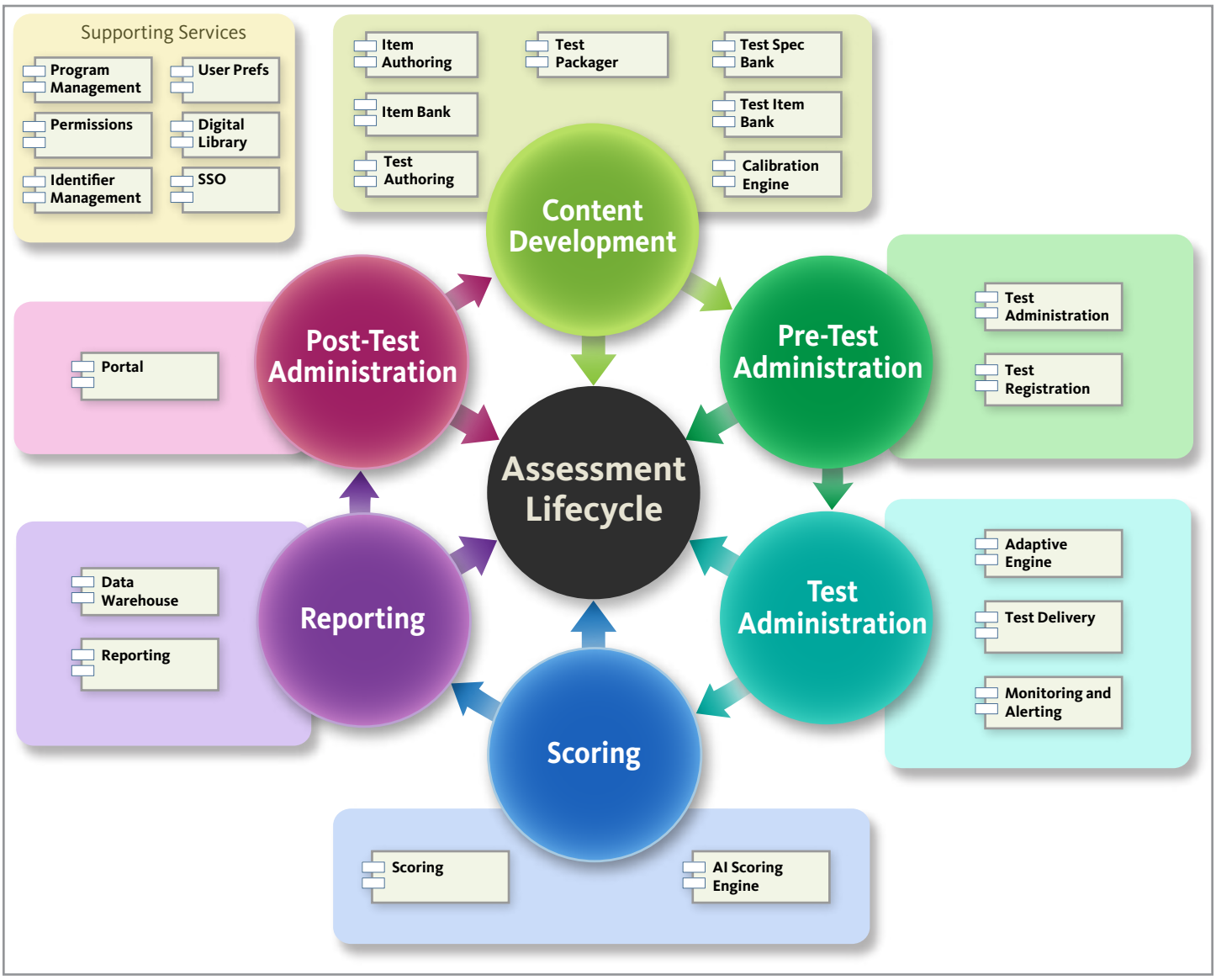

Figure 4.5. Alignment of Logical Components to Assessment Lifecycle

Post-Test Administration

- Portal

Supporting Features

- Program Management
- Digital Library
- SSO
- User Prefs
- Permissions
- Identifier Management

5. Domain Definition

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

measured progress

5. Domain Definition

This section details the assessment system's key concepts by exploring its domain in various visual representations.

Introduction

The approach used in this section is commonly known as the domain model. A domain model identifies the vocabulary, attributes and relationships among all the entities within the scope of the problem domain. This section guides readers in understanding the assessment system through a clear depiction of the concept of the domain. Domain models are also known as Conceptual Entity Relationship Diagrams.

Shown below is a series of domain models that expresses the various aspects of the assessment system: assessment creation, delivery, reporting and shared services.

The following diagrams uses Crowsfoot notation [http://www2.cs.uregina.ca/~bernatja/ crowsfoot.html], which shows relationships and cardinality, or quantities, between domain objects. These diagrams are not intended to show data-level attributes, or to be all-inclusive, but rather to identify a set of critical domain objects of which all components must be aware.

NOTE: Application architecture will define how the domain object attributes will be stored. The attributes must comply with the attributes and names as defined by the interoperability standards for that specific domain object type.

- APIP [http://www.imsglobal.org/apip.html]
- SIF [http://specification.sifinfo.org/Implementation/us/2.5/]

5.1. Assessment Creation Domain

Descriptions:

Item
A composite object that it made up of many item parts and metadata(data providing information about one or more aspects of the item) about that item.

Item Part
Includes things such as the graphics, multimedia, stem(s), item text, option groups, options, etc. that make up an item.

Item Template
A predefined item and parts meant to be used as the starting point for creating an actual item

Test Template
A predefined test specification meant to be used as the starting point for creating an test specification

Testlet
A set of related items that need to delivered together. (e.g. items that are part of a stage in a staged adaptive test)

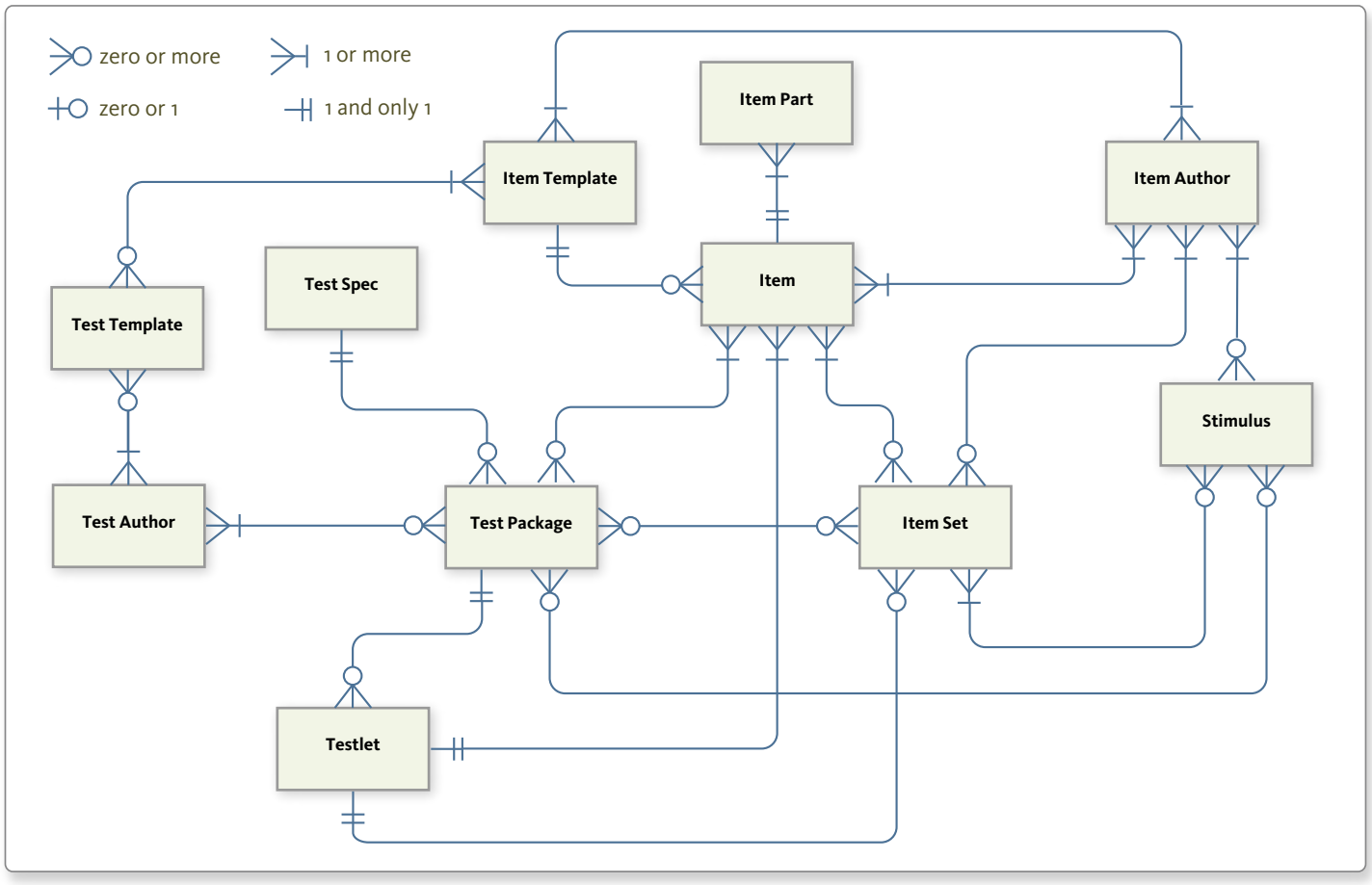

Figure 5.1 Assessment Creation Domain

5.2. Assessment Reporting Domain

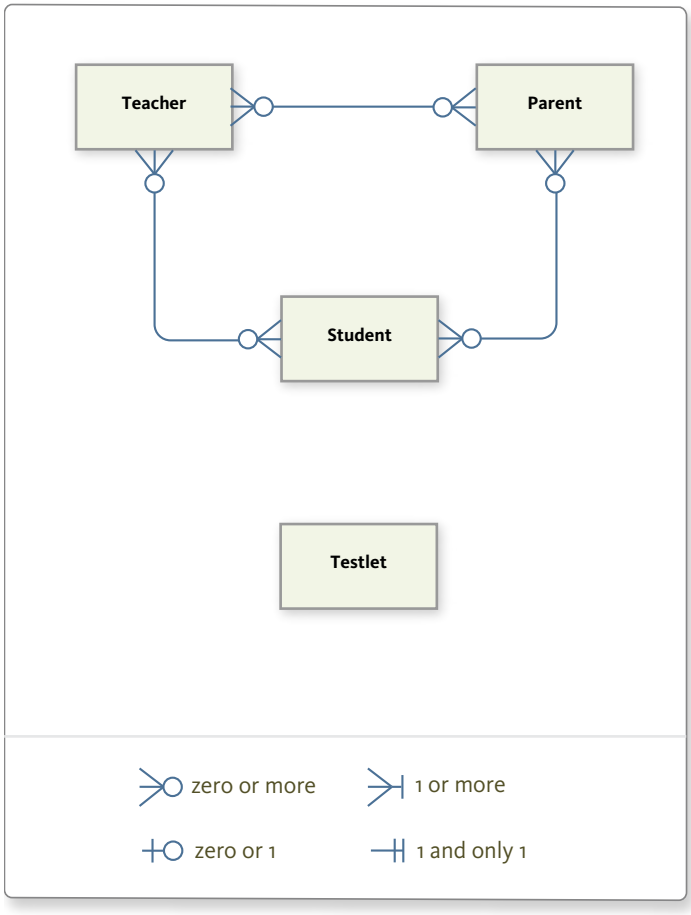

Figure 5.1 Assessment Reporting Domain

5.3. Shared Services Domain

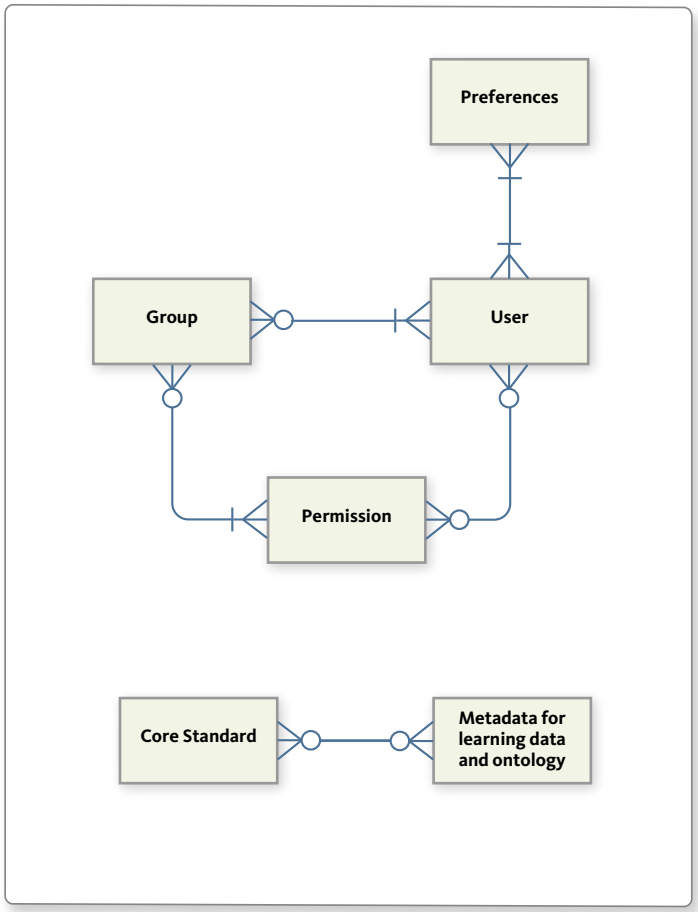

Figure 5.3 Shared Services Domain

6. Deployment and Hosting

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

6. Deployment and Hosting

A significant advantage in the architectural design of the assessment system is allowing for the need for it to be deployed and hosted in a number of different environments. The reason is that the assessment system is to be used for member states, a single state, a district, and a school, which demand varying methods for deploying the same system to all of the above. Therefore an adaptable architecture has been put in place to cater for these scenarios, which we will explore here.

Key considerations for deployment and hosting are:

- The physical location (datacenter) and its attributes such as network connectivity, clustering, security, availability, and backup facilities;
- Application architecture to support data from multiple tenants and partitioning.

6.1. Physical Location

The deployment hierarchy, from the highest level to the lowest level, follows:

- Consortium
- Groups of states (Need to know if this is a possibility)
- State
- LEA (Districts, Counties, etc.)
- School
- Classroom

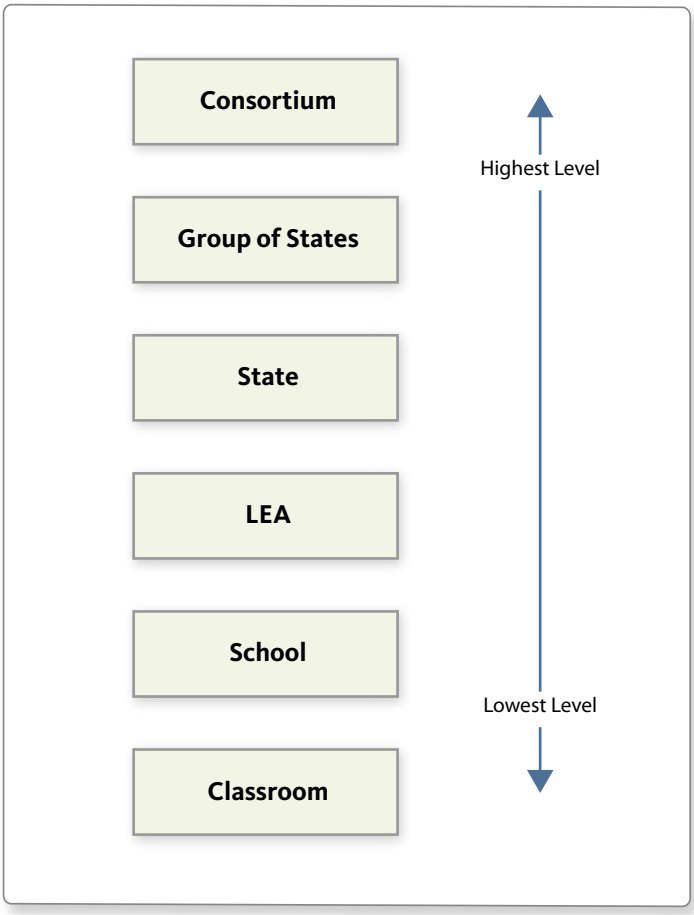

Figure 6.1 Deployment Hierarchy

As a baseline, all components must be deployable at the Consortium level. For performance and connectivity purposes, some components may reside at lower levels in the hierarchy. For example, a state or LEA may use their own Smarter Balanced-certified component. The deployment must consider all other non-functional requirements for these components.

1. Performance and connectivity reasons may force some components to reside lower in the hierarchy.

2. A state or LEA might use their own Smarter Balanced certified component.

6.2. Application Architecture

The proposed assessment system is to utilize multi-tenancy and partitioning in its design, to implement the flexibility that is required on the above hierarchies.

Multi-tenancy is the ability for a single instance of a component to host data pertaining to multiple tenants. For example, if a single Test Delivery system instance hosted at the Consortium level but serves multiple states. The Test Delivery system must be modeled to allow each state access only to data that pertains to the state. (see Figure 6.2) Partitioning creates a smaller, targeted instance of a component or a group of components to be deployed at the lower levels of a hierarchy. Components that are deployed as partitioned must have the ability to sync data with the Consortium-level component. This is most critical for components such as Test Administration, Registration, Delivery and Scoring systems. When components are to be deployed in a partitioned fashion, they will still need to have the ability to synchronize data up to the consortium level component.

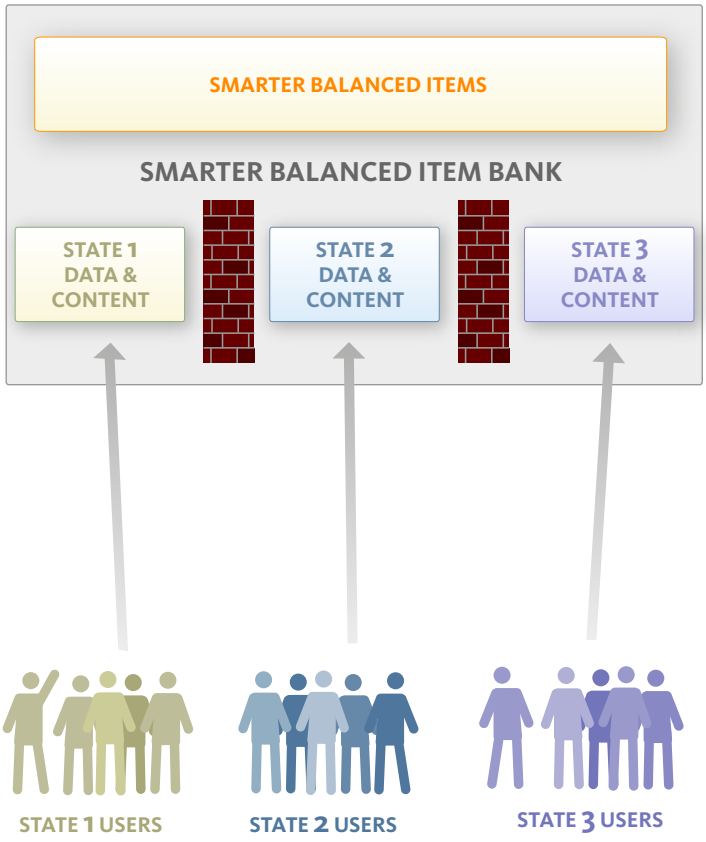

Figure 6.2 Muti-Tenancy

6.3. Scenarios

Following is a list of some possible scenarios. Please note that the following scenarios are for illustration purposes only; it is not an exhaustive list that shows all possible permutations.

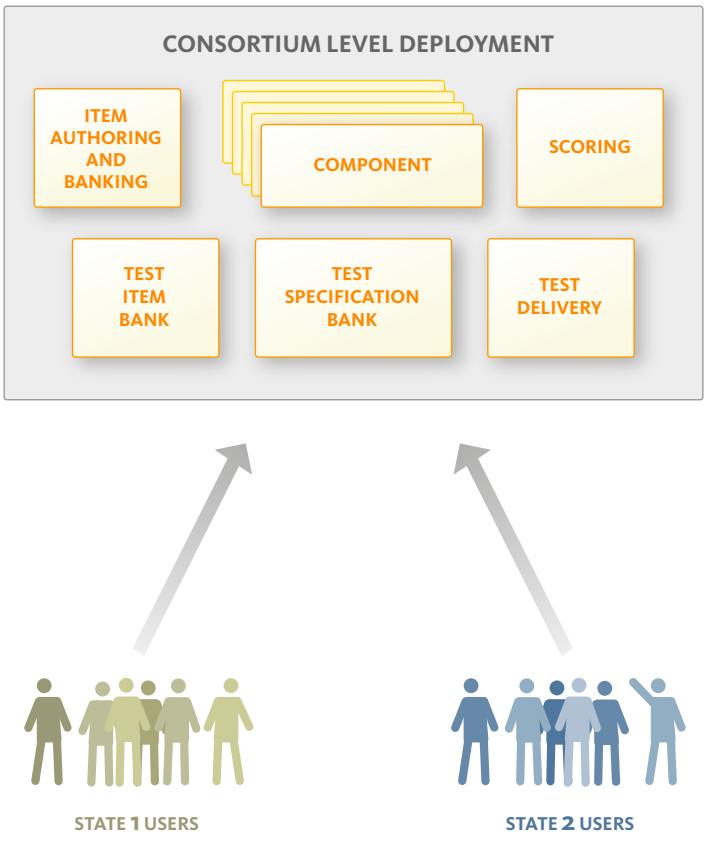

Figure 6.3.1 Deployment Scenario 1

Scenario 1 (Homogenous): All components are deployed at the consortium level, and one or more states use the components without modification.

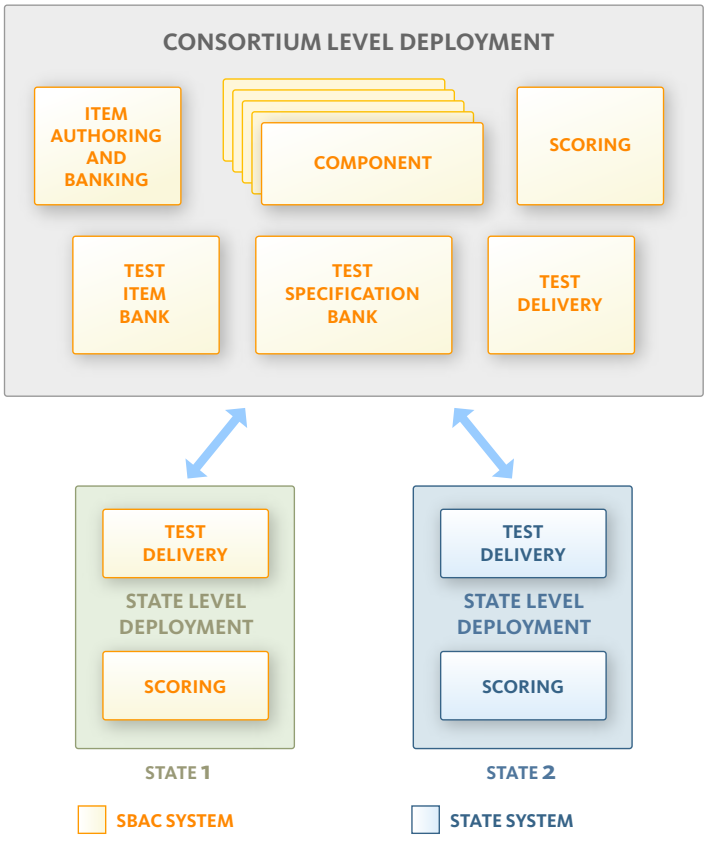

Figure 6.3.2 Deployment Scenario 2

Scenario 2 (Heterogeneous): Some states deploy the Smarter Balanced Test Delivery and Scoring system components at the State level for performance reasons, while others deploy the Test Delivery & Scoring System at the State level and deploy all other components at the Consortium level.

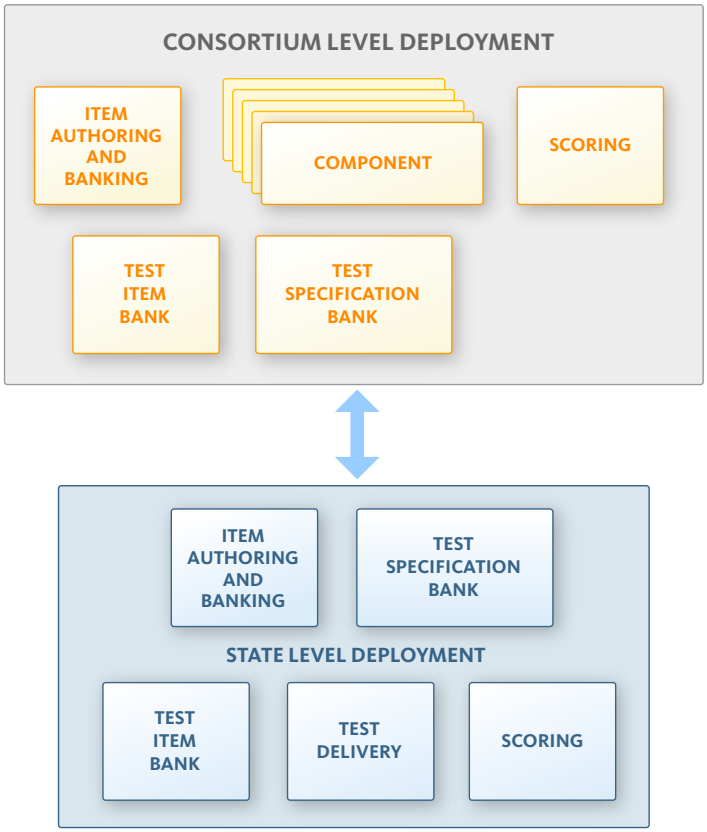

Figure 6.3.3 Deployment Scenario 3

Scenario 3 (Heterogeneous): States may add their own, state-specific items and test with the Smarter Balanced Item and Test authoring components, or use their own authoring tool to add items and tests.

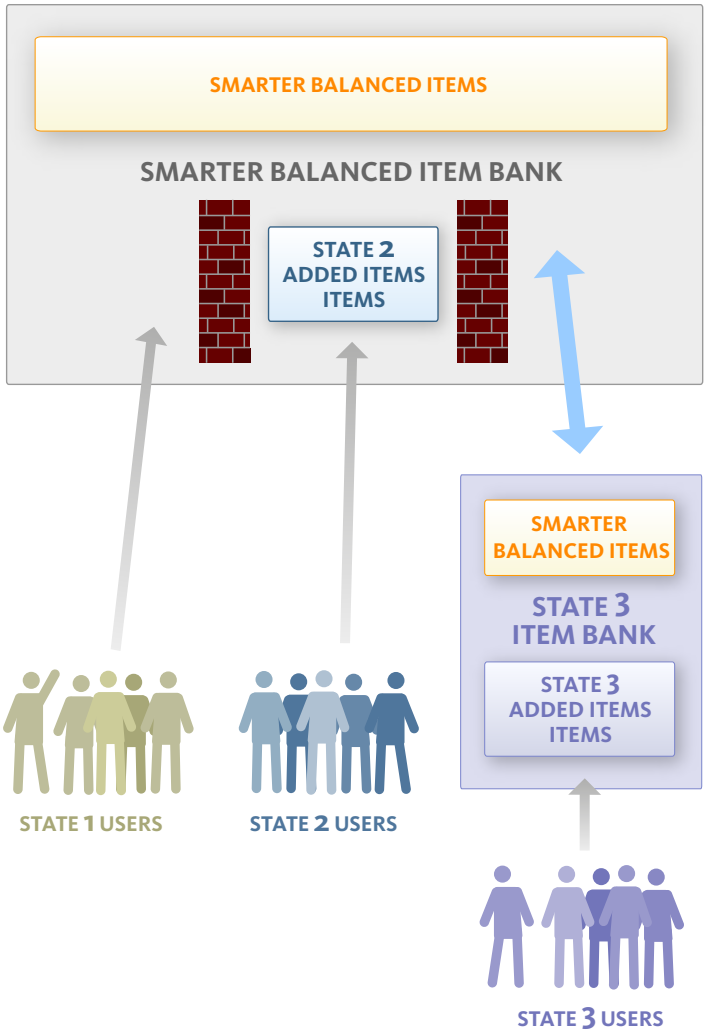

Figure 6.3.4 Deployment Scenario 4

This scenario shows some common models of how the Item Bank can be used. In this scenario:

- State 1 is using the Smarter Balanced Item Bank as-is and only has access to the Smarter Balanced Items.
- State 2 is using the Smarter Balanced Item Bank and have added their items, which are only accessible to state 2.
- State 3 is using their own Item Bank (that is Smarter Balanced certified) and have the Smarter Balanced items and their own items in it.

6.4. Deployment and hosting requirements

This list details the deployment and hosting requirements for the assessment system. They act as guiding principles to the design and implementation of the assessment system regardless of which way the system is to be deployed or hosted.

1. All components must be deployable at the Consortium level and available for use.

2. Architecture and its implementation must support installations at the Consortium level, state level, or the LEA level. It is conceivable that there might be some components that are used at a Consortium or state level while some other components are closer to the end-user.

3. Architecture and its implementation must support cloud-based hosting services, as well as traditional hosting options.

4. Components must support multi-tenancy at the state level and above. (The architecture does not need to support at the LEA level or lower.)

5. Components needs to support partitioning.

6. When components are deployed at the lower levels of the hierarchy, they must still support all security requirements and other non-functional requirements, such as Item Security, Test Security, and Student Data Security.

7. When states or other entities on the hierarchy choose to deploy the Smarter Balanced components deployed at the state level or a lower level, Smarter Balanced items and Test Data Security cannot be compromised. Only approved personnel may have access to the items and data.

8. Smarter Balanced Items can only be exported to Smarter Balanced-certified state systems. Items cannot be exported to LEA's or other organizations lower in the hierarchy.

9. If applicable, the Smarter Balanced instance of Item Bank must support state-specific items.

7. Data Architecture Definition

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

7. Data Architecture Definition

Having described the number of ways that the assessment system could be deployed and hosted across the different levels of the hierarchy, it is now appropriate to detail the data architecture of the assessment system and in particular how it can best serve the intended purposes of the system via its design and execution.

This section will explore the different aspects of the assessment system in terms of data architecture, from creation, delivery, reporting and management. At this point, it is necessary to highlight some conventional schools of thought with regards to database architecture and design, for contextual purposes:

In the past, all data was assumed to be stored in a relational database. This has caused applications to be developed using object relational mapping tools. This has then led to an impedance mismatch between how an application uses its data and how the data is stored. In order for an application to use a rich object hierarchy, the relational database may use multiple tables in order to represent this hierarchy. This causes the need to do multiple joins in order for the application to instantiate the objects it needs. As a result, it causes a lot of computing cycles to be consumed for little benefit.

There has been a movement lately, that has been driven by the explosion of Web 2.0 applications, to move away from using relational databases. This movement is referred to as "NoSQL" [http://en.wikipedia.org/wiki/NoSQL]. There are many different styles of NoSQL databases. This website, NoSQL-database.org [http://nosql-database.org/], lists many types of NoSQL databases and the kind of datastore that they are.

Some of the more interesting documents that may be applicable (but not limited) to the Smarter Balanced system are:

- Document-Oriented Database [http://en.wikipedia.org/wiki/Document-oriented_database]
- XML Database [http://en.wikipedia.org/wiki/Xml_database]
- Graph Database [http://en.wikipedia.org/wiki/Graph_database]
- Key Value Stores [http://en.wikipedia.org/wiki/Distributed_hash_table]

Here is also a ThoughtWorks article on NoSQL [http://www.thoughtworks.com/articles/nosql-comparison], for reference.

The NoSQL databases generally provide for better horizontal scaling than traditional RDBMS databases and can be placed in multiple data centers allowing the data to be synchronized or replicated with other nodes. Most of the NoSQL solutions are open-source and have a great community of users supporting each other; this will make the solution cheap for the schools and school districts. The NoSQL solutions are also better at running on commodity hardware, not requiring investments in special hardware.

Another concept that is worthy of note here, is that the data from one component should not be exposed to another component directly. Drawing from the current best practices, this kind of information and data exchange should be accomplished via the usage of APIs. This is a fundamental change from conventional concepts and it is discussed extensively in this section and throughout the document.

7.1. General data architecture principles

You will find in the following list the guiding principles of the data architecture of the assessment system. These principles apply to all of the following aspects of the assessment system aforementioned: creation, delivery, reporting and management.

1. Components should not access other components' data store directly. A component that wishes to offers its data to other components should implement services to expose the data for other components use. This reduces dependencies on how the data is stored and allows the components to evolve independently and allow the component to store data in the format that is best suited for the component.

2. Use a storage mechanism that fits the intended use of the data.

3. The storage mechanisms must allow for multi-tenancy.

4. Each domain data element is owned by some component, that component must be the source of truth for that data element. For example, the Test Authoring system owns the tests and hence will be the source of truth for tests, and will generate all the new tests and assign them keys which can be referenced by other systems or components. No other system or component should be able to create new tests or modify them. This will ensure that all tests conform to certain standards and data rules.

5. Determine if a history of the domain object needs to be maintained, so that point-in-time data is maintained. For example, if students take the test "123", then the version of the test that was taken would also need to be saved so that it can be referenced later; or the version of the test taken should be saved (embedded) in the score or results.

6. All services and / or databases should not accept data to be stored that does not follow "minimum data needed" rules. For example, to create a user, if at minimum username, password and email are needed then the service should not accept anything less than such.

7. Each entity must have keys generated such that they are unique across the system, such as UUID.

8. When using relational databases, as a principle, tables must be normalized to the third normal form unless there is a strong reason not to do so.

For further reference, this is an article by Martin Fowler, Chief Scientist at ThoughtWorks: Polyglot Persistence [http://martinfowler.com/bliki/PolyglotPersistence.html? utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+PlanetTw+%28Planet+TW %29].

7.2. Assessment creation and management

The banking components in this group fit more into a document style of datastore. When querying the datastore the usual intent is to return the full item, group of items or test specification. If the datastore was relational, multiple joins would be necessary to return those objects. This is not to say that users do not need to query the document database for objects that have certain characteristics (i.e. "Give me all items that have a P value = x"), but that the result of those queries are always well-known document types (item, blueprint etc.).

Another benefit of these style of databases is that they do not enforce schema compliance. This allows documents to only contain the parts that pertain to that document and other documents can contain different parts. The document schemas is then able to grow and change over time; it also simplifies the ability to version those documents. In a relational datastore supporting this capability expands the number of tables necessary and/or creates table data sparseness. Further, when making a change to the structure of the object, usually the database

schema would need to change. With a document database, the database does not actually care what the structure is.

7.3. Assessment delivery

Where the relational database would not be the most efficient, is storing items and the assets that support items. If items are properly pre-processed before the test is delivered to the student, this will allow items to act more like static web pages. This then allows the test delivery component to leverage the scaling capabilities of HTTP servers and content delivery networks. If the user interface controller is browser based, then we can remove the need for dynamic web page creation (e.g. JSP / ASP etc.), so the only dynamic storage need during the test taking, is the student test taking session info (i.e. the response data necessary for the Adaptive Engines). This server side portion only needs to deal with storing responses and calling the scoring and adaptive engines and returning the static url of the next item. This simplifies the horizontal scaling needs of the Test Delivery component. It also allows the component to use a fire and forget resilient queuing of student responses, allowing a slower relational update process to place responses in a relational database. (Resilient queuing = guaranteed message queuing software. e.g. ActiveMQ, rabbitMQ, MQSeries etc.).

The test session state could be stored using a distributed cache in order to facilitate horizontal scaling and durability. (e.g. Ehcache [http://ehcache.org/], Memcached [http://memcached.org/] etc.)

7.4. Assessment reporting

This is the perfect sets of functionality that the relational database is designed to support. The data warehouse will need to support SQL based query capability. It will also be beneficial if the warehouse also supports Online Analytical Processing (OLAP) [http://en.wikipedia.org/wiki/ Online_analytical_processing]. This will allow complicated data mining capability and the support of Pivot Tables [http://en.wikipedia.org/wiki/Pivot_table].

There are two standards that the warehouse should use to support OLAP [http://en.wikipedia.org/wiki/Online_analytical_processing].

- XML for Analysis (XMLA) [http://en.wikipedia.org/wiki/XML_for_Analysis]
- Multi-Dimensional Expressions (MDX) [http://en.wikipedia.org/wiki/MultiDimensional_eXpressions]

By supporting these standards, Smarter Balanced users can choose a range of reporting tools with differing capabilities.

8. Interoperability

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

8. Interoperability

8.1. Interoperability and Standards

Interoperability intention

Smarter Balanced will build all the components in the architecture and provide clear interfaces into every component. This is to enable states to use their own component (or vendor-supplied component). Beyond component level replaceability, there is also integration into other systems like Student Information Systems (SIS).

Interoperability requirements

1. Any component that is built as a result of the Smarter Balanced architecture needs to be replaceable, such that a state can use their own component.

2. Inter-component communication must use current standards (e.g. SIF, IMS etc.) where possible. In the event that a current standard does not cover the need, new extensions need to be emerged.

3. Smarter Balanced architecture must plan for the communication to SIS using prevailing standards.

Assumptions

1. Every school should have a Student Information System (SIS).

To be decided

Note

There is a strong possibility / likelihood that the existing standards (SIF, IMS, APIP etc.) will have significant gaps that will need to be bridged in the eventual assessment system. These gaps will only be identified during detailed application design (and in some cases, not until implementation). There should be no expectation that all of these gaps will be identified during the architecture definition phase (Period A).

Process flow diagrams

This section identifies the points at which interoperability is needed. The following are process flow diagrams with swim-lanes that depict the components involved in the flow. The points that require interoperability are the lines that cross from one swim-lane to another.

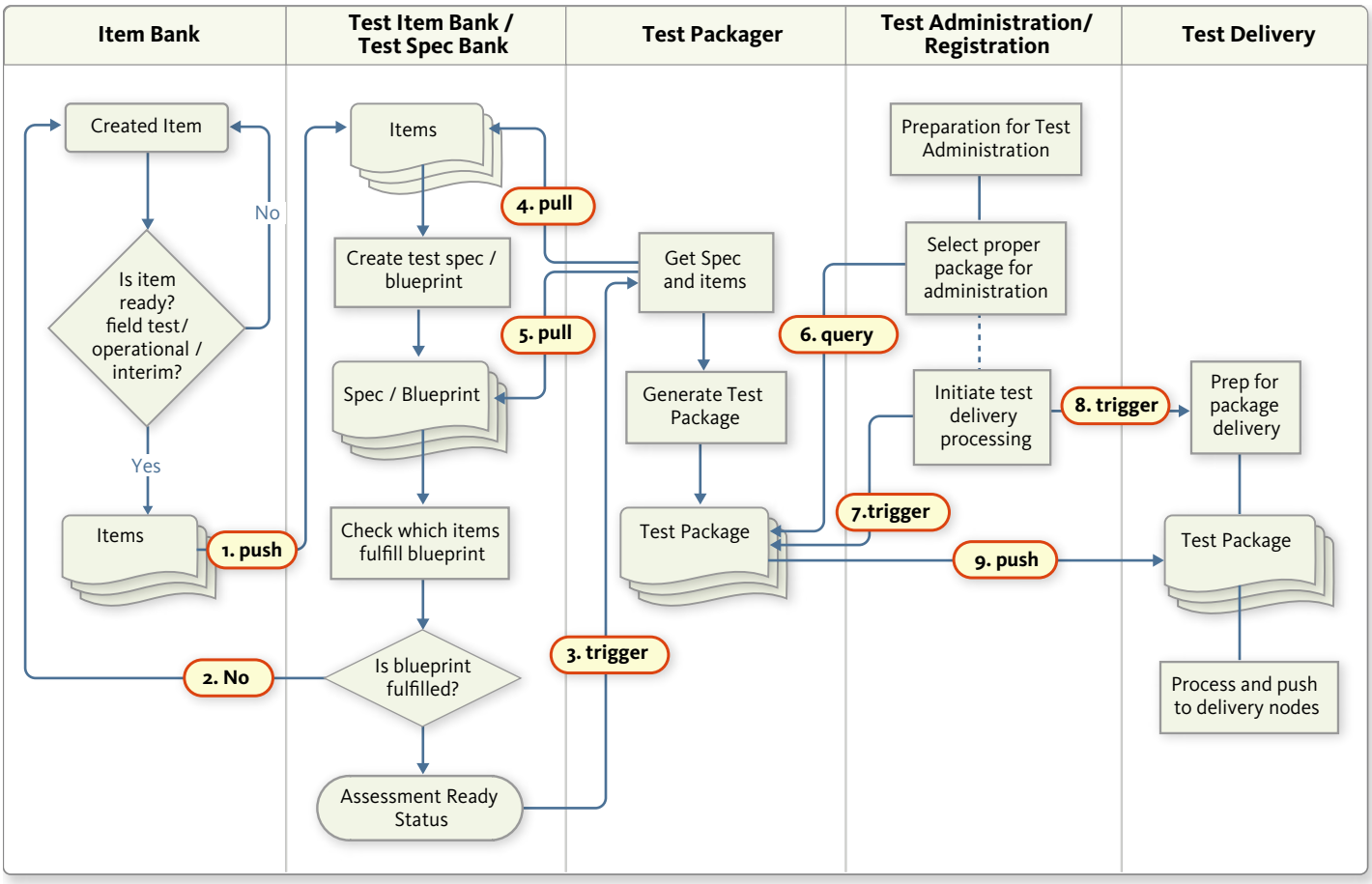

Figure 8.1.1 Item Bank to Test Delivery

Item movement from the Item Bank, packaged with the test specification and passed to Test Delivery.

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
1	Item Bank	Test Item Bank	Item(s)	APIP
2	Test Authoring	Test Item Bank	Query if Items fulfill spec	RESTful API
3	Test Authoring	Test Packager	Trigger start packaging	RESTful API
4	Test Packager	Test Item Bank	Items (pull)	APIP
5	Test Packager	Test Spec Bank	Specs / Blueprints (pull)	Specs need to be defined
9	Test Packager	Test Delivery	Test Package	A test package interoperability standard needs to be created (does APIP support this?)
6	Test Administration / Registration	Test Packager	Available packages query	RESTful API
8	Test Administration / Registration	Test Packager	Trigger move from Test Packager to Test Delivery	RESTful API
7	Test Administration / Registration	Test Delivery	Trigger prep for package delivery	RESTful API

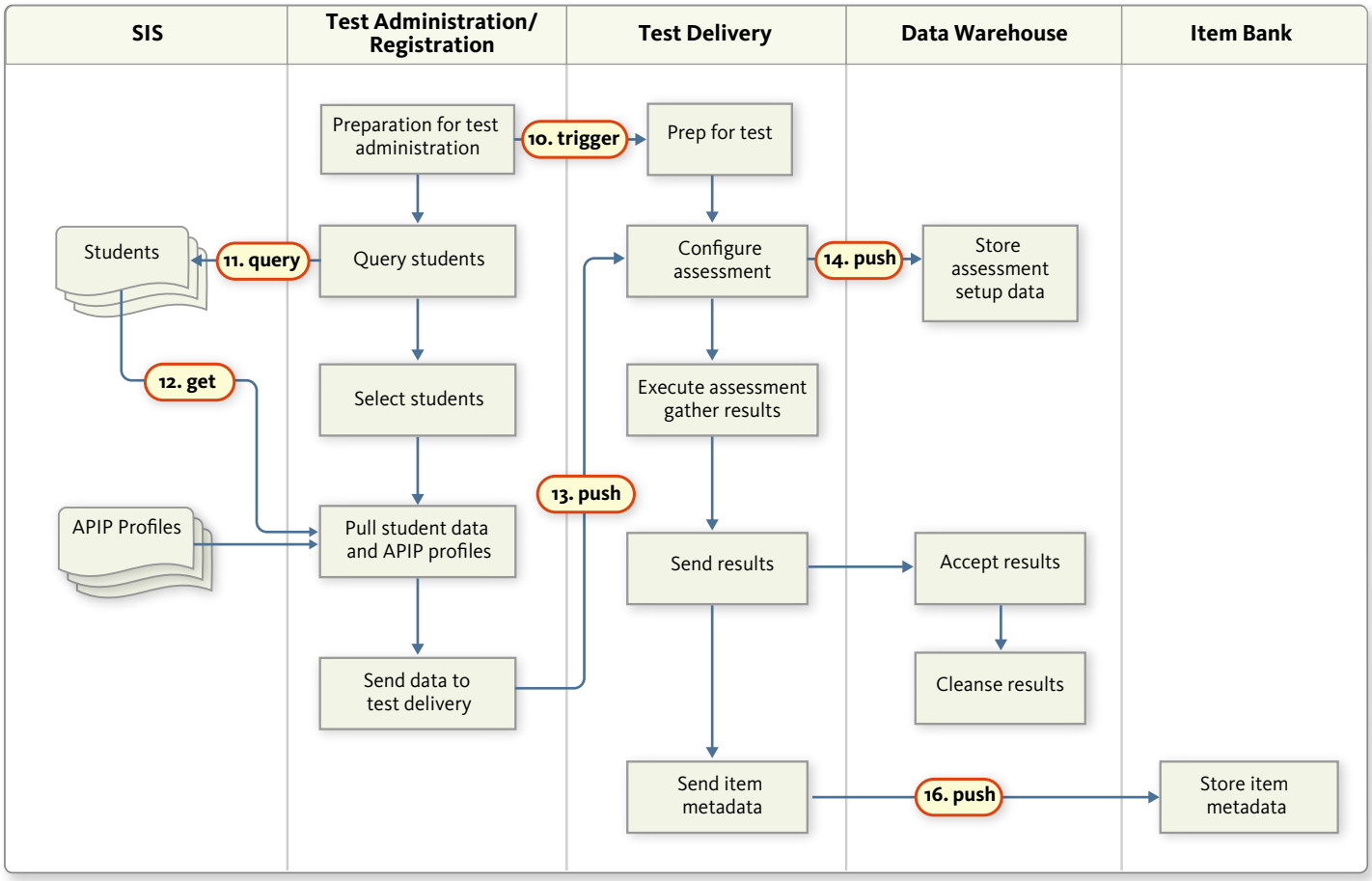

Figure 8.1.2 SIS to Item Bank

Flow of student information and responses through the system.

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
11	Test Administration / Registration	SIS	Student query	SIF (if no SIF-compliant SIS system, use a file import where the file format matches the SIF data structure)
12	Test Administration / Registration	SIS	Student data & APIP profiles	SIF (if no SIF-compliant SIS system, use a file import where the file format matches the SIF data structure)
10	Test Administration / Registration	Test Delivery	Trigger prep for test	RESTful API
13	Test Administration / Registration	Test Delivery	Push student data	SIF
14	Test Delivery	Data Warehouse	Assessment information	Interoperability standard definition needs to be created
15	Test Delivery	Data Warehouse	Assessment results	SIF
16	Test Delivery	Item Bank	Item usage data	Interoperability standard definition needs to be created

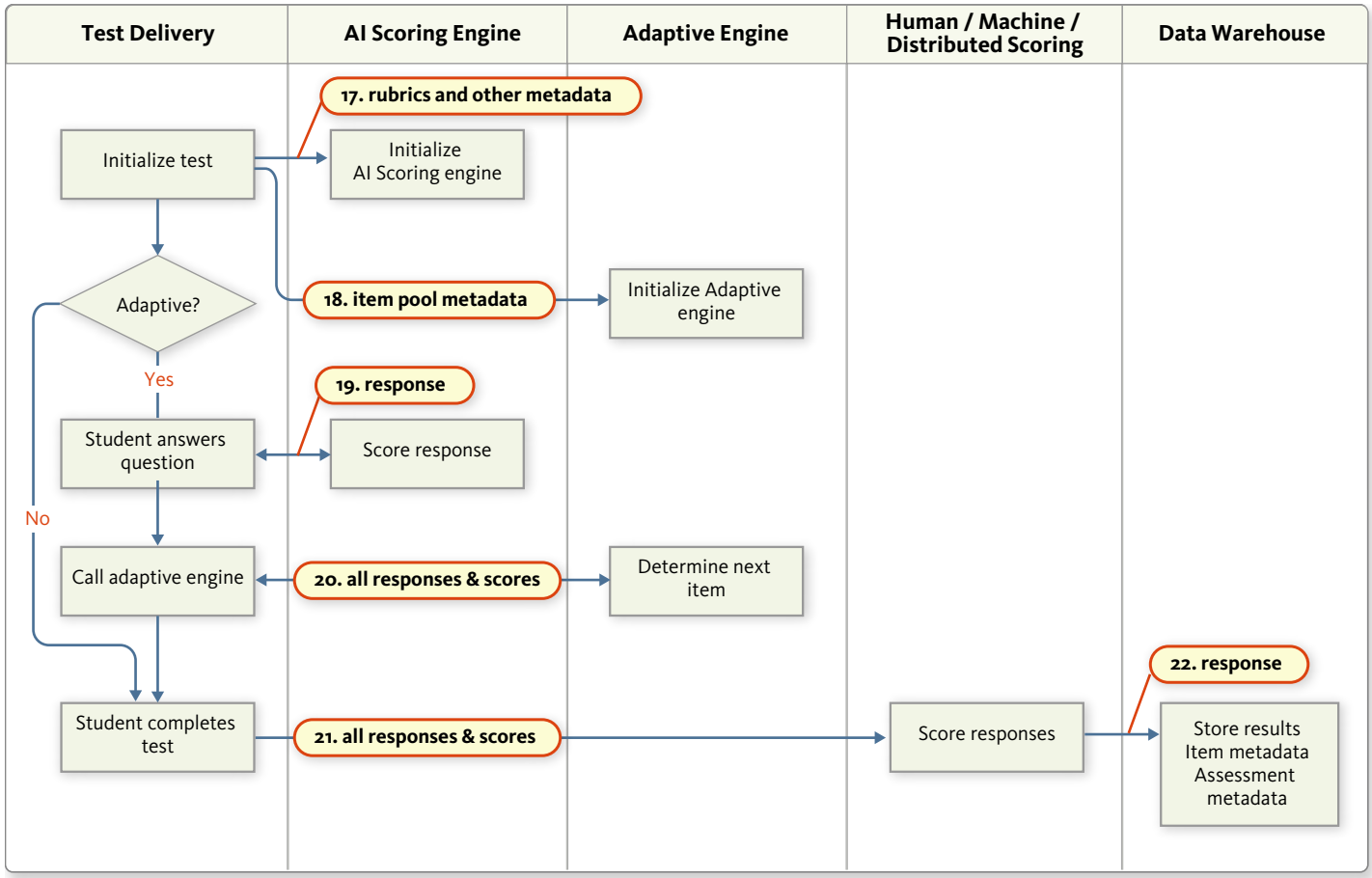

Figure 8.1.3 Test Delivery to Data Warehouse

The interaction of Test Delivery with AI Scoring, Adaptive Engine, Scoring and Data Warehouse.

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
17	Test Delivery	AI Scoring	Item metadata and rubrics	Interoperability standard definition needs to be created
19	Test Delivery	AI Scoring	Single student response	SIF
20	Test Delivery	Adaptive Engine	All responses and scores	SIF
18	Test Delivery	Adaptive Engine	Item pool metadata	Interoperability standard definition needs to be created
20	Adaptive Engine	Test Delivery	Next Item choice	Interoperability standard definition needs to be created
21	Test Delivery	Scoring	All responses and scores	SIF
22	Scoring	Data Warehouse	All responses and scores	SIF

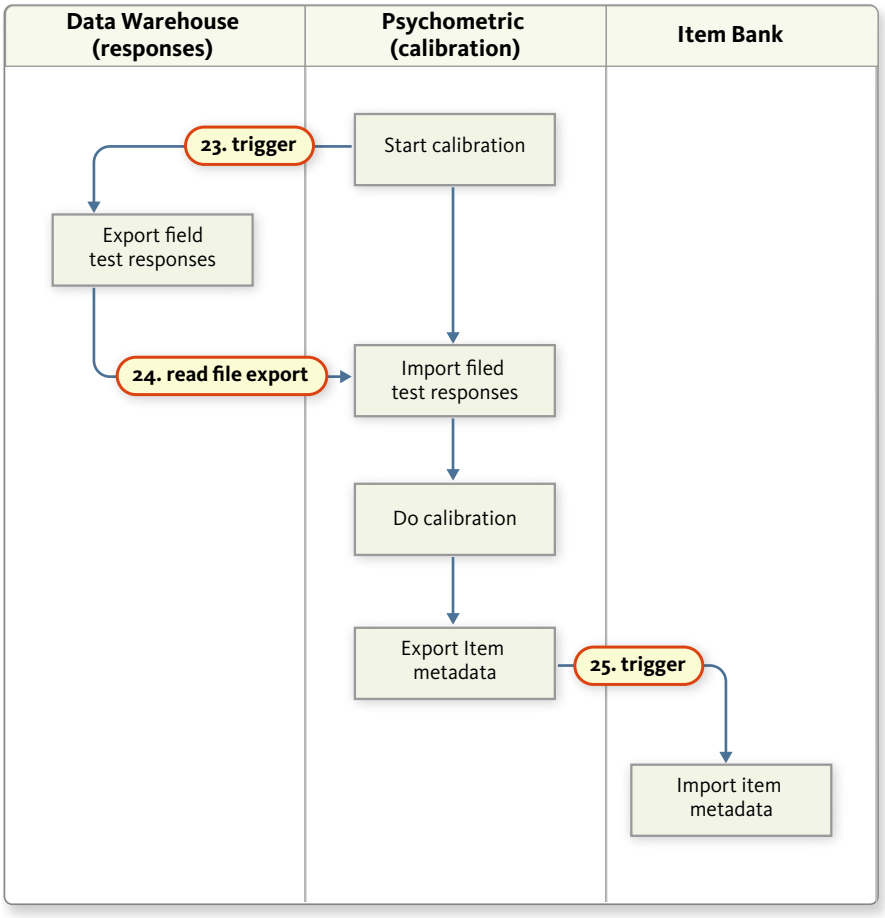

Figure 8.1.4 Data warehouse to item bank

Flow of student information and responses through the system.

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
23	Psychometric (calibration)	Data Warehouse	trigger field test responses report	SIF (where data standards exist) or CSV (Comma Separated Value)
24	Psychometric (calibration)	Data Warehouse	Read export file	SIF (where data standards exist) or CSV (Comma Separated Value)
25	Psychometric (calibration)	Item Bank	export item metadata and trigger Item Bank Import	Interop standard definition needs to be created

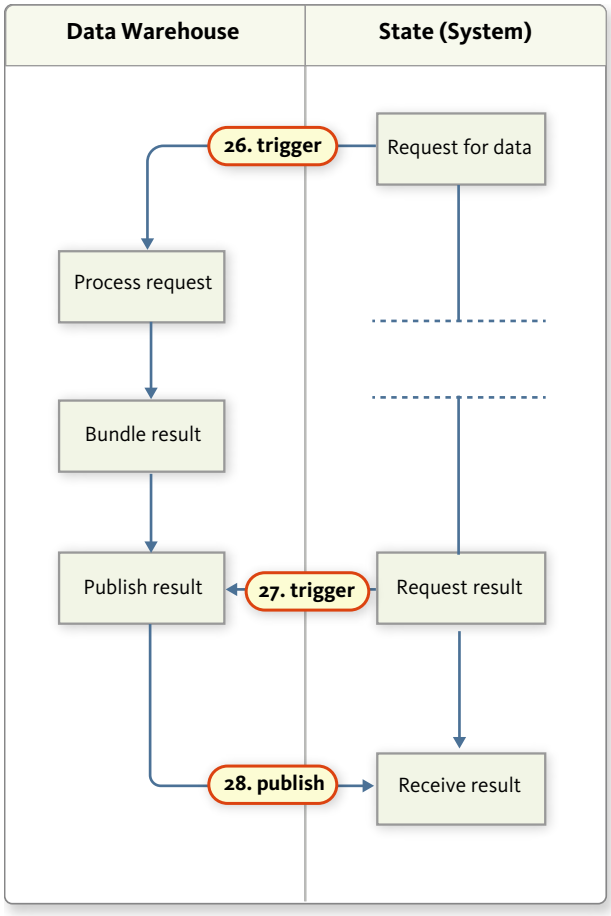

Figure 8.1.5 Data warehouse to State system

Flow of Psychometrician pull of response data to calibrate the items and push them back into the Item Bank.

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
26	State System(s)	Data Warehouse	trigger request for data	RESTful API
27	State System(s)	Data Warehouse	trigger request for data result	RESTful API
28	Data Warehouse	State System(s)	Publish result	SIF (where data standards exist) or CSV (Comma Separated Value)

8.2. Interoperability matrix

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
1	Item Bank	Test Item Bank	Item(s)	APIP
2	Test Authoring	Test Item Bank	Query if Items fulfill spec	RESTful API
3	Test Authoring	Test Packager	Trigger start packaging	RESTful API
4	Test Packager	Test Item Bank	Items (pull)	APIP
5	Test Packager	Test Spec Bank	Specs / Blueprints (pull)	Specs need to be defined
9	Test Packager	Test Delivery	Test Package	A test package interoperability standard needs to be created (does APIP support this?)
6	Test Administration / Registration	Test Packager	Available packages query	RESTful API
8	Test Administration / Registration	Test Packager	Trigger move from Test Packager to Test Delivery	RESTful API
7	Test Administration / Registration	Test Delivery	Trigger prep for package delivery	RESTful API
11	Test Administration / Registration	SIS	Student query	SIF (if no SIF-compliant SIS system, use a file import where the file format matches the SIF data structure)
12	Test Administration / Registration	SIS	Student data & APIP profiles	SIF (if no SIF-compliant SIS system, use a file import where the file format matches the SIF data structure)
10	Test Administration / Registration	Test Delivery	Trigger prep for test	RESTful API
13	Test Administration / Registration	Test Delivery	Push student data	SIF
14	Test Delivery	Data Warehouse	Assessment information	Interoperability standard definition needs to be created
15	Test Delivery	Data Warehouse	Assessment results	SIF
16	Test Delivery	Item Bank	Item usage data	Interoperability standard definition needs to be created

line#	Source Component	Target Component	Domain Objects(s) or Trigger	Suggested Standard
21	Test Delivery	Human Scoring	Item (including rubrics), Student responses	SIF
17	Test Delivery	AI Scoring	Item metadata and rubrics	Interoperability standard definition needs to be created
19	Test Delivery	AI Scoring	Single student response	SIF
20	Test Delivery	Adaptive Engine	All responses and scores	SIF
18	Test Delivery	Adaptive Engine	Item pool metadata	Interoperability standard definition needs to be created
20	Adaptive Engine	Test Delivery	Next Item choice	Interoperability standard definition needs to be created
21	Test Delivery	Scoring	All responses and scores	SIF
22	Scoring	Data Warehouse	All responses and scores	SIF
26	State System(s)	Data Warehouse	trigger request for data	RESTful API
27	State System(s)	Data Warehouse	trigger request for data result	RESTful API
28	Data Warehouse	State System(s)	Publish result	SIF (where data standards exist) or CSV (Comma Separated Value)
23	Psychometric (calibration)	Data Warehouse	trigger field test responses report	SIF (where data standards exist) or CSV (Comma Separated Value)
24	Psychometric (calibration)	Data Warehouse	Read export file	SIF (where data standards exist) or CSV (Comma Separated Value)
25	Psychometric (calibration)	Item Bank	export item metadata and trigger Item Bank Import	Interop standard definition needs to be created

Bulk import and export

There is a need to import and export data between components. It is expected that components format the exported data to the standard defined for that data type.

- Items - APIP
- Item metadata - the defined standard when developed (most likely extensions to IMS QTI / APIP)
- Student Info - SIF
- Student responses - SIF
- Scores - SIF
- Specs / blueprints - the defined standard when developed
- Test Package - the defined standard when developed
- Test Registration - SIF

Usually import and export implies a filesystem transport between systems (i.e. System A exports to a file, that file is moved by LAN, WAN, FTP, email or USB drive to import into System B). This can be supported by implementing a filesystem based Plugin Binary Transport (PBT). It is possible to use other transports also. It is important that these bulk files be secured, it is suggested to use Pretty Good Privacy (PGP) [http://en.wikipedia.org/wiki/Pretty_Good_Privacy] encryptions on these bulk files. It is recommended that any filesystem PBT should provide PGP capability.

9. Non-Functional Requirement Constraints

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

measured progress

9. Non-Functional Requirement Constraints

9.1. Open licensing

This section describes the types of licensing, requirements and recommendations.

Types of Open licensing

Open access(OA)

Open access licensing allows licensing access through the internet without any restriction. This type of licensing can be used for artifacts produced from the architecture phase and other creative content. For a more extensive definition, see the Wikipedia definition of Open Access. Open access [http://en.wikipedia.org/wiki/Open_access_(publishing)].

Open-source software(OSS)

Open-source software [http://en.wikipedia.org/wiki/Open-source_software] licensing can be used for all software components developed as part of Smarter Balanced architecture.

Open licensing requirements

1. All artifacts describing the architecture must be under an Open Access license.

2. All software artifacts produced from period B must be under an Open-source software (OSS) license. If a vendor is selling a proprietary solution, that solution must be made available in an OSS license.

3. Where available, OSS components must be used for building the software systems. This includes, but is not limited to:

   - Operating Systems
   - Tools used for authoring, building and testing the software components
   - Database software
   - Messaging systems

Recommendations

Not all software licenses are compatible. This article [http://en.wikipedia.org/wiki/ List_of_software_licenses] lists some of the caveats of software licensing. In particular, GPL [http:// en.wikipedia.org/wiki/GNU_General_Public_License] licensing has known incompatibilities with other licenses like Apache [http://en.wikipedia.org/wiki/Apache_License]. This article [http:// en.wikipedia.org/wiki/Comparison_of_free_software_licences] compares various licenses.

1. In period B, during construction and/or customization of components, use a consortium-level closed source license but allow components to view each other's source for integration and troubleshooting purposes. When a component is ready for production use, decide on the most appropriate OSS licensing.

2. Use a Open access license like Creative Commons [http://creativecommons.org/] for the artifacts describing the architecture.

Future OSS Project Governance

At the point Smarter Balanced moves from closed source to an open source model, management of the maintenance and enhancement of the produced software needs to be implemented. Each component will need to have a structure put in place in order to manage this. Although numerous processes have been used in the open source community, the following structure tends to surface:

Project Management

A single person or small group of people who are the managers of the the project. They are responsible deciding priorities and determining additions and removing of people who are given commit capability to the source code repository. They also define timelines for releases of that project.

Committers

These people have been approved by the project manager(s) to make changes and additions to the source. They usually have shown a keen grasp of the domain and understanding of the software, and have followed good programming practices as seen by project management.

Contributor

A developer who takes the source code and makes changes based on their own needs and wishes to contribute that change back to project for possible inclusion in the project. "Contributors" who make many contributions to the project that are accepted by project management may become "Committers".

User

A user, uses the software, helps by submitting bug reports, beta-testing the software or suggesting features.

This type of structure is usually sufficient for single projects that work on a single component, but the Smarter Balanced system is a series of many separate components that work independently but are also coupled to the other components of the system. It is necessary to put a controlling structure over these projects to make sure that they don't diverge from their goal of working with one another. This group is usually made up of project managers from the component groups. This group is tasked with coordinating with sub-projects to make sure they interoperate correctly, follow constancy in architecture, follow standard development practices and grow together with a constant vision.

It is also possible to attempt to get some existing open source organizations to sponsor these projects or to form a new foundation to support the continuation of these projects. Some examples of existing organizations are :

- The Apache Software Foundation [http://apache.org/foundation/]
- Eclipse Foundations [http://www.eclipse.org]

9.2. High-availability and scalability

Areas in high availability, scalability and performance to be considered are:

- System performance as perceived by a single user
- Scalability with large volumes of concurrent users
- Resiliency of system and recoverability
- System data capacity with large volumes of data
- High availability of systems

System performance

1. While no specific requirements have been identified during the workshops, test delivery systems must be the most resilient. These systems are highly susceptible to burst modes of operations, with large numbers of concurrent users accessing the system. The combination of Adaptive testing and AI scoring will have a significant impact on system performance and sufficient architecture pre-planning is needed to support this.

2. When tests are delivered, a significant amount of student data must be collected, item, answer, score, comments etc. The architecture must consider data volumes and purging strategies.

3. Network bandwidth and reliability must be considered and the architecture must make appropriate recommendations for critical components.

Requirements that need to be identified in the application architecture for each component

- Total number of users for each component.
- Minimum number of concurrent users that each component must support individually.
- Maximum number of concurrent users that each component must support individually.
- Amount of data that each component will store.

Principles

- Components should scale horizontally. The use of NoSQL technologies and distributed caches will better enable this in the data storage area.
- When components log events or send BI events, it must do so a in "fire and forget" fashion, such that there are no delays to the critical path functions.

9.3. Accessibility

Items must be accessible to fulfill the requirements of special needs students. The system must carry items through the system that conform to the APIP specification. The item authoring and test delivery components must allow for creating and rendering accessible items respectively. Accessibility for special needs users other than students must be taken into consideration. For example, alternate ways of rendering a color pie chart for those who are color blind.

The APIP Accessibility Needs Profile may be stored in an SIS system. This is used by the test delivery component, along with other student data, and therefore a requirement of the delivery system will be to retrieve the profile from outside systems.

APIP will require Smarter Balanced to enhance the APIP standards for accessibility.

Consideration for other types of test delivery devices (such as, Braille) will be needed.

9.4. Technology

Although the workshops did not raise any technology constraints, it was clear that LEAs have limited budgets and may have outdated technology. It is difficult to innovate using older technologies. Because the system will not be in place until 2014, assuming a target technology of 2012-era technologies may be reasonable.

10. Security

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

10. Security

Multiple types of security are required for the system.

- Component-to-component
- User authentication and authorization
- Item-level security
- Student data security

10.1. Component-to-component

Each architecture component must have a consistent capability so it can connect to other components in the system and allow authorized components to use its services. The communication channel between components must ensure that those components cannot be listened to or spoofed. Some techniques that need to be considered are:

- IP filtering
- SSL
- PGP (Pretty Good Privacy)
- SAML, etc.

10.2. User Authentication and Authorization

Users must be authenticated to the system (the system must confirm that they are who they say they are), and also must be authorized to use certain features and functionalities of the components. This user authentication and authorization must be standardized across components to allow seamless access, or users and system administrators will have difficulty maintaining the components. User authentication and authorization must also be configured in each system, to anticipate mismatches between components and difficulties in troubleshooting.

This necessitates a Single Sign On (SSO) solution and component support for the chosen solution. Among the open-source implementations that will be considered is OpenAM [http://en.wikipedia.org/wiki/OpenAM].

There are typically two types of authorization:

Role-based

End-users are given a role or set of roles. Components only permit specific roles to use specific features of the component. This requires a finite list of roles that enables components to code to those roles. The downside of this is that all roles need to be defined before or during implementation time and it will require code change across components when new roles are identified.

Permissions-based

Components define permissions (such as, a unique name, item view, item edit, test create, etc.). These permissions are associated with groups and users are placed in these groups. This method allows an unlimited number of groups to be created without component code changes, but requires that each component poll an external system for the permissions associated with a user.

Student authentication in the Test Delivery component is a specialized mechanism that requires to verify that each student is who the system thinks he or she is.

10.3. Item-level security

Because item exposure is critical, item security must consider the following:

- How are items stored and who has access to the data? It is critical that only authorized users with the correct level of privileges are able to operate on the items.
- How are items transmitted to other systems and how are those systems authenticated and authorized?
- Summative items must be given the highest level of security.

10.4. Student data security

Student data security must comply with:

- FERPA and COPPA.
- State laws regarding data breaches. For example, California has a law called the "California Data Breach Notification Law" (SB 24 – Sep 2012), which requires companies, institutions, and government agencies to provide key details in data breach notification letters and to notify the state attorney general about the data breach.

10.5. Data at rest

Any confidential/sensitive data that is at rest(e.g. password field in the database, export file, SSN in and XML file, etc) needs to be encrypted so that it can not be mistakenly viewed by the wrong party.

11. Technical Architecture Definition

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

11. Technical Architecture Definition

11.1. Server Hardware and Software Requirements

Software requirements: Platform

There are many programming languages and platforms used today. The two most popular platforms are Oracle Java Platform (JVM) and Microsoft's .NET Platform (CLR). While C and C++ are still frequently used, the JVM and CLR platforms support for multiple programming languages and hosting multiple target platforms make them the logical choices for this application. They allow programming an application in multiple languages while running on the same platform, which enables developers to use the most efficient language when solving specific issues. For example, Scala [http://www.scala-lang.org] could be used for parts of a component that require concurrent processing. However, Clojure [http://clojure.org] is good for concurrency, and also works well for functional style programming which supports more mathematical features.

.NET is often considered to only be available for Microsoft platforms, but the open source project Mono [http://www.mono-project.com/Main_Page] allows .NET technologies to run on Linux and OSX operating systems. Commercial versions of Mono are also available for Apple IOS and Google Android. Both platforms are suitable for components of the Smarter Balanced system. Some components can be built in one platform, and others built in the other.

Mono does extend the platforms on which the application may be deployed. However, it does not support all features of the .NET 4.0 platform, may introduce increased support costs, and lags behind .NET updates and new features. Until recently, Mono was supported by Novell. Xamarin, a company founded in May of 2011 by some of the originating Mono developers, now supports the product.

JVM was started by Sun in the mid 1990s and was acquired by Oracle in April of 2009 when they bought Sun. It is supported on most platforms, except IOS, and has been a dominant enterprise platform since the beginning of the millennium. JVM is also well supported in cloud technologies. .NET is still competitive, although it is not as broadly supported.

JVM is the preferred platform for component development. This is not saying that the Java programming language is preferred, but using JVM will allow the use of many languages that interact with each other. Using JVM allows innovation and deployment to non-windows operating systems (OS), enables the output to be deployed to many servers, and continues to be supported for the OSs by Oracle for the foreseeable future.

Software Requirements: Browser

Browser technology advances at a rapid pace. As browsers are updated, they are starting to provide standards support, reducing the incompatibilities between them. Support of older browsers (especially Internet Explorer 6) requires custom code and additional quality assurance testing.

Supporting earlier browser versions may become necessary as component development begins. The Progressive Enhancement [http://en.wikipedia.org/wiki/Progressive_enhancement] development design technique will allow adding features supported by newer browsers without inhibiting older browsers from using the basic features.

Toward that end, the following draft specifications serve as a starting point for the discussion regarding system requirements. Smarter Balanced will consult with member states and districts regarding how to balance the advantages new technology has to offer against the pragmatic budget and logistical issues that face schools and districts. After further analysis has been conducted using the results from the IT Readiness Tool, final recommendations for hardware, browser versions, and operating systems will be made.

User interfaces for all components except for Reporting must support the following browsers at the indicated version and greater:

- Internet Explorer 8+ (Windows)
- Firefox 8+ (Macintosh, Linux & Windows)
- Safari 5+ (Macintosh)
- Chrome 16+ (Macintosh, Linux & Windows)

Reporting may require more varied browser support. Standard reporting features do not require updated browser features, and current reporting solutions handle older browser capabilities. The reporting feature should be tested on the following minimum browser versions:

- Internet Explorer 6+ (Windows)
- Firefox 4+ (Macintosh, Linux & Windows)
- Safari 4+ (Macintosh)
- Chrome (Macintosh, Linux & Windows). Chrome tends to update itself to newer versions and is updated frequently. Testing back versions may be a daunting task. The reporting development team must define a sufficient backward compatibility test plan for this browser.

Hardware Requirements

If the components use a JVM (or .NET / Mono) then specific hardware or operating systems are not required (i.e. needing Solaris running on a Sparc hardware). Any hardware or operation system that runs the JVM is possible. The components application architecture will determine that components hardware requirements but if these components follow the following guidelines they will allow flexible choices.

A components server part(s) should attempt to be as stateless as possible. Where state is necessary, the server should use a distributed cache (e.g. Ehcache [http://ehcache.org/], Memcached [http://memcached.org/], etc.). This will allow for easier horizontal scalability.

When developing parts of the component that can take advantage of concurrency, use the Actor Model [http://en.wikipedia.org/wiki/Actor_model], so that the component can make more effective use of multi-core server hardware.

Hosting environment and Recommendations

The Smarter Balanced assessment system needs to support many different hardware and network topologies. The recommended architectural direction will enable this capability. Because of the requirement for ultimate flexibility, this limits the ability to predetermine the precise hardware and network requirements. Each components application architecture will be better able to determine its physical needs once the requirements are fully fleshed out, but we can give a logical hosting idea that each component can develop towards.

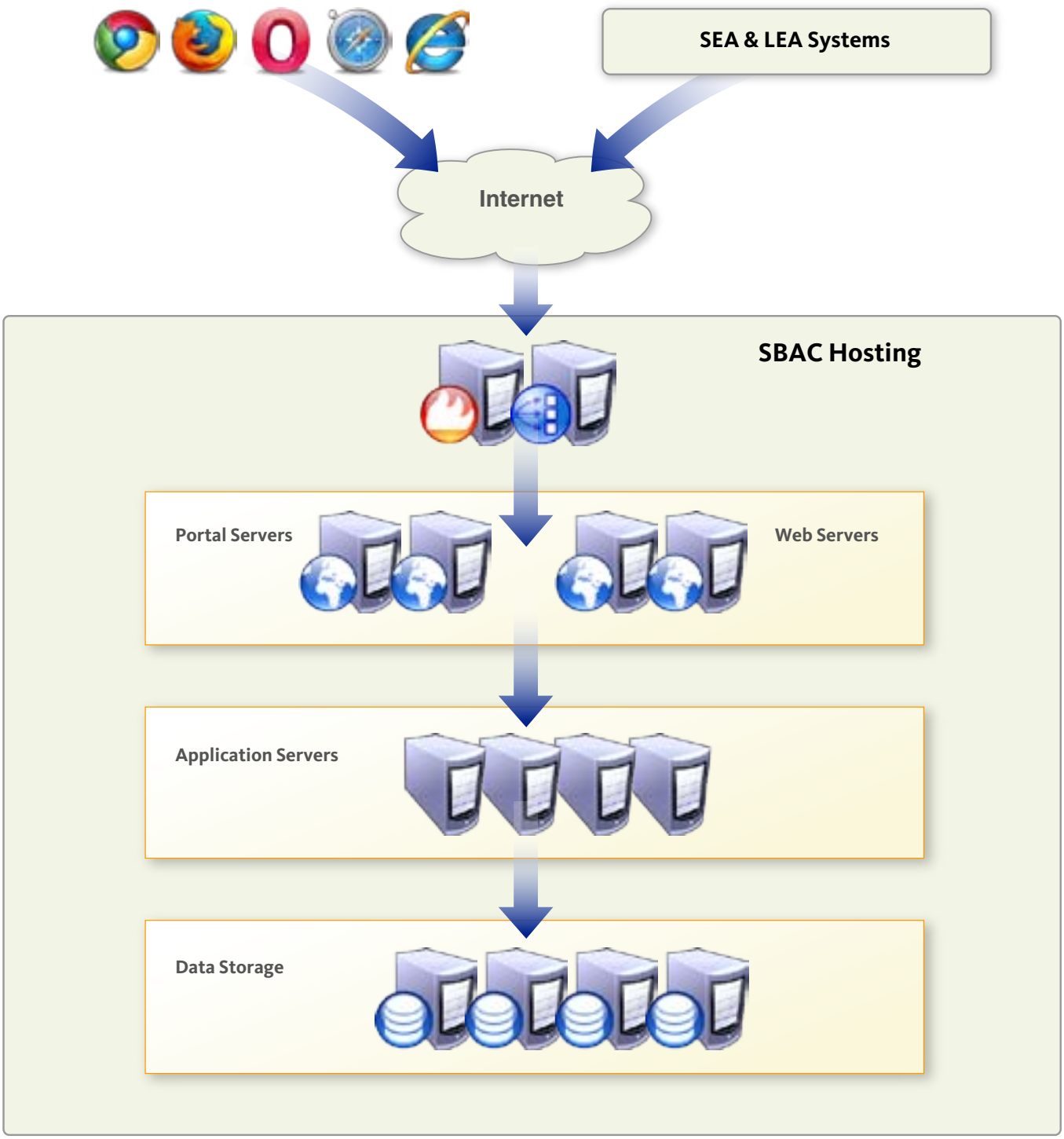

Figure 11.1.1 Logical Hosting Environment

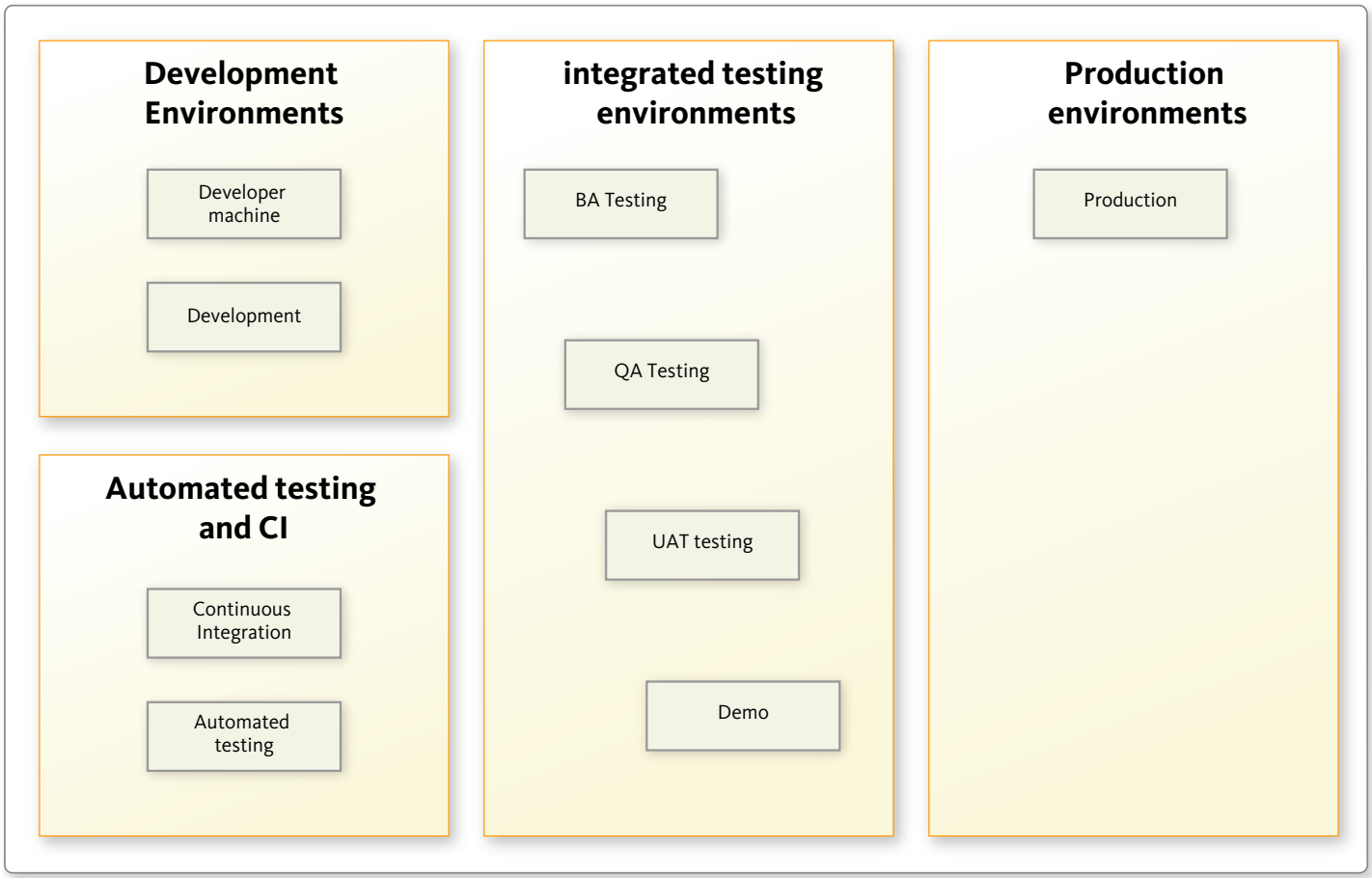

Figure 11.1.2 Development Environments

When a component has this style of hosting application architecture, it will be flexible enough to be deployed to multiple different hosting solutions.

Component Server Matrix

Component	Minimum Component Server Count	Minimum Data Server Count
Item Authoring / Item Bank	2	2
Test Authoring / Test Item & Spec Bank	2	2
Test administration Registration	2	2
Test Delivery	2-N (based on max concurrent usage expectation)	2-N
Scoring	2-N (based on max concurrent usage expectation)	2-N
Data Warehouse	2-N (based on max concurrent usage expectation)	2-N
Reporting	2	0 (Uses Data Warehouse store)
Portal	2	(Depends on application architecture. May be able to share data store of SSO, et al.)
SSO / Permissions / Program Management / User Preferences / Identifier Management	2	2
Monitoring & Alerting	2	(Depends on application architecture)
Digital Library	2-N	2-N

The rationale behind the matrix is that each component server and data server will need to have a minimum of 2 instances, in order to maintain the up-time expected. Meeting the above minimum requirement will mean that the servers will always accessible even if one fails.

Recommendation Virtual Machine (VM) based hosting

System VM [http://en.wikipedia.org/wiki/Virtual_machine] / Virtual Private Server (VPS) [http://en.wikipedia.org/wiki/Virtual_private_server] based hosting will allow the Smarter Balanced assessment system to bypass the usual hardware procurement needs. Estimation of hardware and network requirements before software development is well underway, is usually inaccurate and leads to under or overestimation of the needs. By VM hosting through a hosting provider, the assessment system will be able to procure the required server and data storage in a just-in-time manner. VM providers usually also have a higher speed network backbone to support inter-server communications. They can also support the hardware management and network support. This frees up the assessment system to only need to provide the software support.

The following are some VM hosting providers to consider:

Rackspace [http://www.rackspace.com/cloud]

Amazon [http://aws.amazon.com/ec2]

11.2. Networking Requirements and Diagrams

See also Logical Hosting environment [hosting]

11.3. Database, Data Storage and Archiving Requirements and Approach

By following the recommended architecture, each component can have different storage and archiving requirements. It is expected that the application architecture of those components will need to define those requirements. The following is a list of principles that should be observed:

- Data storage needs to be point-in-time recoverable. The time resolution is dependent on the criticality of the respective data object.
- Student assessment responses must never be lost. If a student has submitted an answer to an item and is presented with another item or test / section completion page, the system must be able to recover all responses including that response.
- Item / test authoring requirements would be based on the Smarter Balanced policy. Smarter Balanced needs to define what is an acceptable loss in case of system failure (i.e. 1 day, 1 hr, 15 min, etc.). It should be noted that the shorter time recovery point, the higher the development and support costs to implement.
- Student responses and other Data Warehouse data needs to be kept for longitudinal use, and Smarter Balanced needs to define the retention lengths for this data.
- Item metadata should be guaranteed delivery from the delivery / warehouse components to the Item Bank (at least once guarantee as opposed to the once and only once guarantee). The Item Bank should be able to accept “re-sent” metadata and be able to gracefully handle redundant data.
- Components should be able to seamlessly recover from single data node failures. When this occurs, components should be able to switch to other data nodes. This means that data storage for a component needs to have a minimum of 2 nodes to support single node failure.
- Smarter Balanced policies need to be explicit about archiving lengths for specific data objects. Some of these policies may be driven by state and federal law.

Master Data Management

[http://en.wikipedia.org/wiki/ Master_data_management]

It is the responsibility of the Program Management [MDM] component to manage and maintain any data and reference data that is needed across components. This is to be considered as the definitive source for that information. Other components are expected to retrieve this data from the Program Management component. They may store their own copies of this data but are

responsible for updating from this data in Program Management when their copy becomes stale. The components should access this data through a REST API [http://en.wikipedia.org/wiki/Representational_state_transfer] provided by the Program Management component.

The Smarter Balanced architecture contains a segregation of where data is modified. The general rule is that data is only modified by the component that owns the data. For example: Item bank owns all Item related data and is the only component that can update this data. Other components consume parts of this data, but never update it. It is recommended that a simple custom solution be used instead of a commercial MDM product since the Smarter Balanced requirements do not demand a sophisticated system with features like "Single version of truth" [http://en.wikipedia.org/wiki/Single_version_of_the_truth] and data governance.

11.4. Systems Management and Monitoring Requirements

All components should use a logging framework that is configurable outside of the component. This will allow components to write log and tracing information in a consistent and configurable way.

Here are the suggested tools:

- JVM - log4j [http://logging.apache.org/log4j],slf4j [http://www.slf4j.org]
- .NET - Log4Net [http://logging.apache.org/log-4net]

For components built on the JVM, the component should use Java Management Extensions (JMX) [http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html]. Applications can expose information about performance, load and other information though a standard interface. Many management solutions support JMX through direct support or through JMX to Simple Network Management Protocol (SNMP) [http://en.wikipedia.org/wiki/ Simple_Network_Management_Protocol] adapters.

Similar to JMX, components on the Windows .NET platform should implement Windows Management Instrumentation (WMI) [http://msdn.microsoft.com/en-us/library/windows/desktop/ aa394582(v=vs.85).aspx]. It performs the same capabilities for Windows components and is build in to the operating system. Windows itself uses this protocol, so all tools capable of monitoring Windows should be able to monitor the components.

Cloud vendors usually offer monitoring capabilities to their solutions. By following JMX / WMI and SNMP standards while implementing components, Smarter Balanced will be able to choose management and monitoring solutions without being tied to a specific vendor.

Possible vendors include, but not limited to:

- Nagios [http://www.nagios.org]
  openNMS [http://www.opennms.org/]
  Hyperic [http://www.hyperic.com]
- CA Technologies
- HP Openview
- IBM Tivoli [http://www-01.ibm.com/software/tivoli/solutions]
- System Management Categories High Availability
- Components that fit in this category need to be highly available and redundant. They are:
- SSO
- Test Delivery
- User Preferences
- Permissions
- Identity Management
- Portal
- Monitoring and Alerting

These components need to expose information as to the status of the component. (e.g. Test Delivery component need to expose the number of connected students). These components also need to be monitored for preventative issues. The machine or VM that they run on must be monitored

for low-memory issues, disk-full issues, processor overloading issues and exceptions. These must cause alerts in the system management software to notify support personnel of possible issues.

Medium Availability

Components that fit in this category need to be available but are not as time critical and do not need to be as redundant.

- All components that are not in the above list

Although these components need to be available, the criticality of their up-time has less impact on the assessment system processes. These components can expose information for the management system to monitor, but alerting could be reduced. The high availability / redundancy needs are reduced, which could in turn reduce cost.

NOTE The management data that each component needs to expose will need to be defined at the application architecture level, as this is the point at which critical capabilities will be fleshed out. It is at this point that the alerting and operational procedures be determined to mitigate the issues.

11.5. Middleware and Integration Software Requirements

This section details the main integration patterns and technology recommendations for messaging, communication and data transfer to data warehouses.

Recommendations relied on the following principles:

1. Favor lightweight integration and frameworks over centralized hub and spoke models or messaging systems. These are easier to test, integrate and have very low requirements on hardware and software.

2. Resist adding business logic in centralized service buses, since they are harder to test and troubleshoot.

3. Favor lightweight RESTful services over integrating at the database level, which stifles emergent changes to the database.

Publish and Subscribe using ATOM feeds

ATOM [http://en.wikipedia.org/wiki/Atom_(standard)] feeds are a lightweight XML-based syndication format and uses HTTP(S) for transport that is secured and reliable. This is best used in scenarios where an application wishes to share its data with other application and instead of using queuing systems, it publishes an ATOM feed. This recent book, Rest in Practice [http:// restinpractice.com/book.html] has some great examples of using ATOM for this purpose.

This pattern should be used anytime data is pushed to a data warehouse and other areas where data needs to be published. Interoperability matrix (section 8.2) enumerates the areas where this is needed. The image below shows this pattern in work using the scenario of the test delivery system and other components publishing data to the data warehouse.

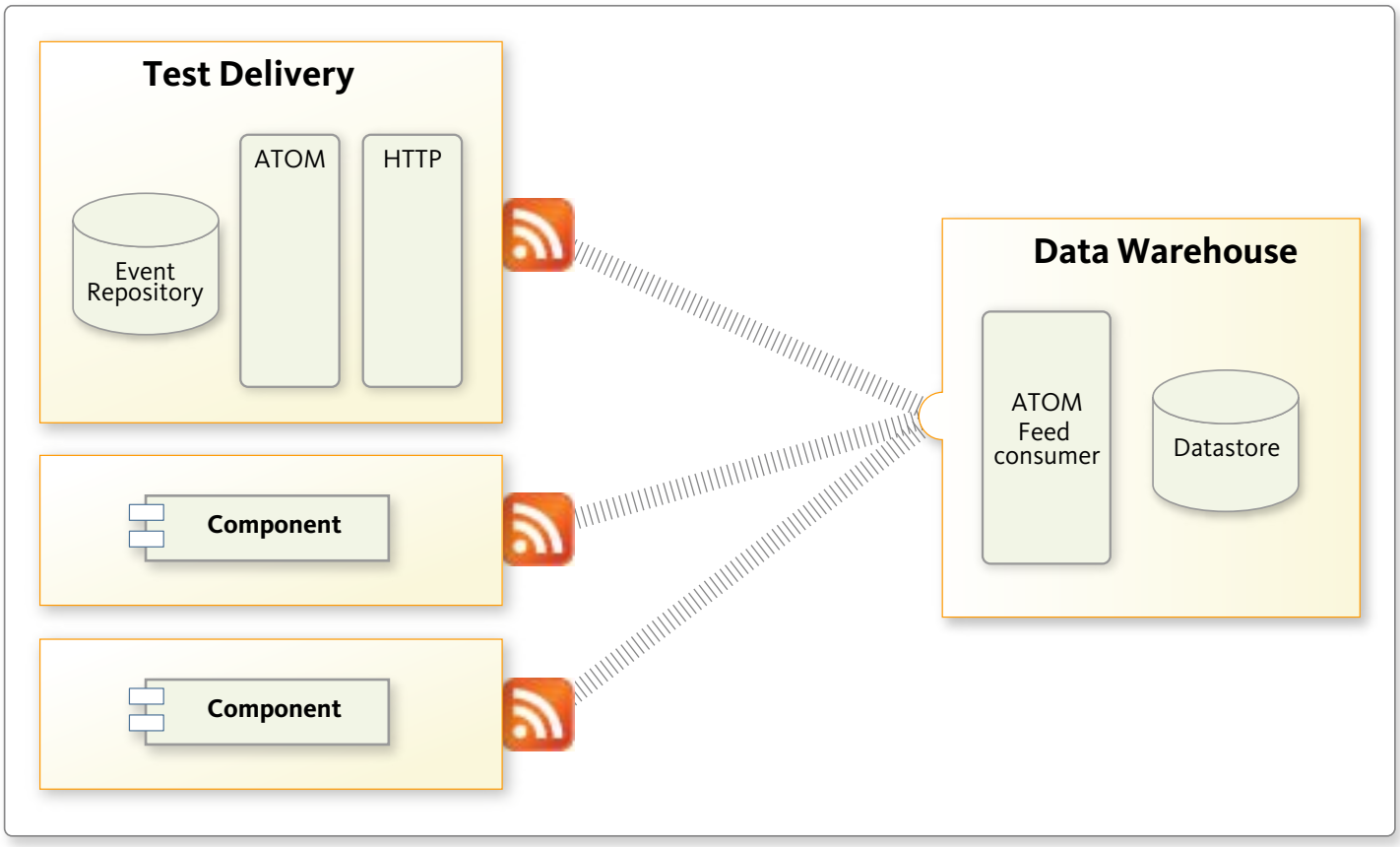

Figure 11.5.1 Publish and Subscribe using ATOM feeds

Point-to-Point communication using RESTful services

REST [http://en.wikipedia.org/wiki/Representational_state_transfer] is recommended for use where point-to-point communication is needed between components either in a fire-and-forget mode or in a request-response mode. REST uses HTTP(S) for transport and message formats can use XML, JSON and the standard HTTP methods. This book [http://restinpractice.com/book.html] is an excellent resource for a deep dive on the subject.

The image below shows this pattern in work using the scenario of the test authoring system querying the Test Item Bank component for available items matching a specification:

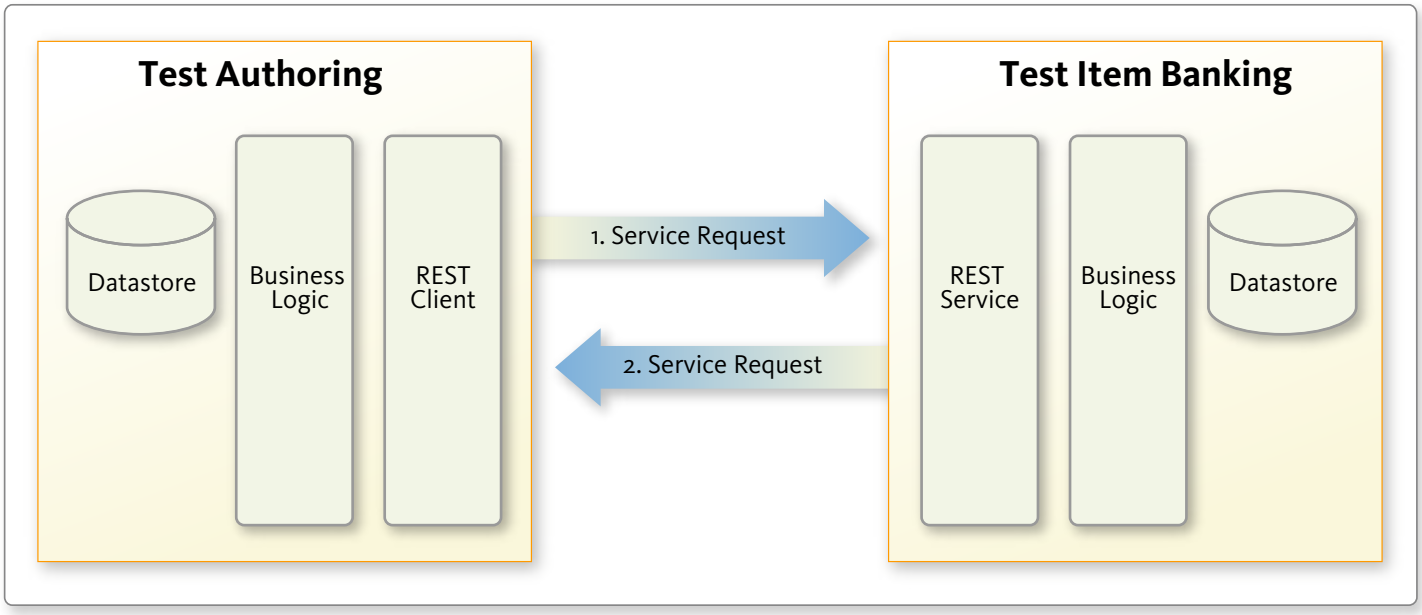

Figure 11.5.2 Point-to-point Communication Using REST

APIP messages over SSH

APIP [http://www.imsglobal.org/apip/] is recommended for use as a message format for items and related information. It is recommended to implement a SSH-based file transfer for the transport, given that these message packets will be large (excess of 500Mb).

Image below shows this pattern in work using the scenario of the item bank transferring items ready for test to the test item bank.

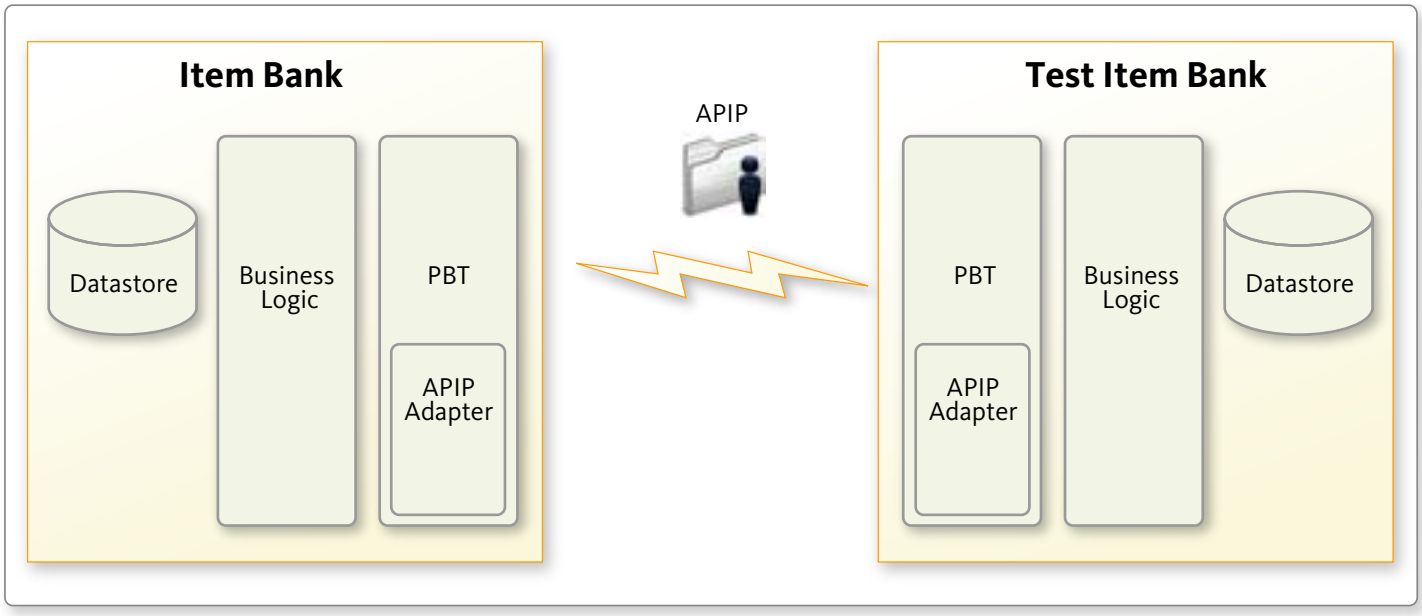

Figure 11.5.3 APIP Messages over SSH

SIF integration using ZIS

SIF [http://www.sifinfo.org/us/sif-specification.asp] is recommended for communication with student information systems (SIS). SIS specifies message format (XML-based) and provides transport using ZIS. It is recommended that the Smarter Balanced assessment system uses the various SEA / LEA ZIS systems for this. When two components need to exchange SIF information, it is recommended to use one of the patterns mentioned here based on the type of communication.

The image below shows this pattern in working using the scenario of the test administration / registration and the test delivery systems exchanging student information from the SIS using SIF and ZIS.

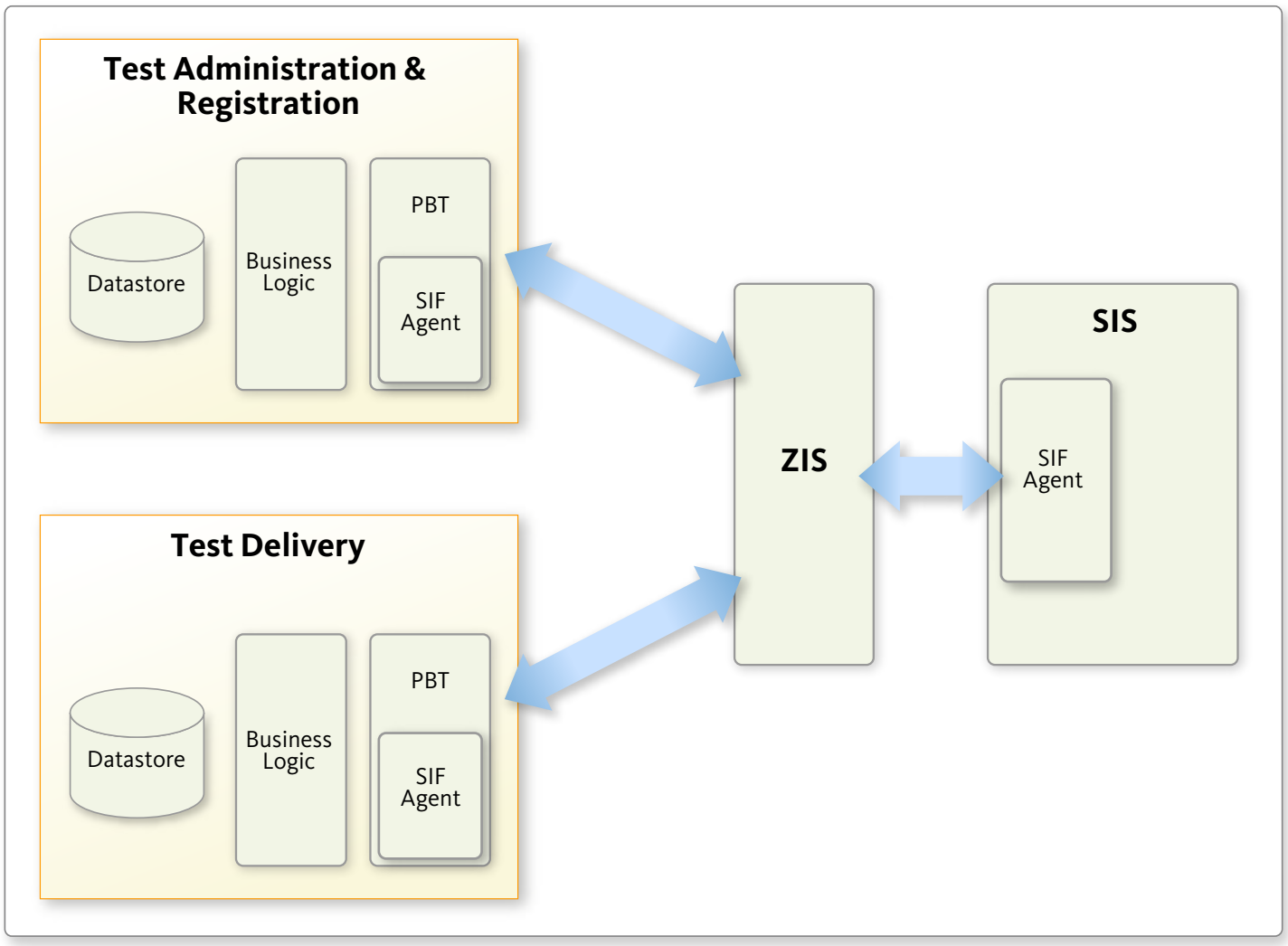

Figure 11.5.4 SIF Integration using ZIS

Resources

- Rest in Practice [http://restinpractice.com/book.html]
- Agile Analytics [http://theagilist.com/book/]
- ATOM [http://en.wikipedia.org/wiki/Atom_(standard)]
- Rome, a framework for publishing ATOM [http://java.net/projects/rome/]
- REST [http://en.wikipedia.org/wiki/Representational_state_transfer]

11.6. Security Requirements and Approach for Applications, Data, and End-user Access

Also see Security [security] End-user access should be controlled using a Single Sign On solution. This solution should support oAuth [http://oauth.net]. Users enter all components through the Portal component. If a component determines that the user is not authenticated, the component should be redirect to the SSO provider.

- All web-based traffic should use Secure Sockets Layer (SSL) [http://en.wikipedia.org/wiki/SSL]
- Any inter-component network communication should use SSL. This should use non standard ports and be firewalled to only accept connections from defined static IP addresses.
- End-user access should be controlled using an Single Sign On solution. Users enter all components through the Portal component. If a component determines that the user is not authenticated, the component should be redirect to the SSO provider. That provider will authenticate and embed a token in a cookie that the target components uses to determine if the user is authenticated.
- Any data that is exported to a file needs to be encrypted. It is recommended to use Pretty Good Privacy (PGP) [http://en.wikipedia.org/wiki/Pretty_Good_Privacy] to do so.

12. Build vs Buy

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

measured progress

12. Build vs Buy

To determine whether to build or buy, the required features of a component must be defined so a weighted matrix can be created to determine the real cost of a buy solution. The matrix also enables comparisons among possible buy solutions, and provides a comparison of the estimated build costs.

This is just a sample of what a Digital library matrix may look like.

Component	Maximum Weight	Vendor A	Notes	Vendor B	Notes	Build	Notes
General							
Trade-off Costs							
Delivery Confidence Level							
Performance							
SSO Integration							
Open Source/ License Type							
Reliability							
Required Skills							
Flexibility							
Sustainability							
Scalability							
Performance							
Security							
Digital Library Specific							
LMS Features							
Interoperability with HR Systems							
Content Management System							
Asset Support							
Collaborative Features							
Authoring Capabilities							
Conversion Capabilities							
Search Capabilities - Semantic and Intelligent							
Metadata Standard							
User Interface							
IP and Licensing Management							
Interoperability for Content - SCORM, IMS Common Cartridge and SIF LearningResource							
Learning Standard Item Support							
Presentation of Information							
Total Points							
Purchase Cost							
Enhancement Cost							
Estimated Difficulty to Enhance							

13. Application Development Model

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

13. Application Development Model

13.1. Objective

The Application Development Model outlined below supports Smarter Balanced system development by multiple vendors. Clear leadership and vendor constraints are required to achieve the envisioned timetable, allowing vendor teams to work as independently as practical whilst reducing integration risk.

13.2. Principles

Early and frequent integration

Rationale

- By integrating early inherent ambiguities are exposed and resolved early in the process de-risking the overall system delivery. By frequently integrating, issues in implementation are discovered early in the process.

Implications

- Automated integration tests to minimize the integration testing costs

Component Versioning

Rationale

- Defining a versioning scheme that is consistent across components, will help with communication of dependencies between components

Implications

- Maintaining a component version dependency matrix

Vendor Collaboration

Rationale

- Support early and frequent integration by enabling clear and frequent communication channels between vendor teams.

Implications

- It will be necessary for vendors to communicate and coordinate feature development which may be unnatural to those vendors culture. Smarter Balanced will need to facilitate the communication and resolve conflicts between vendors.

13.3. Development Practices

- All components use short synchronized iterations (2-3 weeks). All teams iterations should start and end on the same days.
- Common version control. (Recommended Git [http://git-scm.com/])
- Test Driven Development [http://en.wikipedia.org/wiki/Test-driven_development] including the use of Mock Objects [http://www.mockobjects.com]
- Behavior Driven Development [http://en.wikipedia.org/wiki/Behavior_Driven_Development] (Recommended Cucumber Framework [http://cukes.info])
- Continuous Integration (CI) [http://martinfowler.com/articles/continuousIntegration.html] a common CI infrastructure should be used for all vendors to help facilitate cross component integration testing
- Continuous Delivery [http://en.wikipedia.org/wiki/Continuous_Delivery] - All components should be “one click deployable” to its target environment.
- YAGNI [http://en.wikipedia.org/wiki/You_ain't_gonna_need_it] Ya ain't gonna need it - Do not add functionality until it is needed.

13.4. Evolutionary Database Design

Database development should follow the same practices that software based development does. It is necessary that certain practices be followed:

1. Configuration Management: All database artifacts must be in the revision control system to allow the component code and the database it interacts with to be in the same revision control system. This enables the component and the database to be developed and deployed at the same time.

2. Database Sandbox: Create an automated process to create the database sandbox so the component database can be created with all the base, or seed, data that the component needs. This enables anyone who needs a database sandbox to create one without manual intervention.

3. Database Behavior: Like code, database objects have behavior, and that behavior must be enforced by the application layer so the database provides the same behavior. Here is an article about Behavior Driven Database Design [http://www.methodsandtools.com/archive/archive.php?id=78].

4. Tracking changes to the database design and schema: Every change to the database must be coded as migration scripts and checked into the Configuration Management System to allow for automated deployment of database changes in different environments.

5. Continuous Integration: Database changes must be part of the Continuous Integration cycle, as any changes made to the database must be tested and verified with the component. Without this verification, neither the code nor the database changes should be published. After the code is verified with the database, the changes may be published as artifacts to be deployed in other environments. Here is an e-book written about this topic Continuous Integration with Databases [http://www.informit.com/store/product.aspx?isbn=032150206X].

6. Database Design: Database Designers and developers should be working together during component development for cohesive design and development.

14. Scenario Mapping to Component Systems

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

14. Scenario Mapping to Component Systems

This table associates various scenarios with the components they affect. More scenarios will be added as they are discovered or developed.

The scenarios illustrate how users achieve their goals with system components, and highlight the integration points between components.

Category	User Scenario	Components
Item creation & development	Item creation lifecycle	Item Authoring, Item Bank, Test Bank
Scoring	Interim - constructed-response item	Test Delivery, Scoring, AI Scoring Engine
Scoring	Adaptive interim - selected-response item	Test Delivery, Adaptive Engine, Scoring, AI Scoring Engine
Scoring	Teacher creates an interim test human scoring, constructive	Test Bank, Test Spec Bank, Test Authoring, Test Delivery, Scoring
Scoring	Summative, computer-based scoring	Test Delivery, Adaptive Engine, Scoring, AI Scoring Engine
Scoring	Primary, paper-based interim test	Test Bank, Test Spec Bank, Test Authoring, Test Delivery, Scoring
Scoring	Monitoring - performance of rater, summative	Scoring, AI Scoring Engine
Scoring	Mixed paper based & computer-based interim test	Test Bank, Test Spec Bank, Test Authoring, Test Delivery, Scoring, AI Scoring Engine
Scoring	Paper-based summative test	Test Registration, Test Administration, Test Delivery, Scoring
Scoring	Monitor vendor-scored summative test	Scoring, AI Scoring Engine
Scoring	Performance task - interim assessment	Test Delivery, Scoring, AI Scoring Engine
Scoring	Monitoring- crisis papers	Scoring, Monitoring & Alerting
Test creation	Test creation - Smarter Balanced-owned system	Item Bank, Test Bank, Test Spec Bank, Test Authoring, Calibration Engine
Test creation	Test creation - state or LEA or school	Test Bank, Test Spec Bank, Test Authoring
Test creation	Define blueprint	Test Spec Bank
Test creation	Test creation - adaptive test, summative	Test Authoring, Test Bank, Test Spec Bank
Test delivery	Ready the test	Test Registration, Test Administration, Test Delivery
Test delivery	Test delivery	Test Delivery, Adaptive Engine, Scoring, AI Scoring Engine

15. Component Priorities

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

15. Component Priorities

Milestone (end dates)	Component	Procurement Schedule Mapping
August 2012 - Item Pilot	Item Bank	7
	Test Item Bank	14
January 2013 - Pilot Test	Test Package	14, 16
	Adaptive Engine	5, 17
	Scoring Engine (AI & non-AI)	5
	Test Delivery	11, 17, 18, 20
	Test Specification Bank	
	Test Authoring	
	Program Management	
	SSO Permissions	
	User Preferences	
	ID Management	
	Metadata	
February 2014 - Field Test	**Test Registration**	
	Test Administration	16, 19, 20
August 2014 - Project End	**Digital Library**	
	Reporting	15
	Item Authoring	7
	Data Warehouse	
	Portal	
	Monitoring	
	Alerting	
	Item Specifications Bank	

16. Glossary

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

16. Glossary

This section contains two glossaries of terms for the Smarter Balanced Systems Architecture initiative. The first, Inception Glossary, defines some of the terms and items used in the inception period. The second, Architecture Glossary, captures the terms used for the architecture documentation that follows the inception period. Both are here to provide a source of reference.

16.1. Inception Glossary

Below, for reference, are some of the acronyms and terms used in the inception period. In addition, the Architecture Glossary may also be used as a secondary reference.

AI
Artificial Intelligence The ability for a computer and software to score assessments and provide direction for additional assessment items or instructional content.

Application
Computer software designed to help the user perform specific tasks.

ARB
Architecture Review Board Group responsible for the ongoing governance and assurance that the architecture is periodically reviewed and updated; that the standards, practices, patterns and policies are followed; and that solution approaches that further the goals and objectives of the Smarter Balanced are reviewed.

Asset
Digital text, multimedia or images.

ATP
Association of Test Publishers A non-profit organization representing providers of tests and assessment tools and/or services.

APIP
Accessible Portable Item Profile A technical standard that focuses on accessibility in assessment items.

AYP
Adequate Yearly Progress A term used in the No Child Left Behind Act (NCLB) to mean the minimum level of improvement that school districts and schools must reach every year toward achieving state academic standards. AYP is defined independently by each state.

Blueprint
The design for a test. The test blueprint indicates the number of test questions or points related to each competency on the test and the relative emphasis placed on each competency.

Charter
A statement of the scope, objectives and participants in a project.

Cog Lab
Cognitive Lab A method of studying the mental processes one uses when completing a task such as solving a mathematics problem or interpreting a passage of text. An environment where new or modified items are tried to judge their effectiveness.

DLM
Data Lifecycle Management. Managing the flow of a system's data throughout its entire life cycle.

DNU
Do Not Use Describes a state of an item.

DOK
Depth of Knowledge. The complexity of knowledge required by standards and assessments. Four levels: 1-recall, 2-skill/concept, 3-strategic thinking, 4-extended thinking.

Epic
A large feature, or a grouping of smaller features or stories. Also see Story.

FERPA
Family Educational Rights and Privacy Act. Federal law protecting the privacy of student data.

Field Test
Test made up of test items intended to develop and calibrate new assessments.

IEP
Individual Education Plan. Mandated by IDEA, designed to help teachers and students meet the unique educational needs of a student with disabilities, with the intention of enabling that student to achieve improved educational results.

Item Pool
Collection of items or test questions. LEA
Local Education Agency An educational unit within the state. For example, a school district, a Charter School, or a special needs school.

Monitoring
The process of supervising the overall administration of an assessment, including scorers.

MSL
Master Story List. A list of requirements that evolves over time, usually found towards the end of the development process.

NFR
Non-Functional Requirement Criteria that define how a system is supposed to be whereas functional requirements define what a system should do. These can include constraints, attributes or processes.

PARCC
Partnership for the Assessment of Readiness for College and Careers. One of two consortiums that received Race to the Top Assessment federal monies to develop a comprehensive assessment system.

Performance Task
A form of testing that requires students to perform one or more tasks.

Pilot Test
A trial series of new or modified items given to a select group of students.

Platform
The composite of a computer's elements, including architecture, operating system, programming languages and related user interface.

PLP
Personalized Learning Plan Learning goals and objectives based on each individual learner. It may include academic, career, and personal interests.

PNP
Personal Needs Profile A Profile to define individual student needs.

QTI IMS
Question and Test Interoperability specification A standard that defines interoperability for assessment items.

Requirement
An expression of certain characteristics or behavior that software should have. Also see Story.

Retired Item
An item that will no longer be used for an assessment.

Service-Oriented
A set of principles and methodologies for designing and developing software in the form of interoperable services.

SIIA
Software and Information Industry Association A non-profit organization for software and digital content industries.

SIF
Schools Interoperability Framework A non-profit organization that produces open technical standards for interoperability in the education ecosystem, including student information systems, assessments, and learning resources.

SIS
Student Information System A software system that houses and manages data pertaining to students.

SLA
Service-Level Agreement Levels of service defined within a contract.

SLDS
Statewide Longitudinal Data System. Large focus started with the grant program administered by the Institute of Education Sciences (IES) and NCES, designed to aid state education agencies develop and implement longitudinal data systems.

Specifications
As in, item specifications. Definition of the required elements of an item at a low level of granularity.

SRC
Score Reporting Category Category used to measure student understanding in specific content and learning standards in assessments.

Story
A small piece of a requirement that accomplishes a single identified goal in software development.

16.2. Architecture Glossary

The following table defines ambiguous terms that directly reference the architecture document and are commonly found in both the educational and technology fields.

Application Architecture
The design of the internal structure of an application.

Application Development
The development of a software product.

Architecture
The practical art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals, to formally model those systems.

API
Application programming interface, a source code based specification intended as an interface for software components to use to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

ASP
Active Server Pages, a web-scripting interface by Microsoft.

Bandwidth
A rate of data transfer, bit rate or throughput, measured in bits per second (bps)

Binary Transport
A transport implementation well suited to distributed applications.

Cardinality
In database design, the defined cardinality explains how each table links to one another.

Component
One of the units that make up a system.

Concurrency
A property of systems in which several computations are executing simultaneously, and are potentially interacting with each other

Data Warehouse
A database used for reporting and analysis.

Database
An organized collection of data for one or more purposes, usually in digital form. The data are typically organized to model relevant aspects of reality. Also see Relational Database [relational].

Deployment
The process of making a software system available for use.

Domain
A set of common requirements, terminology, and functionality for any software constructed to solve a problem.

Hosting
A facility where software and data are kept.

Identifier
A unique name given to a specific object or a specific class of objects.

Interface
A tool and concept that refers to a point of interaction between components, and is applicable at the level of both the hardware and the software elements.

IP
Internet Protocol, or Intellectual Property.

Item
A composite object that it made up of many item parts and metadata about that item.

JSON
JavaScript Object Notation, a lightweight, text-based, open standard designed for human-readable data interchange.

JSP
Java Server Pages, technology that helps software developers serve dynamically generated web pages based on HTML, XML, or other document types.

LGPL
Lesser General Public License, a free software license published by the Free Software Foundation (FSF).

NoSQL
A broad class of database management systems that significantly differ from the classic model

Tenant
In architecture design, an instance of the software that runs on a server, serving a single client organization. Multitenancy is an instance of the software that runs on a server, serving multiple client organizations (tenants).

XML
Extensible Markup Language, a set of rules for encoding documents in machine-readable form.

17. Release Notes

Assessment System Architecture and Technology Recommendations for the Smarter Balanced Assessment Consortium

Prepared in accordance with RFP SBAC 03 by Measured Progress.

17. Release Notes

The following is a history of the Smarter Balanced System Architecture and Technology Report since its initial release in January of 2012.

Version 2.0.1 – Released on March 21, 2012

This update includes a clarification on system requirements and a few minor fixes.

NEW Added CAT Simulator to High-Level Component Diagram. [107]

CHANGE Replaced instances of SBAC with Smarter Balanced. [116]

CHANGE Add clarification to Browser Requirements in 11.1. [118]

FIX Title correction to Figure 8.1.3. [108]

FIX Fixed split bullet in Section 13.3. [109]

FIX Correction to Diagram 4.1. [110]