RECAP PROJECT
HIGH LEVEL ARCHITECTURE

1.0

Authors
LAKSHMI NARASIMHAN
&
VIJAYKUMAR GUNDAVARAPU
HTC Global Services, Inc.
+1 248 786 2500
lakshmi.narasimhan@htcinc.com
vijaykumar.gundavarapu@htcinc.com

Sponsor
RECAP
New York Public Library

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Notice of Proprietary Information

All information contained in or disclosed by this document is confidential and proprietary to ReCAP.
By accepting this material the recipient agrees that this material and the information contained therein will be held in confidence and will not be reproduced in whole or in part without express written permission.

Distribution: Restricted

Disposal: Shred

File Name: ReCAP High Level Architecture

Copyright © 2013 ReCAP

Revision History

Version	Date	Author	Description	Status
Draft	05/16/2013	Lakshmi Narasimhan & Vijaykumar Gundavarapu	Initial Version – Draft	In Progress
1.0	5/30/2013	Lakshmi Narasimhan & Vijaykumar Gundavarapu	Review comments incorporated	Complete

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Table of Contents

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

1. Introduction

1.1 Purpose

Architecture represents the significant design decisions that shape a system, where significance is measured by cost of change. This document proposes high-level candidate architecture for the ReCAP middleware system. The architecture is described using a number of different views to depict architecturally-significant aspects of the system. It is intended to capture and convey the significant architectural decisions, which have been made on the system.

This version of the document presents high-level aspects of the system and the candidate architecture. As the project evolves through the development life cycle, this document will be updated to reflect the architectural decisions/design for the following (but not limited to):

- Finalized Data View
- Finalized Implementation View
- Finalized Deployment View

1.2 Document Overview

The architecture of the application is represented using the recommendations of the Rational Unified Process guidelines. This document also highlights the development environment, quality requirements and prototyping details.

The UML (Unified Modeling Language) specification of the new ReCAP middleware system has been divided into six views (Rational's 4+1 model):

- **Use Case View** –illustrates and validates the architecture by presenting selected architecturally significant use cases.
- **Logical View** – illustrates the object model of the design. It presents an analysis model, which captures the analysis of the use cases and a design model. This view also describes the logical structure of the system and presents key structural and behavioral elements.
- **Process View** – illustrates the assignment of components to the operating system processes and threads.
- **Implementation View** - describes the physical organization of the software and its components in the production environment.
- **Deployment View** –illustrates the mapping of the software to the hardware and its distribution aspects.

1.3 Audience

The primary audience for this document is the Development team and QA team. The development team will use it to help aid the detailed design during the development phase and ultimately to develop the system. The QA team will use it to ensure testability and also to ensure that proper test cases are written.

Each view as presented in the "Document Overview" section primarily caters to different audience.

- Use-case view – All
- Logical view – Development team
- Process view – Software Integrators (part of the development team), QA team to understand performance and scalability bottlenecks for testing purposes
- Implementation view – Development and Deployment teams(part of development team)

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

- Deployment view – Deployment and Production Support teams

1.4 Definition for Architecture

The Unified Software Development Process [8] defines "Software Architecture" as

"We can think of the architecture of a system as the common vision that all the workers (i.e., developers and other stakeholders) must agree on or at least accept. The architecture gives us a clear perspective of the whole system, which is necessary to control its development. "

This document uses the architecture definitions presented by Software Architecture in Practice and the UML Modeling Language Guide: Software Architecture in Practice [7] defines "Software Architecture" as:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."

UML Modeling language user guide [6] defines "Software Architecture" as:

"An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition. "

1.5 Scope

This document provides the candidate architecture for the ReCAP middleware system, which includes details for each layer (Presentation, Enterprise Services, Data Services, and Data) and the interfaces to ILS, OPAC and GFA LAS systems.

2. Project Goals

This section provides the business needs, project goals and architectural issues with the current system and explains how the new ReCAP middleware system architecture aims to address these issues. This section provides a business perspective to the architecture and establishes architectural goals, assumptions and constraints.

2.1 Project scope and objectives

"*The scope of the project is to expand the vision of the ReCAP facility from a shared storage facility to a shared collection with enhanced access to the patrons of each of the participating libraries by implementing an integrated ReCAP middleware system utilizing established industry architectures. The functional requirements for ReCAP middleware will encompass all of the existing functionality, plus changes and enhancements to improve user experience and collection management*".

The ReCAP project has the following main objectives:

1. Improve visibility of ReCAP shared collection items from any participating institution in existing OPAC systems
2. Display of real-time status of items in ReCAP, including availability for request, restrictions

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

and available pick-up locations
3. Improve services that can be embedded into the online catalog or discovery services of the participating institutions to capture and validate requests made by patrons or by library staff for ReCAP materials
4. Provide real-time tracking for ReCAP materials requested by patrons from the time that they leave the ReCAP until they are returned for refilling
5. Provide tools to support the management of the ReCAP collection, such as collaborative collection development, automated processing of duplicates, or designation of preservation retention.

2.2 Architectural Challenges with current system

1. Shared Collection Visibility – Items placed in shared collection by other partners is not available in the OPAC systems, limiting access of such items to patrons
2. Real-time Availability – ReCAP item status is unavailable in OPAC or ILS
3. Real-time Request Processing – Request processing is batch only with minimal validations and error reporting
4. Real-time Status Reporting – Overall status of items between ReCAP facility and delivery locations are partially captured and distributed across disparate systems
5. Collection Management – No centralized collection management is in place

2.3 How the new Architecture addresses the challenges

1. Shared Collection Visibility – ReCAP middleware consolidates and normalizes ReCAP item and bib records from all three partners and provides nightly feeds to all partner OPAC systems. ReCAP search service provides ability to perform federated search on shared collection from OPAC.
2. Real-time Availability – ReCAP middleware database maintains real-time status of all ReCAP items. Item availability is provided through ReCAP middleware API.
3. Real-time Request Processing – ReCAP middleware maintains validation rules and item status. Request submitted through OPAC forms are validated real-time, processed and recorded in ReCAP middleware database. Users receive confirmation or validation error messages in real-time enabling them to resubmit a valid request.
4. Real-time Status Reporting – ReCAP middleware consolidates a complete view of item status across GFA and ILS systems into middleware database. Consolidated status can be leveraged for tracking and analytics.
5. Collection Management – ReCAP middleware implements centralized automated collection classification algorithm. Middleware provides user interfaces for manual workflow steps such as withdrawal of preservation copies.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

3. Architectural Overview

3.1 Candidate Architecture

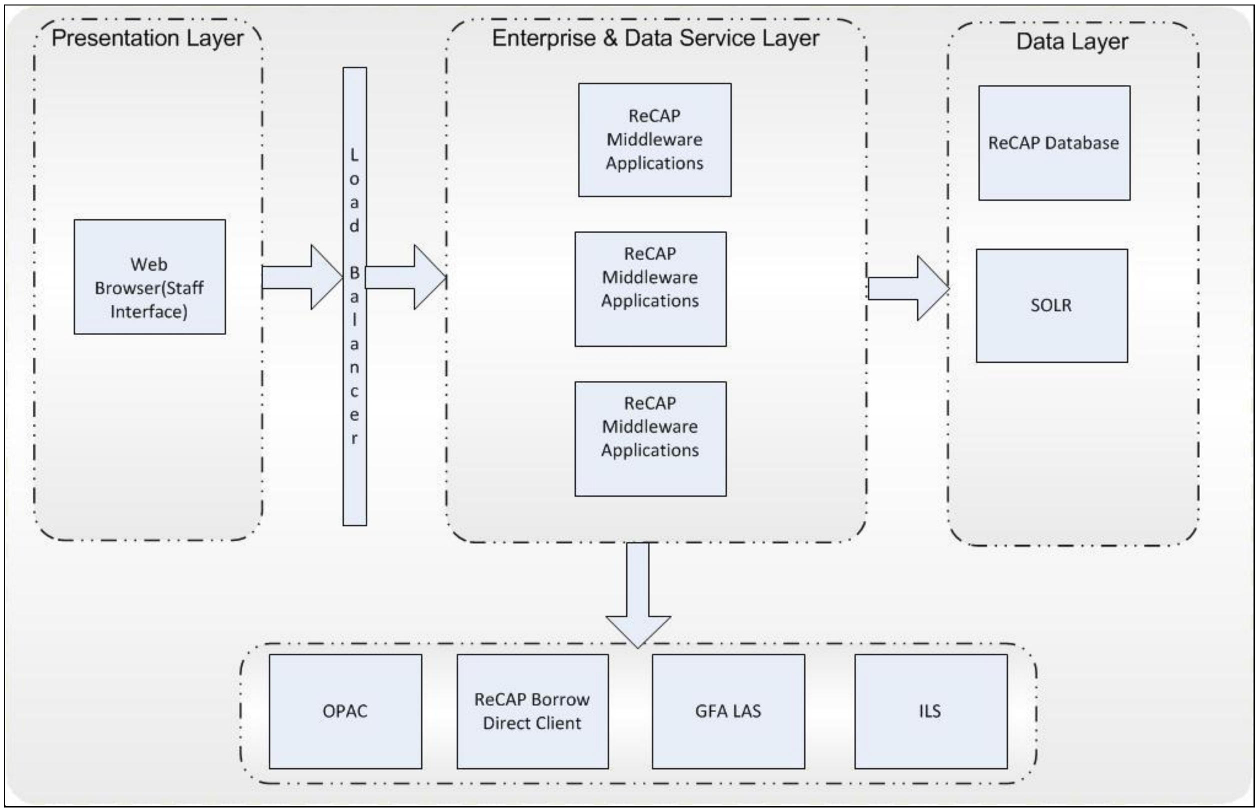

Presentation Layer
Web Browser(Staff Interface)
Load Balancer
Enterprise & Data Service Layer
ReCAP Middleware Applications
ReCAP Middleware Applications
ReCAP Middleware Applications
Data Layer
ReCAP Database
SOLR
OPAC
ReCAP Borrow Direct Client
GFA LAS
ILS

Figure 1- Candidate Architecture

3.1.1 Layers

The architecture includes four distinct layers:

- Presentation Layer
- Enterprise Services Layer
- Data Services Layer
- Data Layer

3.1.1.1 Presentation Layer

The presentation layer deals with user interface aspects of the system. Presentation layer will leverage Kuali Rapid Application Development (KRAD), a framework providing reusable solutions and templates. KRAD is built upon industry standard jQuery libraries providing out-of-box UI components, validations and accessibility to RICE middleware.

3.1.1.2 Enterprise Services Layer

The enterprise services layer encapsulates specific business rules, which are made available to the presentation layer. The presentation layer requests enterprise services, which are then fulfilled by this

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

layer. The architecture envisages providing a seamless enterprise service layer communicating with internal data stores and 3rd party services. The data access layer supports the enterprise service layer by serving the data required.

Enterprise Services is based on a Service Oriented Architecture and leverages Kuali Service Bus (KSB), for service integration. Services will be designed as java spring-based services and will be published on the service bus as remote asynchronous calls. Transaction services will be published as SOAP services and lookup services will be published as RESTFul Services.

Features such as Service Discovery, Quality of Service, Security, Monitoring and Messaging are available as out-of-box features in Kuali Service Bus and can be leveraged during implementation as required.

3.1.1.3 Data Services Layer

The data services layer provides fundamental services to fulfill the business needs (fulfilled through enterprise services) such as Search, Request Item, etc. The data services layer serves data required by enterprise services. Data services support both relational database and Solr.

Services implementing data access to relational database will leverage Java Persistence Architecture (JPA), providing separation of object persistence and data access logic from a particular persistence mechanism (relational database) in data layer. This approach provides the flexibility to change the applications persistence mechanism without the need to re-engineer application logic that interacts with the data layer. Persistence classes are developed following the object-oriented idiom including association, inheritance, polymorphism, composition, and collections. This framework provides the flexibility to express queries in its own portable SQL extension, as well as in native SQL, or with object-oriented criteria.

Services implementing data access to Solr / Lucene search will wrap the Solr RESTFul API's to provide features such as search, filter, sort and navigation.

3.1.1.4 Data Layer

The data layer serves as the data store for all persistent information in the system including the relational database and search engine indexes.

RDBMS data layer will comprise of MySQL cluster. RDBMS data layer will be accessed only from the data access layer via Data Access Objects (DAOs). RDBMS cluster architecture allows a single physical database to be accessed by concurrent instances running across several different CPUs. The proposed data layer will be composed of a group of independent servers or nodes that operate as a single system. These nodes have a single view of the distributed cache memory for the entire database system providing applications access to more horsepower when needed while allowing computing resources to be used for other applications when database resources are not as heavily required. In the event of a sudden increase in traffic, proposed system can distribute the load over many nodes, a feature referred to as load balancing. In addition to this, proposed system can protect against failures caused by unexpected hardware, operating system or server crashes, as well as processing loss caused by planned maintenance. When a node failure occurs, connection attempts can fail over to other nodes in the cluster, which assumes the work of the failed node. When connection failover occurs and a service connection is redirected to another node, users can continue to access the service, unaware that it is now provided from a different node.

A single Solr instance can support more than one index using Solr cores (single index per core). A single large index can be a performance overhead. SolrCloud distributes a single index on different machines,

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

commonly referred as shards. All shards of the same index making one large index are referred as collection. While collection supports index scaling, it does not provide redundancy. Replication of shards provides redundancy and fault tolerance.

Zookeeper maintains the SolrCloud, by distributing the index across shards and federating the search through the collection. SolrCloud uses leaders and an overseer. In the event of leader or the cluster overseer failure, automatic fail over will choose new leaders or a new overseer transparently to the user and they will seamlessly takeover their respective jobs. Any Solr instance can be promoted to one of these roles.

3.2 Rationale

3.2.1 Rationale for using ReCAP Middleware database vs. ILS Transfer

Cross loading MARC records representing holdings of other ReCAP institutions into each ILS was considered as a design alternative. This option would have introduced significant costs and support burdens for partners. In some cases current ILS systems capacity or license thresholds would be exceeded. All three partners are planning to replace their current ILS systems in near future. This option would have incremented the data migration effort from existing ILS to the new ILS.

Loading ReCAP bibliographic and item records in middleware database provides a centralized repository for shared collection without impacting the ILS systems at partner institutions. Middleware database is needed to store the entire item and bib records of all there partner's private and shared collections. Also it is needed to record all the incoming requests from patrons and to maintain the transactions. Hence .middleware database is required irrespective of the decision to synchronize bibliographic and item records. This option comes at a marginal increase to the implementation and ongoing maintenance costs.

Hence the approach of loading shared bibliographic and item records to ReCAP middleware database is recommended over cross loading MARC records to partner ILS.

3.2.2 Rationale for using Kuali RICE vs. other commercial/open source frameworks

Major components required to support the architecture includes Service Bus, Rules Engine, Workflow Engine, Authentication and Authorization and User Interface/Experience framework.

While several open source projects such as JBoss, Spring, and JQuery presented compelling components, these components have to be integrated by the project team to provide a seamless platform for ReCAP middleware.

Kuali RICE framework presents the benefits of open source such as no license costs and vendor dependencies. The framework leverages several industry standard frameworks such as Spring, JQuery, etc. and provides an enterprise grade end-to-end integrated framework well suited for ReCAP middleware development.

Rice is built on a Service Oriented Architecture (SOA) providing common enterprise workflow functionality, customizable and configurable user interfaces with a clean and universal look and feel, and general notification features to allow for a consolidated list of work "action items." Additionally, there are a set of services in Rice that provide identity and access management capabilities and can be used to abstract away from underlying institution-specific identity services. All of this adds up to a re-usable development framework that encourages a simplified approach to developing true business functionality as modular applications.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Kuali Service Bus (KSB) provides service management and routing functionalities. Workflow and Messaging domain is taken care by Kuali Enterprise Workflow (KEW) and Kuali Enterprise Notification (KEN). Kuali Identity Management (KIM) provides services for authentication and authorization management. Also it has Kuali Rules Management (KRM) for business rule development and execution as well as information delivery and analysis.

Kuali foundations commitment to provide and support enterprise scale framework for the higher education and academic library community makes Kuali RICE a compelling choice for this project.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4. Architectural Views

4.1 Use-Case View

The Architecturally significant Use Cases identified during the High Level Architecture definition are listed below. The diagram provides the model for architecturally significant use cases.

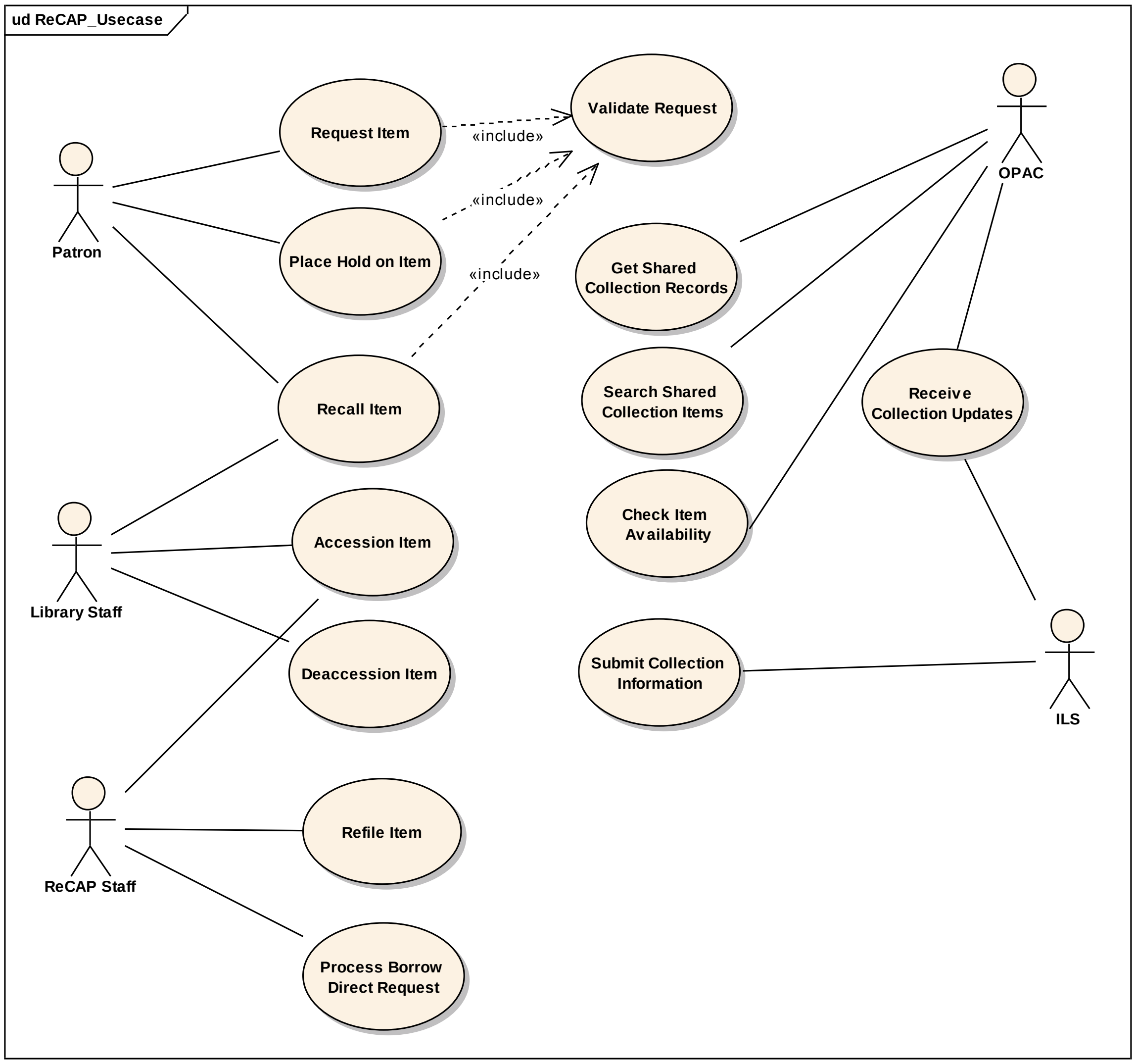


Figure 2 – Architecturally Significant Use-Case View

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

The following table lists the actors (user or system) interacting with the system.

No.	Actor	Description
1	Patron	A library patron is someone who uses a library, a university student or a city resident. Typically, this person gets a library card, browses the available books, CDs, DVDs, etc.
2	Library Staff	A library employee, who is responsible for a collection of specialized or technical information about items and management of items in a library.
3	ReCAP Staff	A Person responsible for day to day activities at GFA facility including accessioning, deaccessioning, filing, re-filing, etc.
4	OPAC	An Online Public Access Catalog (often abbreviated as OPAC or simply Library Catalog) is an online database of materials held by a library Example: Bibliocommons, CLIO.
5	ILS	An integrated Library System (ILS) is an enterprise resource planning system for a library, used to track items owned, orders made, bills paid, and patrons who have borrowed. Example: Millennium ,Voyager

The following lists the architecturally significant Use Cases.

No.	Use Case Name	Architecture Complexity
1	Search Shared Collection Items	Complex
2	Request Item	Complex
3	Validate Request	Simple
4	Place Hold on Item	Complex
5	Recall Item	Complex
6	Accession Item	Medium
7	Deaccession Item	Medium
8	Process Borrow Direct Request	Complex
9	Re-file Item	Complex
10	Check Item Availability	Complex
11	Get Shared Collection Records	Medium
12	Submit Collection Information	Medium
13	Receive Collection Updates	Medium

A brief description of the architecturally significant use cases has been listed below. Each of the use case description includes key business rules and includes reasons for architectural significance.

4.1.1 Search Shared Collection Items

In this use-case the patron will search the OPAC for institution items as well as shared collection items placed by other ReCAP partners. Search for an item in OPAC will initiate search to OPAC's index and ReCAP index. The two search results will be merged to include shared collection items in the search results.

4.1.1.1 Architectural Significance

- Core Functionality
- Complexities –Includes collecting bibliographic and item data from all three partners,

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

normalizing and indexing the data and providing offline feed or API for OPAC systems.

4.1.2 Request Item

In this use-case a patron will request a ReCAP item by submitting the request through a web form. The form will submit the request to middleware API, which will invoke other use-cases to process the request.

4.1.2.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will interact with GFA LAS, ILS and OPAC to process the request. It will create a temporary item record in one of the three applicable ILS depending upon the patron.

4.1.3 Validate Request

Request item use-case will invoke this use-case to validate the request for requested item, delivery location, delivery type, etc. Upon successful validation control will be returned to the main use-case with a confirmation message and upon unsuccessful validation an error message will be returned.

4.1.3.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will validate against ReCAP circulation policies and item availability in the middleware database.

4.1.4 Place Hold on Item

In this use-case a patron will place hold against an item whose status is currently unavailable.

4.1.4.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will maintain a single hold queue for all the partner institutions in a first-in, first-out basis. The hold queue will be automatically propagated to all applicable ILS systems.

4.1.5 Recall Item

In this use-case a patron/library staff will recall an item whose status is currently unavailable.

4.1.5.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will interact with owning or borrowing institution ILS to send the Recall request and maintain the queue in middleware database.

4.1.6 Accession Item

In this use-case Library Staff will upload bib and item records for new ReCAP items. ReCAP middleware will interface with GFA LAS to check the accessioned item status and then apply accessioning algorithm. Applying accessioning algorithm will result in one of the following three scenarios. A valid collection code will be assigned after which the item will be a shared collection item. It might also result in duplicates in

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

which case, the items might be placed under institutional access instead of shared access. Detecting duplicates might also result in libraries withdrawing their items. So the proposed accessions are the items which are sent by Library staff for accessioning and actual accessions are the items which are assigned collection code after running the accessioning algorithm. Item information with assigned collection codes will be returned back to the owning libraries through SFTP drop. ReCAP staff also participates in the accessioning of an item

4.1.6.1 Architectural Significance

- Core Functionality
- Complexities – Accessioning algorithm will be run every time an item is accessioned in ReCAP. Accessioning algorithm includes a tie-breaker to cover most of the scenarios. Match and normalize disparate bib and item data across three partner ILS and GFA LAS. The item barcodes and applied circulation codes data will be returned to owning partner ILS.

4.1.7 DeAccession Item

In this use case Library Staff will initiate a request to deaccession an item through staff interfaces. Based on the collection code a manual approval workflow will be triggered to deaccession the item. ReCAP staff also participates in the deaccessioning of an item

4.1.7.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will run accessioning algorithm to reassign circulation codes for other items after deaccessioning an item. A review/approval workflow will be implemented to manage preservation collections.

4.1.8 Process Borrow Direct Request

This use-case will be invoked by ReCAP staff to process a Borrow direct request. The staff will scan the barcode in the Borrow direct request or enter one if barcode not available. Upon matching the barcode the staff can invoke the request item use case by clicking the confirmation button.

4.1.8.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will provide a thick client interface for barcode scanning to the ReCAP staff. The solution will maintain existing workflow for ReCAP staff and integrate the solution to middleware.

4.1.9 Re-file Item

In this use-case middleware will poll GFA LAS for re-filed items periodically, if an item is re-filed and has no hold or recall queue against it, its status will be changed to available. If a hold/recall queue exists the item will be processed for the first patron in the queue.

4.1.9.1 Architectural Significance

- Core Functionality
- Complexities – ReCAP middleware will actively poll GFA LAS to get the current status of the item. Once the item is checked-in (GFA), ReCAP middleware will process the item for next patron in queue and update corresponding ILS system.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.1.10 Check Item Availability

In this use-case OPAC will request for a real-time availability status of an item from ReCAP middleware. Middleware API will return the status from the index which is maintained in sync with the transaction database.

4.1.10.1 Architectural Significance

- Core Functionality
- Complexities – Real-time Item status will be provided through search API which is maintained in sync with the ReCAP database. Update search engine index without performance degradation.

4.1.11 Get Shared Collection Records

In this use-case OPAC systems will retrieve the other partner's shared collection records from SFTP server. The bib and item record will be normalized during inbound process and will be de-normalized during the outbound process to fit each partner's needs. The outbound records will be limited to other institution's shared collection items.

4.1.11.1 Architectural Significance

- Core Functionality
- Complexities –De-normalizing feeds for five OPAC systems.

4.1.12 Submit Collection Information

In this use-case partner ILS system will provide collection information, new accessioned and updates to bibliographic data through SFTP upload. Middleware will process data from all partners, normalize the data and ingest into middleware database. The normalized data will be updated to ReCAP index.

4.1.12.1 Architectural Significance

- Core Functionality
- Complexities – – Normalizing bib and item data from three ILS systems and de-normalizing feeds for five OPAC systems

4.1.13 Receive Collection Updates

In this use-case the ILS systems will retrieve collection updates from ReCAP middleware through SFTP drops. ReCAP middleware will de-normalize the data sets and provide updated collection information of item records pertinent to requesting institution only.

4.1.13.1 Architectural Significance

- Core Functionality
- Complexities – Identifying the collection update and provide offline export of owning library items only.

4.2 Logical View

The Logical View consists of two models: Analysis model and Design Model.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.2.1 *Analysis Model*

4.2.1.1 Overview

The Analysis Model provides a view of the requirements from the system's perspective. The requirements are refined, structured and the resulting elements are organized into logical groups of similar functionality called analysis packages.

The analysis model contains View-of-participating-classes (VOPC) which map out control, entity and boundary classes.

4.2.1.2 Analysis Packages

The Analysis Model contains the following main packages. The following diagram shows the overall package structure and dependencies:

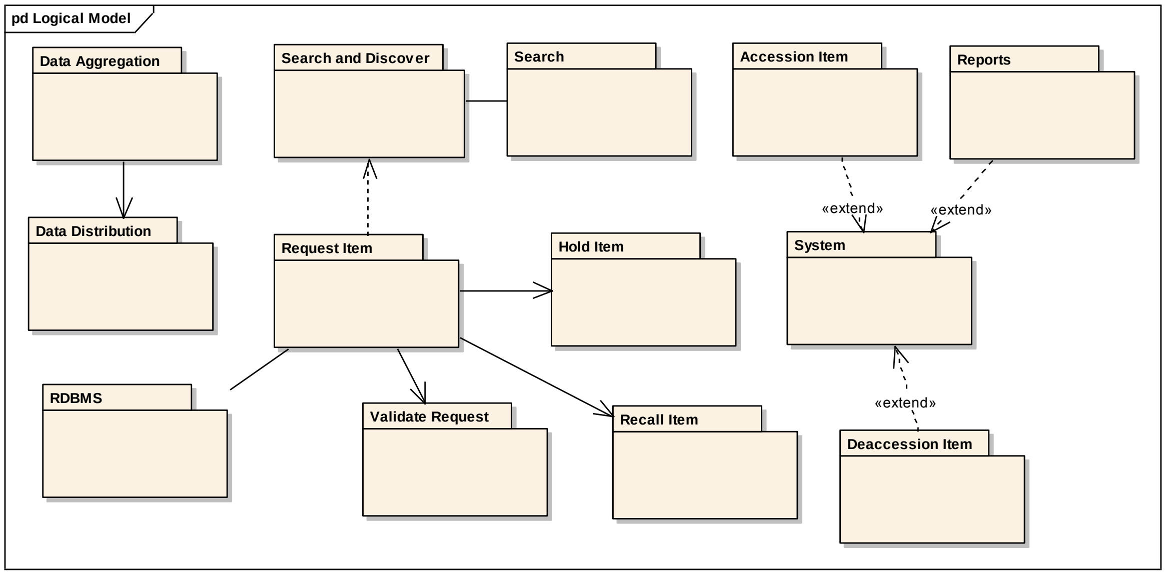


Figure 3 - Analysis Model Packages - Top Level Dependencies

Package Name	Package Description
Data Aggregation	This package contains classes which are responsible for consolidating and normalizing bib and item data from all three partner feeds.
Data Distribution	This package contains classes responsible for de-normalizing data from ReCAP middleware database and then distributes shared collection data to all 3 partners through SFTP uploads.
Search & Discover	This package contains classes which handle all the search requests from OPAC systems.
Validate Request	This package contains classes which are responsible for validating any incoming request.
Request Item	This package contains classes responsible for processing a ReCAP request.
Hold Item	This package contains classes responsible for processing and maintaining hold queue.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Recall Item	This package contains classes responsible for processing a recall request.
Accession Item	This package contains classes responsible for accessioning new items
DeAccession Item	This package contains classes responsible for Deaccessioning existing items
Reports	This package contains classes responsible for generating reports for Library staff.

4.2.1.3 Key Analysis Classes

Significant analysis classes are described below.

4.2.1.4 Boundary Classes

Boundary classes represent the interface between the system and the actors.

Class Name	Class Description
GFA Client	A class responsible for invoking SOAP service to interact with GFA LAS for item status.
NCIP Client	A class responsible for invoking NCIP responders to update/create item details.
Search Service	A class publishing Services related to bib and item records and its real time status.
Barcode Client	A class responsible for handling events related to barcode scanning (Borrow Direct requests)

4.2.1.5 Control Classes

Control classes represent classes that co-ordinate flow between the entities and the boundary classes.

Class Name	Class Description
Authentication Controller	Controller Class which handles all the incoming requests for authentication.
Search Controller	Controller Class which handles all the search requests for shared collection items.
Request Controller	Controller Class which handles all the transactions for item requests.
Workflow Controller	Controller Class which handles all the workflow requests such as Accessioning and Deaccessioning items.
Report Controller	Controller class which handles all the report requests.

4.2.1.6 Entity Classes

Entity classes represent information and associated behavior that must be stored. They are usually persistent.
Important entity classes are listed below:

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Class Name	Class Description
Bib	A bibliographic record is an entry being a uniform representation and description of a specific content item in a bibliographic database (or a library catalog), containing data elements required for its identification and retrieval, as well as additional supporting information, presented in a formalized bibliographic format
Item	An item record represents a physical piece in the library.
Request	Entity that contains all the details for every single request received by ReCAP middleware

4.2.2 Design Model

4.2.2.1 Architecturally Significant Design Packages

The application has been partitioned into four layers:

- Presentation Layer
- Enterprise Services Layer
- Data Services Layer
- Data Layer

The presentation layer deals with presentation aspects of the system. The enterprise service layer isolates business rules and the data service layer from the presentation layer. The enterprise service layer implements common services such as “Search”, “Retrieve” etc. The Data Service Layer deals with data.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

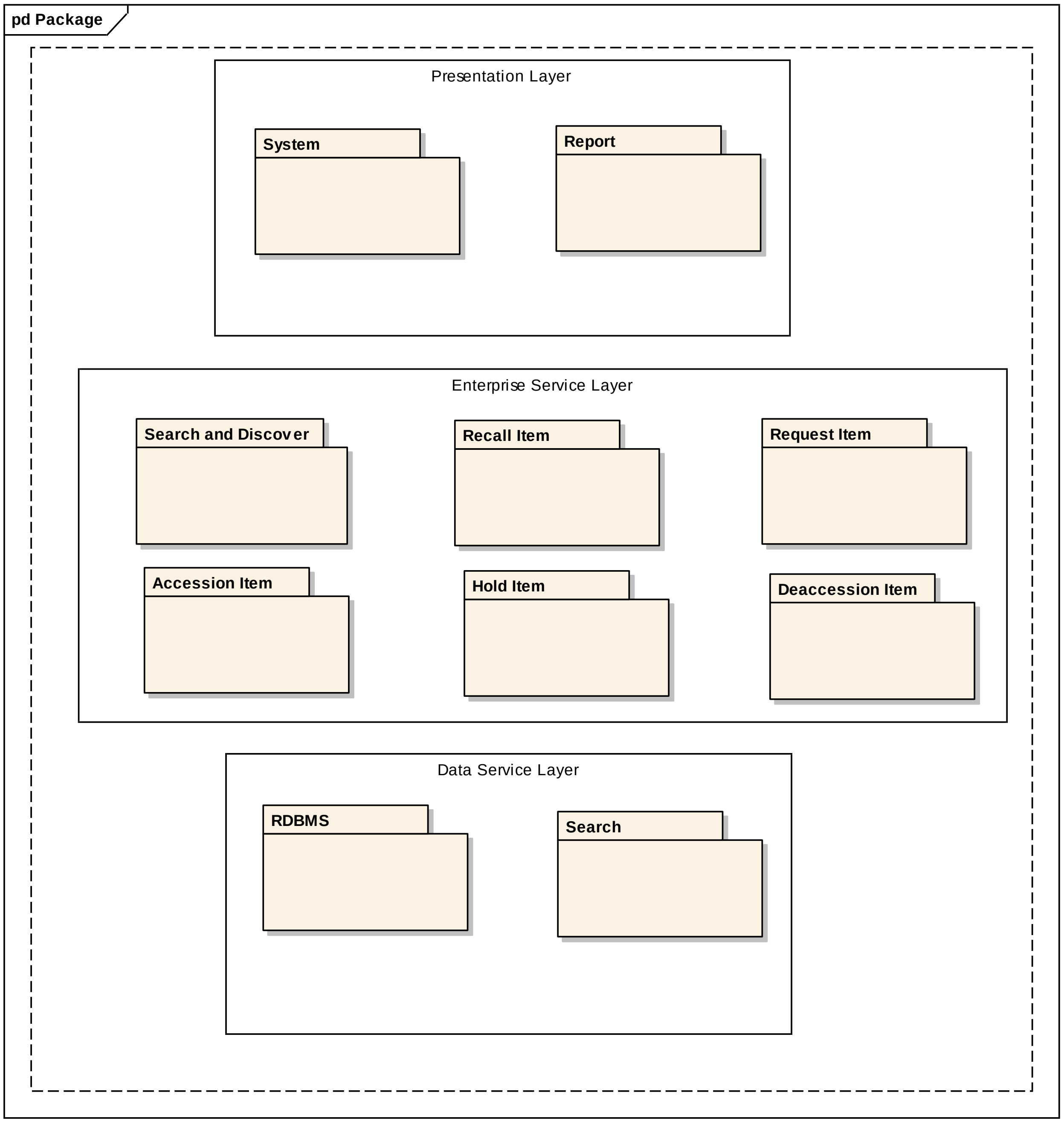


Figure 4 - Package Hierarchy

The presentation layer has been further sub-divided into "System" and "Report" packages. The System package contains the system administration specific implementation of the interfaces. The report package contains the report specific components.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Enterprise Services Layer:

The enterprise services layer contains the “Service” components. The layer uses the “delegate” pattern to delegate to the appropriate architecture (RDBMS or Solr). The underlying services are accessed via a “Façade” bean. Any bean that acts as a façade would contain a delegate class within its package.

The following are sample list of packages that exist within this layer:

Search & Discover – This package contains classes which handle all the search requests from OPAC systems.

Request Item – This package contains classes handling all the logic to process a ReCAP request.

Hold Item – This package contains classes responsible for processing and maintaining hold queue

Recall Item – This package contains classes responsible for processing a recall request.

Accession Item - This package contains classes responsible for accessioning new items
DeAccession Item – This package contains classes responsible for Deaccessioning existing items.

Data service layer contains RDMBS and Search Packages. RDBMS package consists of all classes which interact with the relational database and Search Packages consists of classes which interact with SOLR.

4.2.3 Frameworks, Patterns and Guidelines

Application frameworks are a promising technology for reifying proven software designs and implementations in order to reduce the cost and improve the quality of software.

A framework is a reusable, ``semi-complete'' application that can be specialized to produce custom applications. In contrast to earlier OO reuse techniques based on class libraries, frameworks are targeted for particular business units (such as data processing) and application domains (such as user interfaces)

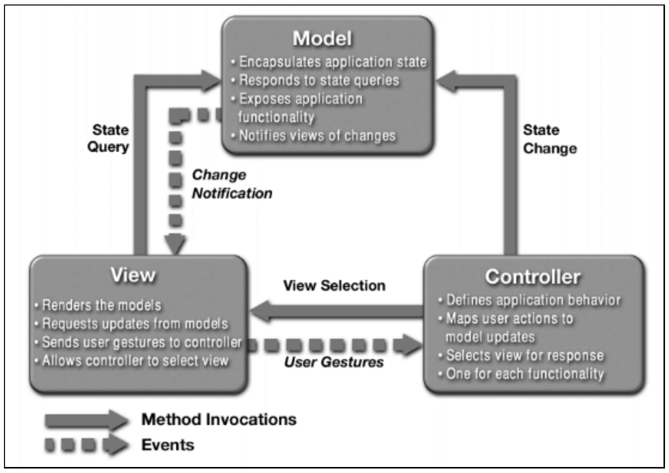
4.2.3.1 Common Patterns

All the diagrams presented in this section are copyright of their respective creators and ReCAP project will use most it not all the below patterns during its implementation.

4.2.3.2 Model-View-Controller (MVC) Pattern

The proposed ReCAP architecture will use MVC model for its core framework. The diagram below (courtesy Sun Microsystems) explains the concepts behind MVC:

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

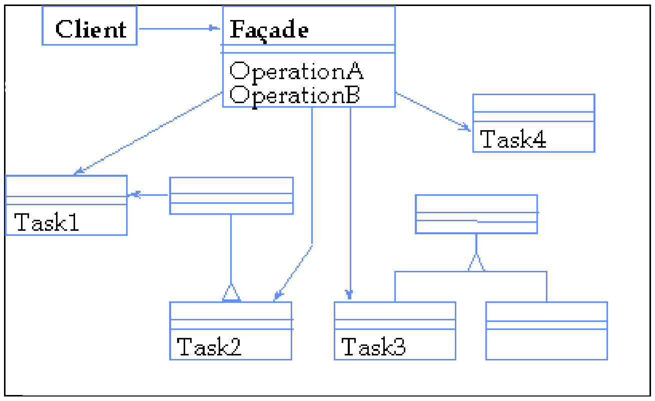
- **Model:** The model represents enterprise data and the business rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model
- **View:** The view renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible for calling the model when it needs to retrieve the most current data
- **Controller:** The controller translates interactions with the view into actions to be performed by the model. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

4.2.3.3 Façade Pattern

The façade pattern is used in the design at many points. Most significant use is via the Service Implementations, where a Service Implementation Class (business service implementation) acts as a façade to the business layer (hiding the business layer complexities) and also provides a simpler interface for the clients to work with.

The diagram below highlights the details of Façade Pattern:

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.



The intent of this pattern is to hide complex underlying structural details with a simpler interface providing following benefits.

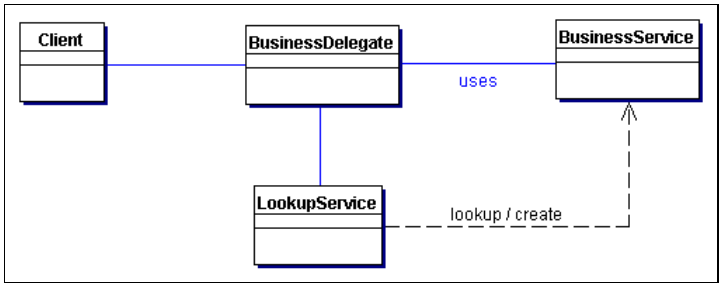
- Provides a simpler interface for the clients to work with
- Reduces number of objects that the client needs to work with
- Promotes weak coupling

4.2.3.4 Business Delegate Pattern

The Business Delegate pattern can be used to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the business tier components. The lookup service could be implemented using the "Service Locator" pattern.

It is normally implemented by defining a business interface, which is implemented by a delegate class and the business component (if the component is being created afresh. If not, then the delegate acts as a façade).

The diagram below illustrates this pattern:



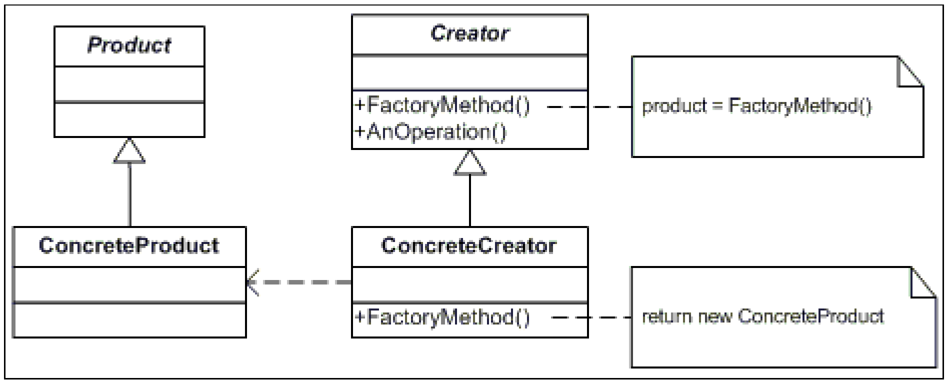
This model will be used wherever multiple implementations are possible for an interface.

4.2.3.5 Factory Pattern

Factory Method is a pattern used for object creation. This pattern helps model an interface for creating an object, which at creation time can let its sub-classes, decide the class to be instantiated. This is called a Factory Pattern since it is responsible for "Manufacturing" an Object. The Factory Pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code.

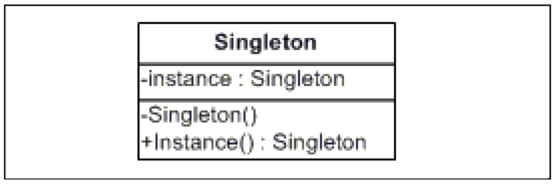
Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

The application layer framework will use this pattern to manage the ingestion factory.



4.2.3.6 Singleton Pattern

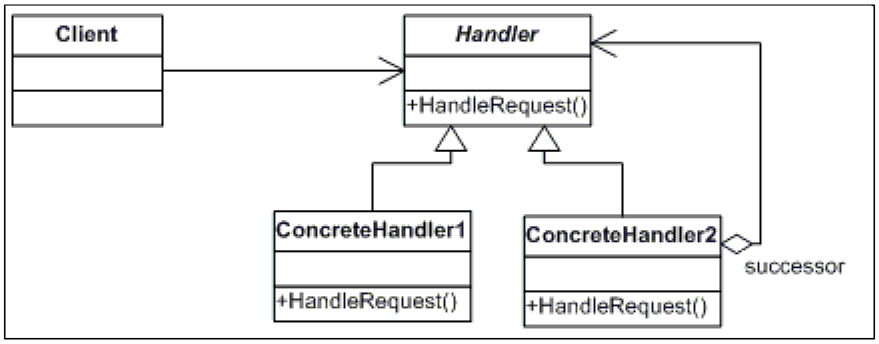
Singleton pattern is used to ensure a class has only one instance, and provide a global point of access to it. It also encapsulates “just-in-time initialization” or “initialization on first use”.
The application framework will use this pattern to initialize single instances of all configuration properties.



4.2.3.7 Chain of Responsibility Pattern

Chain of Responsibility pattern is used to avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. This is achieved by chaining the receiving objects and passing the request along the chain until an object handles it. Each object in chain launches and leaves requests with a single processing pipeline that contains many possible handlers thus creating an object-oriented linked list with recursive traversal.

The application framework will use this pattern to manage several independent steps of output creation.

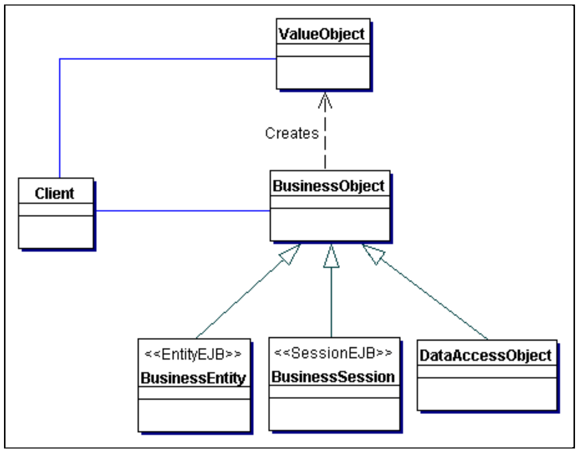


Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.2.3.8 Value Object or Transfer Object Pattern

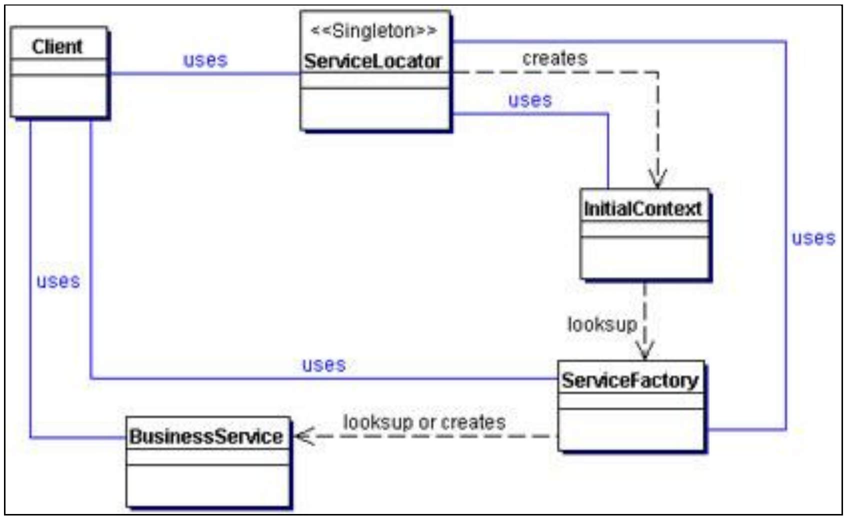
In typical applications there is a need to get several properties or exchange business data between tiers. This exchange requires multiple round trips over a network and could result in poor performance. To improve the performance the business data could be encapsulated into a "Value" or "Transfer" object and passed to the service. Similarly the service could return a "Value" object, rather than having the client make several "get" calls. Typically value objects have a VO suffix.

The application framework will use this pattern to exchange data across tiers. The following diagram illustrates this pattern:



4.2.3.9 Service Locator

Service locator pattern is used to abstract the complexities of initializing all services. Multiple clients can reuse the Service Locator object to reduce code complexity, provide a single point of control, and improve performance by providing a caching facility.
This pattern reduces the client complexity that results from the client's dependency on and need to perform lookup and creation processes, which are resource-intensive. To eliminate these problems, this pattern provides a mechanism to abstract all dependencies and network details into the Service Locator.



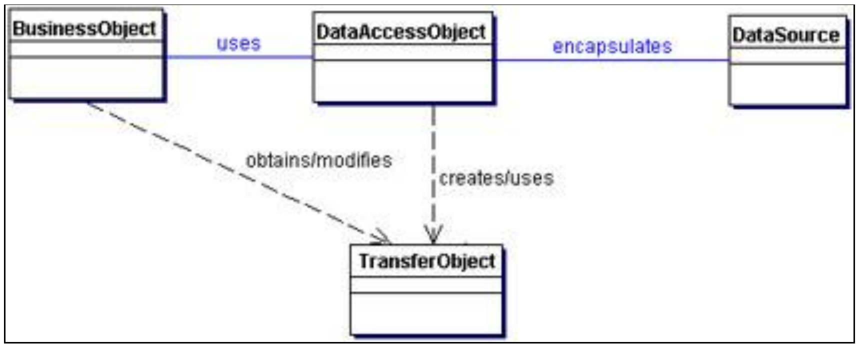
Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.2.3.10 Data Access Object

Data Access Object (DAO) pattern is used to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data.

The DAO implements the access mechanism required to work with the data source. The data source could be a persistent store like an RDBMS, an external service like a B2B exchange, a repository like an LDAP database, or a business service accessed via CORBA Internet Inter-ORB Protocol (IIOP) or low-level sockets. The business component that relies on the DAO uses the simpler interface exposed by the DAO for its clients. The DAO completely hides the data source implementation details from its clients. Because the interface exposed by the DAO to clients does not change when the underlying data source implementation changes, this pattern allows the DAO to adapt to different storage schemes without affecting its clients or business components. Essentially, the DAO acts as an adapter between the component and the data source.

The application framework will use this pattern to encapsulate all data store calls from services.

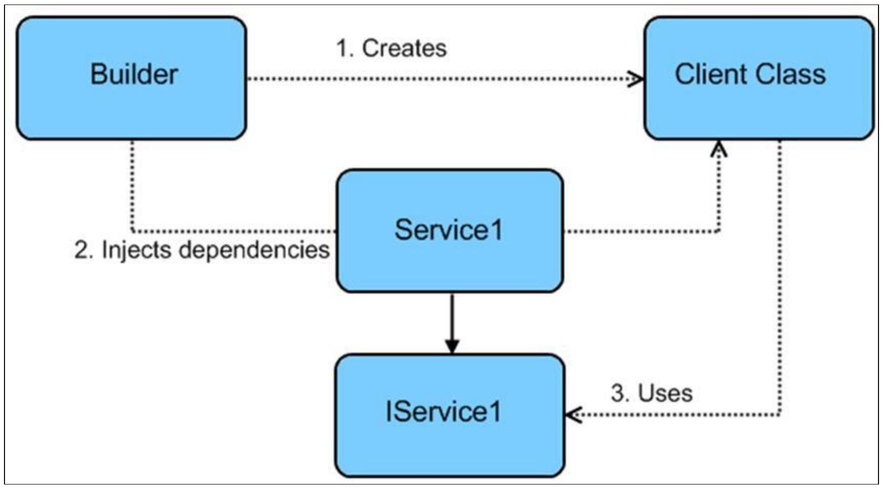


4.2.3.11 Inversion of Control (Spring Framework)

Inversion of Control or IoC is one of the techniques used to wire services or components to an application program. By definition, IoC is “A software design pattern and set of associated programming techniques in which the flow of control of a system is inverted in comparison to the traditional interaction mode.” Simply stated, in IoC, instead of an application calling the framework, it is the framework that calls the components specified by the application.

However, IoC is a broad and generic term. The aspect of IoC that the Spring Framework uses is "Injection of required resources or dependency at Run-time into the dependent resource," which is also known as Dependency Injection. Hence, the service provided by the IoC container of Spring is Dependency Injection.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.



4.3 Process View

The diagram below shows the process views from within various layers with respect to the Web container and clients interacting with the system:

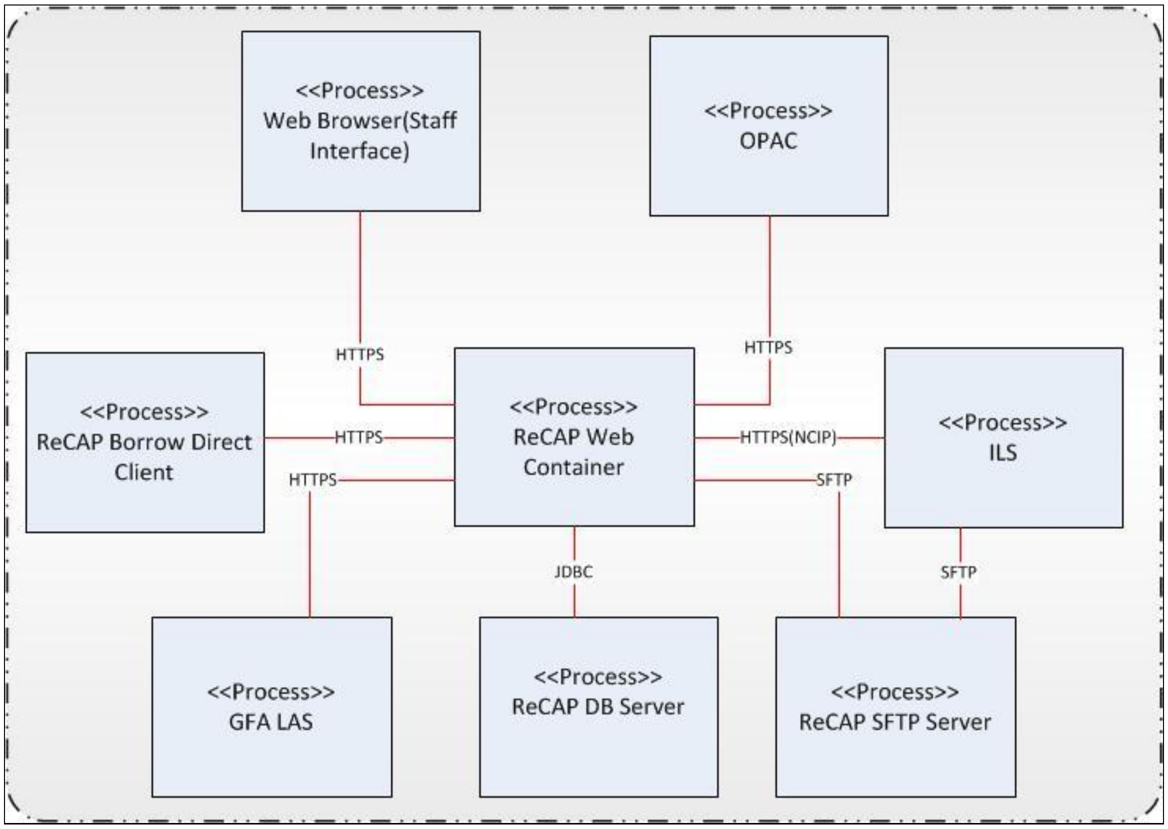

<<Process>> Web Browser(Staff Interface)
<<Process>> OPAC
HTTPS
HTTPS
<<Process>> ReCAP Borrow Direct Client
HTTPS
HTTPS
<<Process>> ReCAP Web Container
HTTPS(NCIP)
SFTP
<<Process>> ILS
JDBC
SFTP
<<Process>> GFA LAS
<<Process>> ReCAP DB Server
<<Process>> ReCAP SFTP Server

Figure 5 - Overall Process View

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.4 Deployment View

The deployment view of the ReCAP system shows the physical nodes on which the system executes and the assignment of the system processes to the nodes. The system can be deployed on different hardware configurations.

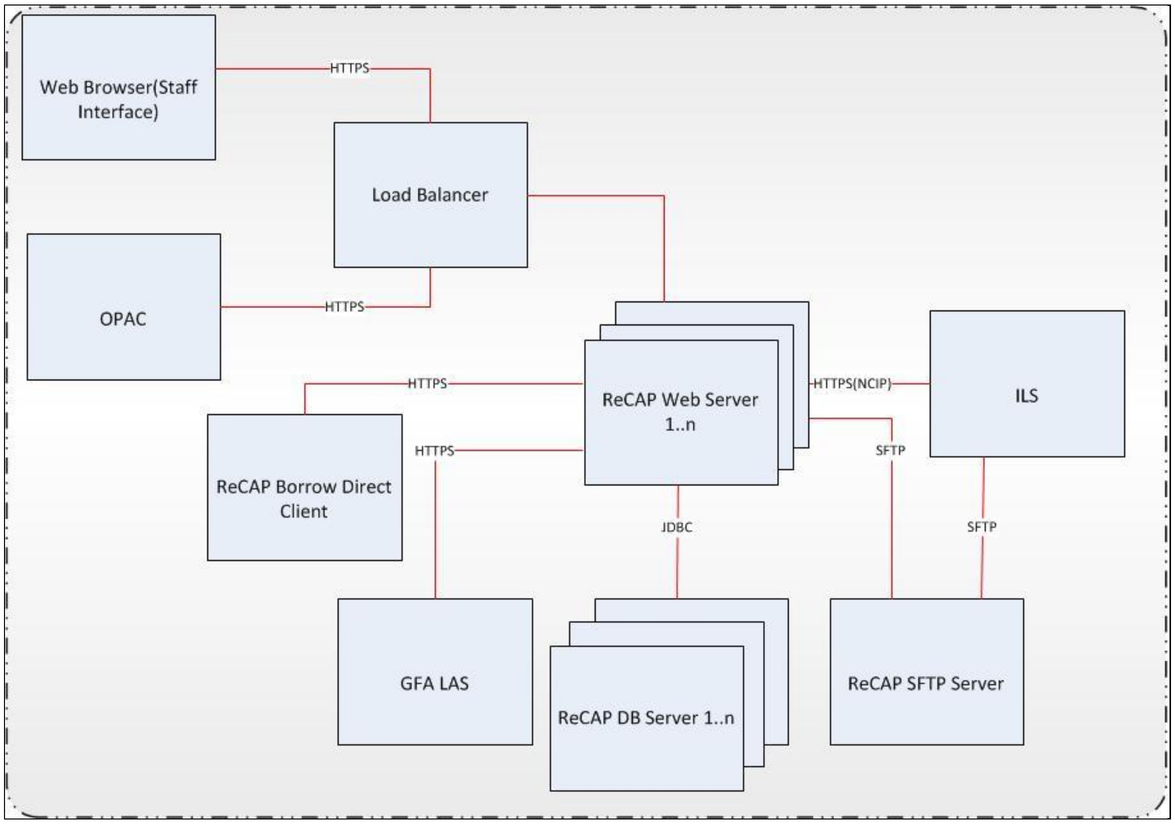


Figure 6 - Deployment View

Please refer to “Size and Performance” section for more details.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.4.1 Amazon Cloud Configuration (Production Instance)

Component	***Name***	**Quantity**
Database	RDS - Heavy Utilization Extra Large Single AZ (15 GB of memory, 8 ECUs (4 virtual cores with 2 ECUs each), 64-bit platform, High I/O Capacity, Provisioned IOPS Optimized: 1000Mbps)	2
	DB Single AZ Storage (250 GB storage)	1
	Provisioned IOPS	1
Web Server	EC2 - Heavy Utilization High CPU Extra Large(7 GiB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of local instance storage, 64-bit platform)	2
	EBS Optimization Fee for the Extra Large Instance	2
	EC2 - Heavy Utilization Standard Medium(3.75 GiB of memory, 2 EC2 Compute Units (1 virtual core with 2 EC2 Compute Units each), 410 GB of local instance storage, 32-bit or 64-bit platform)	1
Storage	EBS Storage(250 GB storage)	1
	Provisioned IOPS	1
Backup Storage	S3 Snapshot of EBS Volumes (500 GB storage)	1
Load Balancer	Elastic Load Balancer	1

4.4.2 Amazon Cloud Configuration (QA & Development Instance)

Component	***Name***	**Quantity**
Database	RDS - Heavy Utilization Large Single AZ (7.5 GB memory, 4 ECUs (2 virtual cores with 2 ECUs each), 64-bit platform, High I/O Capacity, Provisioned IOPS Optimized: 500Mbps)	1
	DB Single AZ Storage (250 GB storage)	1
Web Server	EC2 - Heavy Utilization Standard Large(7.5 GiB of memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB of local instance storage, 64-bit platform)	1
	EBS Optimization Fee for the Extra Large Instance	1
	EC2 - Heavy Utilization Standard Medium(3.75 GiB of memory, 2 EC2 Compute Units (1 virtual core with 2 EC2 Compute Units each), 410 GB of local instance storage, 32-bit or 64-bit platform)	1
Storage	EBS Storage(250 GB storage)	1
Backup Storage	S3 Snapshot of EBS Volumes (500 GB storage)	1

Note: *EC2 Compute Unit (ECU) – One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor*.

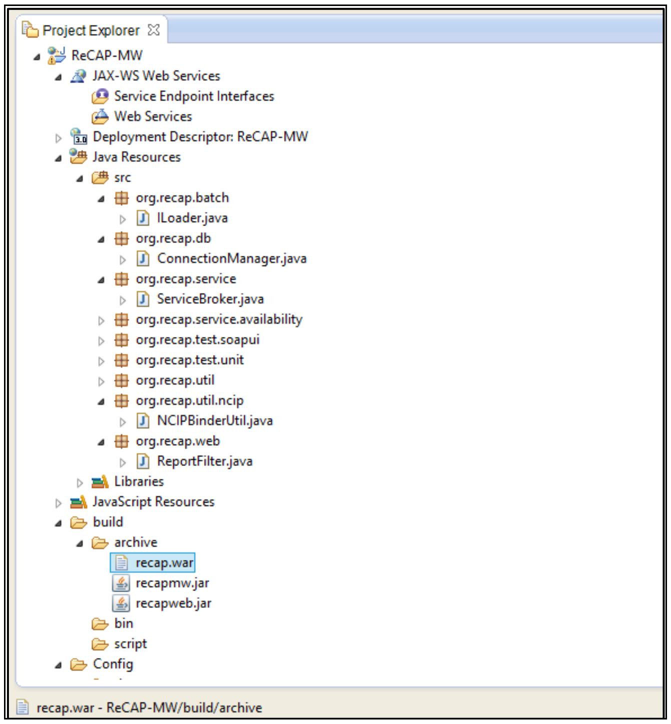
Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.5 Implementation View

The implementation view shown here is only a starting point and will be refined in during development phase.

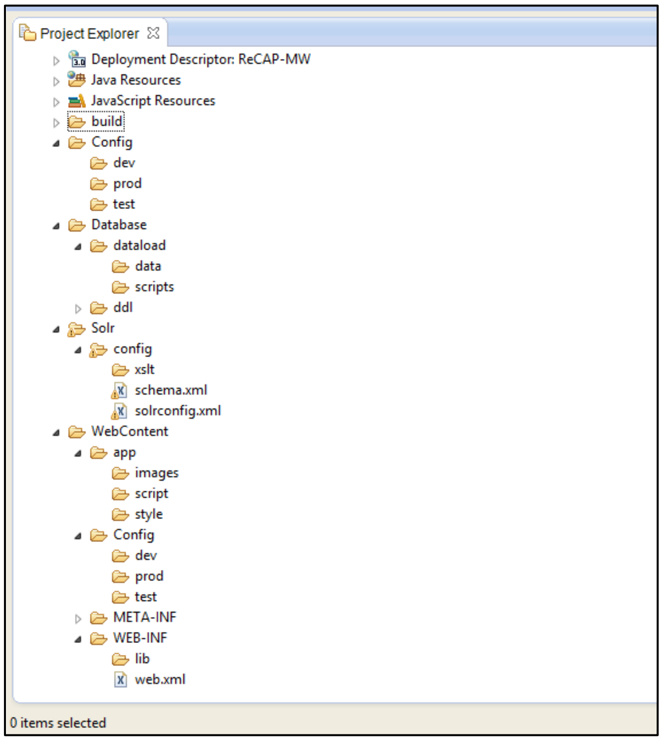
4.5.1 Layers

The layers, packages and its hierarchy are represented using the following diagrams. The package hierarchy starts with standard Java namespace compliant structure and then is divided into two sub-packages for the different layers viz. src – Application (Services, Data Access) and Web Content – Presentation.



```
Project Explorer
◢ ReCAP-MW
    ◢ JAX-WS Web Services
        Service Endpoint Interfaces
        Web Services
    ▷ Deployment Descriptor: ReCAP-MW
    ◢ Java Resources
        ◢ src
            ◢ org.recap.batch
                ▷ ILoader.java
            ◢ org.recap.db
                ▷ ConnectionManager.java
            ◢ org.recap.service
                ▷ ServiceBroker.java
            ▷ org.recap.service.availability
            ▷ org.recap.test.soapui
            ▷ org.recap.test.unit
            ▷ org.recap.util
            ◢ org.recap.util.ncip
                ▷ NCIPBinderUtil.java
            ◢ org.recap.web
                ▷ ReportFilter.java
        ▷ Libraries
    ▷ JavaScript Resources
    ◢ build
        ◢ archive
            recap.war
            recapmw.jar
            recapweb.jar
        bin
        script
    ◢ Config
recap.war - ReCAP-MW/build/archive
```

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.



```
Project Explorer
▷ Deployment Descriptor: ReCAP-MW
▷ Java Resources
▷ JavaScript Resources
▷ build
◢ Config
      dev
      prod
      test
◢ Database
   ◢ dataload
         data
         scripts
   ▷ ddl
◢ Solr
   ◢ config
         xslt
         schema.xml
         solrconfig.xml
◢ WebContent
   ◢ app
         images
         script
         style
   ◢ Config
         dev
         prod
         test
   ▷ META-INF
   ◢ WEB-INF
         lib
         web.xml
0 items selected
```

Figure 7 - Implementation View

4.5.2 Error handling

Error handling will be implemented by leveraging the Exceptions feature of the Java language. The following guidelines are suggested when dealing with exceptions

No.	Exception Guideline	Description
1	Exceptions should not create additional package dependencies	Assume that a client class in package A accesses a class in package B. The class in package B should not throw an exception that belongs to package C (which is used by package B). This produces dependencies between package A and C. (This rule may not apply where Package C is a standard and stable package, such as java.io.)

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

2	Exceptions by package.	If a package's classes throw any exceptions, the package should have its own top-level checked exception. The package should then define exception subclasses for any exceptions that may be handled differently by clients. Good models for this paradigm can be found in the Java packages java.io, java.sql and javax.naming. Sometimes a package is clearly a "sub-package" of another package. In such a case, the sub-package's exceptions can extend the parent package's exceptions. In example of such a sub-package is java.nio.charset (whose exceptions extend java.io.IOException).
3	No blind catches of Exception	A class is responsible for knowing what exceptions it may encounter, and it must treat each exception individually. If the handling of many exceptions is identical, it could be extracted into helper methods.
4	No empty catch-blocks	At the very least, a catch-block should contain an assertion that it should never be reached or a comment stating that it is irrelevant
5	Write sensible throws clauses	Fewer (<3) the number of exceptions thrown, better it is. Always throw exceptions that make sense to the calling class, if not wrap that exception in another, which more closely captures the error type
6	Chaining Exceptions	Always chain exceptions so that the root cause of the error is available for logging it into the error/system log file. This is very useful for diagnosing errors in production environment
7	Use Message Catalogs for easy localization	Use message catalogs for message text of an exception, whose message is directly presented to the end user. This will help the application to be localized or internationalized by just adding another message catalog

Applications when encountering an exception should always log it to the Application/System log. Lower level components should avoid writing to an error/system log.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

4.6 Data View

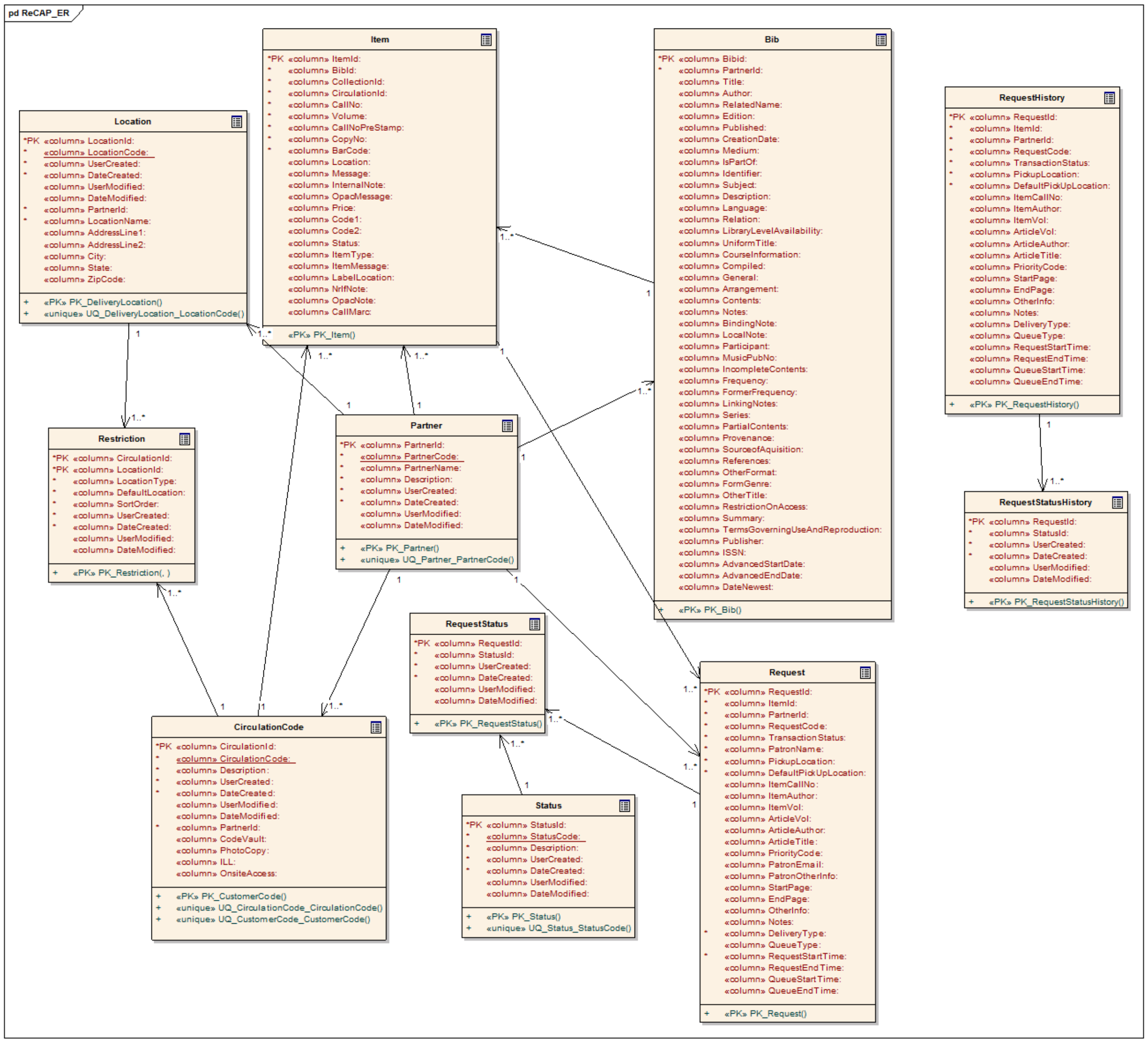


Figure 8 – Entity Relationship

The following table lists all the entities along with their definition:

No.	Entity	Description
1	Item	An item record represents a physical piece in the library

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

2	Bib	A bibliographic record is an entry being a uniform representation and description of a specific content item in a bibliographic database (or a library catalog), containing data elements required for its identification and retrieval, as well as additional supporting information, presented in a formalized bibliographic format
3	Location	Entity that contains list of valid delivery locations
4	Partner	List of names and contacts of the ReCAP partners
7	Request	Entity that contains all the details for every single request received by ReCAP middleware.
8	Status	Entity that contains details of all the statuses applicable for items in ReCAP middleware
9	RequestStatus	Entity that contains details and timelines of all statuses associated to a ReCAP request.
10	RequestHistory	Archive Entity for completed Request (without any patron information).
11	RequestStatusHistory	Archive table for RequestStatus entity.
12	CirculationCode	Entity that contains details about various types of circulation policies for shared collection items
13	Restriction	Entity that contains details about policy restrictions for shared collection items

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

5. Size and Performance

5.1 Scalability

This section explains how the architecture aims to achieve scalability.

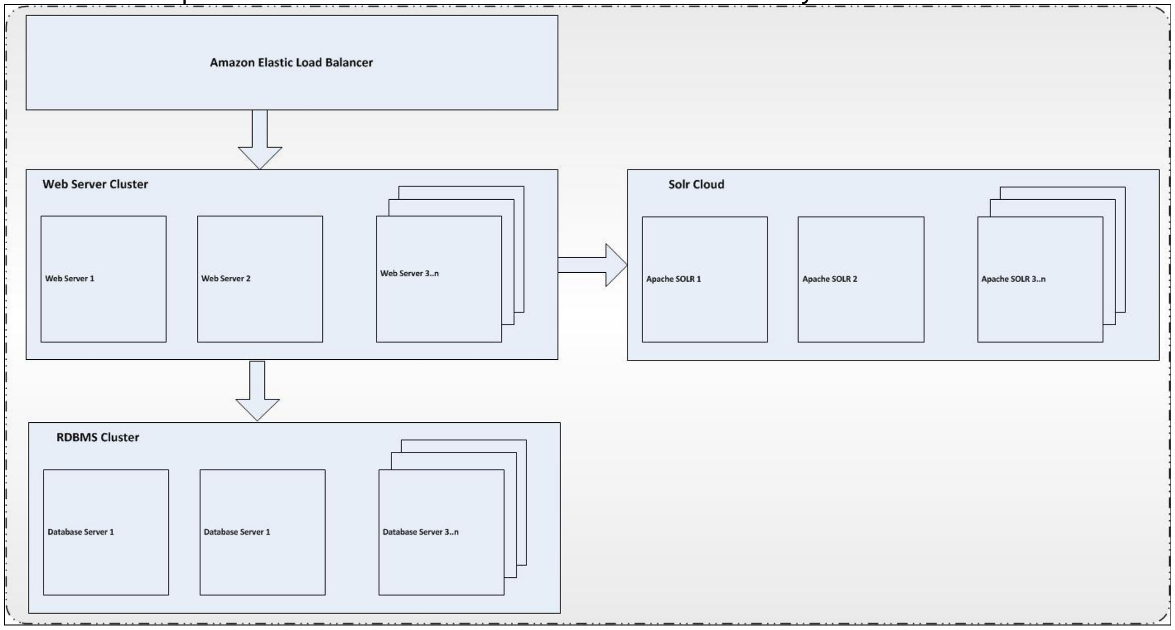


Figure 9 - Scalability

The figure above presents how the architecture aims to be scalable. There are several elements that contribute to scalability:

- Network or HTTP load balancers – These appliances or devices would perform load balancing of HTTP and other protocol specific servers. The actual mechanism of load balancing will depend upon specific device and could include mechanisms like round robin, cookie sniffing etc. In a web environment, the load balancers will balance the load between web servers. The web server maintains state (user specific). This typically means that once a session is established, a user is redirected to the same web server. Relatively inexpensive servers (nodes) could be used for the web servers. Redundant servers could be used to provide high-availability.
- Web Server Cluster – The Web server cluster appears to the client application (Browser or Service Client) as a single server. The Web server provides clustering capability. Although session state could be replicated, it could result in performance hits. The architecture presents design using sticky sessions to provide high availability and fault tolerance without compromising performance. The Web server cluster scales by adding more nodes to the cluster. The applications will need no change when the cluster scales. Again, relatively inexpensive servers could be used to enable linear scaling.
- RDBMS cluster – The RDBMS servers will be clustered to provide scalability. The RDBMS cluster appears as a single server to the user of the database. The RDBMS product takes care of data replication and clustering challenges.
- Solr Cloud – Solr Cloud creates a cluster of Solr servers representing two different shards of a collection (complete index). While shards provide distributive scaling, shard replication provides fault tolerance. Zookeeper takes care of data replication and clustering challenges.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

5.2 Performance

This section presents how the architecture addresses performance related issues.

This section presents various design patterns used to achieve performance.

Basic performance metrics are latency and throughput. Latency is measured as the time elapsed between request and response and throughput as the number of requests handler per second. In an ideal world, the latency should not increase and throughput should scale linearly as the load increases.

Performance related issues needs to be investigated at various points of the architecture. Some common elements that should be subjected to performance tuning are:

- Middleware – Middleware technologies like ESB or other distributed technologies are primary candidates for performance tuning as issues like network round-trips and network latency could become critical.
- Database – Database access and processes like joins and sorts are candidates for performance tuning.
- Search Engine – Solr caching is an candidate to improve search performance by leveraging cached queries and results

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

6. Proposed Development Environment

6.1 Hardware

The core technology will be Java and an implementation of a servlet container. MySQL will be the relational data store and apache SOLR will be the search engine. Application development environment will be hosted in Amazon cloud on Linux. Each developer will have his or her own development setup on PC or Mac and access source code that is stored in a common source control repository such as SVN or CVS.

It is assumed that Partners (NYPL, Princeton and Columbia) will be hosting the development, QA and production environments of NCIP Responders and GFA LAS.

6.2 Software

No.	*Name*	Purpose
1	Eclipse Juno 4.2	Eclipse is a multi-language software development environment comprising a base workspace(Eclipse Public License (EPL))
2	MySQL 5.6	Open source relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases (GNU General Public License)
3	Tomcat 7.0	Apache Tomcat is an open source web server which provides pure Java HTTP web server environment for Java code to run. (Apache License)
4	Kuali RICE 2.2.3	Kuali RICE comprises of a suite of middleware programs (workflow, messaging, identity management), interfaces and Web services around a service bus (Educational Community License)
5	Apache SOLR 4.2.1	SOLR is an open source enterprise search platform written in Java and runs as a standalone full-text search server within a servlet container.(Apache License)
6	Apache Quartz 2.1.7	Quartz is a full-featured, open source job scheduling service that can be integrated with, or used alongside virtually any Java application (Apache License)
7	ProFTPD 1.3.5rc2	ProFTPD is an secured FTP server exposing a large configuration options to the user(GNU General Public License)
8	Jenkins 1.511	Jenkins is a server-based system running in a servlet container providing open source continuous integration features.(Massachusetts Institute of Technology (MIT) License)
9	JUnit 4	JUnit is a unit testing framework for the Java Programming language (Common Public License)
10	SOAP UI 4.5.1	SoapUI is an open source web service testing application for service-oriented architectures (SOA) and provides functionalities like web service inspection, invoking, development, simulation and mocking, functional testing, load and compliance testing (GNU Lesser General Public License (LGPL))
11	Maven 3.0.5	Maven is a build automation tool used primarily for Java projects (Apache License)
12	Java Platforms (Java 7)	Java is a set of several computer software products and specifications that together provide a system for developing application software and deploying it in a cross-platform

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

		computing environment (Freeware)
13	Selenium IDE (1.10.0)	Selenium is a portable software testing framework which provides record/playback tool for authoring tests. (Apache License)
14	Motorola Scanner SDK	A framework providing a single programming interface across multiple programming languages and across multiple system environments for all Motorola scanners
15	Java Swing	Swing is the primary Java GUI widget toolkit developed to provide a more sophisticated set of GUI components

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

7. Prototype/Proof-of-Concepts

The proposed architecture is based on proven industry standard components, hence no specific prototype or proof-of-concept developed. core technology will be Java and an implementation of a servlet container.

8. Quality

Extensibility: The proposed architecture is based on Components adhering to well-defined interfaces and industry based standards (J2EE), so adding new features and implementing new components will not require extensive rework of existing components.

Reliability: The architecture uses standard J2EE architecture and will have the capabilities of Load balancing. Fail-over mechanisms will improve the reliability of the system.

Portability: The Enterprise Service Layer abstracts the platform and protocol specific implementation of common services into a generic set of interfaces. Clients use these interfaces to access the services are not tied to protocol/platform. It will be possible to port applications to different protocols/platforms by implementing the Generic set of interfaces on the client protocol/platform.

By complying with J2EE standards, it will be possible to port the application to multiple server vendors.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Appendix A Glossary

No.	Term	Description
1.	ReCAP	Research Collections and Preservation Consortium (ReCAP) is a storage facility for all the shared collection items.
2.	UML	Unified Modeling Language (UML) is a standardized, general-purpose modeling language in the field of software engineering
3.	GFA LAS	Generation Fifth Applications - Library Archive System is an inventory management system from Generation Fifth Applications which catalogs and controls archival storage of shared collection items in the ReCAP facility.
4.	Patrons	Users of the System who place a request for a shared collection item/items.
5.	ReCAP Middleware	The central component of this architecture which handles search, discover, request processing, reporting and collection management for a shared collection item by interacting with other components in the system.
6.	Shared collection items	Any item stored in the ReCAP facility which can be requested by any of the partners.
7.	Institution items	Any item stored in the ReCAP facility which can be requested only by the owning partner.
8.	Refiling	Refiling is a process of re-shelving the item in the ReCAP facility.
9.	Item Records	An item record represents a physical piece in the library
10.	Bib Records	A bibliographic record is an entry being a uniform representation and description of a specific content item in a bibliographic database (or a library catalog), containing data elements required for its identification and retrieval, as well as additional supporting information, presented in a formalized bibliographic format
11.	Federated Search	Federated search is an information retrieval technology that allows the simultaneous search of multiple searchable resources
12.	SOA	Service-oriented architecture (SOA) is a flexible set of design principles used during the phases of systems development and integration
13.	Temporary item record	An item record which is created in any of the partners ILS which is temporary in nature and requires deletion in the near future
14.	Circulation policies	Rules related to circulation of any given item to a patron

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

15.	Hold	To place a hold on an item means to reserve it. An item that is checked out may have a hold placed on it by another patron who wishes to use it. When the item is returned, the library will contact the patron who is waiting so they may come in and check it out.
16.	Recall	Recall is a special type of request by a library to a borrower for the return of a borrowed item before the due date.
17.	Owning institution	Any institution which borrows an item which belongs to itself is called an owning institution.
18.	Borrowing institution	Any institution which borrows an item which belongs to other partners is called an borrowing institution.
19.	Accession	The process of recording an item and its location in the ReCAP facility into the GFA LAS system.
20.	Barcodes	A barcode is an optical machine-readable representation of data relating to the object to which it is attached. Every item such as books or films and location such as aisle, shelf, and bin has a unique barcode.
21.	Circulation code	A unique identifier for every item which dictates its circulation policy.
22.	Collection Code	A code which is assigned to determine the scope of the sharing of an item.
23.	Customer Code	A unique identifier which is currently used in GFA to identify a group to which an item belongs.
24.	Staff interfaces	User Interface screens for Library staff
25.	Borrow Direct	Borrow Direct is an interlibrary borrowing service offered by all of the Ivy League Universities except Harvard.
26.	Source Control Repositories	It’s a space set aside to maintain code base for the ReCAP middleware project.
27.	SVN(Apache Subversion)	It’s a type of Source control repository.
28.	CVS(Concurrent Version System)	It’s a type of Source control repository.

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

29.	NCIP Responders	National Information Standards Organization Circulation Interchange Protocol (NCIP) is a protocol that is limited to the exchange of messages between and among computer-based applications to enable them to perform functions necessary to lend and borrow items, to provide controlled access to electronic resources, and to facilitate cooperative management of these functions. It's a mechanism by which ReCAP middleware communicates with any of the ILS which belongs to the participating institutions.
30.	Fault tolerance	Fault-tolerant describes a computer system or component designed so that, in the event that a component fails, a backup component or procedure can immediately take its place with no loss of service.
31.	Cluster	A cluster is a group of servers and other resources that act like a single system and enable high availability and, in some cases, load balancing and parallel processing.
32.	Shard	A shard is a term which is used in SolrCloud a feature in Apache SOLR which enables high fault tolerance of SolrCores.
33.	Load Balancer	Load Balancer achieves high fault tolerance by distributing incoming requests across one or more web servers.
34.	Deaccession	It is a process of withdrawing an item from ReCAP facility
35.	RDS	Amazon Relational Database Service (Amazon RDS) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud
36.	AZ	Multi-AZ deployment is for enhanced data durability and availability
37.	IOPS	The ability which Amazon provides to specify or provision the I/O capacity needs is called IOPS.
38.	EC2	Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud
39.	EBS	Amazon Elastic Block Store (EBS) provides block level storage volumes for use with Amazon EC2 instances
40.	Elastic Load Balancing	Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances
41.	S3	Amazon S3 is storage for the Internet. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Appendix B References

No.	Reference	Author
1.	ReCAP Technology Report.doc	Marshall Breeding
2.	ReCAP Grant Overview.doc	ReCAP Team
3.	ReCAP Technology Workshop.ppt	Marshall Breeding
4.	ReCAP Shared Collection Policies.doc	Lizanne Payne
5.	ReCAP Workflows.doc	HTC Team
6.	ReCAP NCIP.doc	HTC Team
7.	ReCAP Library Archival System API Requirements.doc	HTC Team

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.

Appendix C Naming and Coding Standards

Project will adopt the following industry standard naming and coding standards:

Code Conventions for the Java TM Programming Language

Guidelines, Patterns, and code for end-to-end Java applications

J2EE Patterns Catalog

Use or disclosure of the data or information on this page is restricted by the statement of confidentiality set forth on the second page of this document.