OpenLMIS System Architecture v1c (DRAFT)

Revision history:

Date	Author	Description
March 9, 2012	Rick Fant, Ron Pankiewicz (VillageReach)	V1 - First draft
March 19, 2012	Rick Fant, Ron Pankiewicz, James Dailey (VillageReach)	V1b - Updated draft based on input from Gaetano Borriello and team (University of Washington, Computer Science & Engineering Dept.) and additional review
March 29, 2012	Rick Fant, Ron Pankiewicz, James Dailey (VillageReach)	V1c – updated based on advisory team review

For further information, source documents, or if you would like to contribute to this effort, contact ron.pankiewicz@villagereach.org.

Design Overview

This document proposes an initial OpenLMIS reference model system architecture based on commonly adopted enterprise software models. The architecture is focused on providing a highly scalable, distributed, transaction-processing environment that can be deployed on open source and commercially available platforms. The architecture is suited for installations ranging from a community-focused operation serving a small number of users, up to a nation-wide operation supporting thousands of users.

This design follows the Service Oriented Architecture (SOA) model to design a highly scalable core system that will be extended with function-specific application modules. (By SOA, we mean the core system has an API that provides a breadth of services to application-layer modules – e.g., CreateRequisition() – rather than a publish/find/bind environment.) To maximize the flexibility of deployment options and meet the functional requirements and performance goals, the architecture is designed to operate in a single server environment, or scale up and take advantage of multiple physical or virtual servers. The architecture is also compatible with on-premise or off-premise hosting.

In this design, the OpenLMIS system is partitioned into the core system services (Core), and deployment-specific modules and/or applications. The Core is intrinsic to this system architecture, since it serves as the foundation for any specific implementation: it handles data storage, data validation, transaction processing, etc., while ensuring data integrity during all phases of operation. A key requirement is for the Core to be platform neutral - platform in this context being the server operating system, database and object relation mapper. As a result, abstraction layers have been specified that will handle the differences in syntax and function between platform components.

The deployment-specific modules and/or applications[1] deliver functionality to users and external systems, and in turn manage data through the Core APIs. The API abstracts the complexity of the database schema, and helps ensure that all data associated with any object (e.g., a requisition) are handled consistently and correctly by the application layer.

Users will work with the functionality presented by various modules through browser-based web forms, java-script applets on dumb phones, or purpose-built applications hosted on tablets or smartphone with local storage for off-line processing and upload.

Potential OpenLMIS Modules/Applications

- Stores management
- Receiving
- Forecasting
- Requisition
- Procurement
- Order processing
- Distribution
- Dispensing
- Storage capacity model
- Health facility definitions
- Medical item definitions
- Integration to external systems (e.g., warehouse management)
- Pre-configured and ad-hoc reporting

The Core will be made available to the logistics community as a free resource, though an online repository of open source logistics software. Generic application modules and purpose-built applications (e.g., Requisition Management) will be added to the software repository, and available as free resources; these generic modules would not be designed to serve any specific deployment of an LMIS. Implementation teams can adapt the generic modules and applications, as well as create additional deployment-specific modules and applications to meet their users' requirements.

OpenLMIS uses an open source licensing model based on the Eclipse Public License v1.0. This license allows electronic logistics management information systems (eLMIS) implementers to make use of the components they need in creating their specific implementation. The Reference Data Model and System APIs are also made available to implementers under the Eclipse Public License. Pursuant to the license terms, developers making enhancements to the Core must make their modifications publicly available for evaluation and use by other eLMIS implementers. Modules or applications (e.g., web applications, logistics-operational rules, reporting components) will tend to be created and customized on an implementation-specific basis. Such modules and applications would be welcomed as contributions to the code repository, since they will serve as examples for other implementers, even though they would not necessarily be part of the common Core.

By establishing a shared code base, OpenLMIS will (1) leverage the software community's development efforts, (2) make the common eLMIS core and generic applications freely available thereby reducing the cost and time to deploy any new eLMIS, and (3) allow sharing insights and expertise in supporting and enhancing any eLMIS based on a shared design. At the same time, OpenLMIS is not intended to be a specific product or turn-key solution for any eLMIS deployment.

In an area as important and complex as health logistics, any eLMIS must be built around a system architecture that is stable and flexible, and will scale with needs as they evolve over time. This architecture has been designed in accordance with best practices of enterprise systems as have evolved within the IT industry over the past 20 years. The architecture described here is flexible and extensible, thereby suited to the needs of a diverse user community.

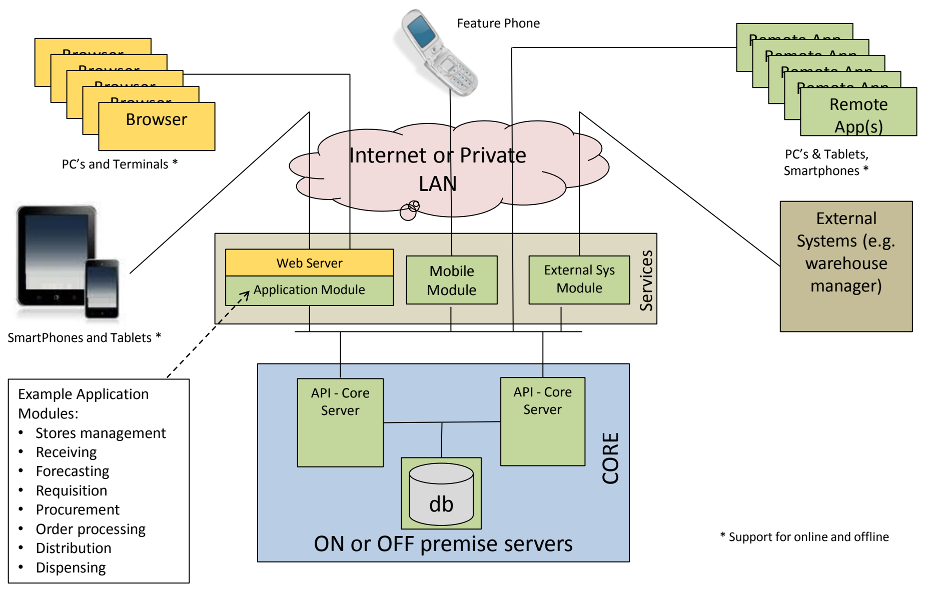
System Architecture Requirements

The typical deployment of OpenLMIS (diagramed below) consists of the Core operating in a server environment located in a local datacenter or in the cloud, a collection of functional modules (e.g.,

[1] See OpenLMIS Architecture diagram later in this document. "Modules" and "applications" implement various levels of functionality; as used in this document, "modules' operate within the server environment while "applications" operate on smart devices (e.g., smartphones, tablets) and interact directly with the Core APIs.

Requisitioning, Order Processing, etc.) running on the same server, or tiered server(s), and a webserver managing presentation and user interaction. Additional gateways support user access through feature phones, while purpose-built applications running on smart devices will connect directly to the Core APIs to support online and offline processing. Each method of interaction with the system makes use of the appropriate locally available network(s), e.g., LAN, WAN, cellular SMS, GPRS.

OpenLMIS Typical Deployment



Example Application Modules:

- Stores management
- Receiving
- Forecasting
- Requisition
- Procurement
- Order processing
- Distribution
- Dispensing

The primary requirements of the architecture include:

Platform neutral. The system Core is designed to be platform neutral, deployable in on-premise or cloud-based physical or virtual server(s). To meet user requirements for locally implemented configurations, the online demonstration deployments will be based on standard Windows and Linux configurations.

Stateless processing. Core services are accessed via REST style interface.

Modules/applications platform agnostic. Modules and applications (web-forms and clients) will be deployable on PC's and mobile devices (phones, tablets, etc.) and aware of connection quality.

Minimum browser requirements. Generic modules will be designed to be compatible with IE9 or later, and Firefox V10 or later. Other browsers (e.g., Chrome) will be evaluated for future support. Since browser-based applications for mobile devices will be specific to a device implementation, support for mobile device browsers will be on a deployment-specific basis.

Connection agnostic. The architecture is compatible with private and public networks that support connectivity between end-user devices and the respective system gateways.

Online and office support. End-user devices with appropriate data/form caching capabilities will allow intermittent connections between modules and browsers, and between the Core and client applications.

Batch processing. The architecture is designed to scale for high-volume transaction deployments – estimated 20,000 transactions in a batch needing to be processed overnight.

Concurrent users. Depending upon the number of supported users and the transaction volume, the system could be deployed on a single server, or distributed across a cluster of servers. The preliminary design goal is to be compatible with high capacity configurations (i.e., deployed on multiple servers), supporting access by up to 500 simultaneous users in mixed task mode (e.g., 150 concurrent data entry/update, 300 concurrent data browsing/reporting, 50 concurrent batch update users).

Data hygiene at point of entry. Module/application design should incorporate data validation at point of entry – including supporting offline support.

Security by role. Core system access is based on user and role with assigned levels of access for viewing, entry, editing, viewing/reporting and auditing.

Transaction models. Transaction models supported include: form-based data entry/editing, batch upload/download (e.g., via spreadsheet or flat file), pre-configured and ad-hoc reports, and real-time transaction processing from external systems.

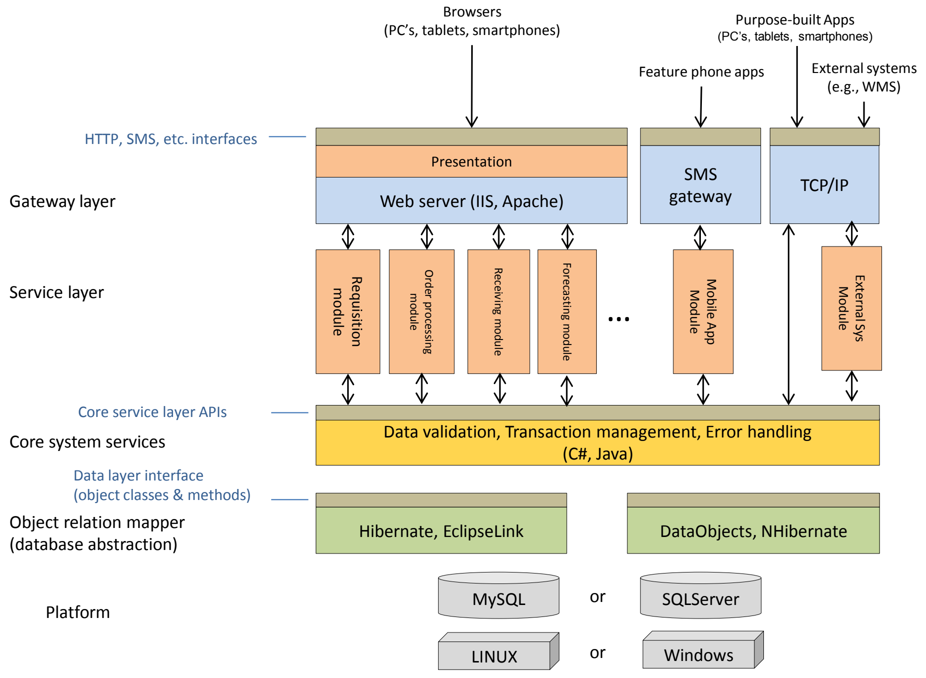
Logging. All transactions are logged (with Core function managed retention policy) for audit purposes.

Roll-back. The Core's API services support transaction roll-back upon failure to complete (state is not preserved across multiple web API calls).

Architecture Introduction

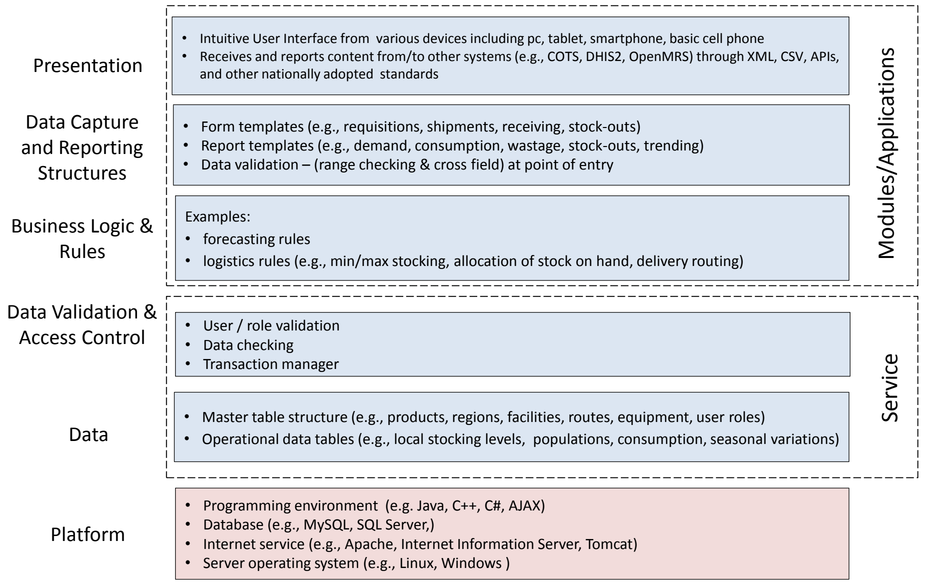
The architecture is a conventional enterprise-level, client-server design using web protocols for communications, and supporting both browser and dedicated client interaction. This architecture, commonly described as an SOA, is germane to many enterprise systems as well as many internet websites; additionally, it is the basis for the rapid expansion in tablet and smartphone applications in today's mobile consumer market. In this context, SOA also delivers an open software architecture that allows adding, upgrading and swapping components without affecting other components in the system. The architecture is based on a comprehensive set of APIs accessed through RESTful services. (The core system has an API that provides a breadth of services to application-layer modules – e.g., CreateRequisition() – rather than a publish/find/bind environment.).

OpenLMIS Architecture



The architecture is designed to be compatible with either open-source platforms (Linux, Java/Python, MySQL, Apache) or mainstream commercial platforms (Windows, SQL Server, .NET, IIS). While some elements of the Core software will necessarily be different due to platform variations (e.g., an object relation mapper), the Core will be based on a common code base as much as possible, to allow implementers maximum leeway in their individual deployments.

OpenLMIS Functions



Presentation
- Intuitive User Interface from various devices including pc, tablet, smartphone, basic cell phone
- Receives and reports content from/to other systems (e.g., COTS, DHIS2, OpenMRS) through XML, CSV, APIs, and other nationally adopted standards

Data Capture and Reporting Structures
- Form templates (e.g., requisitions, shipments, receiving, stock-outs)
- Report templates (e.g., demand, consumption, wastage, stock-outs, trending)
- Data validation – (range checking & cross field) at point of entry

Business Logic & Rules

Examples:
- forecasting rules
- logistics rules (e.g., min/max stocking, allocation of stock on hand, delivery routing)

Data Validation & Access Control
- User / role validation
- Data checking
- Transaction manager

Data
- Master table structure (e.g., products, regions, facilities, routes, equipment, user roles)
- Operational data tables (e.g., local stocking levels, populations, consumption, seasonal variations)

Platform
- Programming environment (e.g. Java, C++, C#, AJAX)
- Database (e.g., MySQL, SQL Server,)
- Internet service (e.g., Apache, Internet Information Server, Tomcat)
- Server operating system (e.g., Linux, Windows)

The OpenLMIS architecture comprises of four major components: the data model, the Core exposed through APIs, web modules and purpose-built applications enforcing business rules and providing specific functionality, and reporting/data-mining modules for reporting, analysis and decision support:

Data Model. This defines how all the operational data (e.g., master item lists), is structured and accessed, along with data validation tables and transaction tables used for maintaining the integrity of the system.

System APIs. The APIs provide programmatic ability to make changes to the database and triggers external events based on validation rules, and programmed criteria. The APIs enforce access control, data validation and transaction integrity. Administrator functions to maintain data validation tables and system parameters are handled through additional APIs.

Modules and applications (user interaction). Web-accessed modules and client applications on smart devices are the means by which end-users make use of all system functionality (e.g., store and retrieve data in the eLMIS based on their specific role in the organization).

Reporting applications. Modules and applications may have pre-configured reports and alerts and ad-hoc reporting including data-mining (search and aggregate) functions.

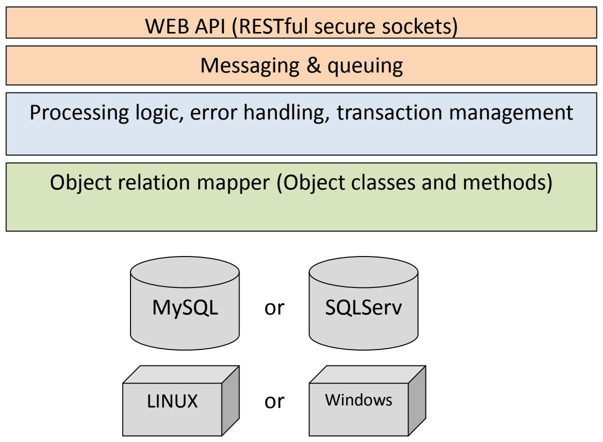
The Core APIs (and associated objects) will support direct interaction with functional modules and purpose-built applications. In this environment, browser-based web interaction is the easiest and most universal approach for data entry and user interaction. Purpose-built applications provide the capability for rich user experiences that can be tightly integrated with additional data sources (GPS, motion sensors, etc.) and be fine-tuned for devices with limited screen and keyboard capability.

We recognize that older browsers (e.g., IE6 and Firefox v3) are still prevalent in public health systems in many low-income countries. However the emerging HTML5 standard will provide a significant improvement in browser capabilities, and thus enhance the functionality that can be presented to users. Accordingly, the functionality and features that will be presented in the generic modules included with OpenLMIS will designed for compatibility with IE9 and Firefox 10, or newer. As browsers with HTML5 capabilities for remote data caching and offline execution become commonplace in the user community, the need for purpose built clients will be reduced.

OpenLMIS Core (Services)

The heart of the OpenLMIS design is the set of services, exposed through APIs that make available the Core's complete functionality. The Core services are also designed to ensure the atomicity of transactions, and the integrity of the data.

OpenLMIS Core Architecture


WEB API (RESTful secure sockets)
Messaging & queuing
Processing logic, error handling, transaction management
Object relation mapper (Object classes and methods)
MySQL
or
SQLServ
LINUX
or
Windows

The Core elements are:

Platform and database. The operating system is at the base of the LMIS. A relational database holds all operational data (e.g., master item list, master facility list), plus data validation tables (dictionaries and range checking) and transaction tables for related to all logistics data. A primary design goal is that no module or application is allowed to read or write directly to the database, but always uses the APIs to access any data.

Database abstraction layer. To provide a common development environment for the object, messaging and API layer, a platform/data-base abstraction layer will provide for a common interface to the higher processing components.

Object classes and methods. The processing functions will be exposed to the modules/applications via the API layer. These will be grouped by logical functions into Objects where Methods and Properties will perform the following functions:

Processing logic. Based on the specific function to be performed, calculations and data-updates will occur. Across all the functions of the Objects and exposed APIs, error return codes will gracefully allow the module/application to correct problems.

Data validation. This part of the Core ensures, for example, that no out-of-range data gets into the database. Data validation range information is primarily found in data validation tables that are established and maintained thru system administrator module/applications.

Transaction consistency. Many of the data updates occurring in the database will update multiple tables and logs. If a problem occurs during the update cycle the transaction monitor will return all data tables to the pre-event condition. If the transaction monitor detects a problem, the modules/applications will then be notified of the error for appropriate action.

Access control. Based on the user and role, this service layer ensures a user has the correct privileges to make changes to the database. If the access control service detects a problem, the modules/application will then be notified of the error for appropriate action.

Messaging & queuing. To provide for acceptance of large volumes of transactions, and to service high-priority requests, this messaging layer will prioritize transaction processing and store low-priority transactions for later processing

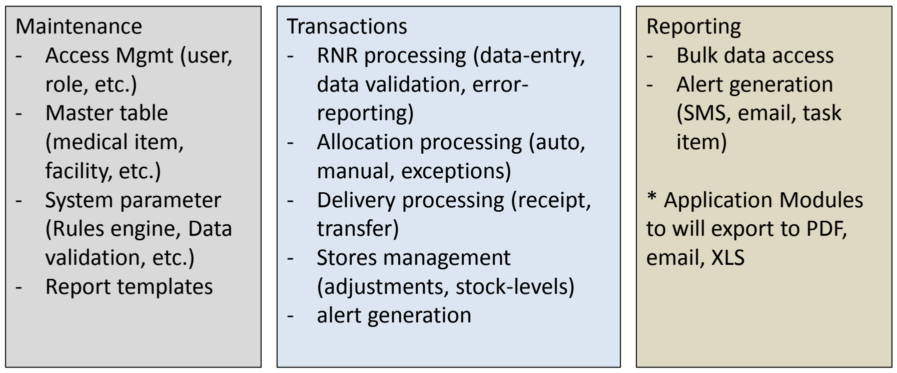
APIs. Based on use-cases for common administrative and eLMIS operations the APIs will expose the database via Objects and Methods for modules and applications

OpenLMIS APIs

The APIs exposed to the modules and applications will be based on the RESTful model of web applications using common XML standards. Modules and applications may elect to operate in secure-encrypted mode using HTTPS or standard internet functions using HTTP (based on business requirements, and available bandwidth and processing power).

OpenLMIS API Structure

Three primary Classes exposed via RESTful secure sockets



Maintenance
- Access Mgmt (user, role, etc.)
- Master table (medical item, facility, etc.)
- System parameter (Rules engine, Data validation, etc.)
- Report templates

Transactions
- RNR processing (data-entry, data validation, error-reporting)
- Allocation processing (auto, manual, exceptions)
- Delivery processing (receipt, transfer)
- Stores management (adjustments, stock-levels)
- alert generation

Reporting
- Bulk data access
- Alert generation (SMS, email, task item)

* Application Modules to will export to PDF, email, XLS

The API structure will be designed around common transaction-based activities intrinsic to logistics operations (e.g., CreateRequisition(), ChangeRequisitionStatus(), AdjustRequisitionLineItemQuantity()).

The object classes and associated methods will be divided into:

Maintenance. These include the set of functions associated with managing the Core and system environment (e.g., Master Item tables, data validation tables, etc) and arules engines (for transaction prioritization and error handling).

Transactions. These include the set of functions associated with handling the logistics-operations updates to the databaseRelated APIs will support posting requests for "off-hour" time-noncritical transactions.

Reporting. These include functions associated with extracting reports, and generating use and system alerts based on transaction data.

OpenLMIS Modules and Applications

By design, the Core APIs abstract the database into objects that are pertinent to logistics operations (e.g., Requisition, Shipment, Receival). For architecture purposes, the OpenLMIS system architecture separates client services that utilize the Core APIs into modules and applications. Modules are server-based components that provide transaction entry and business logic that requires browser-based presentation capability on the client device. Applications are self-contained components that are installed and run on smart devices. Such applications follow the popular model in use in the tablet and smartphone market.

Modules and applications will include the logic that is unique to specific implementation, to implement the business rules associated with the specific eLMIS deployment. The Core has general data validation and transaction integrity monitoring but does not have (and should not have) control of an organization’s business logic. For example, the rules association with stock replenishment amounts in a clinic would reside in a functional module or application that performs calculations based on the organization’s forecasting algorithms, and while using the Core APIs r retrieve consumption data, and issues the appropriate replenishment orders Instructions via the Core APIs for further processing.

The OpenLMIS initiative will develop several generic modules and applications for common use-cases and these will be included the OpenLMIS code repository. These generic modules and applications will be available for customization and re-use, and also will include:

- Application design guides – based on shared learning during the development process;
- Reference documentation for APIs, data-structures, forms management, reporting, etc.; and
- Working source code for business logic implementation that uses the API–based, data-validation environment.