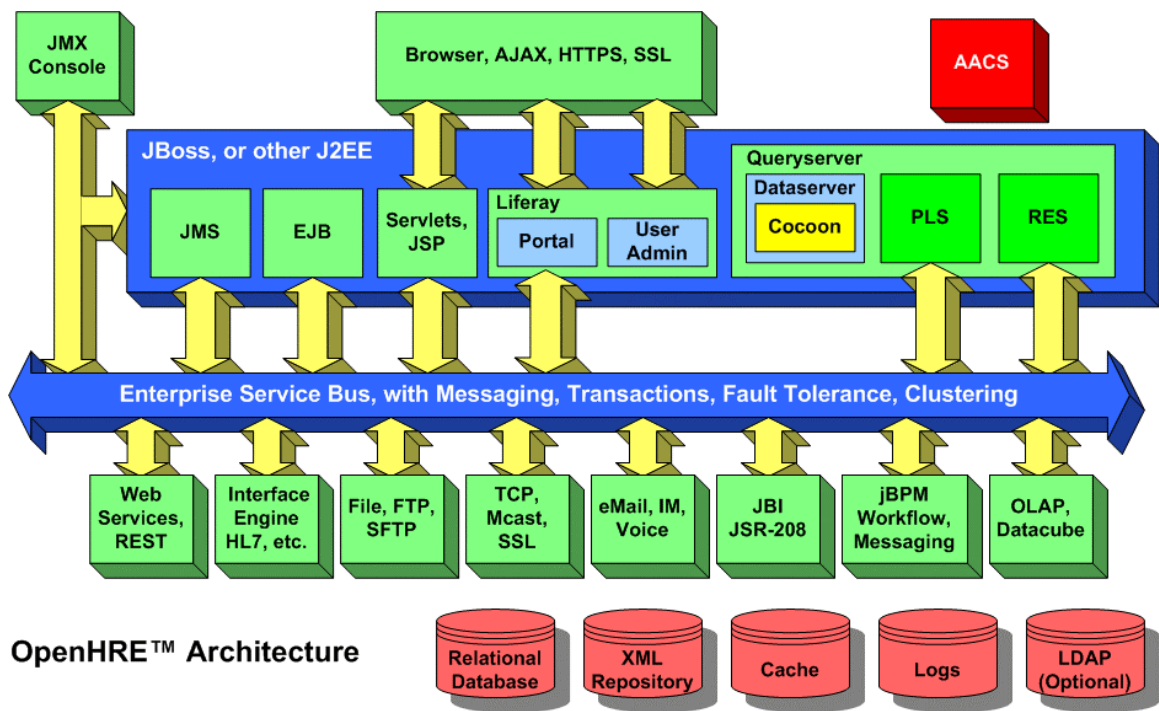


OpenHRE™ Architecture

4/10/2009

OpenHRE™ is the world's first open source toolkit that provides a standard and secure means to exchange data between existing health records systems. With OpenHRE™, health professionals can locate, access, and review clinical data related to a specific health consumer, or acquire de-identified clinical data for population studies.

OpenHRE™ is Java-based open source software. It typically runs on an open-source J2EE platform, such as JBoss or Tomcat, but it can also run on proprietary J2EE platforms such as WebSphere or WebLogic.



Most of the OpenHRE™ user interface is browser-based, using the latest Web 2.0 techniques, including AJAX. Because of the sensitive nature of health information, transmissions to and from the browser are always encrypted using SSL.

OpenHRE™ is compatible with many relational database engines. Most often used with the open source MySQL database, OpenHRE™ is also compatible with databases that have a JDBC driver for Java, including DB2 and SQL Server, for example. OpenHRE™ can also interface to XML repositories, such as the open source eXist engine, and DB2 Viper. Deployments of OpenHRE™ can include both relational databases and XML repositories, or XML documents can be stored as binary objects within relational databases.

Like most open source software, OpenHRE™ is an extension of many other open source software products.

Authentication and Access Control (AACS)

One key component is the Authentication and Access Control (AACS) component, based on an extension of the Liferay Portal (<http://www.liferay.com/>). This provides an enterprise-portal front end for OpenHRE™, including sophisticated user management, with up to three levels of administration, and optional synchronization to LDAP-based enterprise directories.

Authentication is via user id and password, where password complexity and expiration are managed by the system. Repeated failed attempts to logon will result in account lock-out. Idle sessions are automatically logged out. Automated, secure, password reset is provided. Authorization is controlled here for all user-accessible system resources, down to the level of individual portlets and web pages.

The portal features of Liferay are optionally available for use, including moderated content management, email, support for wikis and blogs, and many other pre-built portlet components.

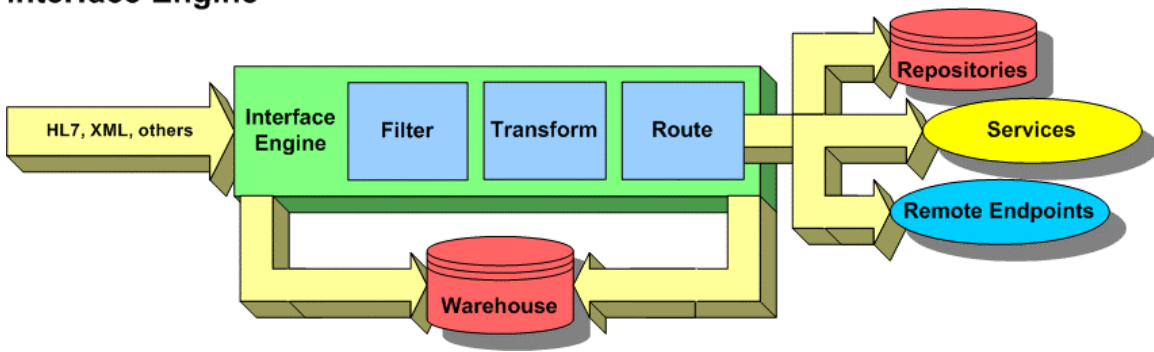
QueryServer

Another key component to OpenHRE™ is the QueryServer. QueryServer is an extension of the UCLA DataServer (<http://www.mii.ucla.edu/index.php/MainSite:DataServerHome>), which was built to handle multiple high-speed parallel queries and transformations of health information, along with a scalable caching system. DataServer itself is based on the Apache Cocoon project.

Browsersoft has extended DataServer to include configurable access to relational databases, which are one of the many data sources available to the QueryServer.

There are many other components to the OpenHRE™ architecture. These are more easily understood by examining the various subsystems and functional blocks.

Interface Engine



□ The OpenHRE™ Interface Engine is an extension of the Mergence Project (<http://www.hmsinc.com/mergence>). Mergence is a high performance, open source HL7 integration engine and application platform. It was developed to fill the need for an open source, fully transactional healthcare integration system capable of handling millions of messages per day, and thousands of concurrent LLP connections. Mergence is based on Apache ServiceMix (<http://www.servicemix.org/>), a lightweight enterprise service bus which conforms to the JBI standard.

Whereas QueryServer is used in applications for pulling data from external sources, the OpenHRE™ Interface Engine is used when data is pushed to OpenHRE™ from external sources. The supported push mechanisms include LLP, JMS, SOAP, File (network and local), FTP, and SFTP.

Mergence includes default message filters, as well as a scripting language for writing custom filters. Mergence also supports filters written in Java if needed for more complex filtering options.

Messages that are accepted by the filter mechanism can be transformed as needed by downstream systems. Transformers are used to extract incoming data and change and format it as needed. Like filters, transformers can be selected from a list, or can be written in several scripting languages through the web interface. Complex transformations can be implemented as a Java class. Incoming messages are first encoded into XML, from which data can be extracted and converted to a Java object, a SQL statement, an XML file, or a variety of other types. Note that both the OpenHRE™ QueryServer and Interface Engine components make extensive use of XSL XML-based transformation technologies, and if needed the same XSL transformations can be re-used by both system components.

Transformed messages can be routed to one, or more, of several configured endpoints. These endpoints can be other services within OpenHRE™, or they can be to external destinations, such as Public Health agencies. Message content and context can both be used to determine the destination routes and formats. Messages can be directed into relational databases as well as XML repositories.

Routing decisions are performed by Apache Camel, which is a powerful rules based routing and mediation engine. Apache ActiveMQ, the premier open source implementation of JMS, handles delivery and message caching.

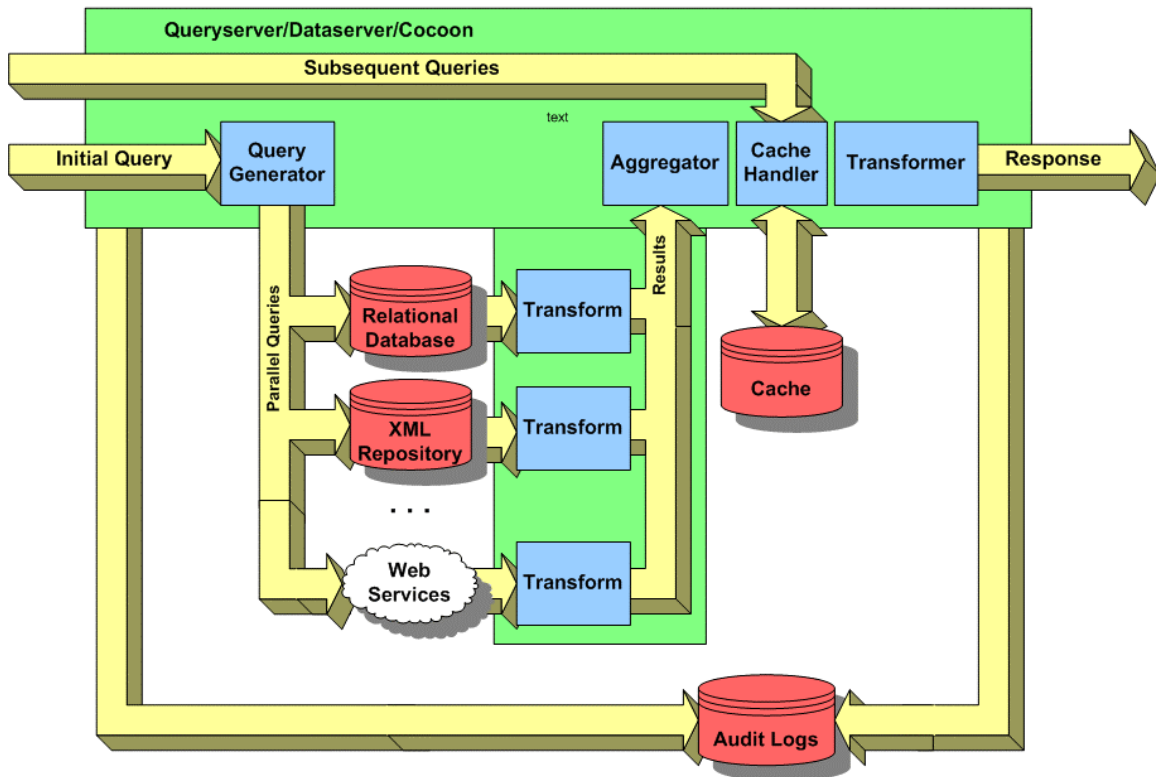
For Logging, Auditing, and Archiving functionality, incoming and outgoing messages are encrypted and stored to an internal message warehouse.

Records Exchange Service (RES)

The OpenHRE™ Records Exchange Service (RES) contains some of the same functional blocks as in the OpenHRE™ Interface Engine. In the Interface Engine these were implemented within the context of a data “push”. In the Records Exchange Service these same functions are implemented within the context of a data “pull”. These functions are implemented and configured using XML-based technologies such as XSL and SAX.

These functions are included inline, rather than factored out into separate services, in order to eliminate extra context switching and increase processing efficiency.

OpenHRE™ Records Exchange Service Processing



The OpenHRE™ Records Exchange Service (RES) is implemented by the QueryServer. Processing is initiated by an external request. This can come from a scheduled timer process, from an internal OpenHRE™ subsystem, or from an outside source. QueryServer follows the HTTPS protocol, so it can handle processing requested by web browsers using HTML, other services using SOAP and XML, AJAX calls requesting XML, HTML, or JSON, and requests for other formats including PDF and image formats.

Ultimately many data sources may need to be queried to satisfy the initial request. It is often the case that it is first necessary to determine which data sources need to be queried. This is the job of the Locator service. Information on the subject of the query, for example patient demographics, is taken from the input request and passed to the Locator service. The Locator service uses one or more registries, “fuzzy” logic, and probabilistic scoring to determine the locations of data regarding the subject.

The Locator is described in greater detail in the following section. It is interesting to note that the Locator follows the Request-Response pattern, it interrogates one or more registries and or data services, it does pre- and post-processing on the data, and it returns its results in a specified format.

Once the locations of the data are known, the RES formats queries to each of the data sources. The format of each query is specific to the original request and the requirements of the data source. The data sources may be relational databases, XML repositories, and other data services, including external services such as NHIN services being hosted by other RHIOs or Health Information Service Providers (HISPs).

The formatted queries are sent out, in parallel, to the located data sources. As the results are returned they may optionally be transformed into a common format using QueryServer transformation services. The standard mechanism for configuring transformations within QueryServer is the XSL stylesheet.

The Aggregator assembles the transformed result streams into a single stream.

RES queries can be quite expensive, involving queries to multiple sources. Returned data can include large data elements, such as complete medical histories and graphics. The ultimate presentation to the user may involve drill-down, with details being displayed on an as-needed basis. Rather than re-issuing the query to the original data sources, the returned data is stored in a local cache repository for an hour or so, to enable the user to access portions of the data. After the initial query, subsequent queries are directed directly at the cache repository.

Instead of using the limited resources of the HTTPSession for the cache, scalable relational or XML repositories are used. This allows the QueryServer itself to be stateless and scalable across many instances.

A final transform may be applied before the response is delivered. Potential output formats include HTML, XML, JSON, PDF, plain text, and images. Transformations may even include side effects such as the updating of relational databases or XML repositories.

As with the OpenHRE™ Interface Engine, all inputs and outputs are compressed, encrypted, and warehoused for logging and audit purposes.

Patient Locator Service (PLS)

The OpenHRE™ Patient Locator Service would be better named the “Patient Locator Service”. It maintains a registry of various demographics that patients have used to identify themselves to care providers, and a set of algorithms used to match these various demographics in order to locate all the care providers of a patient, regardless of variations in identifying demographics.

The OpenHRE™ Patient Locator Service is the central point from which to find the locations of a patient’s health data within a regional or other sub-national, healthcare network. The PLS is also capable of connecting to remote healthcare networks to determine if their affiliated healthcare sites have relevant health data for a patient, and retrieving that location information. The health data itself is not stored in the PLS, only Locators for the data. A Locator often consists of the name of an institution, an identifier

for a particular system within that institution, and a Medical Record Number within that system.

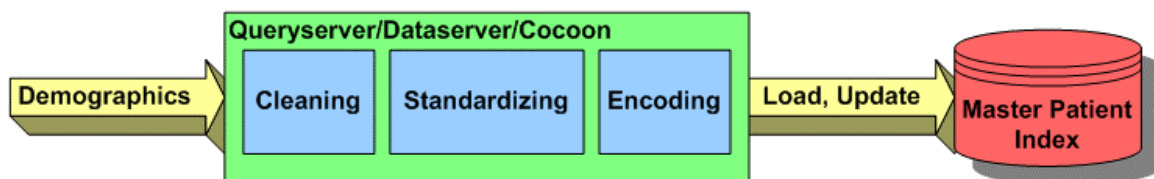
The PLS does house basic demographic data for each patient, and it is this data that is used as a basis for patient lookup. “Fuzzy” matching technology is supplied to allow patient data to be located even if the demographic source data is imprecise.

A key feature of the PLS is the ability to match patient data from multiple systems, even in the face of irregularities in this data. This is accomplished through a combination of pre-processing and on-the-fly processing of the demographic data. Tools are provided to aid the implementer through all the steps in the patient matching process. In all cases the use of these tools is specified through a process of configuration. Additionally, the PLS may be configured as a Master Patient Index (MPI) at an organizational level or for the overall exchange.

We need to capitalize on the speed with which relational databases can find exact matches. On the other hand exact matches are not sufficient to the task, as patient identities are often not entered in exactly the same way by every provider. In order to capitalize on the speed of the database, and still implement probabilistic “fuzzy” matching, we break the problem up:

- Data Preparation and Loading
 - Data Cleaning
 - Standardization
 - Transformation
 - Data Load
- Query and Post-Processing
 - Blocking
 - Comparison
 - Probabilistic Weighting
 - Deterministic Algorithms
 - Results Classification

OpenHRE™ Patient Locator Service Preparation

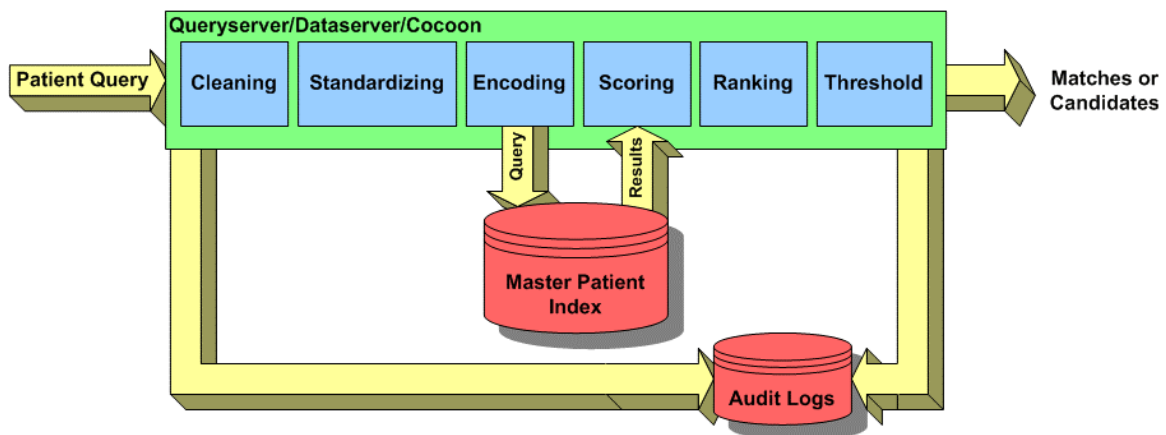


Data Cleaning is the process of eliminating common errors in the demographics data, either in the initial data load or during routine maintenance of the PLS data. Examples of cleaning include removing extraneous spaces and punctuation, applying proper capitalization, and removing obviously bad data values. This process can be accompanied by a report back to the originating institution to facilitate their data cleaning. In the OpenHRE™ PLS, initial data load cleaning is often accomplished through the use of an XSL transformation.

Standardization includes the ability to use standard values for data where variations exist. Examples include substituting “Street” for “St” in addresses, and adding to a record a standard version of a name (e.g. “Charles”) to be used during matching instead of a supplied nickname (e.g. “Chuck”). As before, in the OpenHRE™ PLS, initial data load standardization is often accomplished through the use of an XSL transformation. Transforming nicknames into standard names is accomplished through a configurable Java class.

Transformations and encoding can be used to aid the fuzzy matching process. To work around spelling errors in names, for example, a technique often used is to match on phonetic versions of names, using phonetic transformations such as Soundex, Metaphone, Double Metaphone, and NYSIIS. To aid matching performance it is often useful to pre-record the encoded versions of names. Exact matches on the encoded values are equivalent to “fuzzy” matched on the original values, but can be achieved with greater speed.

OpenHRE™ Patient Locator Service Query



When matching a single patient’s data against a large population of patients stored in the PLS, it is expedient to optimize the process and not spend processing power on checking for matches between records that have little chance of matching. The problem becomes even more acute when cross-checking for matching between the PLS records themselves, which is an $N \times (N-1) / 2$ problem. The performance challenge is helped by filtering or “Blocking” records from the set to be matched. For example, assume that data for the Birth Month of patients is highly reliable. If we block out all record pairs, except those where the Birth Month matches, we stand a good chance of finding all of the valid matches. Blocking is configurable in OpenHRE™. Blocking on an encoded value combines the advantages of block with the advantages of high-speed queries.

When performing an PLS query, the incoming query parameters must undergo the same cleaning, standardization, and encoding process that the loaded data underwent. Matching, scoring, ranking, and thresholding can then be performed completely as part of the database query, or a list of high-probability candidates can be returned for further analysis.

There are many forms of Comparisons that can be specified for the matching process. For example, if two strings are the same, except for a simple transposition of characters, there is an OpenHRE™ Comparator that will return the fact that the strings “almost” match, rather than simply stating that the strings do not match at all. Even though an “almost” match should not count as much as an exact match, enough “almost” matches between two different patient records may indicate that the records are indeed for the same patient. OpenHRE™ contains a number of different Comparators that can be chosen through configuration.

Probabilistic Weights allow for different comparisons to have different weights, according to their ability to discriminate between identification information for different people. Weights may be assigned according to the fields being compared. For example a match on last name may “count” more than a match on first name. Weights may also be assigned according to the frequency of the field values themselves. For example a match on a last name of “Symington” should count more than a match on a last name of “Smith”. OpenHRE™ can be configured with both of these types of weights.

Equally important to the ability to use various weights, is the ability to assign the correct numerical values to the weights. The correct weights to use are not only a function of the transformations and comparisons being used, they are also a function of the patient data population itself. OpenHRE™ contains tools that can be used to estimate the predictive power of the current weights settings, as well as estimating candidate values of the weights themselves, based on the actual patient population data at hand.

An alternative to weighted “Probabilistic” matching is the use of Deterministic algorithms. For example, if your data allows the use of Social Security Numbers, and you have an exact match on First Name, Last Name, and Social Security Number, it is a good bet to declare that a match has been made. OpenHRE™ has the ability to be configured to use deterministic rules on some or all of the match fields. These deterministic rules can be used in lieu of, or in combination with, probabilistic weighting.

The result of the matching process is an overall score of the probability that a match has been obtained. Two threshold values can be set: one that determines the score beyond which we are fairly certain that we have a match, and one that below which we are fairly certain that there is no match. In the “gap” are “potential” matches, which can be reported for further review. OpenHRE™ contains tools by which matches can be manually made or broken, as overrides to the automated algorithms.

Deploying OpenHRE™ includes analyzing the demographics of the patient population in order to optimally specify the matching configuration items detailed above. It may be necessary to iterate this process several times in order to discover the optimal settings to minimize false positive and false negative matches. The process is aided if a representative set of data is available that has the true matches pre-marked, but this is not often available.

PLS Hierarchy

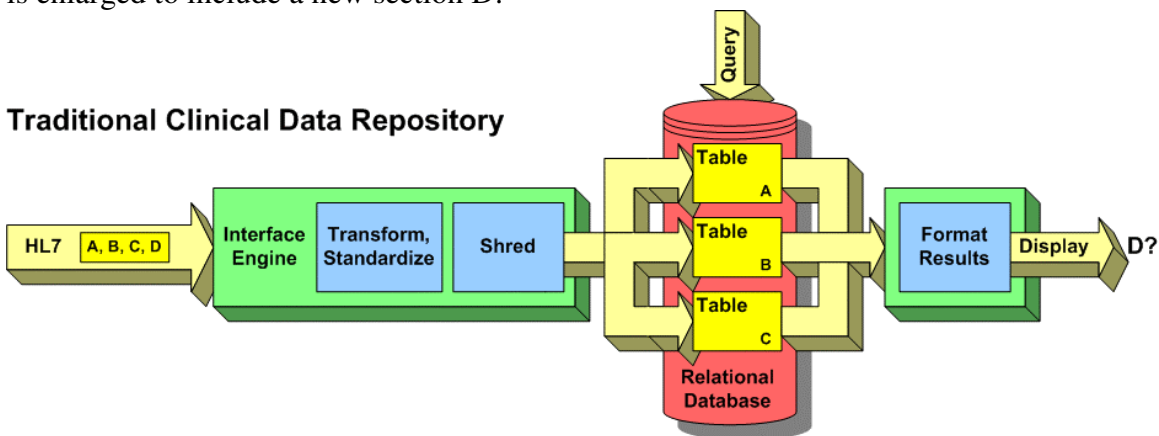
There are many possible configurations for installing the PLS within a federation of Medical Institutions. The PLS can be used to index patient data, not only at the regional or sub-national level, but also within institutions and between affiliates. PLS servers can optionally be arranged in a hierarchical pattern, with each level servicing requests for matches at its level and lower levels.

Data Model

OpenHRE™ uses a document-based data model, with just-in-time transformation to required formats for purposes such as data transfer, display, analysis, and reporting.

Clinical data is highly structured. Traditional schemes for the storage of clinical data involve “shredding” the document structure into a relational model, involving separate tables for each level of hierarchy. This technique requires a rigid document schema that is not susceptible to change.

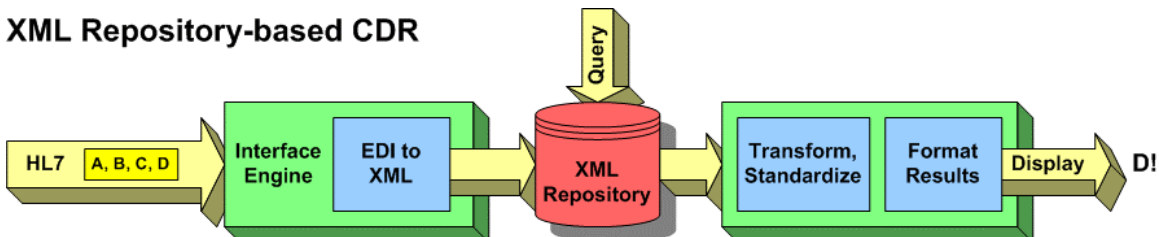
As an example, take the case of an HL7 document containing three levels of hierarchy: A, B, and C. For simplicity we will assume that this hierarchy can be stored in three relational tables: A, B, and C. Our system will require re-design if the document schema is enlarged to include a new section D:



The problem is compounded if we are required to save documents of varying types, formats, and schema revisions.

Our system becomes much more robust if we simply store the documents as we receive them, in a document-oriented repository. In such a system we can transform the data in many different ways, not all of which are known when the data is first collected.

XML Repository-based CDR



Health Record Banks

The Health Record Bank is an innovative concept that combines the speed and simplicity of a Central Repository with the flexibility of a federated system. It is similar to a Central Repository in that only one Health Record Bank needs to be accessed to retrieve all Electronic Health Records for a single patient. On the other hand the entire system is federated in that any consumer of Electronic Health Records needs to be able to retrieve records from a number of Health Record Banks.

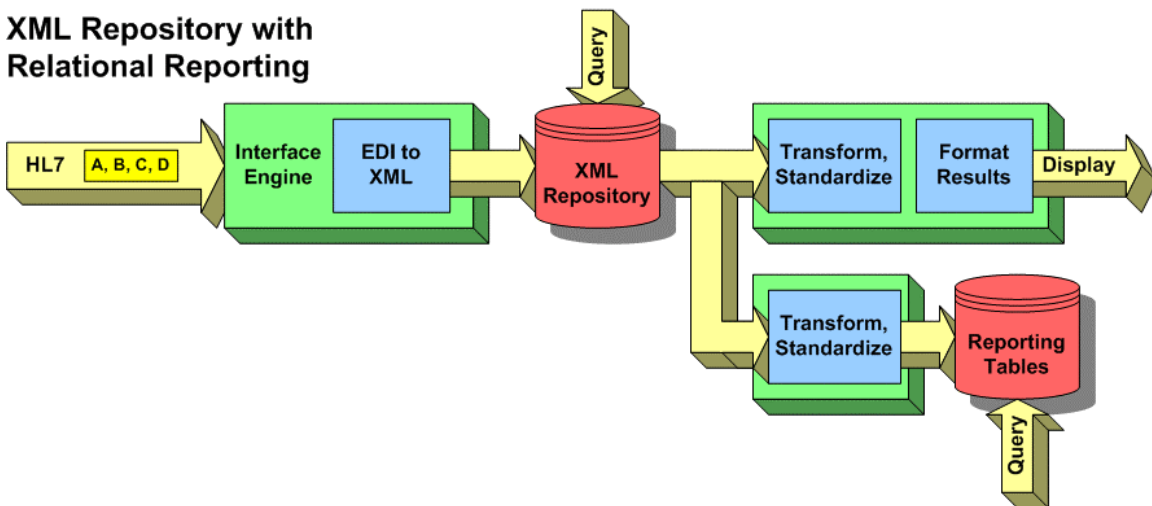
Browsersoft's OpenHRE™ Toolkit can be configured to work with a Central Repository, with a federation of health record sources, or with a combination of these. In this way OpenHRE™ is uniquely positioned to work with a federation of Health Record Banks. Browsersoft's experience with developing and installing centralized and federated OpenHRE™ instances for HIEs across the nation contributes unique skills to the organizations pursuing a health record bank initiative.

The OpenHRE™ Patient Locator Service already has the ability to manage patient identities in a potentially federated environment, and it could be used to implement the Account Locator component of a Health Record Bank.

Reporting Services

Relational databases, given their tabular nature, are very useful in preparing statistical reports, and there are many products available for producing reports from relational databases. For this purpose it is often useful to set up reporting tables to gather statistics from the incoming data.

XML Repository with Relational Reporting

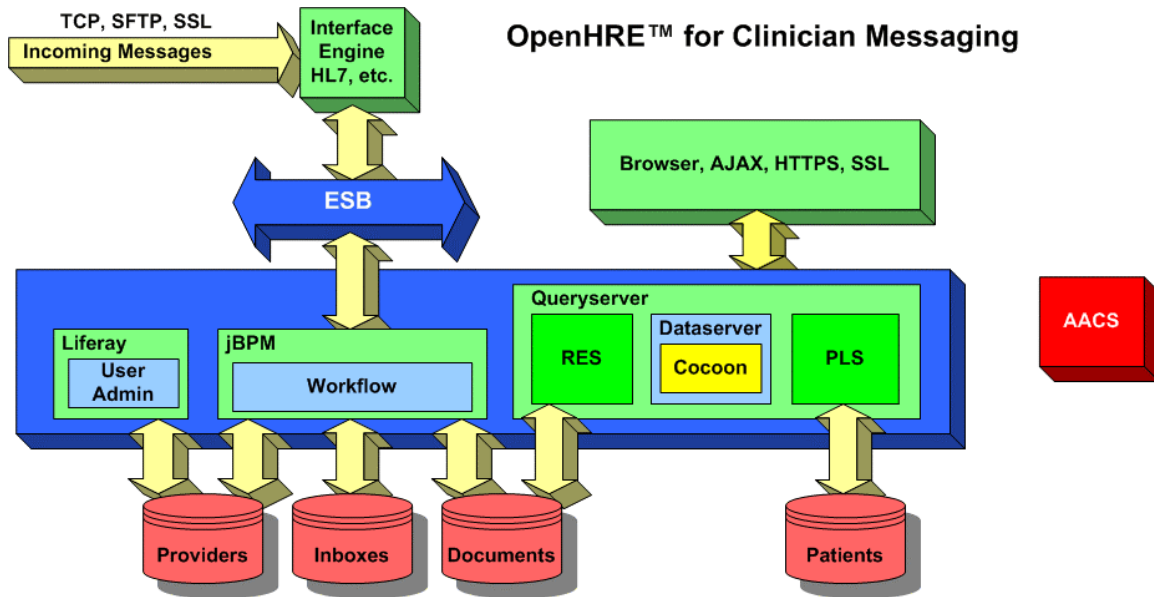


It may seem as if we have made a full-circle since we are now introducing relational tables back into our system, from which we just eliminated them! The difference is that these tables are structured to aid in the reporting of data, rather than the storage of varied data. Often the transformation from the stored data to the data statistics is quite complex, and we can absorb that complexity into the loading of the reporting data, relieving the need for complex processing when the reports are produced.

A good example of transforming stored data into data geared towards reporting is creating a relational data cube for pivot-table “slice-and-dice” tabulation of data measures along multiple dimensions. OpenHRE™ includes the open-source JPivot and Mondrian tools for providing such analysis over a web interface.

Clinician Messaging

OpenHRE™ Clinician Messaging is a general-purpose workflow-based tool used for the communication and processing of clinical data.



The Clinician Messaging process starts with messages, often Lab results, being received by the OpenHRE™ Interface Engine. In this case the incoming message is stored in a Documents repository, and an index to the message, along with the important attributes of the message is forwarded to the jBPM (<http://www.jboss.com/products/jbpm>) Workflow Engine embedded in OpenHRE™.

The Workflow Engine is configured with easily-customized business rules for processing and routing the messages. The attributes of each message include such items as message type and priority, patient name and MRN, origination date and time, and the name of the ordering physician. The business rules are applied to the message attributes to determine the initial disposition of each message. For example, normal results can be routed to a nurse, while abnormal results can be also copied directly to the patient’s physician, perhaps via their pager.

The Workflow Engine maintains one or more statuses for each message. For example, a message can be “in” a physician’s in-box for review, but at the same time can be listed on a “to-do” list for a nurse. A business rule can be added that will forward the “to-do” message to a head nurse if it is not acted on within a configured time-frame.

At minimum, a pointer to each message is stored with an “in-archive” status, so that it can be retrieved and analyzed at a later date.

A graphical tool is available to capture the workflows and configure their business rules. Clinician Messaging processing and routing rules can be configured to be active only for a limited timeframe, or they can otherwise be activated and de-activated as needed. For example, special routing rules can be in effect on weekends, or when a clinician is on vacation.

A Providers Locator Service, similar to the PLS, is included to allow “fuzzy” lookups of health providers. Providers may have email addresses or fax numbers recorded, or they may have “in-boxes” configured within the Clinician Messaging system.

Clinicians may display their incoming messages using the same RES viewers available for Records Exchange. Notes may be added and the messages may be securely forwarded to other destinations, or routed through other workflows.

Note that the actual message documents remain in their repository, and only copies of the document indexes and message attributes are routed through the system.