OPEN IMMUNIZE

Software Requirements & Design Specification

Abstract

Open Immunize is a community software project intended to provide a generic platform upon which jurisdictions can build immunization information systems.
The right thing, done right.

Justin Fyfe, Duane Bender, et al.
justin.fyfe1@mohawkcollege.ca, duane.bender@mohawkcollege.ca

1. Document Information

1.1. Revision History

Name	Date	Reason	Version
Justin Fyfe & Duane Bender	Dec 21, 2015	Initial Version	0.1
Justin Fyfe	February 1, 2016	Includes IMSI, changes to data model, and additional detail on the software interfaces from architectural spikes.	0.2
Justin Fyfe	April 28, 2016	Includes numerous implementation changes for the AMI, IMSI, and RISI.	0.3

1.2. Related Documents

Related Document	Relevance

2. Contents

3. Introduction

3.1. Purpose

The Open Immunization (OpenIZ.org) project seeks to implement a generic platform for deploying immunization information systems within countries, states or provinces around the world. OpenIZ uses an extensible, open architecture which allows for the addition of features such as materials management, analytics, authentication, outbreak management, internet of things, reporting & national data submissions, and much more.

Through this design and the implementation of plugins, it is envisioned that countries can select a package of features which work to achieve an appropriate solution for their environment. For example, a country may select a custom immunization forecasting logic module with a stock management module to support the query of immunizations and stock management capability.

3.2. Project Scope

The scope of this project is to design and implement core functionality to support immunizations, reports and interoperability of the OpenIZ core, a mobile application, web application, and form scanning interface. The scope of the project has several facets:

1. A core services layer upon which the plugin architecture is based. This core service provides an API for access (has no end-user interface) and is used only for storing and retrieving immunization data, stock data, etc.
2. A reference mobile application which can query and store immunization data offline, synchronizing the data when an internet connection is available.
3. A web interface which can be used by physicians in the field to register that immunizations were performed, perform stock operations, etc.
4. An administrative interface which can be used by system administrators to setup new types of immunizations, deploy forecasting logic, perform stock, etc.
5. A reporting interface which will allow users of particular user groups to execute and download reports of either an administrative or informative nature.

Included in the scope will be the interoperability of the solution with a health information exchange. Interoperability scope includes:

1. Communication of patient information to a centralized patient identity registry using IHE PIX
2. Communication of immunization information to a centralized document repository using IHE XDS And IHE PCC IC profiles (HL7 CDA)
3. Communication of provider information using the IHE CSD profile.
4. Communication of stock information to a centralized GS1 service using ASN.3 XML encoding of BMD1 message.

3.3. About the Client

Open Immunize is a community initiative that will, potentially, produce a multitude of clients. The basis for the requirements used for the design of OPENIZ are a collection of those specified as part of projects that have been conducted around the world. This section will describe the general characteristics of such an environment.

The expected clients of this software range from low-to-medium income countries (LMIC) to individual states and provinces in developed countries. This broad base poses a variety of unique challenges. One of the primary challenges of public health professionals working in the field is that of reliable network connectivity and power. The solution shall take into account that feature-rich electronic devices may not be a viable solution and paper based or SMS based interfaces may be used as input into the system. This is true of LMIC but is also true of rural and indigenous regions in developed countries as well.

Furthermore, it is expected that the solution should be able to run on low powered, low cost hardware and server infrastructure. Great care shall be taken during the development and design stages of this project to ensure that optimal performance can be achieved on relatively low powered, low cost machines to achieve the widest possible adoption.

There can also be a severe shortage of "receptor capacity" or a capability on the part of implementing countries to deploy, manage and support these systems. Great care shall be taken in the documentation of the infrastructure, plugin architecture and installation procedures for the service. Where possible, it should be assumed that working knowledge of the underlying technical details is scarce. Multiple deployment options are also supported, including cloud options for greatest reach and custom local deployments where required to meet local legislative or other requirements.

3.4. Team Members

Being a community project, we seek to engage general partners where possible online. Upon commencement the team shall consist of the following members:

Table 1 - Team Members

Name	Role	Organization
Justin Fyfe	Architect	Mohawk College
Duane Bender	Designer	Mohawk College

4. Overall Description

This section provides a high level description of the system.

4.1. Alternative Products

Generic Immunization Information System ("GIIS") is an Albanian product developed by AIRIS solutions based upon a software solution designed for Albania. The GIIS product was deployed in Tanzania as part of the BID initiative funded by the Gates Foundation. The limitations of its architecture were quickly discovered. Key issues included:

- Performance issues with the Tanzania GIIS mobile application,
- Significant security and authorization issues with the GIIS REST services implementation,
- Lack of adherence to best practices in terms of REST interface design and SOA principles,
- Lack of documentation,
- Lack of extensibility and configuration points within the software to support incremental changes,
- Lack of normalization and coherence in the GIIS data-model,
- Lack of unit and integration testing of the GIIS solution.

The OPENIZ project will leverage the learnings and requirements from the Tanzania BID project, but will begin a new code base.

4.2. Project Principles

At a high level, this project seeks to:

- Provide developer extensibility and configuration points in all aspects of the service core,
- Provide extensive documentation for developers and users,
- Provide a disciplined approach for quality assurance, including the requirement of unit testing services on all modules in the OPENIZ project,
- Promote of code-reuse and standards wherever possible as integration points,
- Provide heavily normalized data model where journaling is a first class citizen and not an afterthought,
- Apply Security by design and privacy by design principles.

4.3. Project Features / Deliverables

As stated prior, the OPENIZ project seeks to provide a series of highly customizable components into a series of deliverables that will be used as the basis for implementation. Envisioned as deliverables are:

1. **Immunization Backbone:** An extensible software solution that contains the majority of "online" logic required to perform immunizations, track and merge events from remote sites, perform stock management functions, etc.
2. **User Portal:** A website that can be used by immunization officers in the field to query, record and maintain an immunization record for patients as well as perform stock management duties if required.
3. **Administrative Portal:** A website that can be used by administrators to maintain the backbone configuration including customization of reports, stock items, antigens, etc.

4. **Mobile Clinic App:** A mobile application that can operate offline for long periods and be used to collect immunization data within a clinic.
5. **Mobile Patient App:** A mobile application for patients to connect and retrieve their immunization records.

Of the deliverables, this particular design document will focus on the backbone, web and reference mobile app components. It is expected that mobile experiences will vary from country to country and may need to be customized based on need within those jurisdictions.

The need to customize edge devices speaks to one founding principle of the OPENIZ project that is extensibility.

4.4. User Classes and Characteristics

This section will seek to introduce the user classes. A user class is not necessarily a technical role nor is it a single person, rather, it is a mechanism used to consider how a particular role is expected to interact with the system. A user class will have a series of skills, duties and concerns that will be addressed.

Table 2 - User Classes and Characteristics

User Class	Classifier	Priority
Immunization Officer	Low technical skill, expertise in performing immunizations.	VH
Receptionist	Low technical skill	M
Clinic Manager	Low technical skill, expertise in gathering statistics and managing stock.	H
Regional Manager	Low technical skill, expertise in gathering statistics and managing stock.	M
System Administrator	High technical skill, expertise in systems management.	M
Governing Authority	Medium technical skill, expertise in funding and immunization strategy.	M
Audit / Privacy Officer	Medium technical skill, expertise in privacy legislation.	M
Application Developer	High technical, expertise in creating new extensions.	M
Patient	Technical skill unknown, expertise in adhering to appointments	M

4.4.1. Immunization Officer

The immunization officer user class is defined as an individual whom performs immunizations at local clinics and is responsible for clinical observation (such as adverse reactions, weight, etc.).

4.4.1.1. Skills

The immunization officer within a clinic is typically quite capable of performing immunizations, ensuring that non-expired vaccines are used. The officer will have moderate to low technical skill and is expected to be able to use a simple user interface to enter simple fields like date/time of an action and the result of their action (for example, date/time of immunization and the result or date/time of weight and the measure).

4.4.1.2. Duties

The immunization officer is often busy and the use of the application is often a tertiary duty. It is important that the immunization officer not be distracted from the primary duties of ensuring healthy patients and performing the immunization.

4.4.1.3. Concerns

The immunization officer may have many concerns with the technical solution, including:

- Time, the application must not introduce a bottleneck in the process of immunizing patients.
- Accuracy, the application must provide an accurate depiction of the patient's medical status including weight, history of immunizations, and correct demographics.
- Confidentiality, the application must provide a mechanism that ensures vital information such as immune compromised is shown the officer but not shown to other generic users

4.4.2. Receptionist

The receptionist user class represents an individual who is responsible for preparing the visit by performing tasks such as demographics collection/updates, scheduling of future appointments, etc. The receptionist may be the same person as the immunization officer in some low resource settings, or may be a separate person representing the clinic.

4.4.2.1. Skills

The receptionist is expected to have lower technical skill than the immunization officer, however be adept at using simple technology such as filling in form fields, reading a calendar and following recommendations presented.

4.4.2.2. Duties

The receptionist's primary duties include:

- Onboarding new patients that arrive at a service delivery location, or updating existing patient demographics at presentation
- Ensuring identification is accurate and up to date, including immunization cards and/or health insurance information
- Notifying the immunization officer that the patient has arrived, or placing the patient into the immunization officer's queue, confirming with the patient the purpose of the encounter.

4.4.2.3. Concerns

The receptionist many concerns with the technical solution, including:

- Time: the application must not introduce a bottleneck in the process of queuing patients.
- Difficulty: the application must not be difficult to use and/or present confusing dialogues or options to the receptionist.
- Accuracy: the application must provide validation that ensures that the keystroking of the receptionist does not cause a faulty record.

4.4.3. Clinic Manager

A clinic manager user class represents an individual who is responsible with the operation of a single clinic and ensures that the clinic has sufficient stock perform immunizations. The clinic manager is also

responsible for the transfer of stock to/from a regional distributor, managing the clinics stock balance and ensuring that any technology used within the clinic is not operating in an error state.

4.4.3.1. Skills

The clinic manager is expected to have higher technical skill than the immunization officer does, and should be adept at stock counting, basic math (such as converting vials into doses) and reporting.

4.4.3.2. Duties

The clinic manager's primary duties include:

- Performing stock counts at set intervals and reporting current stock to a regional authority.
- Ordering and receiving stock into the facility and returning expired/unusable stock to another authority.
- Ensuring that any technology used in the clinic is operating in a non-faulted state and that any technology is ready for immunization duties (charged and functional).
- Preparing/running reports which relate to the operation of the clinic such as stock forecasts (days of remaining stock), number of patients immunized, number of outstanding technical issues (failures, etc.), number of patients expected in the coming days, etc.

4.4.3.3. Concerns

The clinic manager may have concerns with the technical solution, including:

- Accuracy: The solution must provide accurate information (as is possible) related to the current stock and function of the clinic such as patients immunized, forecasted patients and consumption of stock, etc.
- Communication: The solution must provide a mechanism for ordering stock in a manner that the manager can see the status of their order can place additional orders, and view balances.
- Security: The solution must provide a secure access layer that prevents unauthorized access to lost/stolen tablets and must ensure the clinic manager does not view information they are not permitted to view.

4.4.4. Regional Manager

A regional manager is an individual who is responsible for the management of a collection of immunization clinics within a specific region such as district, county, city, province, etc. The regional manager is responsible for ensuring sufficient vaccine stock is available for the clinics in their region, and ensuring that immunization coverage meets specified targets.

4.4.4.1. Skills

A regional manager may be of moderate technical skill and, it is expected, should be able to interpret reports and manipulate data within a basic tool like excel, print reports and scan.

4.4.4.2. Duties

The regional manager's primary duties include:

- Organizing stock orders, packing them for subordinate facilities and delivering or facilitating pick-up of the orders.
- Running reports for their district/region and adjusting stock, vaccination campaigns, or outreach programmes.

- Reporting to higher echelons of administration the performance of their region.
- Ordering stock from national distributors of vaccine and ensuring sufficient safety stock to supply their region.
- Device provisioning including onboarding of new users and devices for use within their region and ensuring lost devices are purged/located and user accounts locked under correct conditions.

4.4.4.3. Concerns

The regional manager may have concerns with the technical solution, including:

- Accuracy: The solution must provide accurate information (as is possible) related to the current stock and function of clinics such as patients immunized, forecasted patients and consumption of stock, etc.
- Communication: The solution must provide a mechanism for relaying status of a stock order in a manner that the manager can see the status of their order, can place additional orders, and view balances, pick stock from their current store, etc.
- Security: The solution must provide a secure access layer that prevents unauthorized access to lost/stolen tablets and must ensure the regional manager does not view information they are not permitted to view such as discrete data.

4.4.5. System Administrator

A system administrator is an individual who is responsible for the planning, setup and maintenance of the solution.

4.4.5.1. Skills

The system administrator is typically a highly skilled individual who is responsible for the maintenance of several software systems within a region/district/country. The administrator in some LMIC may have less technical skill but still be familiar with basic database terminology, practices, etc.

4.4.5.2. Duties

The system administrator's duties include:

- Backup of computer databases which contain PHI
- Maintenance of security accounts, and devices permitted to log in within a particular region.
- Setup and installation of software components and their upgrades including networking configuration.
- Advanced technical support, analysis of log files, and other diagnostic tool output.
- Planning of physical architecture, deployment timelines, OID registrations, etc. including the issuance and revocation of PKI certificates.

4.4.5.3. Concerns

A system administrator will typically have several concerns with technical solutions that may include:

- Security: What impact on the network surface area will the solution have, and how will the solution adversely affect the operation of other systems within the enterprise.
- Reliability: What is the reliability of the solution and the burden on the administration that technical support calls will place on resources?

- Cost: What is the cost of maintaining the infrastructure after initial capital costs? What are the costs related to the installation of the system and what, if any, are the licensing impacts on the operating budget and IP of other systems in the network (example: GPL)
- Auditability: What is the traceability of the solution? How easy are deployment mis-configurations to find and diagnose? How difficult are logs to obtain? Do the logs contain sufficient information to quarantine data and/or users and machines in case of breach?

4.4.6. Governing Authority / National Officers

National authority / officers are individuals who are responsible for the planning and maintenance of a national immunization programme. These individuals typically have moderate technical skills and are primarily interested in stock management and reporting functions.

4.4.6.1. Skills

A national officer has data analytics skills and moderate technical skill required to customize reports and manipulate data in Excel. A national officer or programme coordinator may be interested in customizing reports themselves.

4.4.6.2. Duties

The national officer's primary duty is the running of secondary use reports and the leveraging of these reports to perform business intelligence functions. The governing authority is also responsible for the administration and creation of legislation.

4.4.6.3. Concerns

The national/governing authority's primary concern will be that of provenance and governance capabilities of the system. The national officer may be responsible for ensuring that new legislation passed can be implemented within the system and that reports are accurate and up to date.

4.4.7. Audit / Privacy Officers

Privacy officers are individuals who are responsible for the implementation and adherence of the system and its users to policies configured for the jurisdiction. The privacy officer is also concerned about security breaches, performing spot audits to ensure that users are using the system correctly.

4.4.7.1. Skills

The privacy officer is of moderate technical skill, and high domain expertise skill. The privacy officer has the ability to use Microsoft office products, as well as basic BI tools and web-interfaces for detecting security breaches.

4.4.7.2. Duties

The primary duties of the privacy officer include the setup and validation of configured policies within the core application as well as performing routine privacy audits on the system. The privacy officer is also responsible for participating in threat risk assessments and privacy impact assessments.

4.4.7.3. Concerns

The primary concerns of the privacy officer are that the system will enforce consent policies imposed by the deployment jurisdiction, and that any overrides are easily identifiable in any audit logs. The privacy officer will also be concerned with the detail of PIA and TRA assessments performed against the system and will require lots of documentation related to the security services provided by the system.

4.4.8. Application Developer / Implementer / Technical Expert

The application developer, implementer and technical expert class is used to describe those professionals who will be developing integration points for the OPENIZ system for the purpose of implementing the solution in a jurisdiction.

4.4.8.1. Skills

The technical expert is of high technical skill and is able to understand application programming interfaces (API) documentation and how these APIs can be used to control the system for the purpose of their implementation.

4.4.8.2. Duties

The technical expert's primary duties include the development and customization of the OPENIZ solution, as well as the deployment and configuration for a particular deployment. Technical experts may also develop plugins and/or consumer applications of the platform.

4.4.8.3. Concerns

The technical expert's primary concern is that of ease of implementation and integration of the solution. The technical expert expects the system to provide sufficient application programming interface hooks in order for them to sufficiently expand the system to complete whatever implementation work they are performing. The technical expert will also be concerned with the stability and robustness of documentation of interfaces as well as the performance of the system and availability of development tools.

4.4.9. Patient

The patient class is used to describe the consumer of the healthcare services (immunizations) or one of their delegates. This may include parents, relatives, guardians, etc. While patients are directly users of the system per-se, they may play a role in the use of personal portals into the system.

4.4.9.1. Skills

Patients may be of varying skill from complete computer illiteracy, to high technical shrewdness. Patient interfaces should use simple language and only display the necessary information for the patient to understand the data that they are viewing.

4.4.9.2. Duties

The primary duties of the patient class are the attending of appointments registered in the system, and the obtaining of proper patient identification to be identified within the system.

4.4.9.3. Concerns

Primary patient concerns involve the proper and accurate identification of data, the cleanliness (clarity) of interfaces and confirmation prior to submitting any changes to the system.

4.5. IMS Platform

The design of the backbone is not platform specific and could be implemented in a number of different ways. The initial version discussed in this design document will be implemented using the Microsoft server stack, making use of the following technologies:

Table 3 - IMS Implementation Platforms

Technology	Reasoning	Relates To
Microsoft .NET Framework	Execution Environment	Backbone, Web Interface, Administrative Interface
Microsoft SQL Server 2014	Database Environment	Backbone
Microsoft SQL Server Reporting Services	Reporting Services	Reporting
MEDIC Service Core Framework	Robust set of existing plugins available.	Backbone
Microsoft IIS/MVC Framework		Web Interface, Administrative Interface

4.6. Web Portal Operating Environment

The initial version of the web portal will make use of Node.js as the web platform. Node.js offers native support for RESTful APIs and JavaScript, which will be the primary interface method for connecting to the Backbone.

4.7. Mobile App Operating Environment

Reference mobile apps will be created for both the providers and the patients. To achieve the maximum possible device support Apache Cordova will be used as the platform of choice. Cordova is a tool for creating “hybrid” mobile apps that can share a logic codebase but also provide a native user interface for each supported platform.

4.8. Assumptions and Dependencies

The primary risk to implementation is the use of proprietary components upon which the stack will be based. To achieve the lowest cost deployment for LMICs, components that cannot be licensed as free and open source shall be avoided where possible.

5. Requirements

This template illustrates organizing the functional requirements by features.

5.1. User Stories / Use Cases

5.1.1. User Logs In With Valid Credentials

An immunization officer uses the Provider mobile app to log into their provided user account. The device has an active internet connection. Upon login the device sends a unique device identifier to the central authentication system proving its identity. If the user credential is valid, and the device credential is valid, then the system audits the successful login.

5.1.1.1. Functional Requirements

1. The system SHALL provide a login page accepting a user's username and password as authentication mechanism.
2. The system SHALL authenticate the user against a centralized series of credentials or a cached set of local credentials previously entered and validated but not invalidated due to timeout or change.
3. The system SHALL provide a mechanism for storing device credentials including a unique device identifier token that uniquely and unambiguously identifies the device.

5.1.1.2. Security Requirements

4. The system SHALL use a centralized security scheme encrypted using some sort of security token service.
5. The system SHALL provide a mechanism for revoking user accounts in a manner that allows the revocation to propagate to connected devices.
6. The system SHALL use encrypted communications channels secured by certificates to ensure that sensitive information is encrypted between endpoints.
7. The system SHALL provide a mechanism for validating tokens be issued by a trusted party.

5.1.1.3. Quality Assurance Requirements

8. The system SHALL be subjected to a positive authentication test whereby valid user and device credentials are passed to the system. The system SHALL validate these credentials and issue a session.

5.1.1.4. User Interface Requirements

9. The system SHALL present a user login page which allows the user to enter a user name (or equivalent user identification) and a password (or equivalent user identification).

5.1.1.5. Documentation Requirements

10. The system SHALL provide a formal documented method of logging into the system when a user knows their username and password.

5.1.1.6. Interoperability / Communication Interface Requirements

11. The system SHALL communicate a login attempt to a centralized authentication service and SHALL include the username and password entered as well as any other identifying information required. If valid, the system SHALL issue a valid JWT or equivalent OAUTH token to the authentication target (device being authenticated).

12. The system SHALL audit to a central audit repository, the valid access attempt, the session issued and the length of session granted.

5.1.2. User Logs In With Invalid Credentials

An immunization officer uses the system to log into their provided user account. During the login process one of the credentials provided to the system (device or user) is invalid. The system alerts the user to the invalid credential condition and audits the invalid access attempt.

Alternate: The user continues to provide invalid credentials. After the third invalid login attempt the device does not permit another attempt for a 60 second period effectively locking the device.

5.1.2.1. Functional Requirements

13. The system SHALL construct an audit message containing sufficient information to detect a breach and/or invalid access attempt.
14. The system SHALL provide a mechanism for locking out further access attempts whenever an invalid credential is provided.

5.1.2.2. Security Requirements

No additional security requirements identified.

5.1.2.3. Quality Assurance Requirements

15. The system SHALL be subjected to a false user authentication test, whereby an invalid user credential and valid device credential is passed to the system. The system SHALL NOT issue a valid session.
16. The system SHALL be subjected to a false device authentication test whereby an invalid device credential is used and valid user credential is used. The system SHALL NOT issue a valid session.

5.1.2.4. User Interface Requirements

17. The system SHALL provide a descriptive error message when an invalid user credential is provided.
18. The system SHALL NOT disclose the source of the invalid credential, a generic error message about the invalid credential SHALL be provided.

5.1.2.5. Documentation Requirements

19. The system SHALL provide informative documentation instructing system administrators of the appropriate actions to take to resolve the invalid credentials condition.

5.1.2.6. Interoperability / Communication Interface Requirements

20. The system SHALL audit the invalid access attempt, the nature of the authentication failure (reason for failure) and the access attempt the invalid access represents (i.e. third attempt).
21. Upon lockout, the system SHALL produce a high level audit which indicates a severe security breach.

5.1.3. User Resets their Password

An immunization officer wishes to log into the device, however forgets their password. The immunization officer uses the system to reset their password from a registered device. The user enters their username and answers one of the security questions they had setup when their account was created. The user receives an out of band code that they enter into the forgotten password system. The

forgotten password subsystem validates the out of band code provided to the user with the token generated and, if valid, permits the user to enter a new password.

5.1.3.1. Functional Requirements

22. The system SHALL provide a mechanism for storing a security question and answer associated with a particular user account.
23. The system SHALL provide a mechanism for storing an out of band telecommunications method such as SMS or e-mail for sending of out-of-band confirmation data.
24. The system SHALL validate that the “forgot password” request is initiated from a registered device.
25. The system SHALL provide a mechanism for generating a random, 6 digit code and SHALL associate the generated reset code with a user account. The reset code SHALL have an expiry time window of no more than two hours.
26. The system SHALL provide a mechanism for a user to enter the generated code.

5.1.3.2. Security Requirements

27. The system SHALL store the answers to the security questions in a non-reversible encryption format such as a hash.
28. The system SHALL NOT store the generated security PIN and SHALL store a hash of the security PIN generated by the system.

5.1.3.3. Quality Assurance Requirements

29. The system SHALL be subjected to a positive case test whereby the user successfully enters a verification answer, and PIN as generated by the out-of-band PIN generator.
30. The system SHALL be subjected to a false case whereby the user does not enter a valid security question answer. The system SHALL lock the account after three attempts with an invalid security response are entered.
31. The system SHALL be subjected to a false case test whereby a successful security answer is entered however an invalid 6 digit PIN are entered. The system SHALL perform a lockout after three invalid attempts with a security PIN.

5.1.3.4. User Interface Requirements

32. The system shall provide a user interface for entering the username of the user whom forgot their password.
33. The system SHALL provide a user interface to respond to the security question setup by the user after successful retrieval of the security question.
34. The system SHALL provide a user interface to respond with the out of band security code.
35. The system SHALL provide user feedback when an invalid username, security response or PIN code are entered.
36. The system SHALL provide a user warning on all user screens whenever the user sets up an account without an out-of-band telecommunications address or without a security question.

5.1.3.5. Documentation Requirements

37. The system SHALL provide informative documentation instructing users of the password recovery process. This documentation SHALL be provided inline in the web-pages controlling security as well as the administrative/operations manual.

5.1.3.6. Interoperability / Communication Interface Requirements

No additional interoperability or communication interface requirements are identified.

5.1.4. User Reviews Appointments

If login is successful, the immunization officer is presented with a dashboard portal. The user uses the dashboard to review the appointments in the system. The system provides a list of appointments for review and filtering by the user of the point of service device. The system does not disclose appointments for patients outside the immunization officer's responsibility (to be applied by policy, facility, etc.)

5.1.4.1. Functional Requirements

38. The system SHALL provide a mechanism for storing appointments for vaccination. The appointments SHALL be linked to the patient and antigens for which the patient is due at the specified appointment.
39. The system SHALL provide a mechanism for retrieving a list of appointments given a specified date range.
40. The system SHALL provide a mechanism for retrieving a single appointment.
41. The system SHALL provide a mechanism for paging the list of appointments at the server level, reducing the amount of traffic which travels between the point of service application and the backing services.

5.1.4.2. Security Requirements

42. The system SHALL provide a mechanism for enforcing security of the patient appointment data. The system SHALL not disclose appointment information against the will of the patient.
43. The system SHALL capture the intended purpose of use of the data whenever querying patient data. The system will ensure that this POU is used for the policy decision process.
44. The system SHALL audit the disclosure of all clinical and appointment data prior to sending the response to the requesting system.
45. The system SHALL provide a consent override mechanism which allows a user to elevate themselves above the normal security level.
46. The system SHALL provide a mechanism for "most restrictive" policy enforcement. For example, the system SHALL allow a receptionist to only view appointments for a particular clinic however may show appointments for several clinics for an immunization officer.

5.1.4.3. Quality Assurance Requirements

47. The system SHALL be subjected to a positive test case whereby a user accesses the appointment data for which they are allowed to see.
48. The system SHALL be subject to a false test case, whereby a user accesses the appointment data for which they are not permitted to view. The test SHALL ensure that appropriate audits occur and that the user is not permitted to view.
49. The system SHALL be subjected to a break the glass or elevation test whereby a user unsuccessfully accesses patient appointment data however passes an override directive. The system SHALL disclose the requested information.

5.1.4.4. User Interface Requirements

50. The system SHALL provide a user interface which allows a user to view appointments within the system.
51. The system SHALL provide a dashboard component which shows the appointments for the user.
52. The system SHALL provide descriptive error messages whenever an appointment retrieval fails because of technical or privacy error.
53. The system SHALL provide a clear description and button for the user to elevate themselves to a higher security level.

5.1.4.5. Documentation Requirements

54. The system SHALL provide informative documentation instructing users of the appointment process.

5.1.4.6. Interoperability / Communication Interface Requirements

55. The system SHALL provide appointments in an appropriate FHIR resource. These FHIR resources SHALL be chained to the patient being accessed.

5.1.5. User checks-in existing Patient

A patient presents to the clinic for their routine immunization. The receptionist scans their identification which performs an identification search within the system. The receptionist verifies the information and proceeds to “check-in” the patient.

Alternate: The patient presents without an identification card, however has attended the clinic before. The receptionist uses the system to search the local clinic’s user database.

5.1.5.1. Functional Requirements

56. The system SHALL provide a mechanism for uniquely identifying patients
57. The system SHALL provide a mechanism for searching patients based on a series of demographics details such as name, date of birth, mother/father’s name, etc.
58. The system SHALL provide a mechanism for linking multiple identities to a single patient demographic record. This single patient demographic record SHALL be unique per patient.
59. The system SHALL provide a mechanism for performing fuzzy searches on the patient demographics information including fuzzy date of birth match, fuzzy name search, and fuzzy address search.
60. The system SHALL provide a check-in system whereby the patient’s appointment is activated upon arrival and check-in.

5.1.5.2. Security Requirements

61. The system SHALL be configurable such that deployments may customize the manner in which search results are disclosed to the user. Such configurations may include only disclosing exact demographics matches, or confidence levels of match.
62. The system SHALL be configurable such that deployments may customize which demographics fields are disclosed to users of varying user levels. This may be used to prevent unnecessary disclosure of information from the system.
63. The system SHALL audit all demographics searches and check-ins. Audits SHALL include the name of the user performing the search, the search criteria, etc. The system SHALL reject queries which do not include minimum search criteria or user name.

5.1.5.3. Quality Assurance Requirements

64. The system SHALL be subjected to a positive test whereby exactly one match is returned from the demographics information
65. The system SHALL be subjected to a positive test whereby a matching patient is found via an alternate identifier.
66. The system SHALL be subjected to a test whereby a demographics field is removed from the disclosure field list, the system SHALL NOT disclose fields which are not in the disclosure field list.
67. The system SHALL be subjected to a test whereby a user is not supplied to the backbone, the backbone SHALL reject the query request.

5.1.5.4. User Interface Requirements

68. The user interface SHALL expose a search by demographics screen which can be used to search the backbone.
69. The search interface SHALL provide a mechanism for searching by a series of customizable demographics fields. The search interface SHALL be customizable such that user interfaces do not expose search parameters not enabled on the backbone.
70. The search interface SHALL provide a list of search results in accordance with the disclosure demographics fields.
71. The search interface SHALL allow the user to view any one of the returned demographic records and perform a check-in from the view demographics page.
72. The check-in interface SHALL provide a summary of procedures to be performed or SHALL indicate that the user does not have permission to view the scheduled vaccinations.
73. The user-interface SHALL provide a screen for viewing the currently checked in patients and shall provide an option for a receptionist to cancel a check-in.

5.1.5.5. Documentation Requirements

74. The system SHALL provide on-screen user documentation regarding the check-in procedure and demographics search procedure.
75. Each input field on the user interface SHALL provide context aware help (example: tooltip) which explains the use of the feature.

5.1.5.6. Interoperability / Communication Interface Requirements

5.1.6. User copies remote demographics into the system

A patient presents to the clinic for their routine immunization. The patient has never presented to the clinic before, however has a national identification card. The patient presents this which is used by the receptionist to download the patient's demographic data. The system queries the national identification system and presents a series of results to the user. The receptionist selects the appropriate record and indicates that the patient should be imported.

The receptionist continues to check-in the patient.

Variation: The receptionist updates the patient's demographic data and submits the changes. The system conveys this change to the national patient registry.

5.1.6.1. Functional Requirements

76. The system SHALL provide a mechanism for allowing a client to indicate they would like to include remote information from the national infrastructure in the result set of a query
77. The system SHALL provide a mechanism for allowing a client to store a remote demographic in the OpenIZ native data store.
78. The system SHALL provide a mechanism for identifying the provenance (origin) of remote data, and MAY expose this information to remote clients
79. The system SHALL provide a mechanism for designating an identifier within a search as a national identifier or globally unique identifier.

5.1.6.2. Security Requirements

80. The system SHALL provide a mechanism for configuring security and policy enforcement rules required to interoperate within a national infrastructure.
81. The system SHALL provide a mechanism for copying necessary policies related to a remote demographic record into its local data store. The local policies on the imported data SHALL be carried with the record throughout the lifecycle of the object.
82. The system SHALL audit that remote patient demographic data was imported when the import process for the demographic has completed successfully.

5.1.6.3. Quality Assurance Requirements

83. The system SHALL be subjected to the IHE PDQ connect-a-thon tests to ensure that a sufficient level of implementation exists to interoperate within an IHE affinity domain.
84. The system SHALL be subjected to the OpenHIE Client Registry test suite to ensure that the solution can interoperate within the OpenHIE infrastructure.
85. The system SHALL NOT persist any partial data fetched from the national infrastructure, and SHALL rollback any partial updates (due to system availability issues, etc.)

5.1.6.4. User Interface Requirements

86. The system SHALL provide a list of matching patient demographics (based on the national identifier search) and SHALL allow a user to select the most appropriate patient prior to proceeding within a workflow.
87. The system SHALL display the set of demographics which are available from the jurisdictional record.
88. The system SHALL allow the receptionist to update the patient's demographic data and commit the demographics data to the system's primary data store.
89. The system SHOULD provide a mechanism for scanning a barcode or other physical media (RFID, NFC, etc.) attached to the patient's card.

5.1.6.5. Documentation Requirements

90. The system shall provide interface documentation which allows developer personnel to interact with the system in a manner which facilitates the fetching of remote records.
91. The system shall provide user interface documentation which instructs the user that remote patient data is subjected to policies enforced and imported from the jurisdiction.

5.1.6.6. Interoperability / Communication Interface Requirements

92. The system SHALL implement the IHE PDQ (Patient Demographics Query) profile for the purpose of fetching patient demographics from a national infrastructure.

93. The system SHALL implement the IHE PDQ for HL7v3 profile for the purpose for fetching patient demographics from a national infrastructure.
94. The system SHALL implement the IHE PDQm profile for the purpose of fetching patient demographics from a national infrastructure.
95. The system SHALL implement the IHE PIX profile for the purpose of submitting new patient data to the national infrastructure.

5.1.7. User registers a new Patient

A previously unregistered patient presents to the immunization clinic. The patient has never visited the clinic before and does not have a national or regional identification from another clinic. The receptionist gathers the user's demographic details and enters them into the system. The receptionist saves the new demographic record which results in a new patient record within the system. The receptionist saves any existing immunizations the patient has received in the system. The system calculates an immunization schedule for the patient and schedules appointments if necessary. The receptionist reviews the created schedule, and if necessary, continues to check-in the patient.

*Variation***:** This new identification is posted to the national records system which results in a new jurisdictional identifier for the patient.

Variation: The new patient's demographics exactly match the demographics of another patient already registered in the OPENIZ system. The receptionist is shown a warning confirming that this is in fact a new patient or if the patient is a duplicate.

Variation: The new patient's demographics exactly match the demographics of another patient already registered in the OpenIZ system. The patient is registered. At a later time, a district officer retrieves a list of conflicts and resolves the duplicate record. The duplicate record's immunization data is copied into the new patient master file.

5.1.7.1. Functional Requirements

96. The system SHALL permit the ad-hoc registration of a patient and SHALL store that patient locally in the primary data store.
97. The system SHALL permit the entry of back-dated immunizations for the patient. The system SHALL store these as a discrete vaccinations.
98. The system SHALL permit the entry of contraindications which the patient suffers from.
99. The system SHALL calculate the vaccination schedule based on the sum total of the new patient health record including contraindications and existing vaccinations.
100. The system SHALL be capable of determining and tagging duplicate patient entries.
101. The system SHALL provide a mechanism for resolving duplicate patients.

5.1.7.2. Security Requirements

102. The system SHALL only ever auto-merge or should only allow receptionist merges of duplicate entities if they are exact matches (i.e. deterministic matching)
103. The system SHALL communicate new demographics to the national patient registry over a secure channel.
104. The system SHALL audit the creation, merge, or update of records to the configured audit repository. The audit SHALL contain the user involved in the action and the patient demographic involved in the action.

5.1.7.3. Quality Assurance Requirements

105. The system SHALL be subjected to a test whereby a receptionist user creates a new demographic which does not have a matching demographic. The system SHALL NOT flag the new demographic as a duplicate.
106. The system SHALL be subjected to a test whereby a receptionist user creates a new demographic which exactly matches an existing record. The system SHALL mark the duplicate.
107. The system SHALL NOT submit data to the national infrastructure prior to confirming successful creation of the local data.

5.1.7.4. User Interface Requirements

108. The system SHALL provide free text entry fields for patient names, addresses, existing identifiers, etc.
109. The system

5.1.7.5. Documentation Requirements

5.1.7.6. Interoperability / Communication Interface Requirements

5.1.8. Patient presents and is past due / has no appointment

A patient presents to the clinic without an appointment, or is late for an existing scheduled appointment. The receptionist uses the system to look up the patient's records, and looks at the missing (past-due) immunization events within the system. The receptionist requests the system to generate an on-demand immunization recommendation, the system creates an appointment with the past-due immunizations scheduled on the current date. The receptionist checks-in the patient for the created appointment.

Variation: The system displays the past due vaccination and automatically generates an updated vaccination schedule and displays any warnings if appropriate (i.e. some vaccines may be unsafe or may require a different dosing).

5.1.8.1. Functional Requirements

110.

5.1.8.2. Security Requirements

5.1.8.3. Quality Assurance Requirements

5.1.8.4. User Interface Requirements

5.1.8.5. Documentation Requirements

5.1.8.6. Interoperability / Communication Interface Requirements

111.

5.1.9. Patient presents and provides new demographics

A patient presents to the clinic for their routine immunization. The patient informs the receptionist that their demographics information (phone number, address, etc.) has changed. The receptionist keys the changed data into the system and saves the patient's demographic information.

5.1.9.1. Functional Requirements

112.

5.1.9.2. Security Requirements

5.1.9.3. Quality Assurance Requirements

5.1.9.4. User Interface Requirements

5.1.9.5. Documentation Requirements

5.1.9.6. Interoperability / Communication Interface Requirements

113.

5.1.10. Immunization officer performs vaccination

After being checked in, the patient waits a predetermined amount of time. The immunization officer calls the patient into a private area to discuss their immunization history, and reviews the immunizations to be given for the specific encounter. The physician starts the encounter recording measurements (such as height, weight, etc.) and adjusts the list of immunizations to be given based on what is considered safe. The patient receives the immunizations from the immunization officer.

5.1.10.1. Functional Requirements

114.

5.1.10.2. Security Requirements

5.1.10.3. Quality Assurance Requirements

5.1.10.4. User Interface Requirements

5.1.10.5. Documentation Requirements

5.1.10.6. Interoperability / Communication Interface Requirements

115.

5.1.11. Patient has an adverse reaction to immunization

After receiving the immunization, the patient is instructed to wait a certain time period before leaving the clinic. During this time the patient develops a rash/fever/other reaction. The immunization officer uses the system to record the adverse event (i.e. updates the immunization encounter).

Alternative: After going home, the patient starts to develop an adverse reaction to the vaccine given during their encounter. The patient may access the system (example: via PHR) to note that they had an adverse reaction to the vaccine.

5.1.11.1. Functional Requirements

116.

5.1.11.2. Security Requirements

5.1.11.3. Quality Assurance Requirements

5.1.11.4. User Interface Requirements

5.1.11.5. Documentation Requirements

5.1.11.6. Interoperability / Communication Interface Requirements

117.

5.1.12. Patient enters immunization history

Prior to attending a local school, the patient is requested to use a publicly available form to indicate the vaccinations which they have received in the past. This form is given to a receptionist of a local clinic, or

regional manager, or completed online. The form submits data with a “holding” status, until the information can be verified by a qualified medical professional (immunization officer). The immunization officer updates the status of the immunization record from “holding” to “active” to indicate the information was correct.

5.1.12.1. Functional Requirements

118.

5.1.12.2. Security Requirements

5.1.12.3. Quality Assurance Requirements

5.1.12.4. User Interface Requirements

5.1.12.5. Documentation Requirements

5.1.12.6. Interoperability / Communication Interface Requirements

5.1.13. User authorizes a new application

A user (immunization officer, patient, etc.) has downloaded a new application from a mobile application store and wishes to allow the application to access their immunization record. The mobile application provides the central OPENIZ repository in the jurisdiction with its application key, and the user credentials. The OPENIZ server matches this with patient credential using a token and associates the application with the user’s profile. The application can now access functions as prescribed by the jurisdiction for the particular application.

5.1.13.1. Functional Requirements

119.

5.1.13.2. Security Requirements

5.1.13.3. Quality Assurance Requirements

5.1.13.4. User Interface Requirements

5.1.13.5. Documentation Requirements

5.1.13.6. Interoperability / Communication Interface Requirements

120.

5.1.14. National officer enables a new application

After reviewing an application on the mobile application store, the national officer decides that an application meets criteria for a need within their jurisdiction. The national officer enables the application on their service by allowing the application key and selecting the user roles / application functions that the application is allowed to operate. This information is communicated to the OPENIZ backend.

5.1.14.1. Functional Requirements

121.

5.1.14.2. Security Requirements

5.1.14.3. Quality Assurance Requirements

5.1.14.4. User Interface Requirements

5.1.14.5. Documentation Requirements

5.1.14.6. Interoperability / Communication Interface Requirements

5.1.15. District / Regional / National Officer runs summary reporting

A regional officer wishes to determine the performance of their immunization programme within their jurisdiction. The officer uses the reporting engine of the solution to run a series of reports which illustrate the performance of their region.

Alternate: The national officer uploads a new report to the reporting engine and selects which users may view the report and specified parameters they are permitted to view.

5.1.15.1. Functional Requirements

122.

5.1.15.2. Security Requirements

5.1.15.3. Quality Assurance Requirements

5.1.15.4. User Interface Requirements

5.1.15.5. Documentation Requirements

5.1.15.6. Interoperability / Communication Interface Requirements

5.2. Other Non-Functional Requirements

5.2.1. Performance Requirements

List any performance requirements if available. State any performance requirements and their rationale. This will help implementers understand the intent and make suitable design choices.

123. The system SHALL be capable of performing simple queries and returning resources from the local data storage device in a reasonable amount of time.
124. The system SHALL provide a mechanism for compressing inbound and outbound data.
125. The system SHALL provide a mechanism for fragmenting and bundling data. The system SHALL allow consumers to dictate how this bundling occurs in order to minimize traffic. This requirement is waived when standardized interfaces are implemented.

5.2.2. Safety Requirements

Specify requirements that are concerned with the possible loss, damage or harm that could result from the use of the deliverables of this project. Refer to any policies that are being enforced.

126. The system SHALL persist all outbound messages and SHALL track the response to outbound messages. Unsuccessful messages SHALL be flagged and the system SHALL provide a mechanism for re-sending data.
127. The system SHALL persist all inbound messages and their responses for any operation which modifies data. The system MAY persist the entire inbound request and/or response message but SHALL at least persist the unique message identifier. This functionality is related to exec-once requirements.

128. The system SHALL NOT communicate PHI over unsecured channels and SHALL reject any messages which are not sent over encrypted channels.
129. The system SHALL use node authentication when communicating with other infrastructure components. Node authentication SHOULD be used for end-user devices.
130. The system SHOULD use a local root authority for node authentication purposes but SHALL at minimum allow the trusting of a list of certificates if a root authority is not supported.

5.2.3. Security Requirements

Identify any requirements related to security or privacy issues. Define any user identity authentication and authorization requirements. Refer to any policies or regulations that are being enforced

5.2.4. Quality Assurance Requirements

Specify any quality characteristics of the software that are important to either the implementer, or customer. Some examples are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative and verifiable requirements when possible. At the least, clarify the relative preference for various attributes such as ease of use over ease of learning.

6. Interface Considerations

This section outlines the requirements of any external interfaces required to implement the project.

6.1. User Interfaces

Describe the logical requirements of a user interface that are required. This may include prototype screen captures, diagrams, product style guidelines, layout constraints, standard buttons that will appear on screens. Keyboard shortcuts and error message standards may also be listed here.

6.2. Software Interfaces

Identify any software interface that this project will provide. Include database services, libraries, tools, and integrated components.

Table 4 - Software Interfaces

Software Package	Type	Provider	License

6.3. Communications Interfaces

Describe any requirements associated with communications functions required by this product. This could include e-Mail, web-browsers, network server communications, protocols, etc…

Table 5 - Communications Interfaces

Interface	Service	Method / Standard	Provider
Immunization Management	OPENIZ Core Store	FHIR DSTU2 / IC+XDS	OPENIZ
Immunization Recommendation	OPENIZ Forecaster	FHIR DSTU2 / IC+XDS	OPENIZ
User Accounts	OPENIZ Authorization	OAUTH	OPENIZ
In-Application Reporting	SSRS Reporting	SSRS	Microsoft SQL Server 2008+
Decision Support Services	NxBRE DSS	RulesML	NxBRE
Document Consumer	OPENIZ XDS	IHE XDS.b	MEDIC / MSFT XDS
Document Source	OPENIZ XDS	IHE XDS.b	MEDIC / MSFT XDS
FHIR Service Core	Core FHIR Services	FHIR DSTU2	MEDIC SVC Core
Auditing	ATNA Auditing	ATNA + DICOM	MEDIC AVIC
HMIS Reporting	TBD	TBD	TBD
Patient Identity Source	Patient Identity	PIX	MEDIC CR
Patient Identity Consumer	Patient Identity	PIX	MEDIC CR
Patient Demographics Search	Patient Identity	PDQ	MEDIC CR

7. Solutions Architecture

7.1. Solution Architecture

OpenIZ provides a loosely coupled open system architecture. Figure 1 illustrates the major components of the OpenIZ platform where each bidirectional arrow represents a communications channel over an open standard.

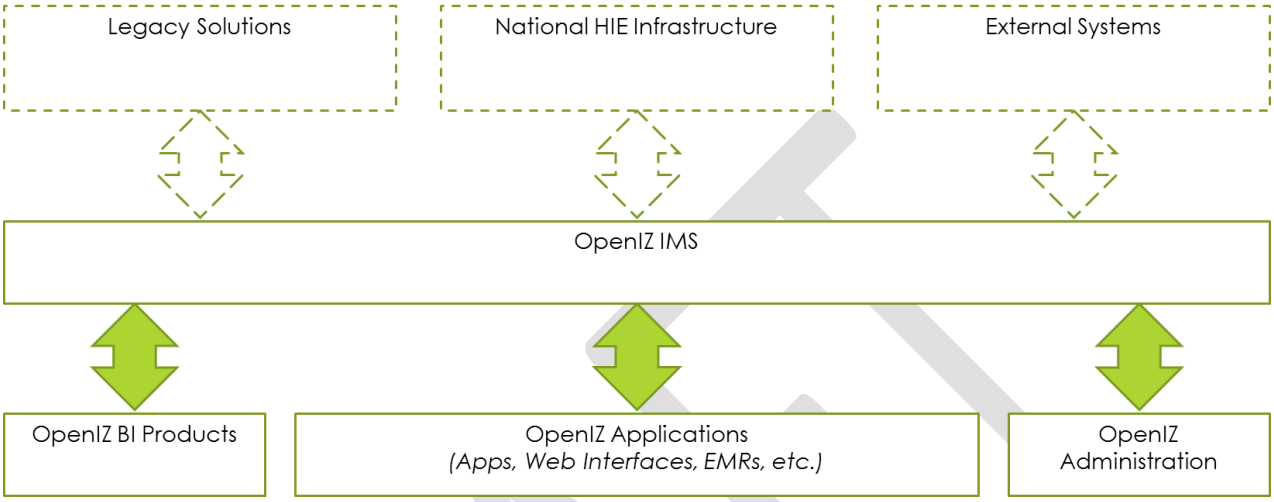


Figure 1 - OpenIZ System Architecture

The major components of the architecture are:

- **OpenIZ Immunization Management System (IMS)**: The IMS is the primary platform component of the OpenIZ platform. The OpenIZ IMS is responsible (at a high level) for:
  - Maintenance of individuals' immunization records
  - Scheduling and maintenance of immunization appointments
  - Forecasting immunization schedules and demand
  - Integration with infrastructural systems such as Logistics Management Information Systems (LMIS), Health Management Information Systems (HMIS), educational systems, etc.
- **OpenIZ Analytics Products:** Represent products which perform business intelligence functions. These are typically commercial BI products which are configured to work the OpenIZ backbone and data services. These types of interfaces are typically used by regional, district and national immunization coordinators.
- **OpenIZ Applications:** Represent applications such as EMRs, HISs, Mobile Applications, and custom websites which use the IMS to convey data to end users. This also include s the reference implementations of the patient and provider mobile applications.
- **OpenIZ Administration:** Represent administrative interfaces and services which are responsible for the maintenance and health of the OpenIZ IMS engine. These include configuration, reconciliation and other services which end users (clinicians, immunization officers, receptionists, etc.) do not require access to.

7.2. Network / Physical Architecture

Todo

7.3. Software Architecture

7.3.1. OpenIZ's Immunization Management System Architecture

The Immunization Management System (IMS) portion of OpenIZ is based heavily upon the micro-services architecture. In this architecture, a series of pluggable services implement a series of contracts. Whenever a function unit wishes to perform a unit of work it will ask the host context (IServiceProvider) to get the currently configured service provider.

The service types provided by OpenIZ's IMS are illustrated in Figure 2.

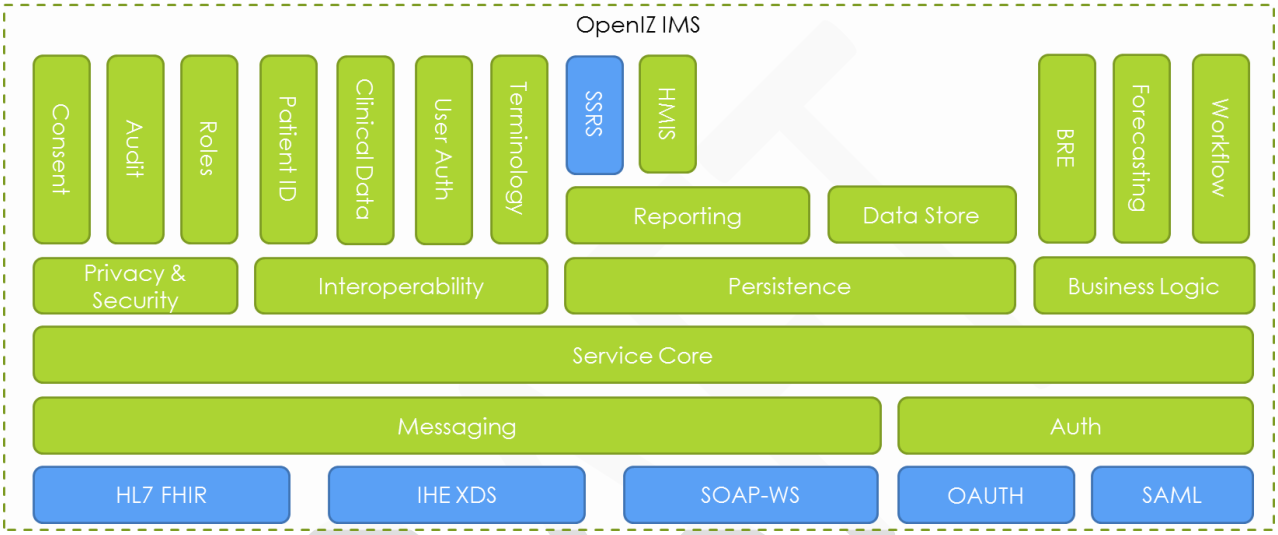


Figure 2 - OpenIZ IMS Component Architecture

Each service is described in more detail in Table 6 with those services provided by the MARC-HI Service Core framework marked in red.

Table 6 - OpenIZ IMS Services

Service	Contract	Description
Messaging	IMessageHandlerService	The message handler service is started upon application start/stop and is used to receive messages, parse them into a canonical form.
	IMessagePersistenceService	The message queue service allows messaging services to queue inbound messages that need re-processing.
Auth	IAuthService	Auth services are used to expose authorization schemes to OpenIMS applications, and offer centralized user credentialing.
ServiceCore	IVaccinationService, IAppoitmentSchedulingService, IPatientRegistrationService, IProviderRegistrationService, IAdverseEventService,	These clinical data services are responsible for the orchestration of underlying functions to perform the specified operations they define. For example: The appointment scheduling

	IStockService, IObservationService IHealthFacilityService	service would be responsible for finding recommended dates for a particular clinic.
	IConceptService	The concept management service is responsible for managing the internal concept dictionary found within the OpenIZ database.
Consent	IPolicyDecisionService, IPolicyInformationService	The consent service contract identifies a service that has the ability to apply consent rules (enforcement) against a set of results before prior to being disclosed.
Audit	IAuditorService	The audit service is responsible for handling audits generated from the IMS and either persisting them locally, or sending them to a central audit service.
Security/Roles	IIdentityProviderService, IRoleProviderService	The role provider service is responsible for determining user action permissions based on claims given by OpenIZ applications. It is also responsible for handling maintenance of the roles/users/devices/applications/etc.
Patient ID	IPatientIdentityService	The patient identity service is responsible for establishing patient identity, performing demographics searches etc.
Clinical Data	IClinicalDataService	The clinical data service is responsible for the consumption and publication of clinical data to/from OpenIZ's IMS.
Terminology	ITerminologyService	The terminology service is responsible for communicating with centralized terminology services to resolve terminologies when no internal concept dictionary candidate is found.
HMIS	IHealthManagementInformationService	The HMIS integration service is responsible for communicating data to a central HMIS system.
Data Store	IDataPersistenceService	The data persistence service is responsible for taking the internal canonical model of the OpenIZ IMS and translating that data into the physical data storage unit.
Business Rules (BRE)	IBusinessRulesService	The business rules service is fired in a series of places and allows deployments to easily extend the OpenIZ IMS behavior's business logic.
Forecasting	IForecastingService	The forecasting service is used primarily by the immunization management service and is focused on the forecasting

		of vaccine schedules. Forecasters operate in a slightly different manner than pure BRE in that Forecasting services are permitted access to make decisions on the entire patient context whereas BREs are executed on a single message interaction context.
Workflow	IWorkflowService	The workflow service is a planned service which can be used to manage and execute workflows based on a series of triggers. The workflow service is primarily hooked in on relevant events from the datastore.
Notification	INotificationService	The notification service is used to alert other systems of real-time data storage events. This is the hook that will most likely be used when implementing pure ODD in OpenIZ IMS or when merges or patient registrations occur.

The service execution flow is represented in Figure 3.

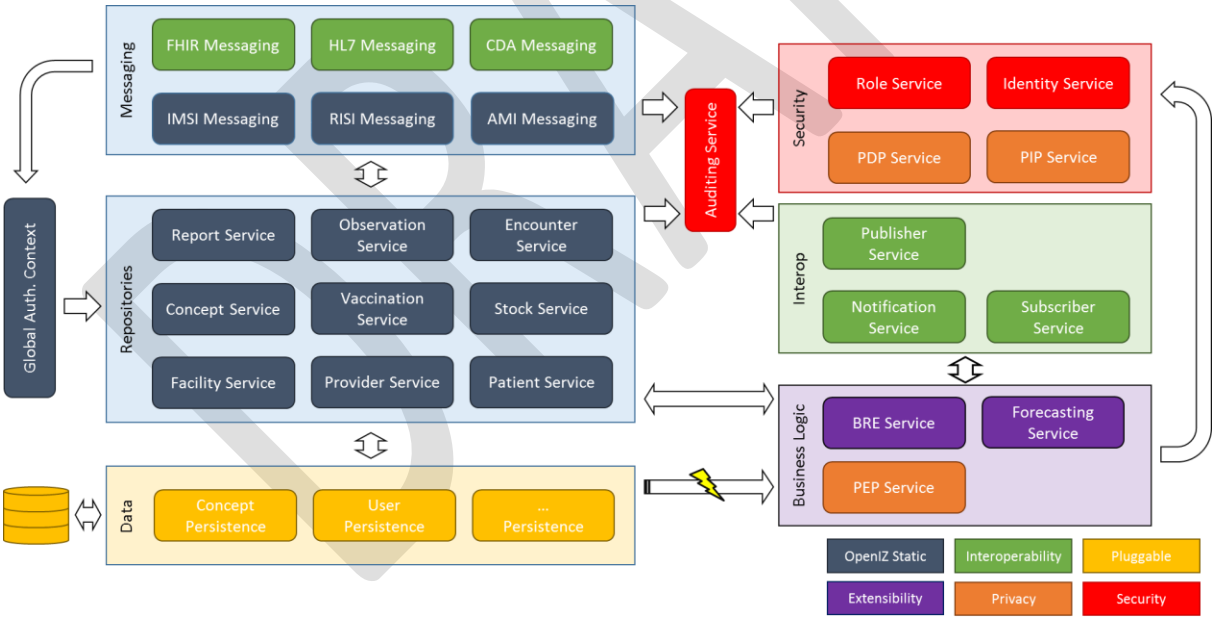


Figure 3 - IMS Component Execution Flow

7.3.1.1. Message Handling Services

IMessageHandlerServices represent services which are responsible for running listeners which operate and translate inbound message traffic into the View Model classes to be used in the common service orchestrator.

7.3.1.2. Message Persistence Services

The IMessagePersistenceService is responsible for the persistence of all inbound message traffic and can be used to store a message log of inbound messages, provide dead-letter-queuing (if required) and execute once capabilities.

The message persistence service has two major components, the persisting of messages and the persisting of message info records. MessageInfo instances contain basic information such as the interface on which the message was received, date/time, etc.

7.3.1.3. Daemon Services

The IDaemonService is not a service contract per-se, rather it is a scaffold interface (contract) which can be used by services which need to operate as a daemon within the OpenIZ application context. Daemon services are started as application context start and shut-down at application context stop. Daemon services are started in the order in which they appear in the application host's configuration file.

7.3.1.4. Authorization Handler Services

An IAuthService is a specialization of the message handler service which is specifically targeted at authorization of OpenIZ client application, devices, etc. The authorization handler service exposes a security token service (STS) which clients may use to get a signed token asserting that the client is valid. The authorization handler service is only implemented when OpenIZ is hosting the ACS. Depending on the environment this may not be desirable and a third party ACS may be used.

This can be useful when the IMS messaging handlers are configured to use federated security. In the case where the IMS messaging handlers use federated security, it is imperative that any authorization service used is configured to issue the same token type as the IMS service is expecting.

7.3.1.5. Clinical Data Repository Services

The clinical data repository services are responsible for orchestrating other OpenIZ service instances to perform a business function of work. The service orchestrator is primarily concerned with wrapping the complexity of data storage of elements in a consistent way.

These repository services differ from the data persistence services in that the data persistence services provide raw access to the database in an independent way. It is the raw data services which persist and raise events related to persistence whereas the clinical repository services represent a series of convenience methods wrapping the raw data persistence classes.

Service orchestrator operations are as follows:

Business Unit	Method	Description
Patient		The patient repository service is responsible for providing functions related to the searching, matching, storage and versioning of entities which carry the class code of PAT.
	Find	Returns a list of all patients whose demographics match the supplied lambda expression and/or pre-defined search parameters. This method ensures that patients are of appropriate status (not obsolete).

	Insert	Registers the specified patient, returning an object which indicates whether there were detected duplicates and any duplicate records (if privacy controls of the user applies).
	GetDuplicates	Searches for all patients which are flagged as duplicates which can be merged.
	Obsolete	Obsoletes the specified patients.
	Save	Updates the provided patient data in the database, validating the objects using specified business rules services.
	Merge	Merges the provided patient records into the specified survivor. SVN style merging is performed. For example, if A and B are to be merged into C then two new versions of C are created (merging B , merging C) and a new version of A and B each are created obsoleted.
	UnMerge	Splits a patient into a series of patients. This operation can only be done to “undo” a merge. In this scenario, if A and B were merged into C, the method will accept a version identifier for B which is to be re-constituted (un-obsoleted). Version C is tagged for manual review to remove data.
	Get	Retrieves the specified patient from the datastore.
Provider		The provider management service is responsible for the maintenance of provider information in the IMS data model.
	Search	Retrieves a list of a providers which match the provided LINQ expression.
	Register	Performs the necessary operations to store the provided provider data in the IMS data store.
	Obsolete	Obsoletes the provided provider in the IMS data model.
	Update	Updates the provided provider information, creating a new version of the provider within the data store.
	Get	Retrieves the specified provider from the data store.
Administration		The administration repository service is responsible for the storage and maintenance of substance administrations within the IMS data store.

Appointment		
Place		
Organization		
UserEntity		
DeviceEntity		
PlaceEntity		
Encounter		
Observation		

7.3.1.6. Concept Management Service

7.3.1.7. Stock Management Service

7.3.1.8. Timer Services

The timer service is an example of an unaltered IDaemonService, it makes no modifications to the underlying base service contract.

Timer service jobs are implemented via the ITimerJob interface. Their schedules are dictated by the timer service's configuration mechanism. The default timer job defined in the MARC-HI ServiceCore framework uses the application configuration file to manage the execution of timer jobs. OpenIZ may implement additional functionality in future releases to allow database based configuration to occur.

7.3.1.9. Audit Services

7.3.1.10. Consumer Services

The subscriber services are responsible for the retrieval of remote information from remote systems. When configured, the service orchestration services will execute external subscriber services and aggregate results with local results from the local data store. This behavior can often be controlled within the query to the service orchestrator. This behavior can be configured in the following modes:

1) **Local Only –** Remote demographics are never acquired. This mode is used when an ADT feed is used from a central EMPI to feed local patient data.
2) **Remote Only –** Remote demographics are always acquired and local patient records are created for referential integrity of the database only.
3) **Remote Null Set –** Local demographics and remote demographics are queried. If no local results are found then a remote call is done.
4) **Aggregate –** Local demographics and remote demographics are collected and the results are aggregated into a single result set.

Additionally the service orchestrator can create remote demographics records locally (copy) in order to maintain referential integrity of the data store.

7.3.1.11. Data Persistence Services

7.3.1.12. Business Rules Services

The business rules services are responsible for the execution of business rules based on system events. There are two types of services which can be classified as business rules:

1. **Event Based:** These services implement IDaemonService and subscribe to system events for which they are interested. For example, a BRE which validates a user's password is of correct length would subscribe to the PasswordChanging event of the IIdentityProvider.
2. **Explicit Call:** These services implement the IBusinessRulesService<T> and is executed on-demand. These type of business rules services are only called from repository services that require explicit business functions to be performed. An example of this would be an IBusinessRulesService<Patient> which may call : DetectMerge() or Validate() functions

By default, OpenIZ has a basic business rules engine service which provides access to both services as Microsoft ClearScript engine.

7.3.1.12.1 JavaScript Business Rules Engine

The JavaScript business rules engine allows implementers to describe their business rules as a series of JavaScript functions. These functions are broken into three types of function calls. Interfacing with the IMS is performed via the interface objects listed in . .

Interface	Helper	Description
OpenIZEngineService	OpenIZ.Engine	The engine interface is used for controlling subscriptions and registrations to the core OpenIZ engine.
OpenIZXRepositoryService	OpenIZ.X	This interface is a direct wrapper of the repository services and is used to call functions on those services. Where X is the name of the repository service listed in .
OpenIZConfigurationService	OpenIZ.Configuration	This interface is used to interface with the OpenIZ configuration system.
OpenIZSecurityService	OpenIZ.Security	Interfaces to the security system. Use this interface for interfacing with the identity, role and security providers.
OpenIZForecasterService	OpenIZ.Forecaster	Interfaces to the forecasting service.
OpenIZPersistenceService<X>	N/A	Interfaces to the persistence services.

Service handlers can be acquired by calling getService on the OpenIZ.Engine helper, dropping the OpenIZ moniker. For example, to acquire the patient data persistence service:

```
var patientPersistence = OpenIZ.Engine.getService(“PersistenceService<Patient>”)
```

Signal handlers are JavaScript event handlers which are registered via the connect method on the service instance.

```
var patientPersistence = OpenIZ.Engine.getService("PersistenceService<Patient>");
patientPersistence.Inserting.connect(function(sender, evt) {
        // Business rule : When a patient dies notify an external system
        if(evt.data.deceasedDate != null)
        {
                var reporting = OpenIZ.Engine.getService("Elbonia.OpenIZ.Reporting.DataReportingUtil");
                reporting.myCustomReportMethod(evt.data);
        }
});
```

Additional JavaScript interfaces can be implemented as described in Microsoft’s Jscript interface documentation () and registered with the OpenIZ BRE Jscript engine via configuration.

7.3.1.13. Forecasting Service

OpenIZ forecasting services (IForecastingService) is implemented by service classes which perform on-demand forecasting / scheduling. Passive forecasting (based on events) should be done by a daemon service which subscribes to events on the persistence layer.

Forecasting services generate instances acts with min/max times representing the minimum safe date, maximum safe date and suggested date for vaccinations. The forecasting service's proposal method accepts a parameter of type Patient and optionally any relevant data (existing vaccination SubstanceAdministrations, Observations representing AEFIs).

Acts created from the forecaster are mood code PRP or "Propose" (in RIM speak, the forecaster proposes an Act to occur). Illustrates a sample proposal from the forecaster in which a PENTA vaccine is proposed to occur at Good Health Clinic for patient Jamie Smith.

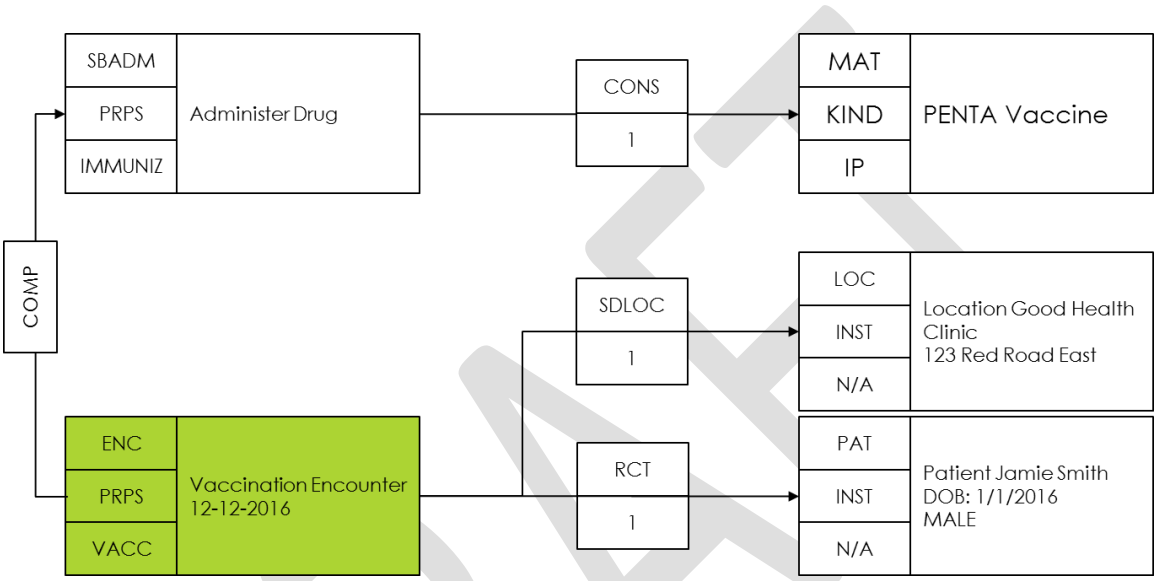


Figure 4 - Proposed Vaccination Encountered

More information about how to read these data model diagrams can be found in .

7.3.1.14. Workflow Services

7.3.1.15. Publisher Services

Publisher services are used to notify external systems when certain events occur within the IMS. These services are observers, meaning they start are IDaemonService instances which subscribe to events in the persistence layer and take appropriate action when events occur in the underlying persistence service.

7.3.1.16. Configuration of the IMS

The first version of the OpenIZ IMS backbone will leverage the MARC-HI ServiceCore framework components heavily. These components are configured via application configuration files. This will introduce some overhead on large-deployments as configuration files will need to be shared among the application hosts performing a particular role within the OpenIZ infrastructure.

Some components of OpenIZ such as forecasting and protocols are configured via a central database (or rather, their behavior is controlled by the central data store for OpenIZ).

7.3.1.17. Plugin Management

Plugin management is performed via a series of assembly attributes which are embedded in the assembly manifest of the IMS plugins. The following attributes are to be used for identifying plugin metadata:

Table 7 - Plugin Management Attributes

Attribute	Use	Description
AssemblyVersion	R	Identifies the version (major.minor.revision.build) of the plugin. This information is used for dependency information.
AssemblyInformationalVersion	O	An informational version which is displayed on the administration and management service interface.
AssemblyDescription	R	A human readable description of the plugin to appear on the administrative interface.
AssemblyCopyright	O	Copyright information and/or use restriction messages.
AssemblyPlugin	R	Identifies the assembly as a plugin. The plugin attribute identifies the minimum version of the OpenIZ core which is required to run the plugin.
AssemblyPluginDependency	O	Identifies the name and version of a dependency upon which the plugin must have installed.

Additionally plugins may embed database modification scripts into their assembly manifest. These database scripts are stored in an XML information format similar to Liquibase whereby “features” are identified and relevant “install” and “uninstall” SQL commands are included. These SQL statements may have guard conditions that are maintained by the database configuration technology selected.

Each feature file is also assigned an RDBMS invariant name that indicates the database management system for which the installation script is intended (in the case that a plugin works with more than one RDBMS, for example: the ADO.NET message persistence schemas).

7.3.1.18. Security Architecture

All the components of OpenIZ are designed to consider how data is access securely from each layer and between each component. This architecture requires that all access to method calls to secured services pass an instance of IPrincipal which represents the authenticated user context within the current execution pipeline.

There are four major concepts to the OpenIZ security architecture:

- **Identities:** Represent an identification of a security asset such as user, device or application. For example, the user jsmith would represent a user identity.
- **Principals:** Represent an authenticated identity (or collection of identities) representing a single session. Principals have an identity (the user/device/application accessing the IMS) as well as a series of claims about the identity (such as role/device/application/authentication method/etc.)
- **Policies:** Represent a definition of some action or group of actions applied against the OpenIMS system (such as login, create role, etc.) or data within the OpenIMS data store (such as privacy policies applied to data). Policy definitions are maintained by policy information providers.

- **Permissions:** Represent a granting of access or rights to a policy for a principal. The decision on whether a principal is granted a permission to perform an operation is made by a policy decision provider.

The creation of an IPrincipal instance can be from a local authority (such as simple SQL database authentication) or from a remote authority (such as SWT, JWT, etc.).

7.3.1.18.1 Basic Security

The default OpenIZ messaging services (FHIR, IMSI, etc.) can be configured to use HTTP basic authentication. This authentication mechanism is tied into the WCF pipeline and uses the current implementation of IIdentityProvider to authenticate username and password in the HTTP header. The device identity is established via the TLS client certificate sent in the HTTP request.

Applications connecting to a HTTP Basic security service are furthermore required to send their application identifier (as a UUID) and application secret in the X-OpenIZClient-Authorization HTTP header. This header has the same format as the BASIC Auth header and includes the client id and secret as a base64 encoded string.

Claims can also be sent using this scheme via the X-OpenIZClient-Claim HTTP header. Claim values are base64 encoded in format: claimURI=claimValue. The X-OpenIZClient-Claim HTTP header values repeat and the authentication pipeline ensures that the user is permitted to make the claim provided.

For example, the HTTP headers for user “Jsmith” on client e612f88c-3ba3-40fe-8cd6-792836b2088c making claim that purpose of use is TREATMENT would be:

```
POST /fhir
Authorization: BASIC anNtaXRoOnBhc3N3b3Jk
X-OpenIZClient-Authorization: BASIC
ZTYxMmY4OGMtM2JhMy00MGZlLThjZDYtNzkyODM2YjIwODhjOjc0MzQyYTE1MTY4YTQxODNhOWU2ZTllZTFmMGUxZWQ0
X-OpenIZClient-Claim: dXJuOm9hc2lzOm5hbWVzOnRjOnhhY21sOjIuMDphY3Rpb246cHVycG9zZT1UUkVBVE1FTlQ=
Content-Type: application/json+fhir
Content-Length: 2394

{
```

7.3.1.18.2 Federated Security

Figure 5 illustrates how a remote client can obtain a token from a federated security token service (STS) representing an IPrincipal and pass it to the OpenIZ IMS. The creation of a local IPrincipal is controlled by a local IIdentityProviderService implementation. It is imperative that the ACS generate a token format which is suitable for the IMS messaging interface to consume (i.e. the configurations match), otherwise the IMS will have no mechanism for verifying tokens.

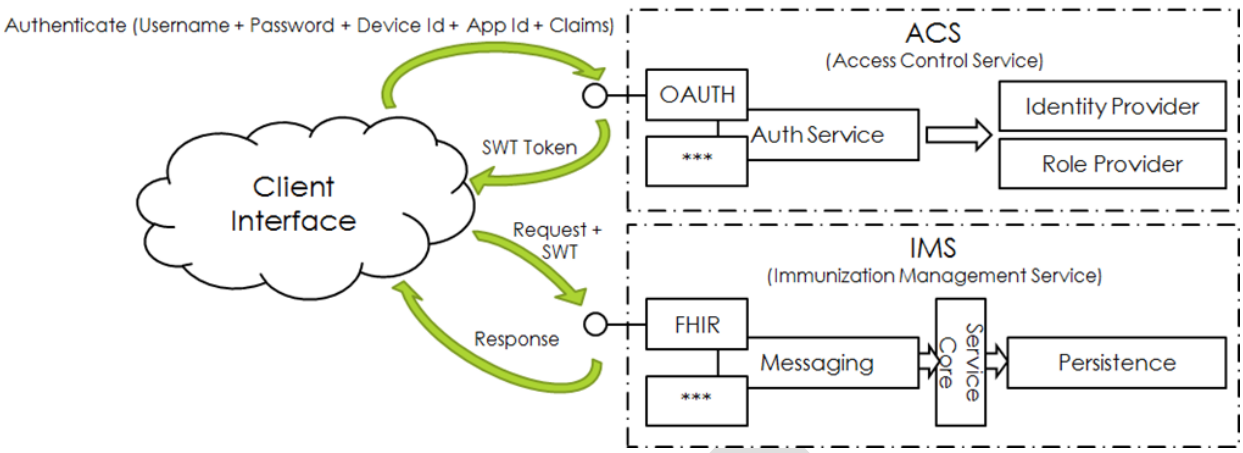


Figure 5 - Security Architecture

Any ACS service can be used with OpenIZ, however it is recommended that the ACS being used support the OAuth token service's password grant and provide client/device authentication via TLS and/or HTTP basic auth.

7.3.1.18.3 Default OAuth ACS Implementation

OpenIZ provides an implementation of an OAuth STS which generates JSON Web Tokens (JWT) compatible with OpenIZ. The default implementation of the OAuth STS only supports password and token refresh grant types.

The returned value is a JWT token which may subsequently be used by the client to access IMS service interfaces. The JWT token validator is inserted into the WCF's WIF pipeline and ensures that the token is signed by a trusted ACS and that the token has not expired.

The default ACS implementation performs node authentication (authentication of the device) using the TLS certificate passed in the SSL transport layer. The device certificate used to connect to the ACS forms the basis of authenticating the node and may be explicit (using the DeviceEvidence field in the SecurityDevice table) or chained (to a root CA that the ACS trusts).

Applications are authenticated using the HTTP BASIC auth scheme described in the OAuth 2.0 specification. The application is expected to pass its client_id and client_secret as a username/password in HTTP Authorize header.

The client can make claims about the request by using the X-OpenIZClient-Claim HTTP header. This header is in the format claimType=claimValue and is base64 encoded. Multiple claims are separated by a comma.

The following example represents a request for token for user jsmith from client e612f88c-3ba3-40fe-8cd6-792836b2088c making claim that purpose of use is TREATMENT.

```
POST /oauth2_token
Content-Type: application/x-www-urlform-encoded
Authorization: BASIC
ZTYxMmY4OGMtM2JhMy00MGZlLThjZDYtNzkyODM2YjIwODhjOjc0MzQyYTE1MTY4YTQxODNhOWU2ZTllZTFmMGUxZWQ0
X-OpenIZClient-Claim: dXJuOm9hc2lzOm5hbWVzOnRjOnhhY21sOjIuMDphY3Rpb246cHVycG9zZT1UUkVBVE1FTlQ=
Content-Length: 204
```

```
grant_type=password&username=jsmith&password=password123&scope=http://demo.openiz.org/fhir
```

7.3.1.18.4 OpenIZ Claims

The implementations of IPrincipal should be claims based. In a claims based principal, the authenticated user information contains a series of claims about that user such as their name, organization, the reason for access, etc. The claims used in OpenIZ are listed in Table 8.

Table 8 - OpenIZ Claim Types

Claim	Value	Use
urn:oasis:names:tc:xacml:2.0:resource:resource-id	String	The identifier of the resource to which the claim is about.
urn:oasis:names:tc:xacml:2.0:action:purpose	PurposeOfUse	Indicates the reason why data is being queried. Used for policy enforcement decisions. Valid values are drawn from the PurposeOfUse concept set.
urn:oasis:names:tc:xacml:2.0:subject:role	String	The clinical roles that the user has.
urn:oasis:names:tc:xspa:1.0: subject:facility	Url	The facility identifier to which the principle belongs.
urn:oasis:names:tc:xspa:1.0: subject:organization-id	String	The organization identifier to which the principal belongs.
urn:oasis:names:tc:xacml:1.0: subject:subject-id	String	The distinguished name of the principal.
http://openiz.org/claims/grant	String	The policies to which the user has been granted access by the ACS.
http://openiz.org/claims/device-id	String	The identifier for the security device from which the principal is operating.
http://openiz.org/claims/application-id	String	The identifier for the security application from which the principal is operating.
http://schemas.microsoft.com/ws/2008/06/identity/claims/role	String	Security roles to which the user belongs.
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	String	The user name of the principal.
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/authentication	String	The authentication result of the principal.

http://schemas.microsoft.com/ws/2008/06/identity/claims/authenticationinstant	DateTime	The instant in time when the principal was authenticated.
http://schemas.microsoft.com/ws/2008/06/identity/claims/authenticationmethod	String	The method of authentication used.
http://schemas.microsoft.com/ws/2008/06/identity/claims/expiration	DateTime	The date/time that the principal's authentication no longer is valid.
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/sid	UUID	The security identifier of the principal. This is the UUID of the user.

7.3.1.19. Policy / Privacy Enforcement Architecture

The enforcement of privacy and policies is handled through a series of services within the OpenIZ solution. From a high level, three different types of services are involved:

- **Policy Information Provider (PIP)** – Is responsible for storing information related to the policies. The information point is responsible for maintaining a list of IPolicy objects which contain the name, oid, handler (C# class which is executed upon policy decision), and elevation control.
- **Policy Decision Point (PDP)** – Is responsible for making a decision related to a policy (or series of policies) for a given securable. The decision outcome is one of the following options:
  - **Deny** – The principal has no authorization to access the requested securable or policy.
  - **Elevate** – The principal can access the securable or policy however they require additional authentication (such as 2nd level password, TFA, etc.)
  - **Grant** – The principal is granted access to the specified securable or policy.
- **Policy Enforcement Point (PEP)** – Is responsible for listening to events from the OpenIZ system and leveraging the decision and information points to enforce the policy decision. This implementation can vary between jurisdictions however by default involves either the masking (i.e. there is something here you can't see), redaction (i.e. removal of information), or partial disclosure of records.

The process for enforcement is illustrated in Figure 6.

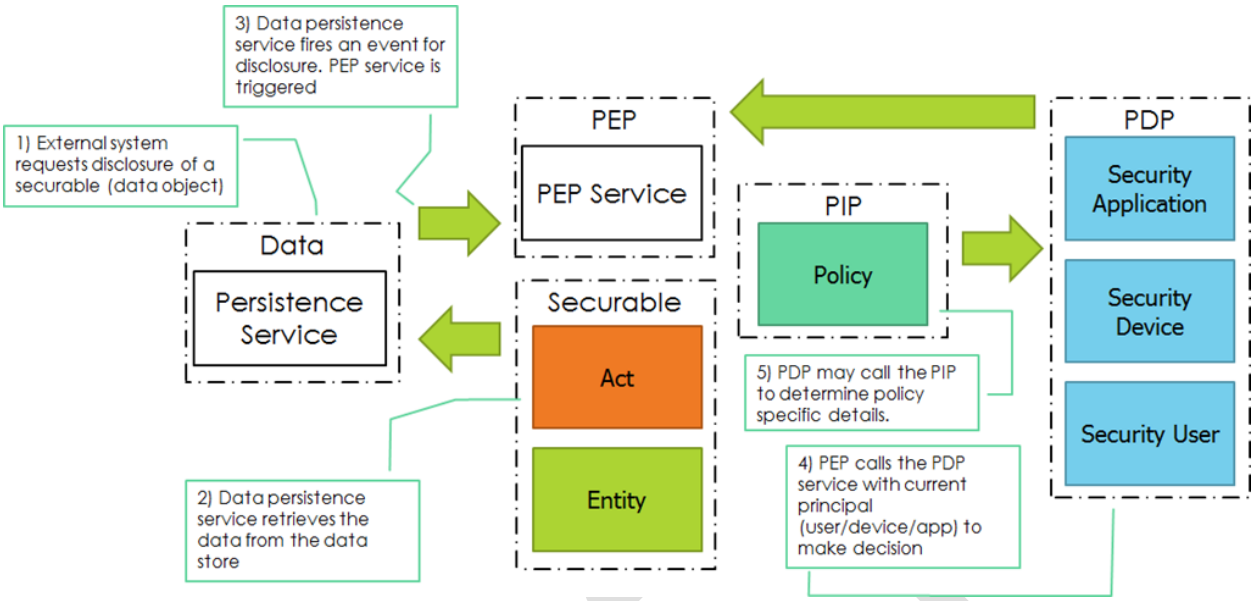


Figure 6 - Policy Enforcement Architecture

Policy enforcement may happen declaratively via enforcement of security attributes on code (most notably the PolicyPermission and PolicyPermissionAttribute classes). The default policies included in OpenIZ are listed in . The IMS is expected to be aware of all policy identifiers, clients and services accessing the IMS are merely to be aware of local policies which may have an impact on their function.

Table 9 - OpenIZ Policies

Name	OID	Description
Access Administrative Function	1.3.6.1.4.1.33349.3.1.5.9.2.0	Identities which possess this policy permission are granted access to all administrative functions of OpenIZ.
Change Password	1.3.6.1.4.1.33349.3.1.5.9.2.0.1	Allows an identity to change any other user's password.
Create Role	1.3.6.1.4.1.33349.3.1.5.9.2.0.2	Allows an identity to arbitrarily create a user role.
Alter Role	1.3.6.1.4.1.33349.3.1.5.9.2.0.3	Allows an identity to modify roles, including role membership.
Create Identity	1.3.6.1.4.1.33349.3.1.5.9.2.0.4	Allows an identity to create arbitrary identities (users).
Create Device	1.3.6.1.4.1.33349.3.1.5.9.2.0.5	Allows an identity to create arbitrary devices.
Create Application	1.3.6.1.4.1.33349.3.1.5.9.2.0.6	Allows an identity to create arbitrary applications.
Login	1.3.6.1.4.1.33349.3.1.5.9.2.1	Grants an identity the login permission.
Unrestricted Clinical Data	1.3.6.1.4.1.33349.3.1.5.9.2.2	Identities which possess this policy permission are granted

		access to all clinical functions of the OpenIZ IMS.
Query Clinical Data	1.3.6.1.4.1.33349.3.1.5.9.2.2.0	Allows an identity to execute any query against clinical data.
Write Clinical Data	1.3.6.1.4.1.33349.3.1.5.9.2.2.1	Allows an identity to create and/or update clinical data.
Delete Clinical Data	1.3.6.1.4.1.33349.3.1.5.9.2.2.2	Allows an identity to obsolete clinical data.
Read Clinical Data	1.3.6.1.4.1.33349.3.1.5.9.2.2.3	Allows an identity to fetch arbitrary records.
Override Disclosure	1.3.6.1.4.1.33349.3.1.5.9.2.3	Allows a user to override a disclosure deny.
Client Administrator	1.3.6.1.4.1.33349.3.1.5.9.2.10	Allows a user to access a client administration function.

7.3.1.19.1 Most-Restrictive Enforcement

OpenIZ's default policy decision service provider operates on a basis of most-restrictive with default DENY. In this evaluation scheme policy decisions are created as follows:

- If the principal has no data associated with the policy, the result of the decision is DENY,
- If the principal has one rule associated with the policy via role, device, or application then the result of the decision is the rule's configuration,
- If the principal has multiple rule instances configured via role, device or application then the result of the decision is the most restrictive option.

For example, John Smith (user jsmith) is a member of USERS, CLINICAL and is accessing OpenIZ from application ReaderApp.

Policy	**From USERS**	**From CLINICAL**	**From ReaderApp**	**Effective Set**
Access Administrative Function				***DENY***
Change Password				DENY
Create Role				DENY
Alter Role				DENY
Create Identity				DENY
Login	GRANT		GRANT	GRANT
Unrestricted Clinical Data		***GRANT***		***GRANT***
Query Clinical Data		GRANT (implied)		GRANT (implied)
Write Clinical Data		GRANT (implied)	DENY	DENY
Delete Clinical Data		GRANT (implied)	DENY	DENY
Read Clinical Data		GRANT (implied)		GRANT (implied)
Override Disclosure		GRANT	DENY	DENY

7.3.1.20. Report Services

It is expected that OpenIZ implementations will leverage several different types of reporting engines based on features of the report engine. For example, Jurisdiction A may choose SSRS whilst another may choose JasperReports.

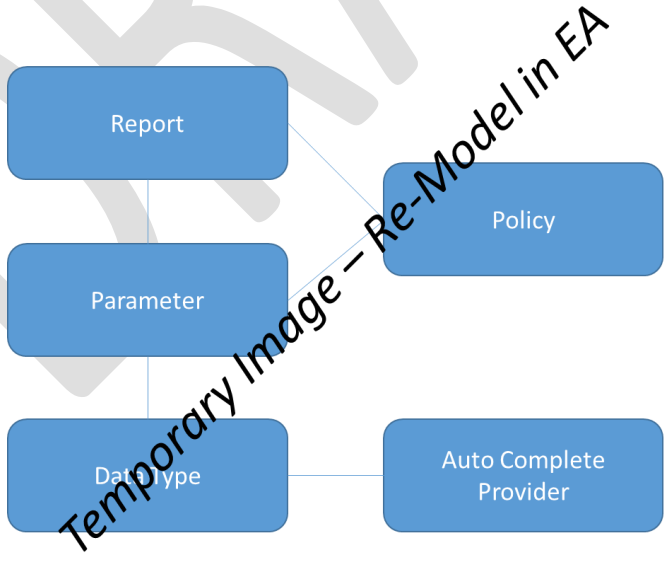
In order to ensure that client applications are given a consistent interface with which to generate reports, the OpenIZ IMS provides the IReportProvider service. This service is responsible for:

1. Exposing an enumeration of report objects which can be used by callers to determine the installed reports on the report manager.
2. Maintaining the security attributes of the report, providing a the ability to restrict report access based on policies.
3. Exposing the parameters that a particular report can accept for generation.
4. Executing the report and exposing the resultant files to callers.
5. Installing / reflecting reports to the backing report engine. For example, the IReportProvider may provide a mechanism for reflecting JRXML or RDL report files and deciphering parameters and report titles.

The IReportProviderService service is merely an extension of the IDataPersistenceService<Report> interface.

7.3.1.20.1 Model

The IReportProvider service exposes a series of canonical objects which describe the reports contained in the execution engine. Illustrates the objects exposed and their relations.



Describes the report canonical model in more detail.

Class	Property	Type	Description

Report	(N/A)	VersionedEntityData	The report class identifies a single report within the report execution engine which can be executed by a principal.
	Source	XElement	The complete source of the report. This is the RDL or JRXML, etc. which comprises the report.
	Name	String	The name of the report.
	Description	String	A long form description of the report, its intended purpose, etc.
	ProviderId	String	The identifier by which the execution engine knows the report as.
	Policy	SecurityPolicy[]	One or more policies which a user must posses in order to execute the report. These policies are AND.
	Parameters	ReportParameter[]	One or more report parameters which can be used to render the report.
ReportParameter	(N/A)	VersionedAssociation	The report parameter class identifies a single report parameter which can be applied to report.
	Name	String	The human readable name of the report parameter.
	Description	String	A long form description of the report parameter such as described in a help document.
	ProviderId	String	Identifies the id of the parameter as the report execution engine understands the report parameter.
	Order	Int32	The order in which the report parameter should be displayed.
	Policy	SecurityPolicy[]	One or more policies which the principal must posses in order to populate the parameter.
	Default	Object	The default value for the report parameter.
	DefaultProvider	IValueProvider	The IValueProvider which can be used to ascertain a default value at runtime.
	Type	ReportDataType	The type which represents the type of data in the parameter.
ReportDataType	(N/A)	IdentifiedData	The report data type class is used to describe a type of data which can be populated into a report. This can be simple types like: Date, String, etc. or complex types like

			SecurityUserSelector, VillageSelector, etc.
	Name	String	A human readable name for the report data type.
	SystemType	Type	The underlying system type which this datatype represents.
	Description	String	The description of the report data type.
	Values	Object[]	One or more values which are acceptable for the report parameter type.
	ValuesProvider	IValueProvider	Represents a value provider which can be used in lieu of a static list of allowed values. This is often used for validating report parameter values as well as a source for auto-complete information.

7.3.1.20.2 Value Providers

A value provider represents a simple implementation of the IValueProvider interface which is used by report parameters and parameter types to either provide values for an auto-complete list, default value, etc.; or validate a value given by a caller. IValueProvider extends the IEnumerable<Object> interface.

The methods provided by a value provider are listed in .

Method	Return	Parameters	Description
GetEnumerator()	IEnumerator<Object>	None	Gets a list of allowed or acceptable values based on the current authentication context.
Validate	Bool	Object	Validates that the provided object is valid according to the current security context.

7.3.2. OpenIZ's Administration & Configuration Architecture

TODO:

- Discuss deployment
- Discuss cloud controller
- Discuss concept of REALM

7.3.2.1. Administration Management Interface (AMI)

The administration management interface (AMI) is used to control and harmonize the configuration of OpenIZ instances. The AMI function is three fold:

1. **Cloud Control** – The AMI is responsible for distributing the configuration of an OpenIZ cluster (example: in a private cloud) across application servers.
2. **Configuration –** The AMI is used by an administration tool to control the service interface and configuring an instance of OpenIZ.

3. **Device Management –** The AMI is used by administrators to on-board new devices, manage their security certificates.

The service represents a hybrid of configuration, management of the Security* tables (applications, devices, etc.) as well as control over security certificate services. The AMI service uses CAS to ensure that the access to the service is only performed by administrators with appropriate functions. The exception to this rule is are the security user functions for update whereby a user may update their own information.

7.3.2.1.1 Operations

Illustrates the operations that are supported by the administration management interface (AMI).

Resource	Operation	Description
csr/	POST	Submits a new certificate signing request (CSR) to the administrative management interface. Depending on the configuration for the service the CSR may be approved automatically.
	GET	Searches the application management interface for existing certificate signing requests.
csr/{id}	DELETE	Rejects the specified CSR
	PUT	Updates the CSR. This mechanism is used to accept the CSR and generate a public key.
	GET	Gets the specified certificate signing request.
certificate/	POST	Posts a new certificate directly to the certificate manager.
	GET	Performs a search of all accepted certificates.
certificate/{id}	DELETE	Revokes the specified certificate placing it in the CRL.
	GET	Gets the specified certificate binary.
Crl	GET	Gets the current certificate revocation list (CRL).
configuration/	GET	Gets the current full service configuration for the service cluster.
configuration/{section}	PUT	Updates the specified configuration section.
	GET	Gets the specified configuration section.
configuration/plugin	POST	Indicates that a new system wide plugin should be installed on next startup of IMS hosts.
	GET	Gets a list of plugins on the specified IMS host.
configuration/plugin/{id}	PUT	Updates the plugin manifest for the specified plugin identifier.

	DELETE	Deletes the specified plugin from the IMS host.
	GET	Gets the plugin manifest for the identified plugin.
configuration/plugin/{id}/bin	POST	Uploads the binary assembly to the IMS host.
	DELETE	Deletes the specified plugin binary.
	GET	Gets the specified plugin binary.
device/	POST	Creates a new SecurityDevice entry in the IMS' configuration.
	GET	Searches the IMS' data store for security devices.
device/{id}	PUT	Updates the specified security device information.
	DELETE	Deletes the specified security device information.
	GET	Gets the identified security device.
application/	POST	Creates a new SecurityApplication entry in the IMS's host configuration.
	GET	Gets the specified security application information from the IMS configuration store.
application/{id}	PUT	Updates the specified application manifest.
	DELETE	Deletes the specified application identifier.
	GET	Gets the specified security application information.
user/	POST	Creates a new SecurityUser. Note that the security principal on the request must have the CreateIdentity policy.
	GET	Searches the IMS' data store for SecurityUsers.
user/{id}	PUT	Updates the specified security data store.
	DELETE	Deletes the specified security user from the identity provider.
	GET	Gets the specified user information from the identity provider.
role/	POST	Creates a new role in the role provider.
	GET	Searches the role provider for the specified roles.
role/{id}	PUT	Updates the specified role.
	DELETE	Deletes the specified role.
	GET	Gets the specified role data.
policy/	POST	Creates a new policy in the data store.
	GET	Performs a search of the current policy information provider's policies.
policy/{id}	PUT	Updates the specified policy information (if permitted)

	DELETE	Deletes the specified policy (if permitted)
	GET	Gets the detailed policy information provided.

7.4. Communications / Interoperability Architecture

The primary mechanism of communication with the OpenIZ IMS is via the interoperability and communications layer. These services allow multiple user interfaces, or point of service devices, to connect with the IMS. The overall communications architecture for the IMS is outlined in

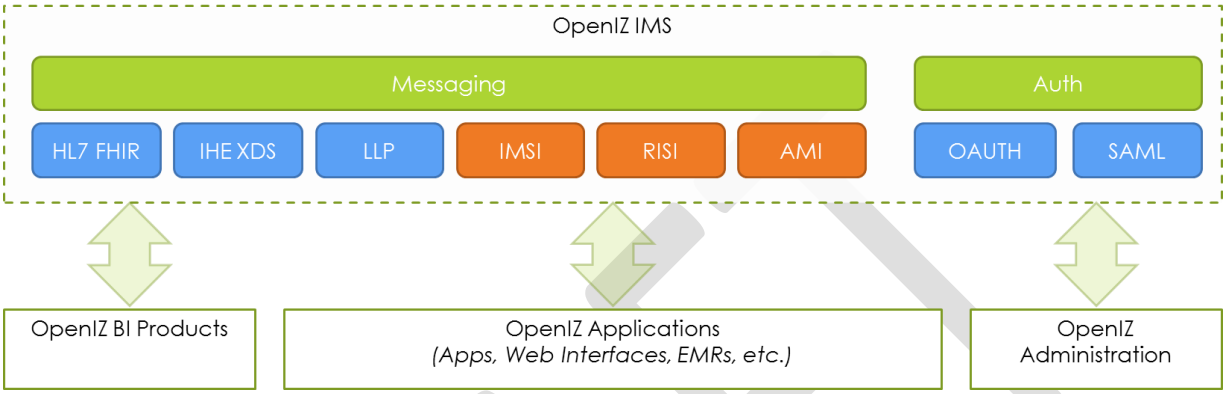


Figure 7 - IMS Messaging Interfaces

The interfaces are detailed in Table 10.

Table 10 - IMS Messaging Interfaces

Interface	Standards	Version	Description
IMSI	Proprietary	1.0	The Immunization Management Service Interface provides a raw 1-1 mapping between REST and the backing IMS data store.
RISI	Proprietary	1.0	The Report Integration Services Interface provides a direct aggregate view of the backing database as well as queries supporting reporting services such as MS-SSRS or JasperReports. The RISI interface is readonly and intended to be used for BI purposes only.
AMI	Proprietary	1.0	The Administration Management Interface allows access to a variety of administrative functions including restarting services, deploying plugins, configuration and meta-data editing. It has no access to the clinical store and is available to administrators only.
LLP	HL7 V2 over MLLP	2.5	The HL7v2 interface provides ADT, QBP, VXQ, VXU and SRM messaging support and is intended to provide a bridge to IMS functionality.

XDS	IHE XDS + IHE IC	XDS.b	The XDS.b interface supports the export and import Immunization Content (IC) CDA documents.
FHIR	HL7 FHIR	1.0.2	The HL7 FHIR interface supports the manipulation of IMS resources using the HL7 FHIR standard.

7.4.1. Communicating with the IMS

All interfaces to the IMS are secured with one of the allowed authorization schemes. These include SAML, OAUTH or Basic AUTH. Communications with the IMS protocols are made using the Windows Communication Foundation (WCF) framework and/or the NHAPI HL7v2 framework for LLP.

There are six interfaces that the IMS will eventually support:

- Immunization Management Service Interface (IMSI) – A proprietary, REST based interface which provides a near 1-to-1 mapping of clinical functionality in the IMS to clients.
- Report Integration Services Interface (RISI) – A proprietary, REST based interface which provides a series of interfaces for acquiring the location and parameterization of system reports for use in the IMS.
- Administration Management Interface (AMI) – A proprietary, SOAP based interface which is used to perform administrative operations on the nodes in an OpenIZ cluster.
- HL7® FHIR™ - Fast Health Interoperability Resources (FHIR) is a standardized REST interface for interacting with the IMS.
- HL7® Version 2 – HL7v2 interfaces provide VXU capability within the IMS and are implemented per CDC specification for immunization reporting.
- HL7® CDA – Interface which provides and imports CDA documents containing immunization summaries.

7.4.2. Immunization Management Service Interface (IMSI)

The IMSI interface provides a 1-to-1 mapping of business model artifacts to a wire level representation. The IMSI is primarily designed for clients which will require sending and receiving large amounts of RAW data from the IMS for purposes like offline synchronization.

The IMSI does no “messaging” of data and data values in the underlying business data model are exposed. The IMSI interface’s principles are:

- Provide referentially valid data to clients as it exists in the IMS data store.
- Accept referentially valid data from clients as it was presented to the IMS.
- Faithfully store and reproduce the objects received on the wire with minimization of duplication.

7.4.2.1. Transport

The IMSI service is a RESTful service which exposes. The semantics REST API is described in .

Operation	URL	Method	Description
Create	/{resource}	POST	Instructs the IMS to create the specified resource.
Create / Update	/{resource}/{id}	POST	Instructs the IMS to update the specified resource if it exists or create it if it doesn’t.

Update	/{resource}/{id}	PUT	Instructs the IMS to update the specified resource.
Query	/{resource}?{query}	GET	Instructs the IMS to perform the specified query.
Read	/{resource}/{id}	GET	Instructs the IMS to fetch the most recent version of the specified resource.
History Query	/{resource}/{id}/history	GET	Instructs the IMS to fetch all history of the specified item.
History Read	/{resource}/{id}/history/{vid}	GET	Instructs the IMS to fetch a specific version of the record.
History	/{resource}/history	GET	Instructs the IMS to fetch all changes performed on the specified resource.
Obsolete	/{resource}/{id}	DELETE	Instructs the IMS to obsolete the specified resource.

7.4.2.2. Response Codes

The IMSI will respond with a series of HTTP response codes which detail the outcome of the response. In addition to the HTTP response code, an object will be returned which allows the client to compute the reason why the HTTP error was returned.

Lists the response codes and their meaning in the context of the ISMI.

Code	**Error**	**Description**
200	OK / No Error	This error condition indicates that the entire operation succeeded and the response represents the desired operation.
201	Created	This status code indicates that the resource was created on the server and additional processing may occur.
302	Moved	This status code indicates a redirect. This is used when a request is made to an IMSI interface on a server where a remote IDataPersistence is configured (such as in a load balancing or migration scenario).
400	Bad Request	This response code indicates that there was no possible way for the IMSI to comprehend the request sent to it. This is typically done when a low level processing instruction fails (such as bad compression data)
401	Unauthorized	This response code indicates that the requested resource requires permissions which are above the current permission set of the user. This may indicate that the current user is ANONYMOUS and is trying to access a protected resource, OR may indicate a desire by the IMSI interface for the client to elevate themselves.
403	Forbidden	This response code indicates that the authorized user is not permitted to access the requested resource. This represents a full DENY policy decision.

404	Not Found	This response code indicates that the resource requested cannot be found.
405	Method not allowed	This response code indicates that the client attempted to perform an operation on the IMSI interface which is not permitted.
	Conflict	This response code indicates that an update failed because of a formal validation constraint.
410	Gone	This response code indicates that the resource being requested DID exist, however has since been obsoleted.
415	Unsupported Media Type	This response code indicates that it is not possible for the IMSI to process the request content. This error typically occurs when the client is submitting data which is not JSON nor XML.
500	Server Error	This error indicates that the server encountered an error while trying to perform the operation.
503	Service Unavailable	This error indicates that the IMSI service is temporarily unavailable due to startup or partial startup.

7.4.2.3. Resource Types

The IMSI exposes resources contained in the underlying business model API layer. In addition to these resources, the IMSI exposes several "meta" resources which wrap complexities around appointment/scheduling. The IMSI specific resources are found in the http://openiz.org/imsi rather than the http://openiz.org/model namespace.

Table 11 - Resource Types

Resource	Namespace	Description
Concept	http://openiz.org/model	Represents a concept such as "arm" or "OPV".
ReferenceTerm	http://openiz.org/model	Represents the wire representation of a concept such as a CVX or ICD code.
Act	http://openiz.org/model	Represents a general action that is or was performed such as a concern.
Observation	http://openiz.org/imsi	A classification of Act which represents the observing of some value. Sub-classed resources are TextObservation, CodedObservation, and QuantityObservation.
PatientEncounter	http://openiz.org/model	A classification of Act which represents an encounter that the patient has or will have with a health provider.
SubstanceAdministration	http://openiz.org/model	Represents the administration of a substance to a patient.

Entity	http://openiz.org/model	A generic class representing an unclassified entity.
Patient	http://openiz.org/model	A classification of an entity representing a Patient.
Provider	http://openiz.org/model	A classification of an entity representing a healthcare provider.
Organization	http://openiz.org/model	A classification of an entity representing an organization.
Place	http://openiz.org/model	A classification of an entity which represents a place where health services are delivered.
Material	http://openiz.org/model	A classification of entity which represents some sort of material such as a kit, classification of drug or boxed item.
ManufacturedMaterial	http://openiz.org/model	A classification of entity which represents a manufactured material. Manufactured materials are those acquired from a manufacturer.
Bundle	http://openiz.org/model	A collection of OpenIZ elements bundled for the purpose of transporting referenced objects.
ConceptSet	http://openiz.org/model	A series of concepts which are grouped together for some purpose such as roles, statuses, etc.
Drug	http://openiz.org/imsi	A classification of manufactured material which represents an administrable drug.
Vaccination	http://openiz.org/imsi	A classification of substance administration which represents the administration of a vaccine.
Appointment	http://openiz.org/imsi	A classification of PatientEncounter which represents the intent or proposal to have an encounter.
AdverseReaction	http://openiz.org/imsi	A classification of Act representing a concern (wrapped in observation) which describes an adverse reaction to some agent.
ConceptRelationshipType	http://openiz.org/model	Represents classifications of the concept relationship types.

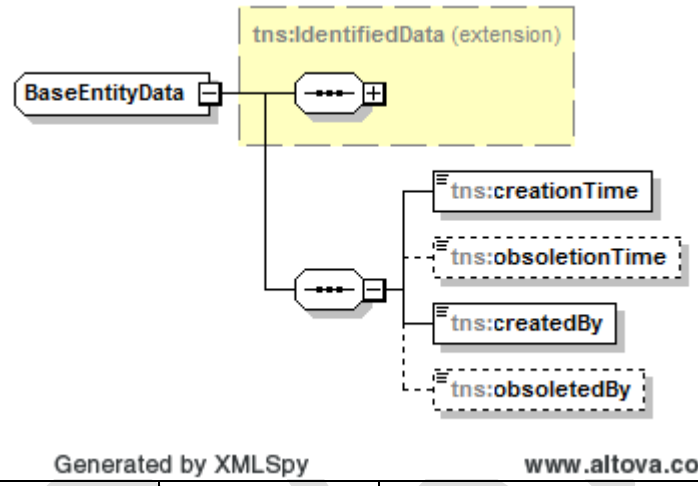
PhoneticAlgorithm	http://openiz.org/model	Represents phonetic algorithm information.

7.4.2.3.1 Base Types

The IMSI resources extend a series base classes which, while not resources per se, are included here to ensure the type definitions which follow are more concise.

Table 12 provides a summary of the base entity data class. The base entity data class is use to convey data which is related to a data entity where the data is attributed.

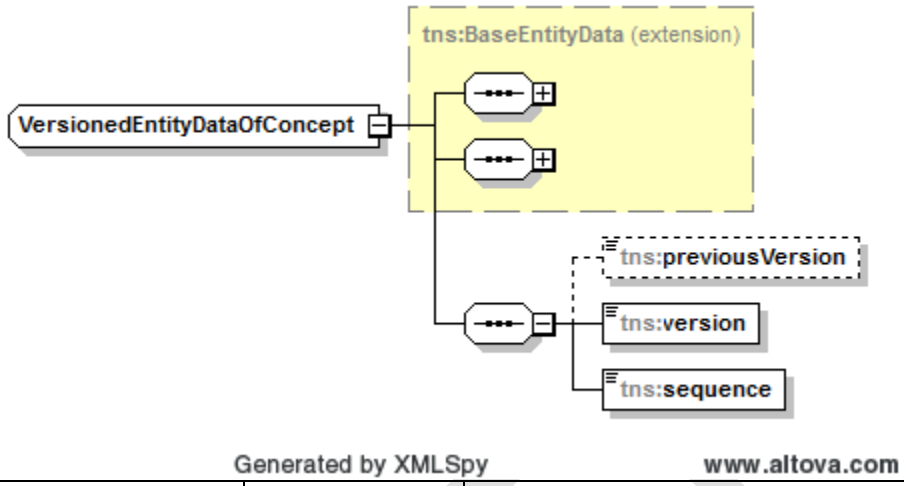
Table 12 - Base Entity Data



Element	Type	Description
id [1..1]	UUID	Uniquely identifies the business model entity.
creationTime [1..1]	DateTime	Identifies the full timestamp (from the service) when the specified data was created.
obsoletionTime [0..1]	DateTime	When present, indicates the date/time that the entity became or will become obsolete.
createdBy [1..1]	UUID	Identifies the user who was responsible for the creation of the data model entity.
obsoletedBy [0..1]	UUID	Identifies the user who was responsible for the obsoletion of the data model entity.

Table 13 illustrates the versioned entity data class is a generic class and indicates common data elements which are required for entities which are versioned in the OpenIZ system.

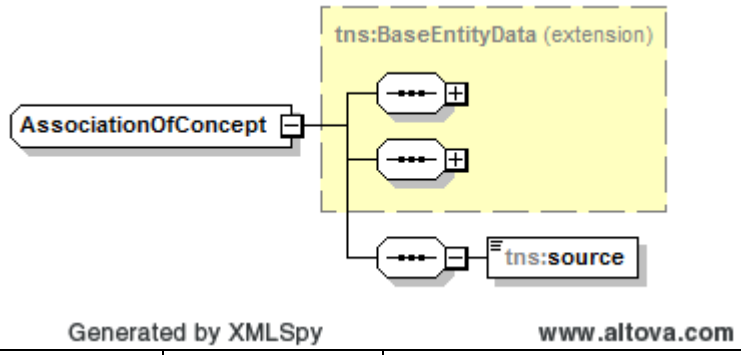
Table 13 - Versioned Entity Data



Element	Type	Description
previousVersion [0..1]	UUID	The version identifier of the previous version of this entity.
version [1..1]	UUID	The unique identifier for the represented version.
sequence [1..1]	int	An incrementing serial number which is used to identify the order in which the version was created. This number is not consistent between systems in value, however must be consistent in ordering.

Table 14 conveys how the association class is used to track a simple association between a source and the current item. The association classes point to the source of their association.

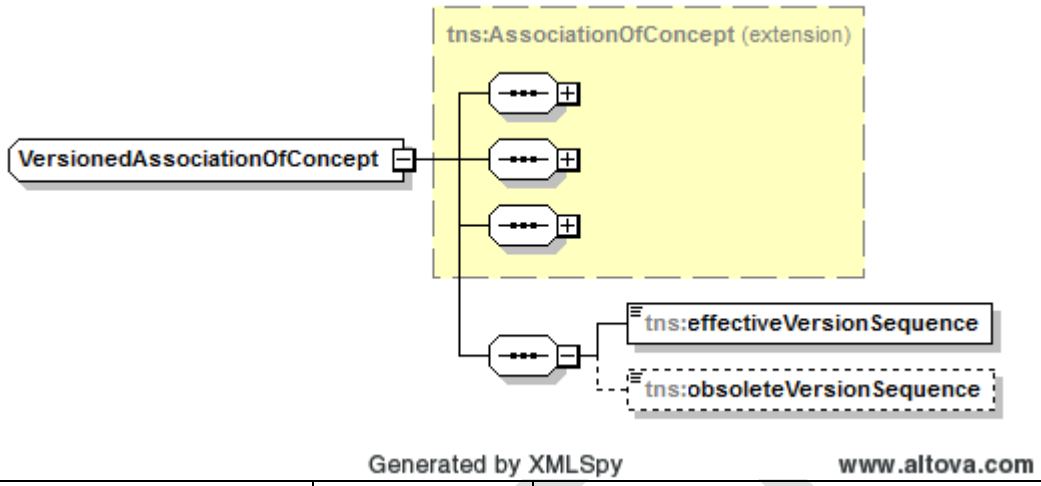
Table 14 - Association



Element	Type	Description
source [1..1]	UUID	The source data entity to which the associative class is associated.

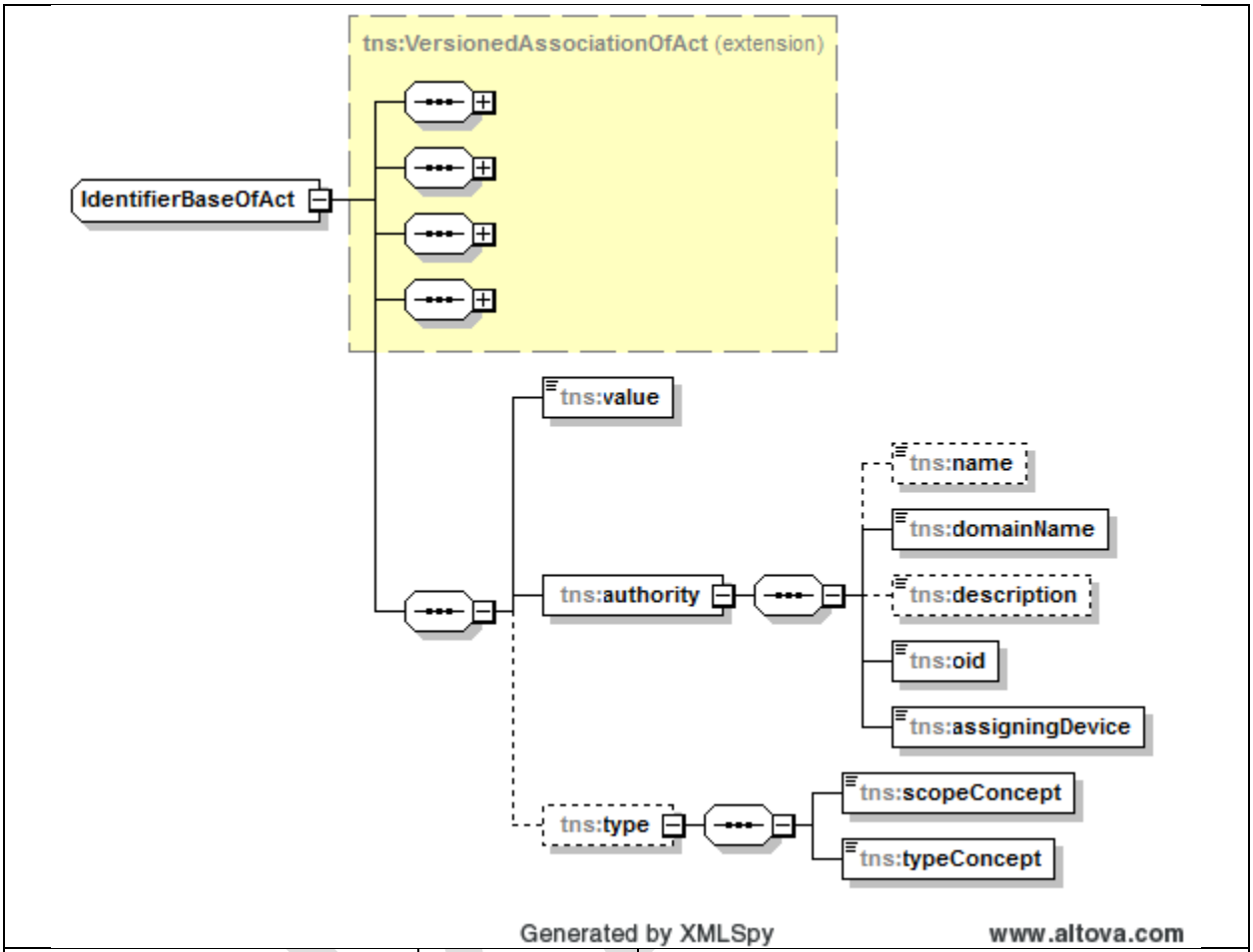
The versioned association class represents a special association whereby the associative entity is applied to a specified series of versions (Table 15).

Table 15 - Versioned Association



Element	Type	Description
effectiveVersionSequence [1..1]	int	The effective version of the ***source*** entity to which the associative entity is active.
obsoleteVersionSequence [0..1]	int	The version of the ***source*** entity where the associative entity is no longer active.

Alternate identifiers (identified as element “identifier”) are represented in entities as illustrated in .



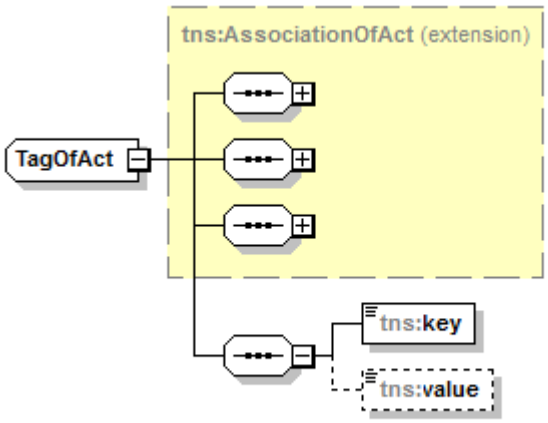
Element	Type	Description
value [1..1]	String	The value of the identifier as assigned by the specified authority.
authority [1..1]	Authority	Represents information about the source authority from which the identifier is assigned.
authority.name [0..1]	String	The optional name of the authority. For example: Good Health Hospital Systems
authority.domainName [1..1]	String	The domain name of the authority. In HL7v2 speak the CX.4 value. Example: GHHS
authority.description [0..1]	String	A human readable description of the assigning authority.
authority.oid [1..1]	String	The OID of the authority. Example: 1.2.3.4.5
authority.assigningDevice [0..1]	UUID	The identifier of the SecurityDevice which is allowed to assign identifiers.
type [0..1]	Type	Identifies the type of the identifier. Example: Business, Stock, etc.
type.scopeConcept [1..1]	UUID	Identifies the allowed scope of identifier use. This maps to the classCode on Entity and Act classes where the identifier can be used.

type.typeConcept [1..1]	UUID	Identifies the type of identifier. Example: Stock, Business, etc.

7.4.2.3.2 Extensions & Tags

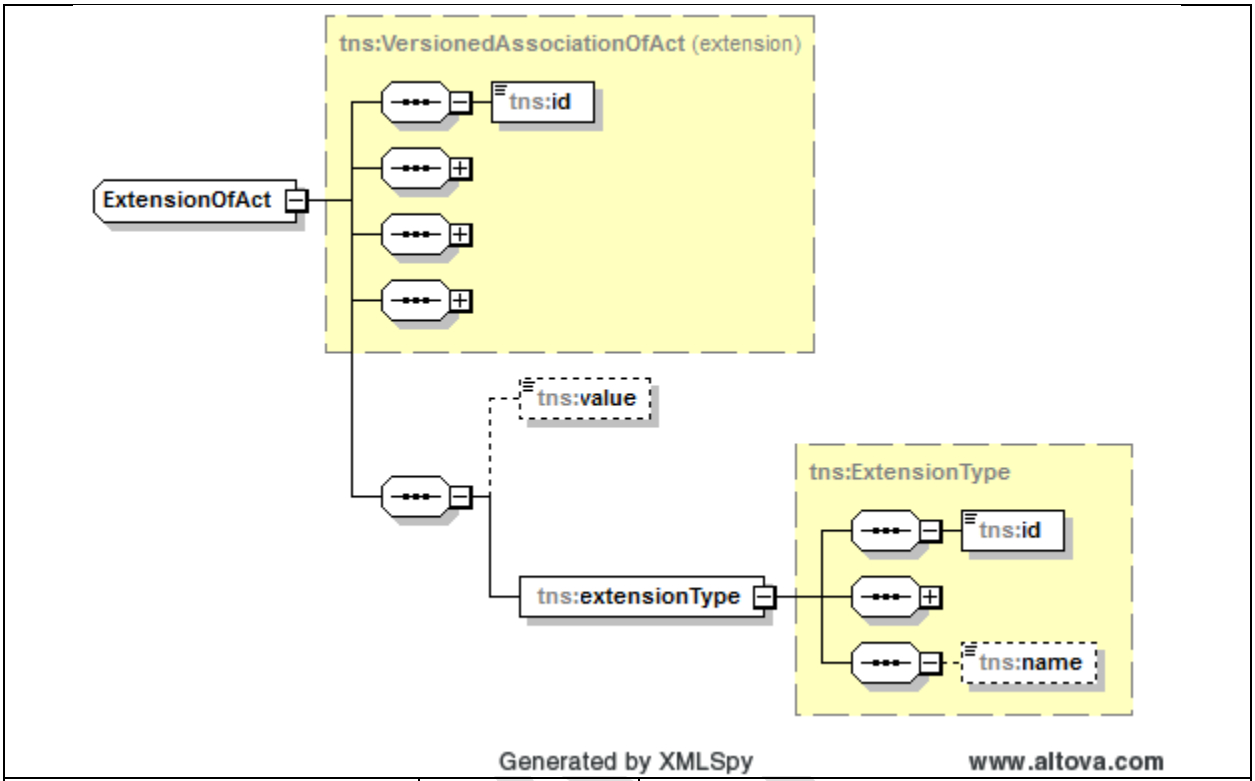
The IMSI interface represents the extension and tag values stored in the underlying data model. Extensions and tags are used to extend IMS data model classes to support use cases not envisioned in the original design.

Tags are version independent data elements which are attached to acts and entities and used for things such as workflow control, and security.



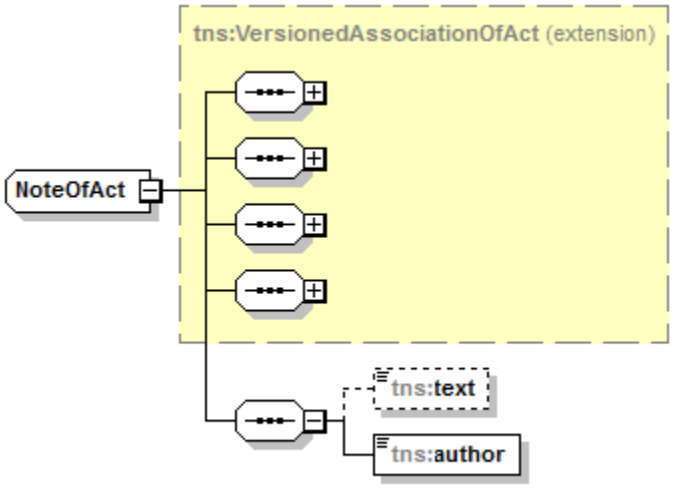
Element	Type	Description
key [1..1]	String	Identifies the type of tag applied to the act/entity.
value [0..1]	String	Carries the value of the tag. Some tags may be indicator flags (i.e. the presence of the tag indicates something).

Extensions are used to represent data elements which are associated with the clinical meaning of an act or entity. Unlike tags, extensions can carry more robust data and typically result in new versions of the associated entity and acts.



Element	Type	Description
value [0..1]	Base64binary	The serialized value of the extension represented in the instance.
extensionType [1..1]	Extension Type	The type of extension. Indicates how the value of the extension should be serialized and/or parsed and interpreted.
extensionType.id [1..1]	UUID	Uniquely identifies the extension type.
extensionType.name [0..1]	String	The human readable name of the extension type.

Notes are textual information which are intended to be displayed verbatim to human participants. Notes, like tags, are version independent.

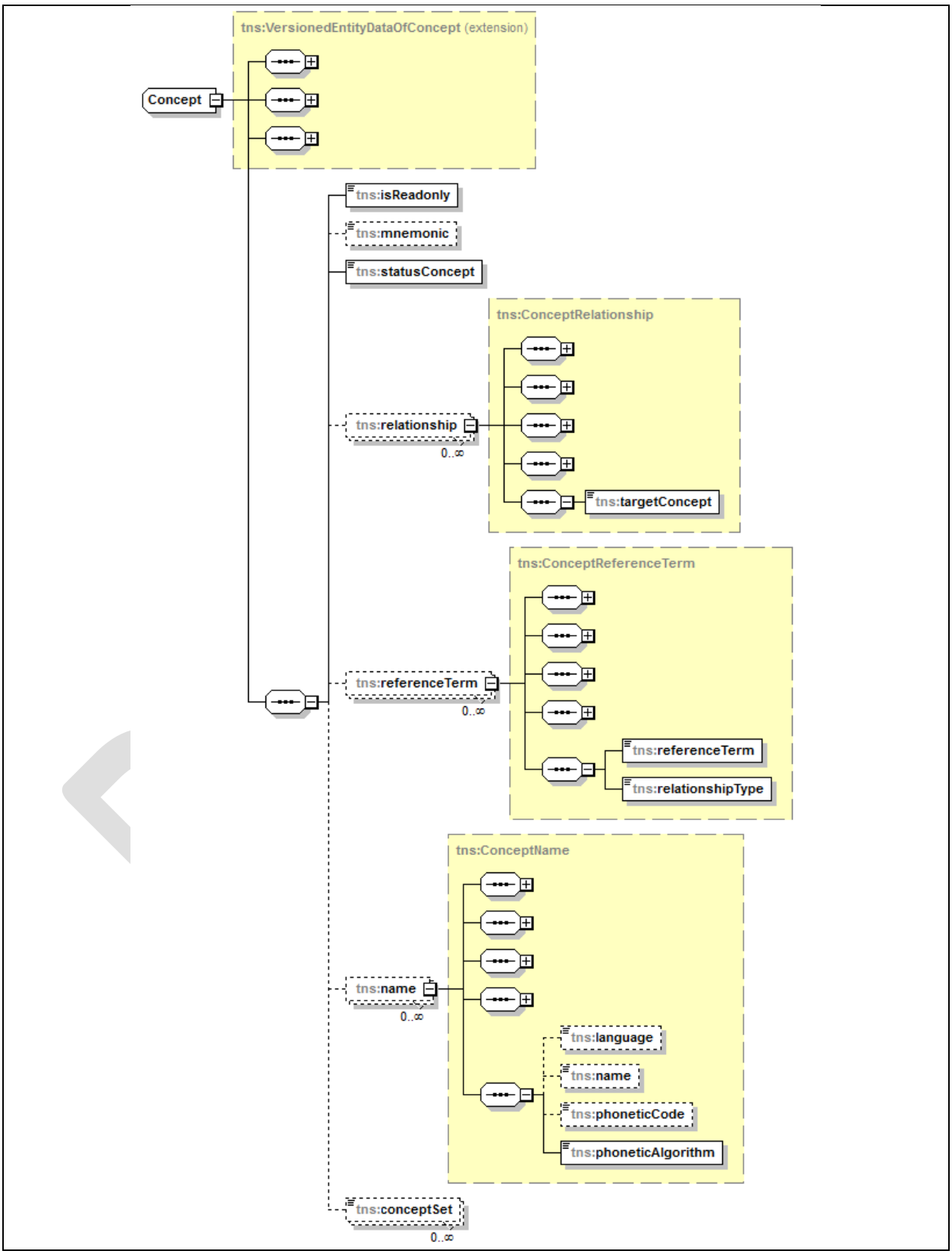


Element	Type	Description
text [0..1]	String	The textual content of the note attached to the act or entity.
author [1..1]	UUID	Identifies the author of the note.

7.4.2.3.3 Concept

The concept resource can be used to get, search, fetch history, create, update and obsolete clinical concepts used in the OpenIZ IMS system. Concepts are used to represent abstract facets of care delivery. For example: arm, Polio Vaccine, Allergy, etc.

Table 16 - Concept

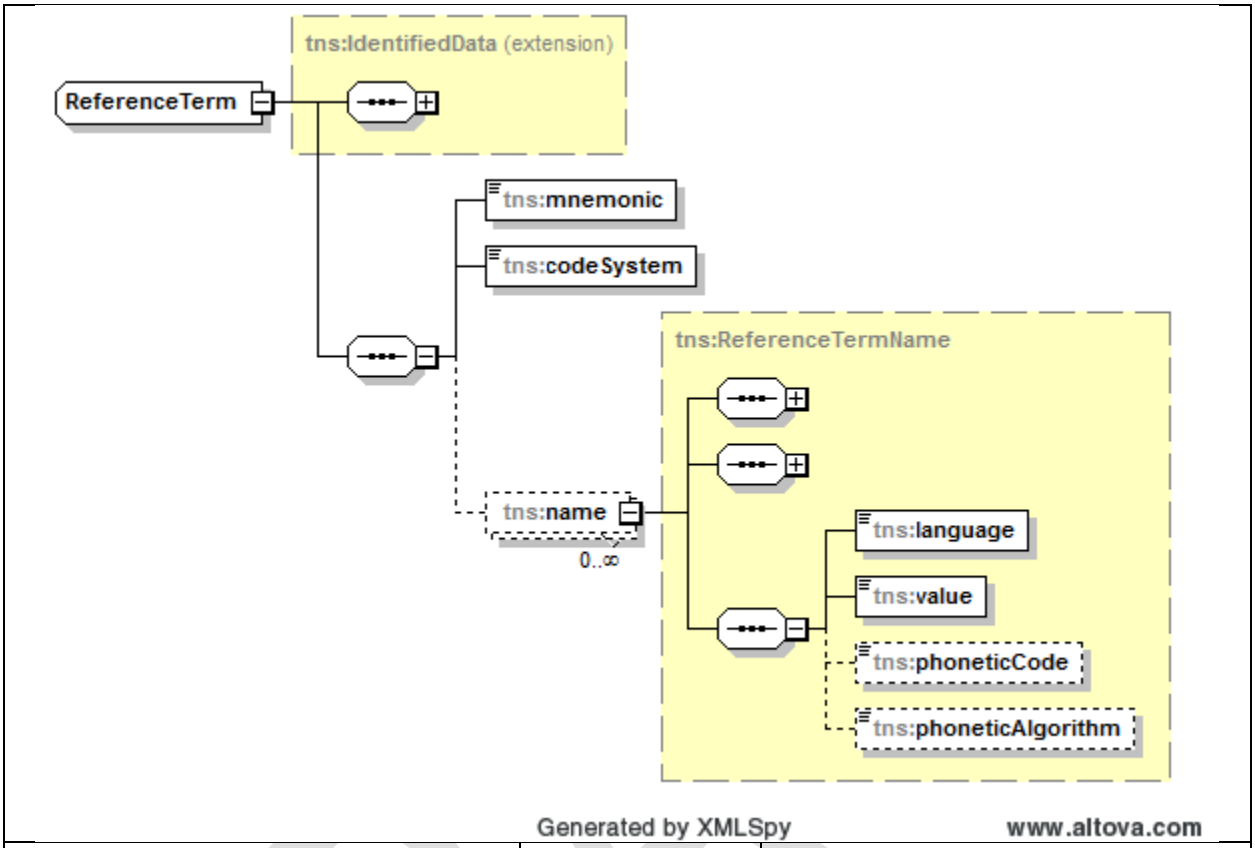


Element	Type	Description
isReadonly [1..1]	bool	Indicates whether the concept is readonly (i.e. no changes can be made)
mnemonic [0..1]	String	A invariant string which can be used to reference the concept in queries.
statusConcept [1..1]	UUID	The identifier of the status concept which represents the current status of the concept as of the represented version.
relationship [0..*]	Relationship	One or more relationships that this concept has with other concepts.
relationship.targetConcept [1..1]	UUID	Identifies the target concept which this concept is associated with.
relationship.relationshipType [1..1]	UUID	Identifies the concept relationship type (same-as, inverse-of, etc.)
referenceTerm [0..*]	ReferenceTerm	One or more reference terms (wire terms) which can be used to represent the concept.
referenceTerm.term [1..1]	UUID	Identifies the term identifier which contains
referenceTerm.relationshipType [1..1]	UUID	Identifies the type of relationship the concept reference term has to the concept. Example: narrower-than, same-as.
name [0..*]	Name	One or more names which can be displayed to represent the concept.
name.language [1..1]	String	The language code in which the name value is represented.
name.value [1..1]	String	The value of the name which can be displayed.
name.phoneticCode [0..1]	String	The phonetic code (for searching) of the name.
name.phoneticAlgorithm [1..1]	UUID	The identifier of the phonetic algorithm which can be used to generate the phonetic code value.

7.4.2.3.4 ReferenceTerm

The reference term resource can be used to create, search, obsolete, and get information related to reference terms used in the OpenIZ system. Reference terms are used to represent concepts on the wire and are primarily referenced for interoperability reasons.

Table 17 - Reference Term

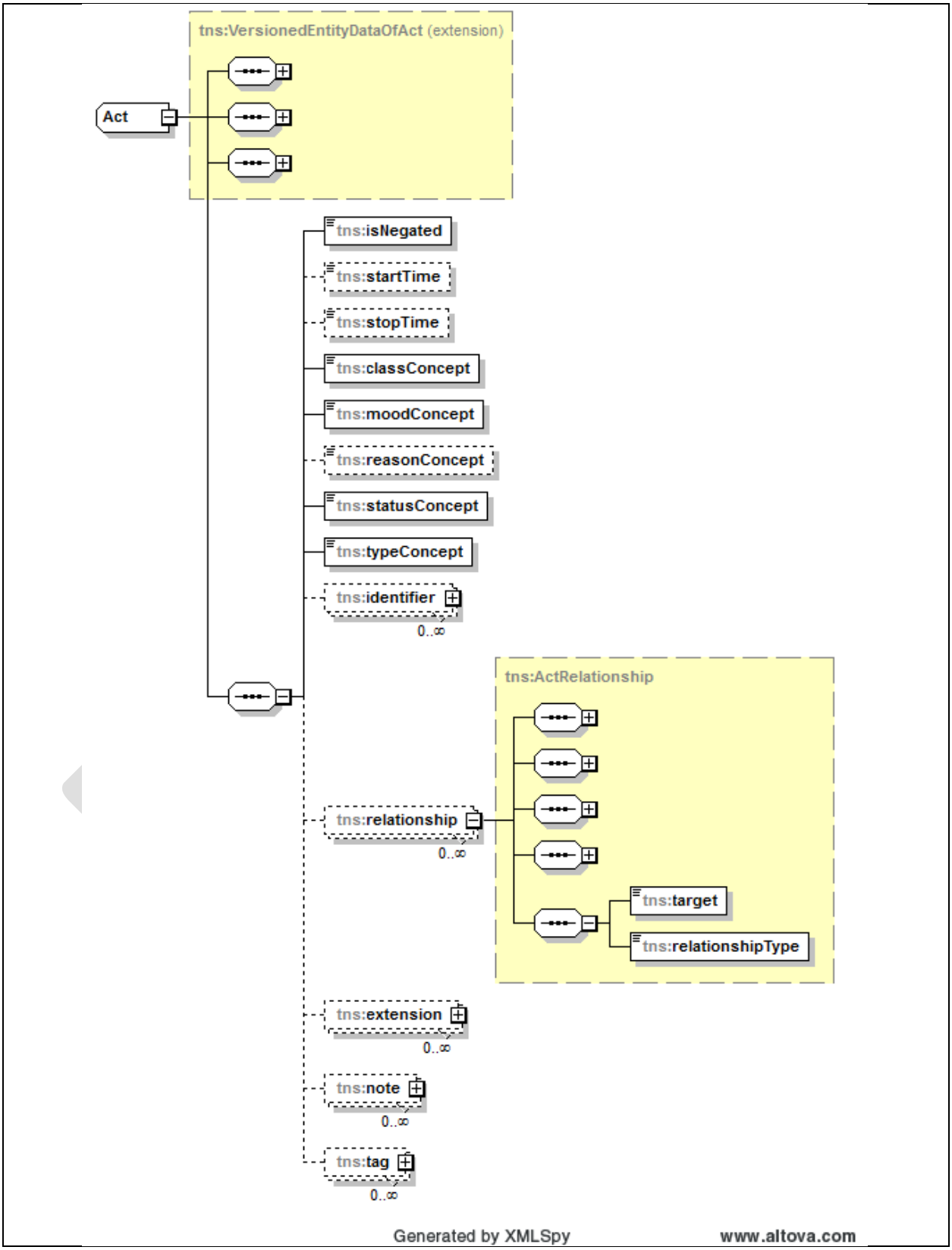


Element	Type	Description
mnemonic [1..1]	String	A mnemonic which is used to represent the reference term on the wire.
codeSystem [1..1]	UUID	Identifies the code system to which the reference term belongs (example: ICD10, LOINC, etc.)
name [0..*]	Name	One or more displayable names from the reference term's standard. These display names are used in interoperability contexts.
name.language [1..1]	String	The ISO language code in which the value of the name is represented.
name.value [1..1]	String	The value of the reference term name in the specified language.
name.phoneticCode [0..1]	String	The phonetic code of the name.
name.phoneticAlgorithm [0..1]	UUID	The phonetic algorithm identifier used to generate the phoneticCode.

7.4.2.3.5 Act

Identifies acts which cannot be classified into one of the concrete act types (observation, substance administration, etc.).

Table 18 - Act


tns:VersionedEntityDataOfAct (extension)
Act
tns:isNegated
tns:startTime
tns:stopTime
tns:classConcept
tns:moodConcept
tns:reasonConcept
tns:statusConcept
tns:typeConcept
tns:identifier
0..∞
tns:ActRelationship
tns:relationship
0..∞
tns:target
tns:relationshipType
tns:extension
0..∞
tns:note
0..∞
tns:tag
0..∞
Generated by XMLSpy
www.altova.com

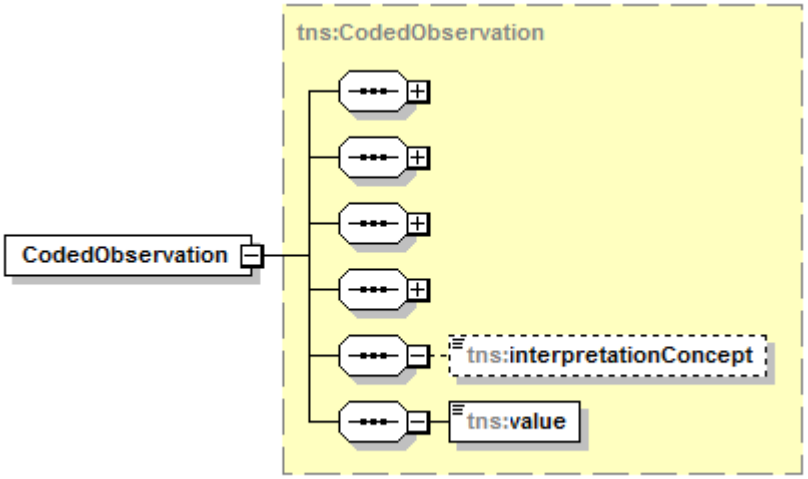
Element	Type	Description
isNegated [1..1]	Boolean	Indicates whether the act which is represented in the response is a negation. Example: OPV NOT given.
actTime [1..1]	DateTime	Identifies the time that the act occurred.
startTime [0..1]	DateTime	Identifies the time when the act started or is scheduled to start.
stopTime [0..1]	DateTime	Identifies the time when the act ended or is scheduled to end.
classConcept [1..1]	UUID	Identifies concept which classifies the act. For example: classifies observations, administrations, etc.
moodConcept [1..1]	UUID	Identifies the mood of the act. For example: classifies events from intent to perform acts.
reasonConcept [0..1]	UUID	Identifies a codified reason as to why the act took place (or didn't)
statusConcept [1..1]	UUID	Identifies the codified status of the act at the current status.
typeConcept [1..1]	UUID	Identifies the type of act. Further classifies the act (i.e. makes observation a reaction observation)
identifier [0..*]	Identifier	Provides one or more alternate identification schemes for the act.
relationship [0..*]	Relationship	Provides one or more relationships between the current act and other acts. This is used to link acts together by encounter.
relationship.target [1..1]	UUID	The target act of the relationship.
relationship.relationshipType [1..1]	UUID	Identifies the type of relationship such as "component of", "subject of", etc.
extension [0..*]	Extension	One or more extensions which are used to extend the clinical data contained in the act.
note [0..*]	Note	One or more notes which are used to provide human readable data related to an act.
tag [0..*]	Tag	One or more tags which are used to further classify an act.

7.4.2.3.6 Observation

The observation resource is an IMSI specific observation class which encapsulates the functionality of the three types of observations for convience. Observation allows a consumer to quickly query any three of the observation types.

A coded observation represents an observation whose value is a concept. This type of observation is used whenever an observation is made which can be codified. For example: Observed reaction, severity, etc.

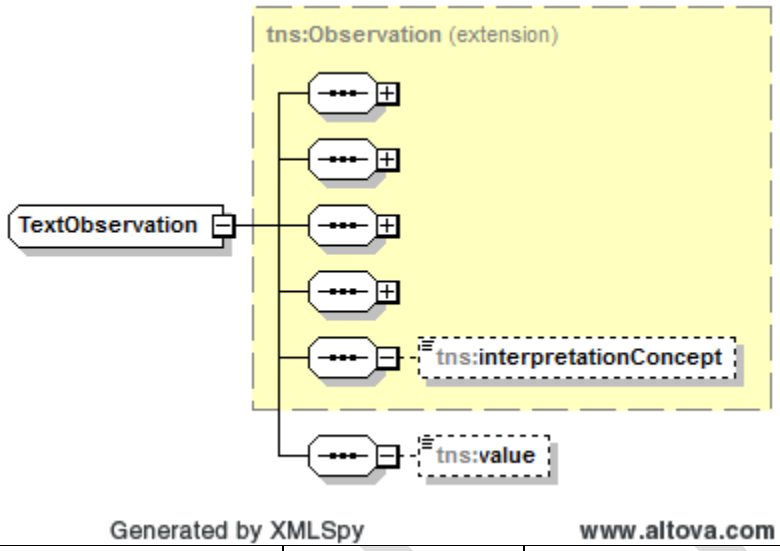
Table 19 - Coded Observation



Element	Type	Description
interpretationConcept [0..1]	UUID	A concept which provides an interpretation of the observation's value (High, Low, Nominal, etc.)
value [1..1]	UUID	The identifier of the code which represents the value of the observation.

A text observation is used to store an observation whose primary value is textual. These observations are not intended to be used by processing routines, rather are intended for human consumption.

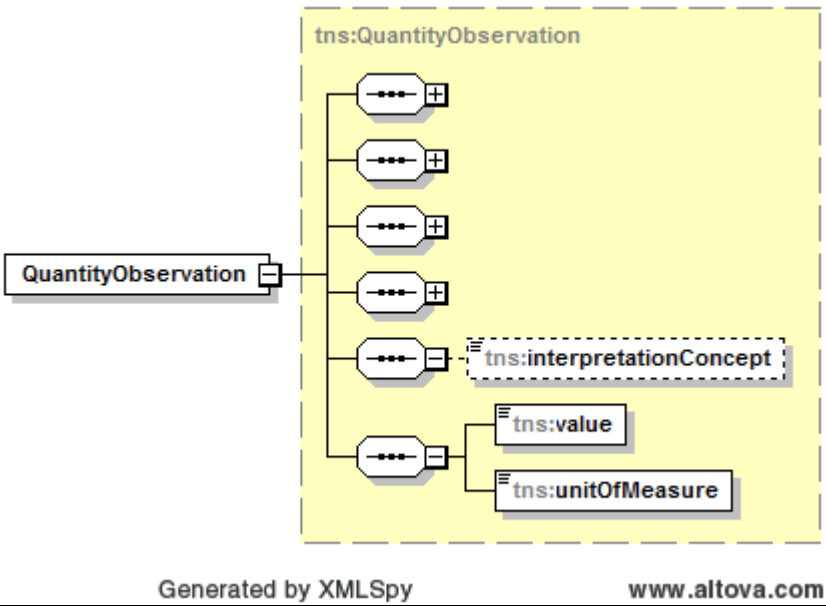
Table 20 - Text Observation



Element	Type	Description
value [0..1]	String	The textual value of the observation.

Finally, quantity observations are used to convey an observation where the observation value was obtained by quantitative measurement. For example: weight, height, bmi, etc.

Table 21 - Quantity Observation

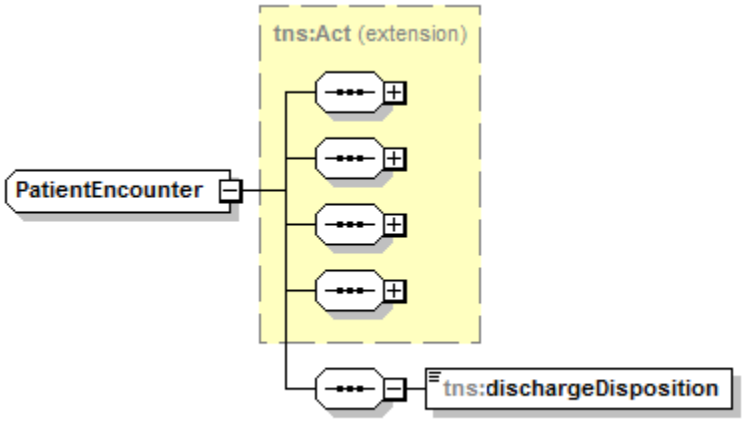


Element	Type	Description
Value [1..1]	Decimal	The numerical value of the observation.
unitOfMeasure [1..1]	UUID	A concept identifier which indicates the units of the value measure.

7.4.2.3.7 PatientEncounter

Patient encounters are used as a means to convey information related to an encounter which the patient has with the health system or is intended to occur (in the case of an appointment).

Table 22 - Patient Encounter

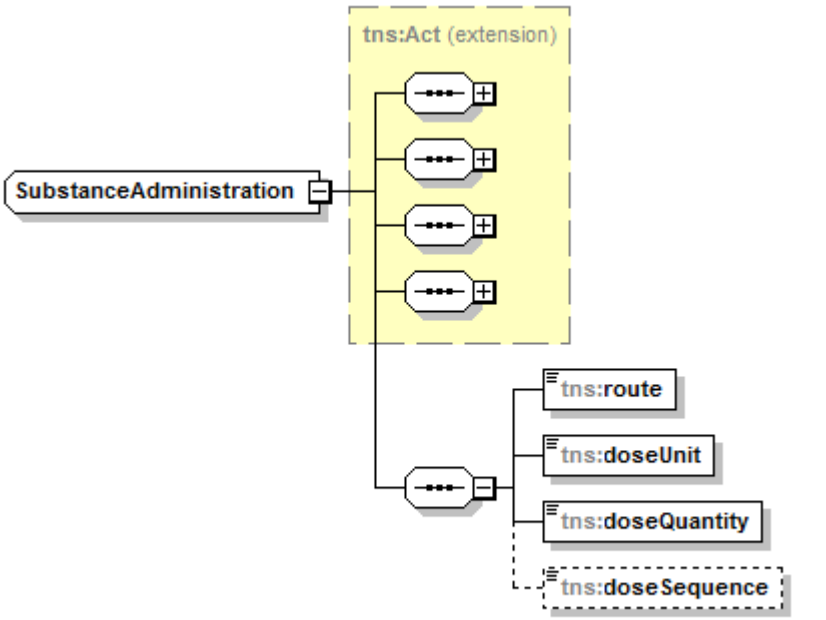


Element	Type	Description
dischargeDisposition [1..1]	UUID	A concept which indicates the method in which the patient left the encounter.

7.4.2.3.8 SubstanceAdministration

Substance administrations represent the administration of any substance to a patient in the context of an encounter. These can include vaccinations, booster shots, EpiPen (anti-histamine) administrations, etc.

Table 23 - Substance Administration

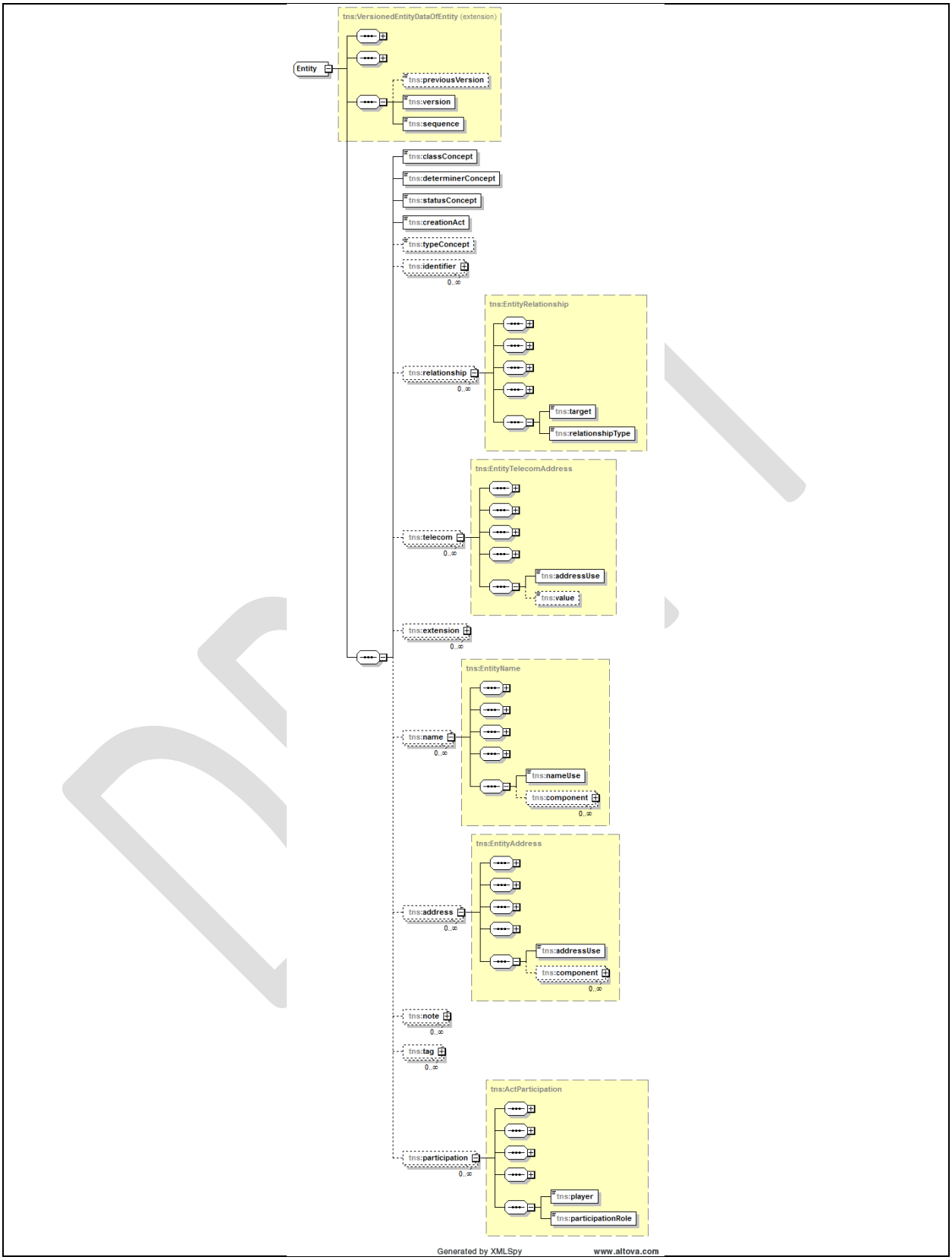


Element	Type	Description
route [1..1]	UUID	The concept which identifies the route in which the substance was administered.
doseUnit [1..1]	UUID	The unit of measure in which the doseQuantity was administered.
doseQuantity [1..1]	Decimal	The quantity of substance administered.
doseSequence [0..1]	Int	The sequence of the dose.

7.4.2.3.9 Entity

The entity resource is used to represent entities which cannot be classified in one of the other entity subclasses (patient, material, etc.).

Table 24 - Entity

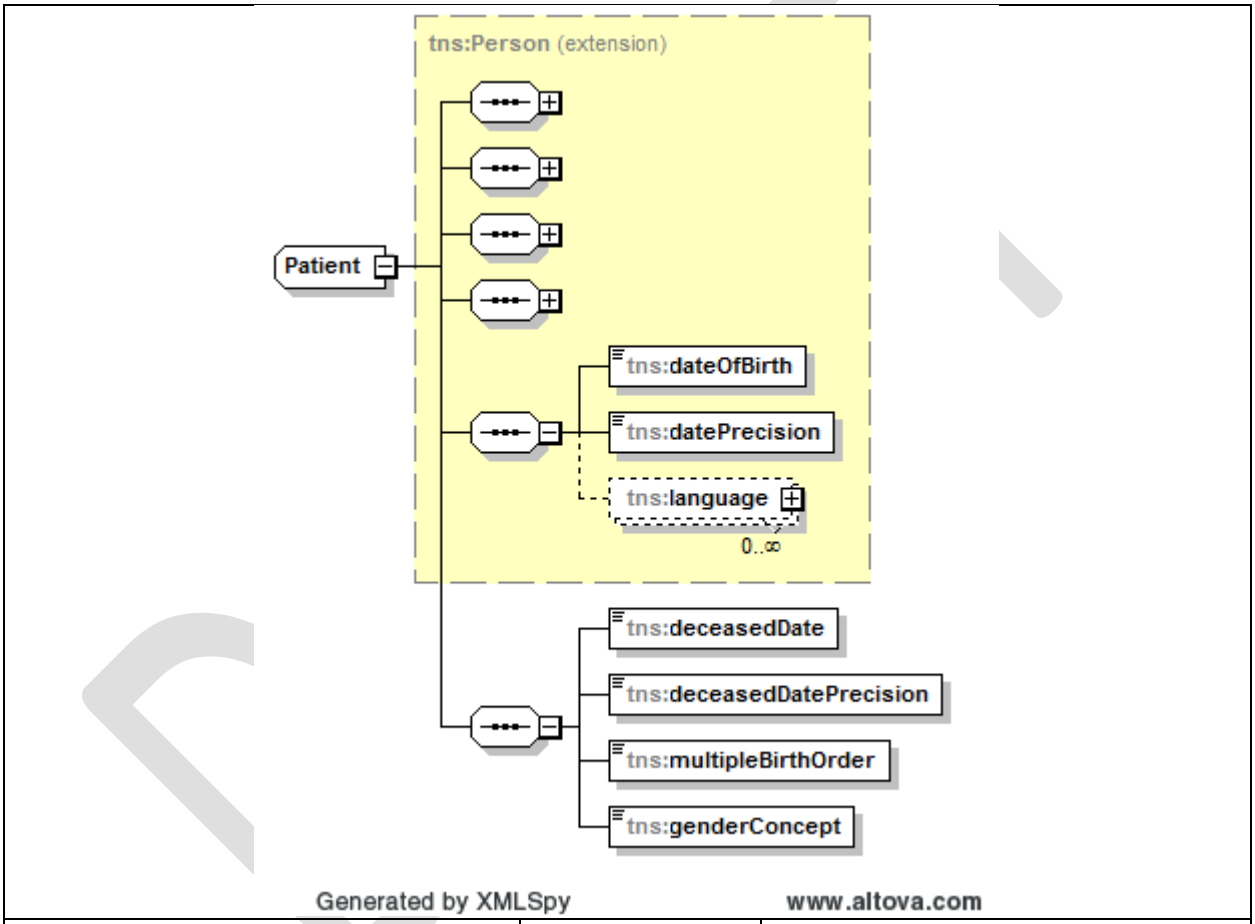


Element	Type	Description
mnemonic [1..1]	String	A mnemonic which is used to represent the reference term on the wire.

7.4.2.3.10 Patient

The patient resource represents a specialization of the person resource which is used to represent data related to a patient role.

Table 25 - Patient

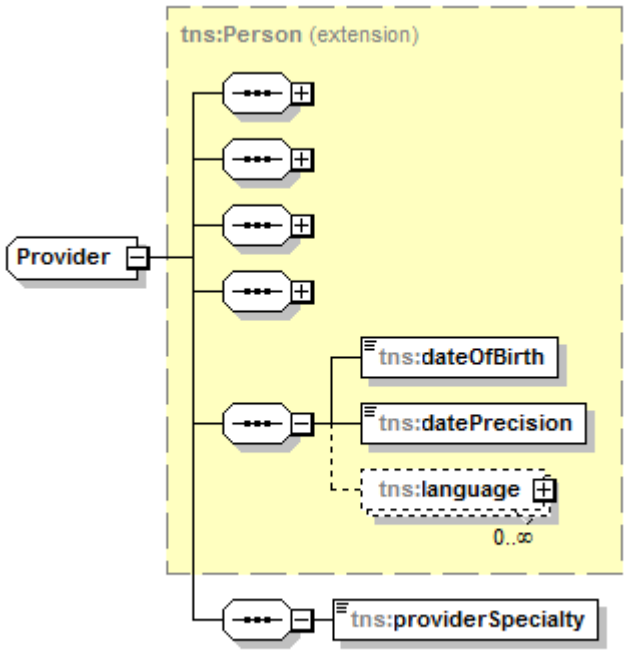


Element	Type	Description
deceasedDate [0..1]	DateTime	When populated, identifies the date that the patient died.
deceasedDatePrecision [0..1]	DatePrecision	Identifies the precision of the deceased date.
multipleBirthOrder [0..1]	Int	When populated, indicates that the patient was a part of a multiple birth.
genderConcept [1..1]	UUID	Identifies the gender of the patient.

7.4.2.3.11 Provider

A provider represents a specialization of a person which is charged with the delivery of health care services to patients.

Table 26 - Provider

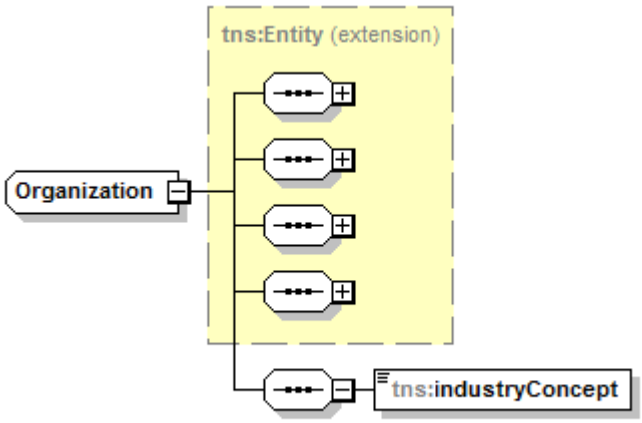


Element	Type	Description
providerSpecialty [1..1]	UUID	A code indicating the type of provider represented by the provider entity. (example: Nurse, CHW, etc.).

7.4.2.3.12 Organization

An organization represents an administrative construct which is used to deliver or organize care, delivery and manufacture of other entities.

Table 27 - Organization

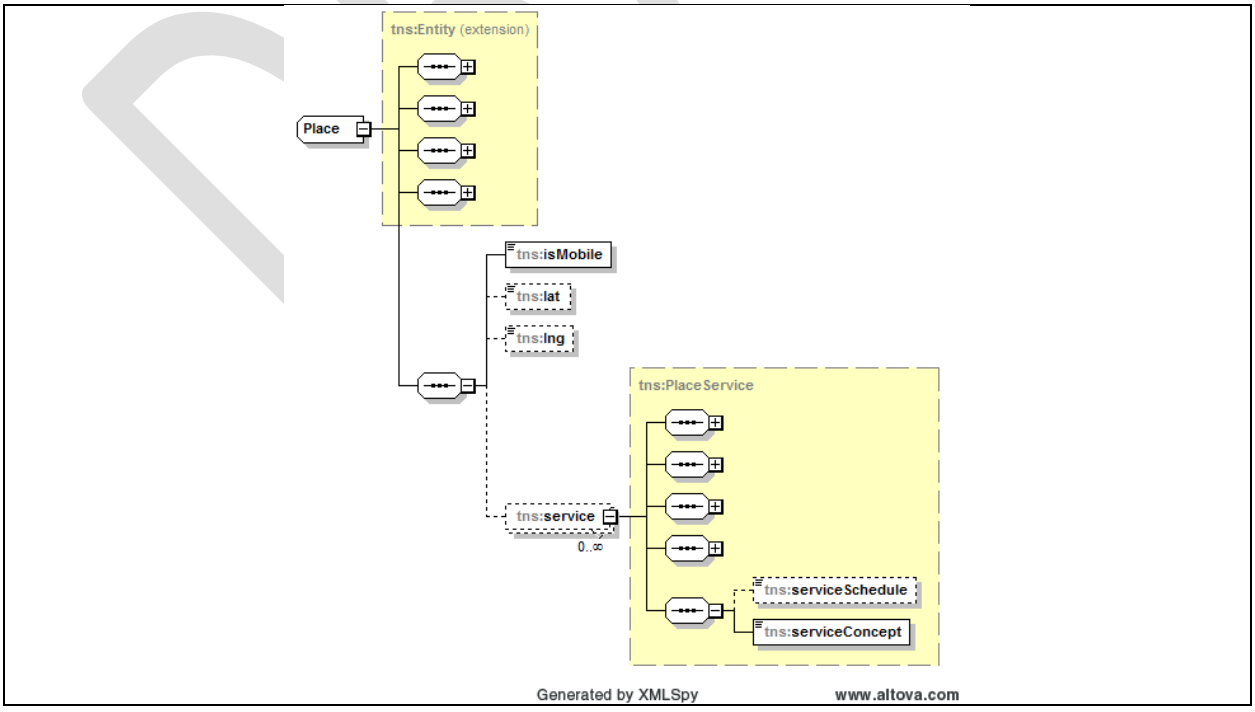


Element	Type	Description
industryConcept [1..1]	UUID	A code indicating the industry in which the organization operates. Examples: supply, hospital, etc.

7.4.2.3.13 Place

A place represents an entity which is a physical location where health delivery services are delivered such as clinics, hospital, mobile immunization clinics, etc.

Table 28 - Place

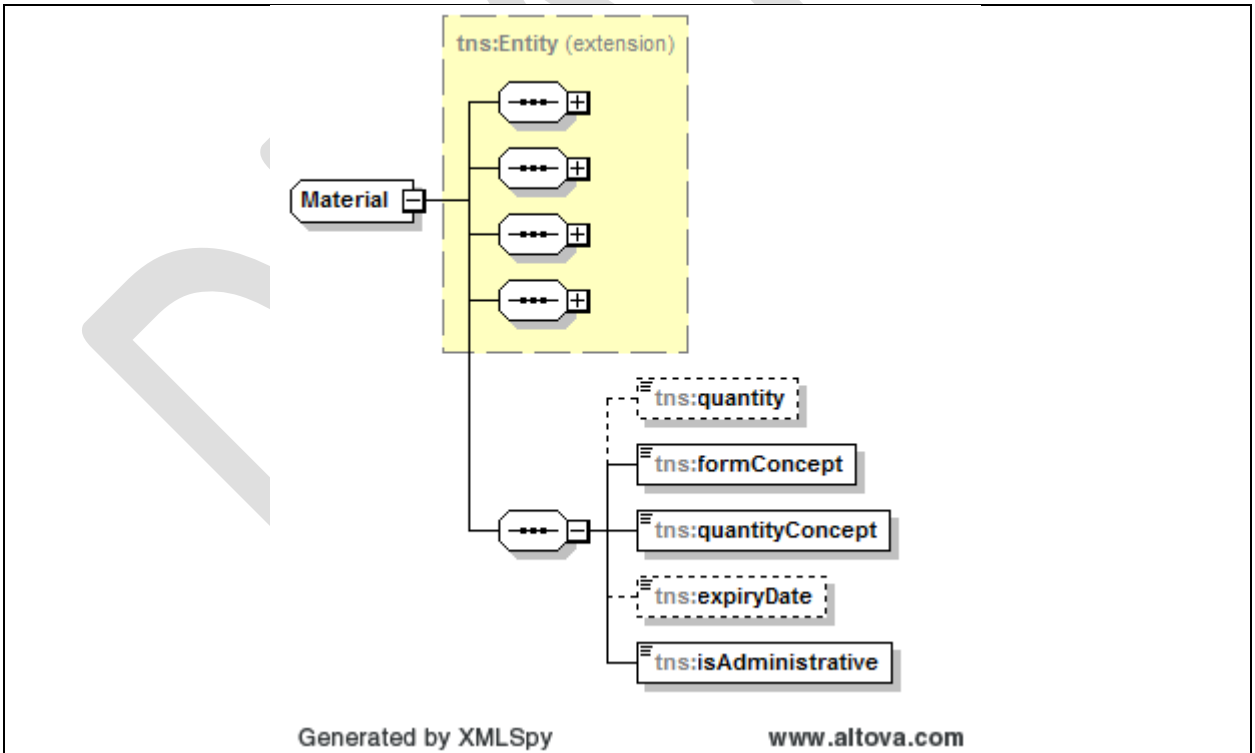


Element	Type	Description
isMobile [1..1]	Boolean	An indicator which identifies whether the delivery location is a mobile location.
lat [0..1]	Float	Indicates the latitude of the place.
lng [0..1]	Float	Indicates the longitude the place.
service [0..*]	Service	One or more services which the place offers.
service.serviceSchedule [0..1]	Xml	An XML representation of the scheduled of the service availability.
service.serviceConcept [1..1]	UUID	The type of service provided.

7.4.2.3.14 Material

A material represents an entity which can be distributed, consumed, packaged, or used. A material can be a physical material such as a distribution kit, device, etc.

Table 29 - Material



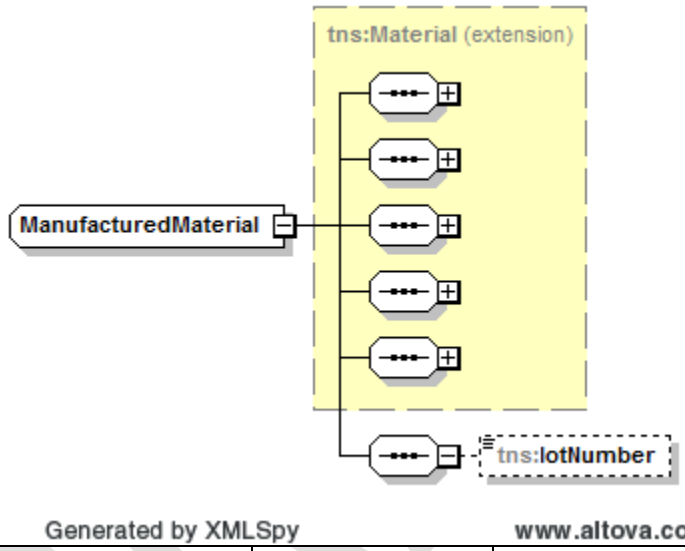
Element	Type	Description
quantity [0..1]	Decimal	Identifies the quantity of material in a container or group if populated.
formConcept [1..1]	UUID	Identifies the form of the material. For example: liquid, gas, boxed, etc.

quantityConcept [1..1]	UUID	Identifies the units of measure for the material when administered or consumed.
expiryDate [0..1]	DateTime	The time when the material will expire or did expire.
isAdministrative [1..1]	Boolean	When true, indicates that the material is not a consumable and is merely an administrative construct.

7.4.2.3.15 ManufacturedMaterial

A manufactured material represents a material which is acquired from a manufacturer.

Table 30 - Manufactured Material

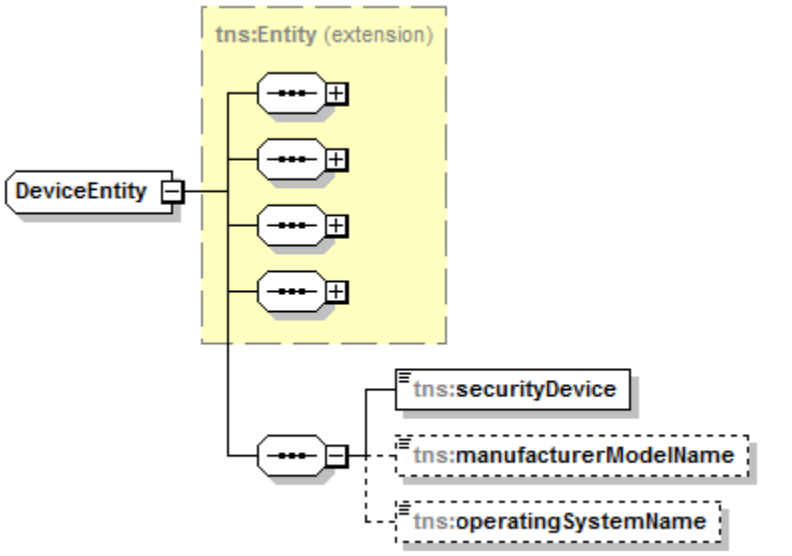


Element	Type	Description
lotNumber [0..1]	UUID	The lot number of the manufactured material.

7.4.2.3.16 Device

A device represents an entity which is physically used. A device entity is differentiated from a SecurityDevice in that the DeviceEntity is used primarily as information augmenting clinical data whereas a SecurityDevice contains security identity information.

Table 31 - Device



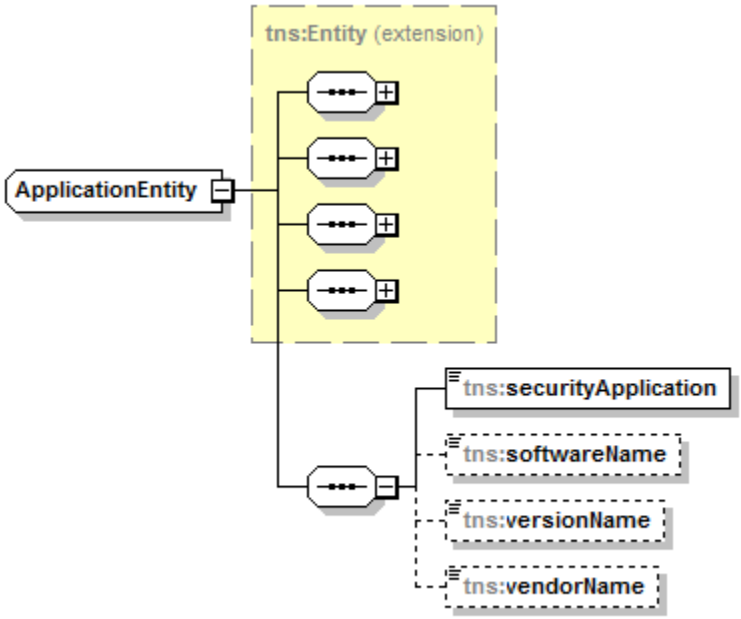
Element	Type	Description
securityDevice [1..1]	UUID	Identifies the security device which the device entity represents.
manufacturerModelName [0..1]	String	Identifies the manufacturer name of the device entity.
operatingSystemName [0..1]	String	Identifies the operating system of the device entity.

7.4.2.3.17 User

The user entity represents information in the clinical context of the OpenIZ data store. It is differentiated from a SecurityUser in that the user entity is focused more on the clinical data related to a user (used for provenance) whereas a SecurityUser is primarily concerned with the authentication and security enforcement.

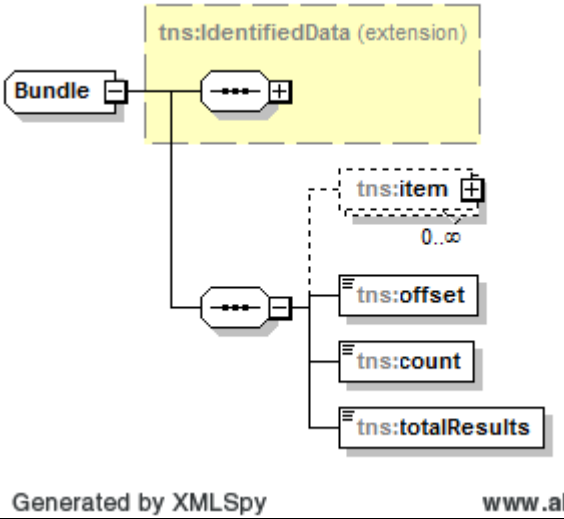
7.4.2.3.18 Application

The application entity represents information in the clinical context of the OpenIZ clinical data store.



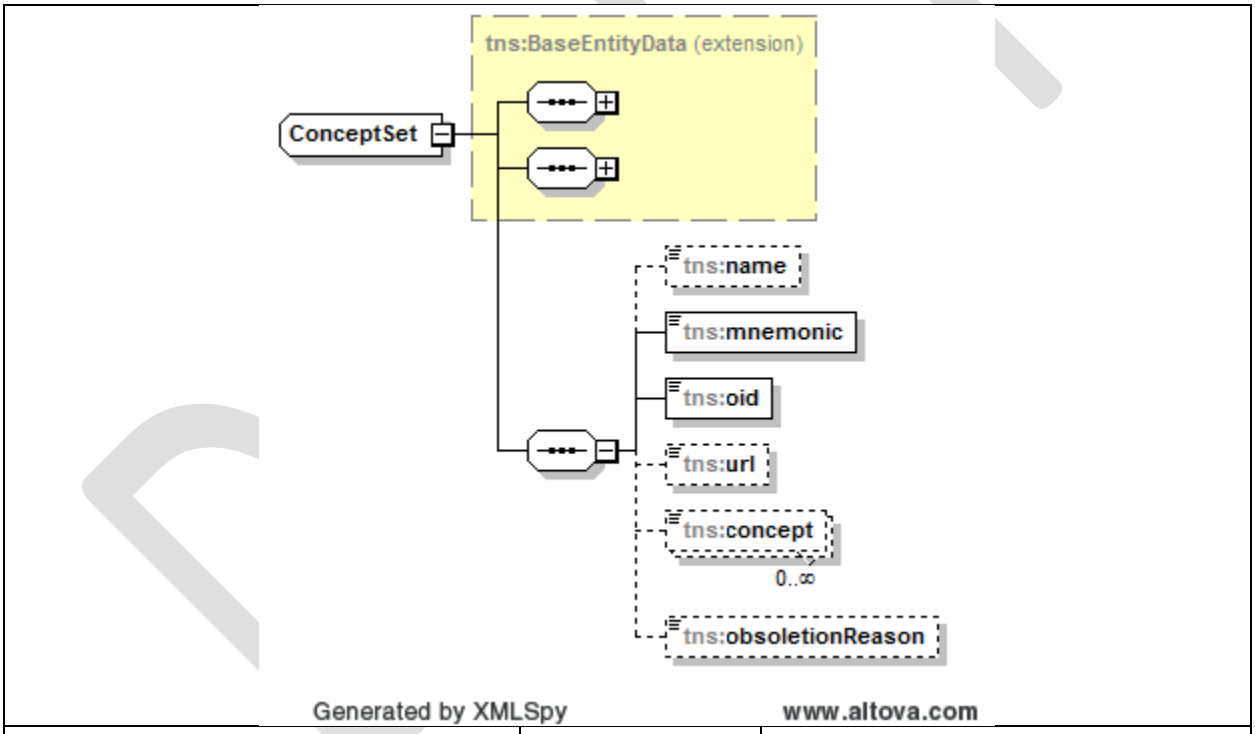
Element	Type	Description
securityApplication [1..1]	UUID	Identifies the SecurityApplication which the application entity represents.
softwareName [0..1]	String	The name of the software package.
versionName [0..1]	String	The name of the version of the software package.
vendorName [0..1]	String	The name of the vendor which manufactured the software application entity.

7.4.2.3.19 Bundle



Element	Type	Description
item [0..*]	Resource	Represents the contents of the bundle which are the resources being transported to another system.
offset [1..1]	Int	The offset of the bundle in the case of a large result set.
count [1..1]	Int	The count of the primary items in the bundle.
totalResults [1..1]	Int	The total number of primary items in the bundle.
entryItem [0..1]	UUID	Identifies the entry point in the bundle when the bundle is used to package a single result with referenced items.

7.4.2.3.20 ConceptSet



Element	Type	Description
mnemonic [1..1]	String	A mnemonic which is used to represent the reference term on the wire.

7.4.2.3.21 Drug

7.4.2.3.22 Vaccination

7.4.2.3.23 Appointment

7.4.2.3.24 AdverseReaction

7.4.2.4. Bundling Resources

All IMS data contract members are independent and do not include deeply nested data. Rather, each IMS object contains a reference to other objects via properties identified by UUID. When fetching and/or querying a record, callers can instruct the IMS to bundle these references into a resource bundle which contains the specified object.

This pattern is used because it prevents deep nesting and maintains references to data elements even when JSON is used.

7.4.2.5. MIME Types / Message Encodings

The IMSI supports both XML and JSON encodings. The control over which encoding is to be used is performed via the Accept header in the HTTP message. The allowed content types are listed in

Content-Type	Description
application/xml application/xml+openiz-ims	An XML formatted message will be sent or is included in the body.
application/json application/json+openiz-ims	A JSON formatted message will be sent or is included in the body.

7.4.2.6. Compression

The IMSI interface supports two-way compression for all communications. This means that a client can not only request compressed data from the IMS but can submit compressed data to the IMS. This functionality is expected to be useful when mobile clients are disconnected for long periods of time and are required to submit large amounts of data.

Requesting the IMSI to perform compression on a response requires the use of the Accept-Encoding header. Submitting compressed data to the IMSI requires the Content-Encoding header.

The IMSI will accept and produce compressed data in either GZIP or DEFLATE formats. If accept-encoding is not understood the X-CompressResponseStream header will be set to “no-known-accept” and uncompressed data will be sent to the requestor. Additionally if a Content-Encoding is sent to the IMSI which it cannot process a 400 (Bad Request) response will be sent to the requestor.

7.4.2.7. Query Parameters

7.4.2.7.1 Element Filters

All query parameters in the IMSI interface are mapped directly to data contract objects. For example, if a resource contains an element named “createdBy” the filter of the same name will perform a filter of data by the createdBy.

It is also possible to chain query parameters using a dotted notation. For example, to query by the name of the user which created a resource one can filter “createdBy.userName”. In this scenario any chained

parameters match the data element name on type. Since createdBy is of type SecurityUser, elements in SecurityUser can be chained.

7.4.2.7.2 Control Parameters

IMSI queries also provide a series of response control parameters. All control parameters begin with an underscore.

Parameter	Operations	Description
_bundle	Get, Version Get	Instructs the IMSI to bundle resource references.
_expand	Search, History, Get, Version Get	Instructs the IMSI service to expand properties and include them in the result. Using this option will force the delay-load of properties in the IMS model.
_all	Get, Version Get	Instructs the IMSI to expand all properties and include them in the result. This option forces the delay load of all data in the IMS model and can have performance implications.
_since	History	Instructs the IMSI to only return versions in the version history which have been created since the specified version identifier.
_offset	Search	Instructs the IMSI interface to offset search results by the specified number of result.
_count	Search	Instructs the IMSI interface to include only the indicated number of results in the return bundle.

7.4.2.7.3 Modifiers

Modifiers are applied to operators of particular type and can be used by consumers to perform simple filter tasks. The list of modifiers are found in .

Modifier	Operator	Example
Equality	?a=b	?userName=SYSTEM
Less Than	?a=<b	?creationTime=<2016-01-01
Less Than or Equal To	?a=<=b	?value=<=6.3
Greater Than	?a=>b	?value=>2
Greater Than or Equal To	?a=>=b	?creationTime=>=2016-01-01
Approximately	?a=~b	?mnemonic=~FOO
Not	?a=!b	?userName=!smithj

7.4.2.7.4 And / Or Semantics

Boolean logic semantics on the IMSI are limited to a simple scheme. If more than one parameter of the same name appears in the query string, the values are ORed to one another. Unique filter parameters are ANDed. For example:

?createdBy.userName=SYSTEM&mnemonic=~EVN

Equates to any record created by a user with username SYSTEM and having a mnemonic approximately matching EVN, whereas:

?createdBy.userName=SYSTEM&mnemonic=~EVN&mnemonic=~Event

Matches any record created by a user with username SYSTEM having mnemonic approximately matching EVN or mnemonic approximately matching Event.

7.4.2.7.5 Guard Conditions

There are several times when queries will require filtering on a chained parameter where the query executor wishes to guard or only filter a specific type of traversal. These guard conditions are used to select only traversals whose classifiers match the guard condition. For example, to filter on only Patients whose legal given name is John, the query would be as follows:

/Patient?name[L].component[GIV].value=John

Here the query guards that only names with NameUse.Mnemonic (the classifier for EntityName) matching L are filtered and the component having ComponentType.Mnemonic (the classifier for EntityNameComponent) has a value of GIV. In the query filter builder for the IMSI the actual expression is as follows:

```
o => o.Names.Where(guard => (guard.NameUse.Mnemonic == "L")).Any(name =>
name.Component.Where(guard => (guard.Type == "GIV")).Any(component =>
(component.Value == "John")))
```

Guard conditions are useful when filtering collections where classifier carries some semantic meaning. For example: name, addresses, participations, entity relationships, etc.

7.4.3. Report Integration Services Interface(RISI)

The RISI interface is primarily designed to abstract applications from the reporting technology used in an OpenIZ deployment. The RISI exposes data related to reports, and facilitates the execution of reports in a secure manner.

7.4.3.1. Transport

The RISI interface is very simple and designed solely for the purpose of exposing data for reporting purposes.

The operations for the RISI are described in .

URL	Method	Description
/report	POST	Instructs the RISI that a new report metadata object (with report data) should be created on the IReportProvider.
/report/{id}	PUT	Updates the specified report meta-data, resulting in a new version.
/report?{query}	GET	Queries the IReportProvider per the provided query parameters.
/report/{id}	GET	Reads the specified report manifest from the report provider.
/report/{id}	DELETE	Obsoletes the specified report.
/report/{id}/output.{format}?{parms}	GET	Executes the specified report returning a result in {format}.

/report/{id}/source	GET	Downloads the report source file as found on the report execution engine.
/report/{id}/parm	GET	Gets the parameters which are required for the report.
/report/{id}/parm/{parmId}/values	GET	Gets a list of auto-complete parameters which are applicable for the specified parameter.
/type	GET	Gets all registered report parameter types.
/type	POST	Creates a new report parameter type.
/type/{id}	GET	Retrieves detailed information for the specified report parameter.
/type/{id}	PUT	Updates the information related to the parameter type.
/type/{id}	DELETE	Obsoletes the report parameter data type.

7.4.3.2. Response Codes

See 7.4.2.2 for response codes.

7.4.3.3. Compression

The RISI interface supports compression of requests and responses. See 7.4.2.6 for more information.

8. Data Architecture

8.1. Conceptual Data Model

This section seeks to describe and discuss the concepts found within the OpenIZ data design. It does not describe the data entities, nor their fields and seeks to provide context to the logical data model which follows.

The OpenIZ data model can be described as a series of conceptual domains. These domains are used to conceptualize how the entirety of the IZTW data model functions. The logical domains within the OpenIZ data model can be described as:

- **Security:** Security tables are primarily used for the purpose of securing the OpenIZ data and enforcing of policies. OpenIZ's policy system is described in the solution architecture section of this document. Security tables also deal with users in roles.
- **Clinical:** These tables represent the primary way in which clinical data is stored. The OpenIZ clinical data model is a derivation of the HL7® Reference Information Model (RIM). The HL7 RIM can be described as a series of base classes describing all events as "entities playing roles participating in acts".
- **Stock Management:** These tables represent a series of ledgers for each of the "places" represented in OpenIZ. Stock management tables primarily deal with tracking transfer orders to/from places and organizations.
- **Protocol / Workflow:** These tables represent metadata that is used by forecasting and DSS tools to enforce vaccination protocols. Patients are enrolled into a series of protocols or workflows that can be used to track adherence to best practices.
- **Concepts:** These tables are used to control the vocabulary used within the OpenIZ data model. OpenIZ uses a series of core concepts in the clinical tables and the concept tables permit the mapping of these internal OpenIZ concepts to wire-level codes in HL7® FHIR™, or HL7® CDA™.

8.1.1. Clinical Domain

The clinical domain within OpenIZ is loosely based on the HL7 reference information model (RIM). The OpenIZ clinical domain does not represent a holistic implementation of the RIM, however borrows some of its concepts to represent and organize clinical data.

Figure 8 graphically illustrates how the elements within the clinical domain relate to one another.

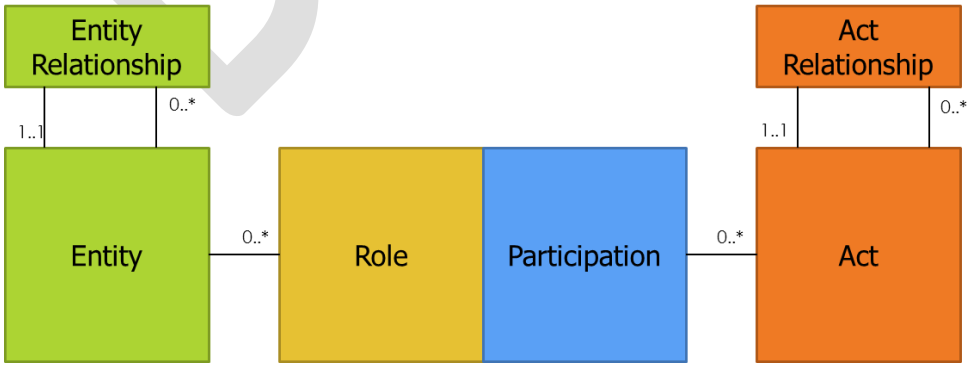


Figure 8 - Conceptual Clinical Model

- **Entities:** An entity represents a person, place, organization, or thing (syringe, antigen, vaccine, etc.). Entities can be related to one another via an entity relationship such as place belonging to an organization or series of materials belonging to a vaccination kit.
- **Roles:** Roles represent a type of part an entity plays. For example, a Person entity may play the role of a provider or a patient.
- **Participations:** Participations are the link between an act and an entity via a role. A participation is used to describe how an entity participates in the carrying out of an act.
- **Acts:** An act represents an action performed. An example of an Act may be an encounter the patient has with a provider to receive a weight or immunization. Acts may be related to one another, for example an encounter may fulfill an appointment, or an observation may be a part of an encounter.

In this context we can represent many different clinical scenarios. In order to make conceptualizing data elements in the OpenIZ persistence store easier, much data documentation leverages an information model cards as illustrated in Figure 9.

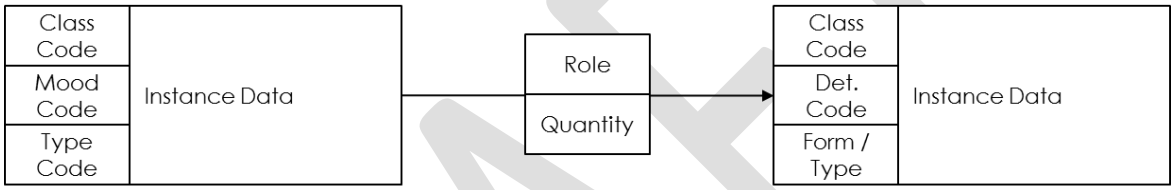


Figure 9 - Sample information model representing Participation

8.1.1.1. Entities

An entity within the OpenIZ data model is used to represent a person, place, or thing. Entities represent the who, which and where aspects of an action. Entities are further classified into several subclasses illustrated in Figure 10.

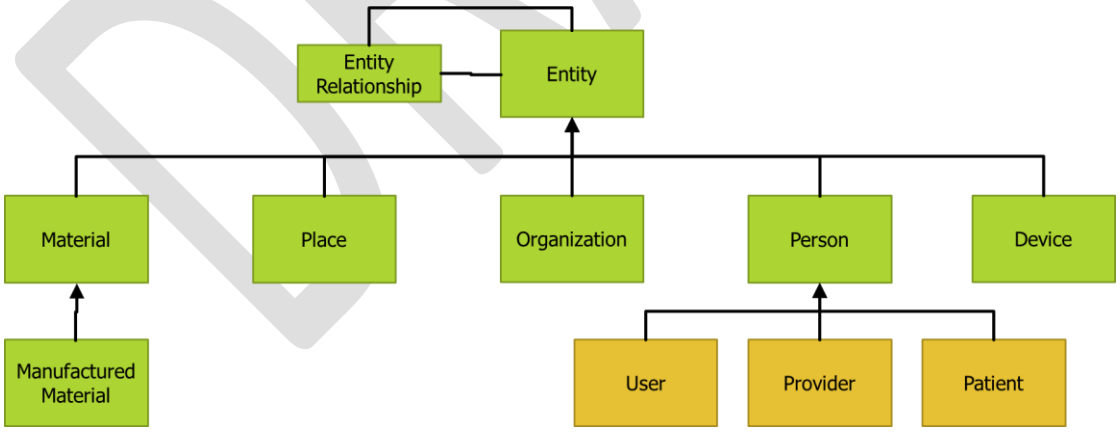


Figure 10 - Entity Classes

Entities are further classified by their class code and determiner code. The determiner code of an entity is responsible from differentiating a type of an entity (i.e. antigen, dose number, material type, etc.) and an instance or series of instances of an entity (an actual vial of vaccine, a box of syringes, etc.). Most

entities within the OpenIZ system are expected to be stored as instances of entities, classes of entities will primarily be restricted to materials whereby a class of antigens (OPV for example) will have both instances (vials of OPV) and sub-classes of representing dose numbers (OPV0 – OPV3). Provides a summary of the entity classes and how they are classified in the data model.

Entity Class	Class Code	Description
Entity	ENT	An entity is the base class used to represent a person/place/thing in the OpenIZ data model.
Material	MAT	A material represents a physical thing to which participates in the delivery of care. For example: a syringe, a box of vaccine, etc.
Manufactured Material	MMAT	A manufactured material represents a material which is manufactured such as diluent, vaccine, syringes, etc.
Place	PLC	A place represents a physical location where health services are provided.
Organization	ORG	An organization represents an administrative structure which employs providers, operates clinics, etc.
Person	PSN	A person represents a human being.
Patient	PAT	Represents a person who receives health services.
Provider	PVD	Represents a person who provides health services.
User	USR	Represents a person who actively uses the system.
Device	DEV	Represents a physical object on which health services data is entered, stored, etc.
Application	APP	Represents a piece of software which is used to access, record or update medical data.

Determiner codes are used to classify whether a tuple in the entity table represents an actual thing or a classification of things. There are three types of determiner classifications for objects listed in .

Determiner	Description	Example
Instance	The entry in the database represents a one or more occurrences of an entity.	A patient, a facility, an organization, etc.
Kind	The entry represents a classification of entities.	A type of material, a type of organization, a type of device.
Quantified Kind	Represents a kind of object which is quantified such as X number of Y.	A dose of vaccine (5ml, etc.)

Figure 11 represents a sample information model of a kit of BCG vaccine. The model illustrates how a single tuples in the OpenIZ model are related to one another to represent a kit. The material “BCG Vaccine” is comprised of one dose of “BCG Antigen”, one “Syringe”, and one dose of “BCG Diluent”. A box of “BCG Vaccine” represents 25 single doses of “BCG Vaccine” (meaning 25 syringes, 25 antigen and 25 diluent). Finally, from this we can order (via instantiation or inheriting a kind) a kit of GSK 5ml BCG vaccine. This derivation will have the same rules applied (in that this vaccine must have a syringe, antigen, and diluent.

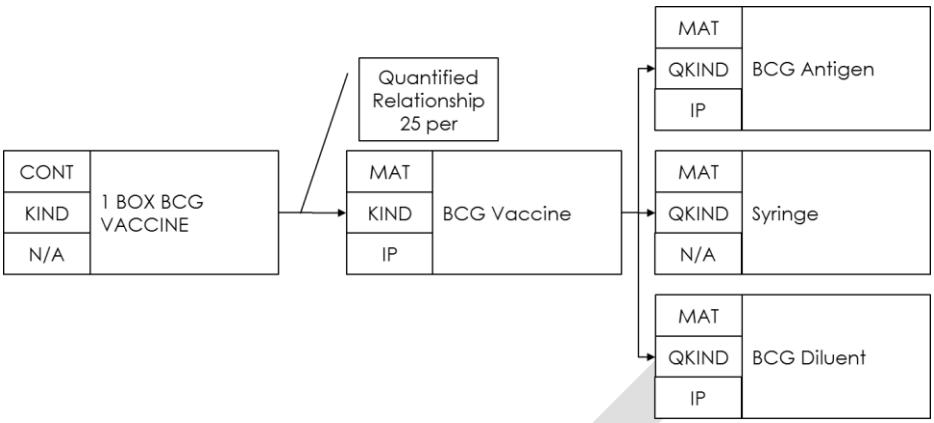


Figure 11 - Representing BCG as a combination of materials

Figure 12 represents a complex scenario of kitting a vaccine and then deriving a specific type of kit. The relationship between the instantiation and the generic concepts of the material would have a type of "Manufactured" meaning the "GSK 5 ml BCG" is a manufactured representation of the generic BCGvaccine.

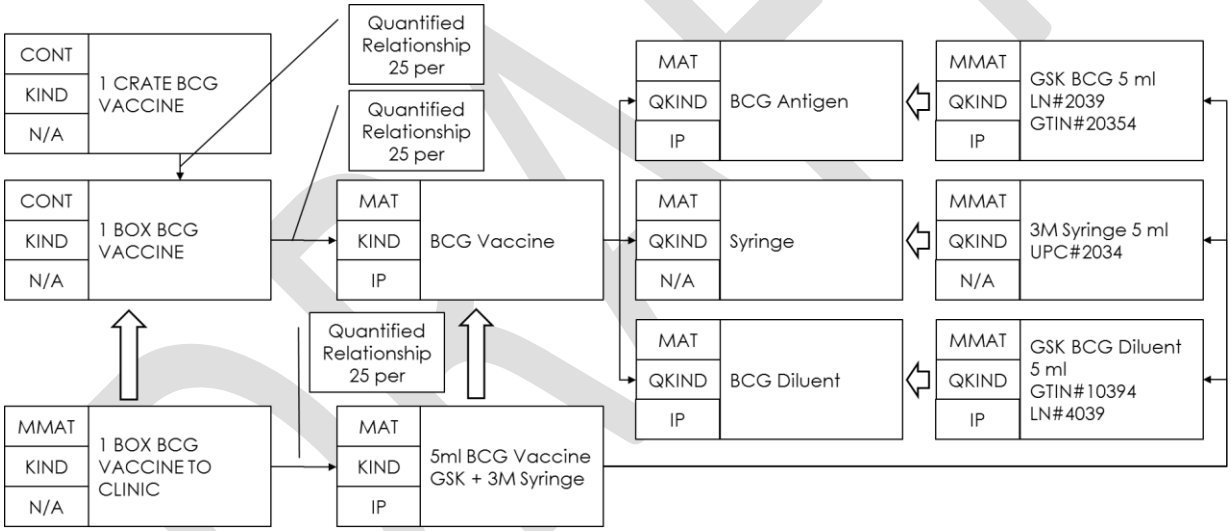


Figure 12 - Instantiating BCG vaccine kit to Manufactured materials

The OpenIZ model also supports more simplistic representation of a "BCG Vaccine" if kits are not required by the jurisdiction that is implementing OpenIZ. Figure 13 illustrates a valid representation of BCG in a jurisdiction where only the antigen is tracked.

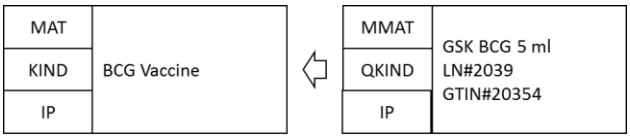


Figure 13 - Simple BCG to manufactured material

8.1.1.2. Acts

Acts in the OpenIZ data model represent actions taken by entities to other entities. In this lens, we can say that an act represents everything that "happens" to entities. This mechanism of storage allows the OpenIZ data model to adapt to different jurisdictions with relative ease.

There are five different types of acts that are supported by the default OpenIZ schema (Figure 14).

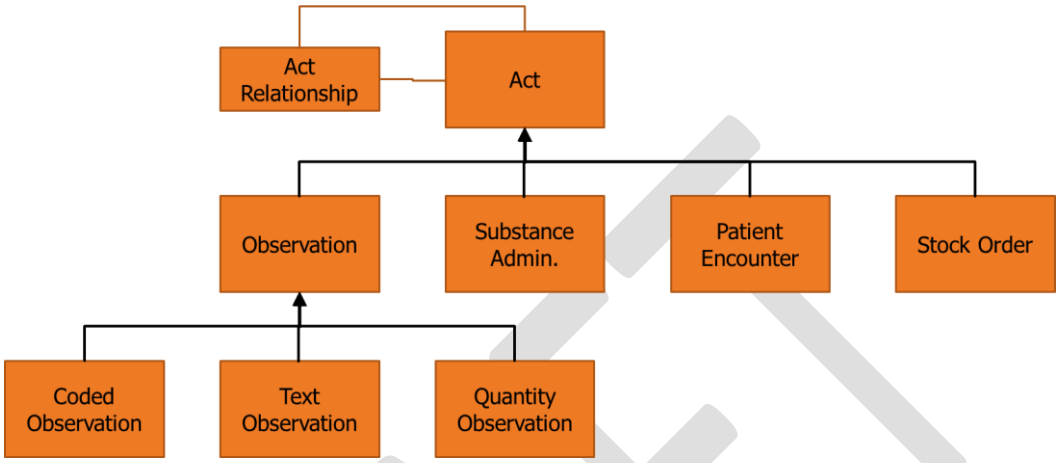


Figure 14 - Act classes

- **Patient Encounters:** Represent an act whereby a patient presents, or is intended to present for care.
- **Observations:** Represent an act whereby an entity observes something. Observations can include codified observations such as diagnoses, textual values such as a free-text description of an event, or a quantity such as weight or heartrate.
- **Substance Administrations:** A substance administration is representing an act whereby a substance, such as a vaccine is administered to a patient.
- **Transfer Order:** An order is a special type of stock act and represents a request by an entity to transfer stock from one place to another.

Acts are classified into one of these four types based upon their ClassConcept code. This dictates what type of Act the particular tuple in the database represents. Additionally, Acts are classified by Mood via the MoodConceptId. The mood of an act identifies the mood or method of operation of the act, moods include:

- **Proposal:** A proposal to perform the act, typically a forecaster or some automated DSS process will create acts with this mood code.
- **Intent:** Represents an intent to perform the act. These represent a human entering an intent to attend an encounter, or perform an observation.
- **Event:** Represents an act that *actually* occurred. This represents an action taken by a human.
- **Request:** Represents a request by one human to another human perform the action. Examples may be requesting a transfer order or requesting an encounter be done (i.e. referral).

Often time acts are related to one another via the ActRelationship entity. Act relationships are important, as typically acts cannot be standalone. For example, one cannot simply “Observe” something without having an encounter. An example of several types of interactions within OpenIZ are described below:

- **Patient Presents to receive an Immunization:**
  In this use case the primary act is a PatientEncounter whereby participants include the clinician performing the vaccination (performer), the patient (record target), the clinic (service delivery location). The patient may be weighed prior to receiving the immunization that represents a component act of type Observation, and the administration of the vaccine represents a substance administration with relations to one or materials administered (Figure 15).

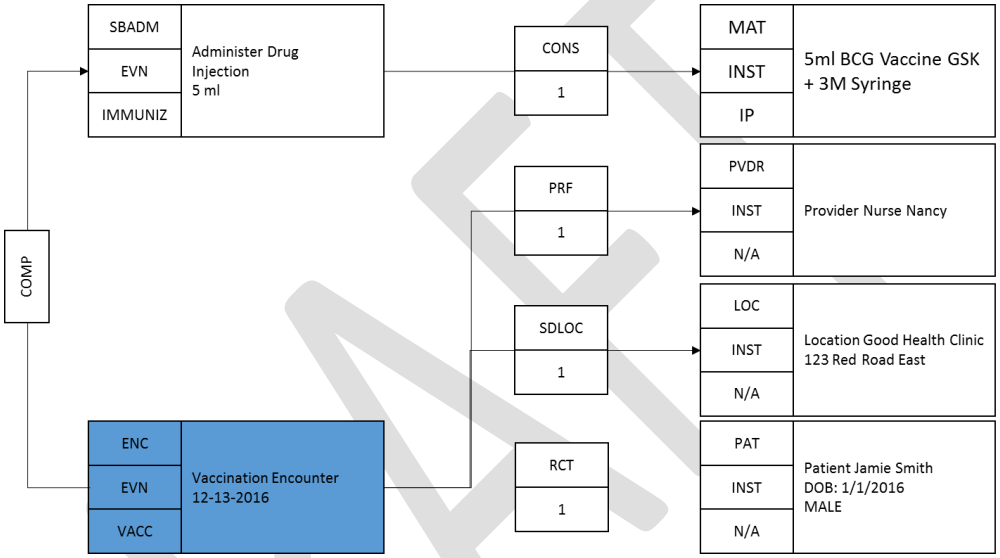


Figure 15 - Sample Vaccination Encounter model

- **Scheduling a second dose of an antigen:**
  In this use case the primary act is a PatientEncounter with mood of *Intent* (i.e. I intend to have an encounter) and date in the future. The PatientEncounter may have one or more substance administrations with mood of *Intent* that represent the vaccinations that are intended to be given at the appointment (Figure 16).

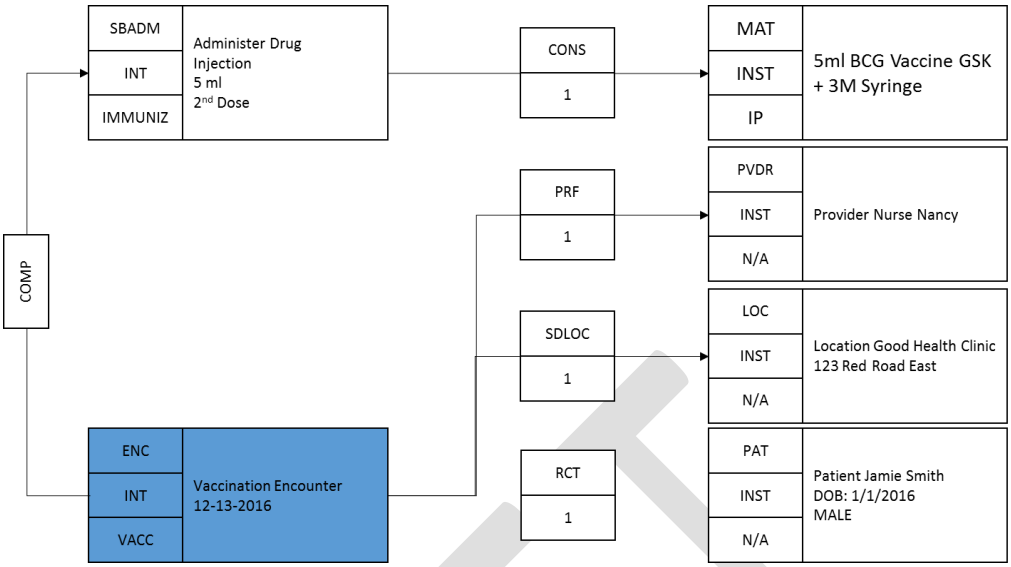


Figure 16 - Vaccination Appointment model

- **Patient Presents for Appointment:**
  In this use case, the patient presents for the scheduled appointment. The encounter fulfills (FLFLS) the appointment request (Figure 17).

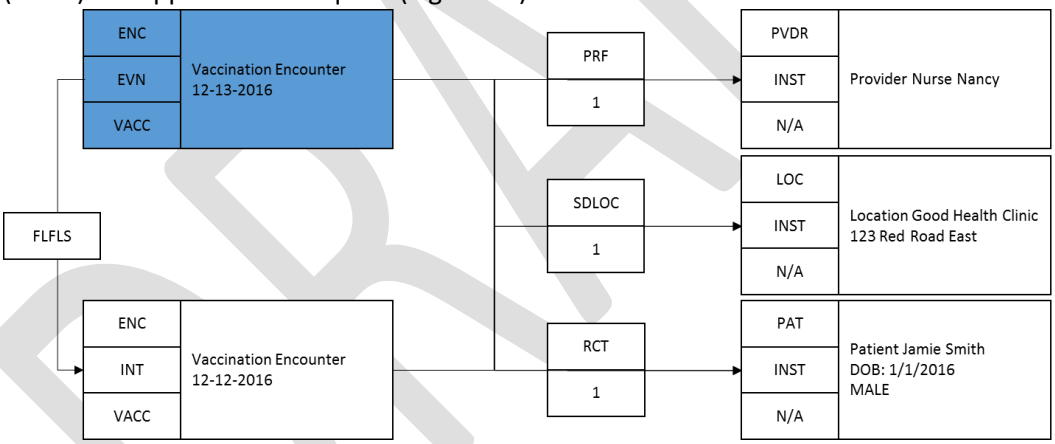


Figure 17 - Vaccination appointment fulfilling an appointment

- **Forecasted vaccination:**
  In this use case a decision support system identifies a patient which needs to receive a vaccine. The forecasting engine may create a PatientEncounter with mood of *Proposed* that indicates that a computer system is proposing an action to occur. When the patient presents for their vaccination the clinician creates a new PatientEncounter with mood of *Event* that fulfills the PatientEncounter with mood of *Proposed* (i.e. the clinician is saying "I am acting on your proposal").

8.2. Logical Data Design

This section seeks to describe and discuss the methodology behind the OpenIZ data design and to describe the logical data design and schema. Note that the logical data design is often different from the physical schema as the physical schemata are adapted for the restrictions of the RDBMS in which they

operate. The logical data design is also separate from the business data model design though the concepts are translatable.

8.2.1. Design Notes

This section outlines several design principles which were used in the creation of the OpenIZ data model. Please note that any diagrams are informational, the source of truth are the detailed data dictionary tables.

8.2.1.1. Versioning

All data tables in OpenIZ are versioned. This includes everything from display names of concepts, entities, acts, etc. This versioning allows an administrator to view a record as it was when it was created by a clinician.

This is important not only for a medical/legal reason, but for understanding why an action was taken and/or submitted. The versioning of major elements is performed using a few core concepts:

- **Object Table:** The core table contains the object's identifier and any immutable attributes of the object. For example, the Entity table contains the immutable "class code" attribute in the object table.
- **Object Version Table:** The object version table contains the object's versioned attributes. That is, attributes which may change over the lifetime of the object. Each version is assigned a unique identifier and a version sequence number. The version table also keeps track of version history via a replaces column so that versions are linked to one another.
- **Associative Tables:** All associative tables to a versioned object carry an EffectiveVersionSequenceId and ObsoleteVersionSequenceId which allow the OpenIZ queries to establish whether a relationship was active at a particular time an action was taken, and also allows OpenIZ to establish the user which was responsible for adding/removing the association.

Some tables are versioned in a different manner; these tables are typically tables where versioning is not as important for tracing purposes. These tables typically have a CreationTime/CreatedBy and ObsoletionTime/ObsoletedBy column and may have a pointer to a previous record that represents an edit.

The key message is that no UPDATE statements are (and should never) be performed on the OpenIZ database.

8.2.1.2. Tags and Extensions

Often times there are country specific requirements that cannot be stored in the default OpenIZ data model. The data model has two methods of storing and reproducing such data.

Tags represent version independent, non-structural data associated with an Act or Entity. Tags are often used for tagging workflow values associated with an Act (such as editing approval process), security tags, and/or attachments. Tags are primarily used for storing metadata.

Extensions are structural attributes that modify the meaning of an Act or Entity. Extensions differ from tags in that they add meaning to an entity or act, and thus, are versioned. Extensions can be thought of as representing data specifically associated with the entity. Examples of extensions include 10-cell

address of a patient, a stock consumption policy of a place. Extensions should be used in the last resort, when no other mechanism exists to store the data in the natural data model.

8.2.2. Entity Relationships

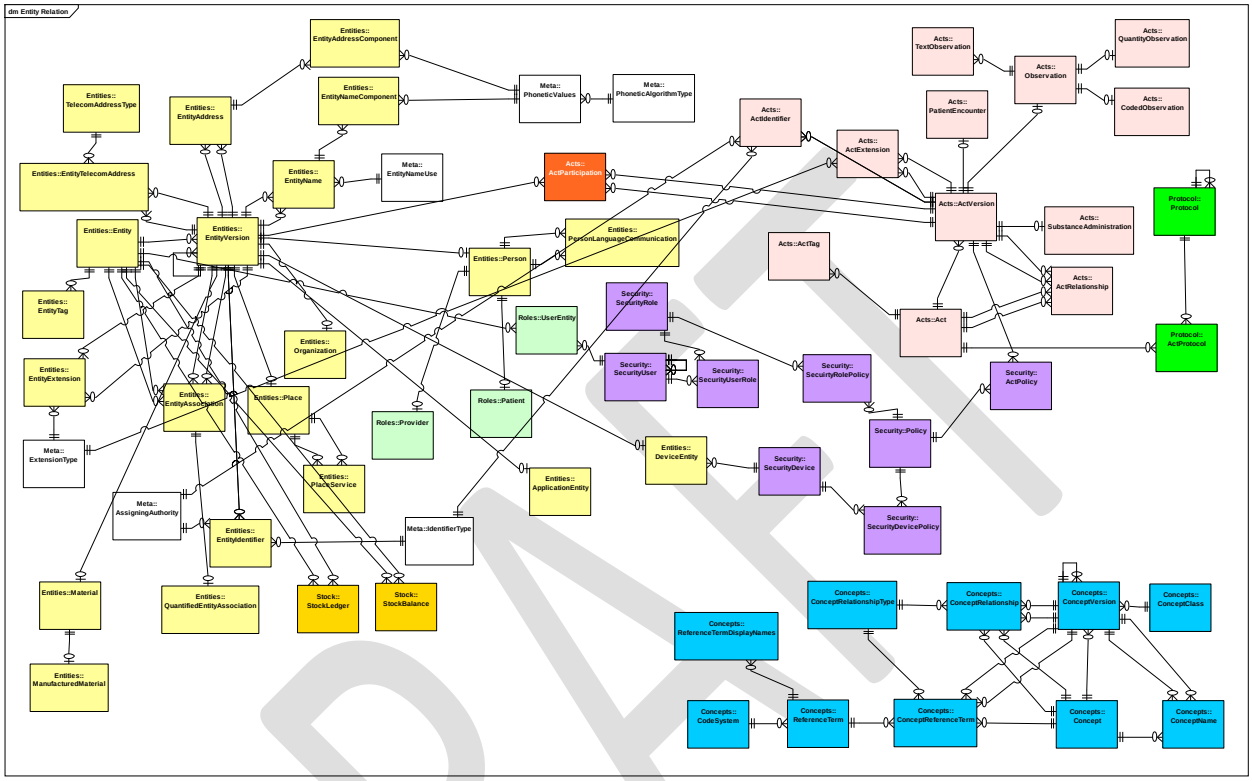


Figure 18 - OpenIZ Entity Relations

Table 32 - OpenIZ Entity Dictionary

Entity	Classification	Purpose
Entity	Clinical:Entity	The entity table provides a master list of all entity data within the data model. This table is used to store attributes related to the abstract superclass “Entity”.
EntityVersion	Clinical:Entity	The entity version table provides data related to the versioning of entity data within the OpenIZ system. All key entity attributes (minus classification) are stored in the entity version table.
EntityTelecommunicationsAddress	Clinical:Entity	The entity telecommunications address table provides a 1..n relationship through which telecommunications addresses (email addresses, phone numbers, fax numbers, etc.) can be stored and related to a particular entity version.

TelecommunicationsAddressType	Metadata	The telecommunications address type table is used to store data related to the type of telecommunications addresses used. This can be configured so that handlers may “contact” these addresses.
EntityAddress	Clinical:Entity	The entity address table is used to store addresses that are related to a particular entity. These addresses may include physical addresses, mailing addresses such a PO Boxes, etc.
EntityAddressComponent	Clinical:Entity	The entity address component table is used to store the component pieces of an entity address. These components may include the street address, city, village, zip code, geo-locator, etc. All address components are phonetically searchable.
EntityName	Clinical:Entity	The entity name table is used to store 1..n names related to an entity. An entity name may be a place name such as “Good Health Hospital”, a person name such as “Legal”, “Maiden”, etc.
EntityNameComponent	Clinical:Entity	The entity name component table is used to store the components of an entity name. These may include the given, family, suffix or prefix portions of a name.
EntityNameUse	Metadata	The entity name use table is used to store a master list of allowed uses of an entity name. This table is also responsible for ensuring that an appropriate name use is used in conjunction with the associated entity. For example, a “Person” entity name may not use an “Organizational” name use.
EntityIdentifier	Clinical:Entity	The entity identifier table is used to store a list of 1..n identifiers associated with an entity. An entity identifier can be thought of in many ways, for example: - An MRN for a Patient - A GTIN or UPC for a Material - An OU name for an Organization - A license number for a Provider
EntityIdentifierType	Metadata	The entity identifier type table is used to identify how an entity identifier is to be used and selected. An example, a GTIN entity identifier type may not be used with a Person entity as a Person does not carry a GTIN.

AssigningAuthority	Metadata	An assigning authority an a globally unique domain identifier which qualifies which system, organization or authority has the ability to assign new identifiers within a domain. An example of an assigning authority may be a system which generates MRNs, a jurisdictional EMPI, a global trade organization, etc.
EntityExtension	Metadata	An entity extension represents a non-classified, extended field for the entity. Extension fields are used primarily to store data related to an entity that does not belong in the standard OpenIZ datastore. For example, there may be a jurisdictional requirement to store the Race or EthnicGroup of a Person.
EntityExtensionType	Metadata	An entity extension type qualifies the type of extension represented in the EntityExtension table. The type identifies an extension handler responsible for de-persisting the data.
EntityTag	Metadata	An entity tag represents a simple Key/Value pair data that is version-independent for a particular entity, that is to say that the updating of the tag does not produce a new version of the entity. Tags can be used for things like workflow tagging, quarantine control, or any other application specific use.
Person	Clinical:Entity	The person table is used to store data related to the Person subclass of the Entity superclass.
PersonLanguageCommunication	Clinical:Role	Identifies the languages of communication in which the person can be contacted.
Patient	Clinical:Role	The Patient table is used to store data related to the Patient subclass of the Person superclass. The patient table adds fields for date of birth, multiple birth order, deceased time, etc.
Provider	Clinical:Role	The provider role table is used to store data related to the provider subclass of the Person superclass.
Organization	Clincial:Entity	An Organization is a table used to store data related to the Organization subclass of the Entity superclass. An organization represents a legal entity, which is not a

		person, who has a mandate to deliver healthcare.
Place	Clinical:Entity	A Place is a table used to store data related to a physical place where care is delivered.
PlaceService	Metadata	The PlaceService table is used to store data related to the services available at a particular place. The services in the scope of OpenIZ may represent immunization services, weight services, emergency services, etc.
Material	Clinical:Entity	The Material table is used to store data related to the Material subclass of the Entity superclass. A material represents a physical thing or supply that is used in the delivery of care. This can include syringes, diluents, vaccines, kits, gloves, etc.
ManufacturedMaterial	Clinical:Entity	A manufactured material represents a material that can be ordered from a manufacturer. This includes syringes, diluents, gloves, etc. This is a specialization for a Material in that a material can technically be a self-assembled group of materials which may not necessarily be orderable by a manufacturer (for example: a vaccine kit which is an administrative material)
EntityRelationship	Clinical:Entity	The entity association table is used to associate entities to one another. This can be used in a variety of ways including - Linking places to an organization - Linking materials together in a kit - Linking persons together, for example a mother/father of a patient - Linking places together in an administrative hierarchy. - Linking providers with patients.
ActParticipation	Clinical:Participation	The ActParticipation associative entity class is used to associate a particular version of an act with a version of an entity participating in the act. Example of participations include : RecordTarget, Author, Performer, Subject, etc.
Act	Clinical:Acts	The Act table is used to store the abstract data related to a particular act.

ActVersion	Clinical:Acts	The ActVersion table is used to store abstract data related to a particular point-in-time version of an Act or subclass of an act.
ActRelationship	Clinical:Acts	The ActRelationship table is used to link two acts together in some way. Examples of an ActRelationship include fulfillment (i.e. an encounter fulfills an appointment), componentization (i.e. encounter has an observation), etc.
SubstanceAdministration	Clinical:Acts	The SubstanceAdministration table is used to store the administration of substances (like vaccines) to the patient.
Observation	Clinical:Acts	The Observation table is used to store additional data related to an observation that occurs during the course of an encounter.
QuantityObservation	Clinical:Acts	The QuantityObservation table is used to store additional data related to an observation whose value is a quantity. For example, an observation of weight, blood pressure, height, etc.
CodedObservation	Clinical:Acts	The CodedObservation table is used to store additional data related to an observation whose value is a code. For example, observing the patient has blue eyes.
TextObservation	Clinical:Acts	The TextObservation table is used to store additional data related to observations which have textual content such as observation
PatientEncounter	Clinical:Acts	The PatientEncounter table is used to store additional data related to an encounter that the patient has with the health system. For example, an encounter may represent a single fulfillment of 6 week vaccinations.
ActTag	Metadata	The ActTag table represents a series of version independent metadata tags applied to the act. These can be workflow, security or categorization tags.
ActExtension	Metadata	The ActExtension table represents a series of extended data elements that can assigned to an act's version. This can be to store additional data related to the act itself not representable in the core data framework.

StockLedger	StockManagement	The stock ledger is used to store stock transactions (like a bank account) performed against a Place's stock level. A stock ledger has a 1..1 mapping to a deduction, deposit or transfer of materials between places.
StockBalance	StockManagement	The stock balance table is used to store a running total or balance of a material at a particular place.
StockOrder	StockManagement	A stock order represents a special type of Act (a non-clinical act) which is a request to order stock to/from a place. Mood codes for this include: - Propose – When a planning function proposes an order - Intend – When an order should be placed - Event – When an order has been placed
QuantifiedActParticipation	Clinical:Act	A quantified act participation is a specialization of an act participation that has the ability to convey the number of entities participating in the act. In particular, quantified act participations represent entities which are classes instead of instances of entities.
QuantifiedEntityRelationship	Clinical:Entity	Represents a specialization of an entity association whereby the container entity has a specified number of Source in Target entity.

8.2.3. OpenIZ Concept Model

One of the central design principles in Open Immunize is that of customizable code and valuesets to be used within the OpenIZ system. The concept subset of tables within the datamodel are used to store the master dictionary of concepts used within the OpenIZ system.

There are four major facets of the concept system in OpenIZ:

- **Concept Classes:** A concept class is a classification of what a concept represents. Concept classes are used for validation and organizing the concept dictionary. Examples of a concept class could be UnitOfMeasure for concept which are units of measure.
- **Concepts:** A concept is a central idea of an object or attribute of an object within the OpenIZ database. A concept is, technically speaking, an abstract object which describes a real world thing. For example, the concept of an Arm, or the concept of OPV.

- **Reference Terms:** A reference term is a wire level representation of a concept. A reference term represents a manner in which a concept can be represented to another system. For example, the concept of OPV in the HL7 CVX reference term set is '01'.
- **Concept Sets:** A concept set represents a collection of concept that may be used for a particular purpose. Concept sets are used to restrict the complete set of concepts within the OpenIZ database to those applicable in a particular scope. For example, an entity classification concept must be selected from the EntityClass concept set.

Figure 19 illustrates the relationships between the concepts tables found within the OpenIZ database.

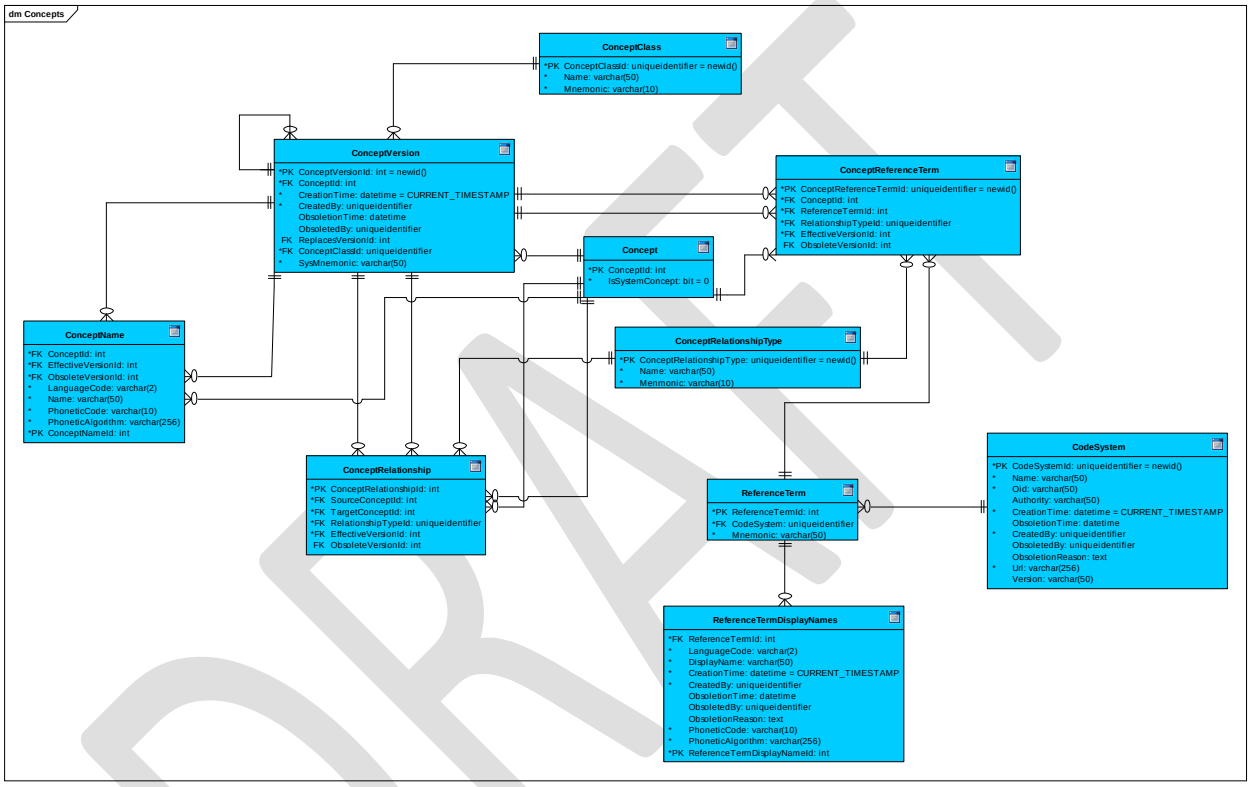


Figure 19 - OpenIZ Concept Table Relations

By default, several concepts are included in the default installation of OpenIZ including language codes, vaccine codes, entity and act classifications, status codes, etc. Additional codes may be added by users and/or may be customized to their environment.

Table 33 - OpenIZ Concept Data Dictionary

Table	Column	Type	Description
ConceptClass	(None)	N/A	The concept class table stores a complete list of concept classifications.
	ConceptClassId	UUID	Represents a unique identifier for the concept classification.

	Name	VARCHAR	Represents a human readable name for the concept classification. Example: Class Codes
	Mnemonic	VARCHAR	Represents a system mnemonic for the concept class. The mnemonic does not change even if the human readable Name column does.
Concept	(None)	N/A	The Concept table stores the key data related to a concept. The Concept table represents immutable concept properties that cannot be changed once a concept is created.
	ConceptId	UUID	A unique identifier for the concept.
	IsSystemConcept	BIT	An indicator which identifies whether the concept is a system concept (i.e. no further versions can be created, cannot be obsoleted, etc.).
ConceptVersion	(None)	N/A	The concept version table is used to store mutable properties of a concept. All edits to a concept's attributes result in a new version being created in the ConceptVersion table.
	ConceptVersionId	UUID	A unique identifier for the concept version.
	VersionSequenceId	INT	A sequence identifier for the version which allows for establishing a time-independent record of version order.
	ConceptId	UUID	The concept to which the version applies.

	CreationTime	DATETIME	The instant in time when the concept version became active (was created). Should default to the current database timestamp.
	CreatedBy	UUID	The user who was responsible for the creation of the version, or the system user if the installation process created the concept version.
	ObsoletionTime	DATETIME	When present, identifies the time when the concept version did become obsolete. This is used whenever a new version is created, the old version is obsoleted.
	ObsoletedBy	UUID	Indicates the user who was responsible for the obsoletion of the record.
	ReplacesVersionId	UUID	Identifies the concept version that the current version of the concept replaces.
	ConceptClassId	UUID	Identifies the classification of the concept as of the version tuple.
	SysMnemonic	VARCHAR	A unique mnemonic used by the system to lookup the concept. The system mnemonic is primarily used for validation purposes where a concept's identifier does not represent a consistent identifier across deployments.
ConceptName	(None)	N/A	The concept name table represents a

			series of human readable names for the concept at a particular version. This facilitates searches as well as translation.
	ConceptNameId	UUID	A unique identifier for the concept name
	ConceptId	UUID	The concept to which the concept name applies.
	EffectiveVersionSequenceId	UUID	The version of the concept when the name did become active.
	ObsoleteVersionSequenceId	UUID	The version of the concept when the concept name was obsoleted.
	Name	VARCHAR	The human readable display name for the concept.
	LanguageCode	CHAR	The ISO639-2 language code for the concept display name.
	PhoneticCode	VARCHAR	The phonetic code for the display name. This is used for phonetic “sounds-like” searches of concepts.
	PhoneticAlgorithmId	UUID	The phonetic algorithm used to generate the phonetic code. This allows deployments to use METAPHONE, SOUNDEX or custom phonetic algorithms appropriate for the language used.
ConceptRelationship	(None)	N/A	The concept relationship table is used to link concepts to one another. Concept relationships can represent equivalency between concepts,

			parent/child relationships, etc.
	ConceptRelationshipId	UUID	The unique identifier for the relationship.
	SourceConceptId	UUID	The concept that represents the source of the relationship.
	TargetConceptId	UUID	The concept which represents the target of the relationship
	EffectiveVerisonId	UUID	Identifies the version of the source concept where the relationship did become active.
	ObsoleteVersionSequenceId	UUID	Identifies the version of the source concept where the relationship is no longer active.
	RelationshipTypeId	UUID	Identifies the type of relationship the concepts have.
ConceptRelationshipType	(None)	N/A	The concept relationship type represents allowed types of relationships that a concept can have.
	ConceptRelationshipTypeId	UUID	The unique identifier for the concept relationship
	Name	VARCHAR	The human readable name of the concept relationship type.
	Mnemonic	VARCHAR	An invariant value that represents the type of relationship typically used by software components.
ReferenceTerm	(None)	N/A	A reference term represents a wire level code that can be used to represent the concept.
	ReferenceTermId	UUID	A unique identifier for the reference term.

	CodeSystemId	UUID	The code system in which the reference term belongs.
	Mnemonic	VARCHAR	The wire level code mnemonic for the reference term.
ConceptReferenceTerm	(None)	N/A	An associative entity that links a concept to one or more reference terms and indicates the strength of the map.
	ConceptReferenceTermId	UUID	A unique identifier for the concept reference term map
	ConceptId	UUID	The concept to which the reference term is linked.
	EffectiveVersionSequenceId	UUID	The version of the concept where the reference term map became effective.
	ObsoleteVersionSequenceId	UUID	The version of the concept where the reference term map became obsolete.
	ReferenceTermId	UUID	The reference term which is associated with the concept.
	RelationshipTypeId	UUID	Identifies the relationship (or strength) that the reference term has with the concept. For example: SAME_AS, NARROWER_THAN, etc.
CodeSystem	(None)	N/A	The code system table represents a master list of all code systems from which a reference term can be drawn.
	CodeSystemId	UUID	A unique identifier for the code system entry.
	Name	VARCHAR	A human readable name for the code system. For example: ICD10

	Oid	VARCHAR	The object identifier that identifies the code system in an interoperable manner.
	Authority	VARCHAR	The organization name who has authority over the code system. Example: World Health Organization.
	CreationTime	DATETIME	The time when the code system entry was created. Default to the current timestamp in the RDBMS.
	CreatedBy	UUID	The user that was responsible for the creation of the code system.
	ObsoletionTime	DATETIME	When populated, indicates the time when the code system record is obsolete.
	ObsoletedBy	UUID	Identifies the user who was responsible for obsoleting the record.
	ObsoletionReason	VARCHAR	The textual description as to why the record was obsoleted.
	Url	VARCHAR	A URI that uniquely identifies the code system. This is primarily used when exposing the code system over REST interfaces.
	Version	VARCHAR	A textual description of the version of the code system that this record represents.
ReferenceTermDisplayNames	(None)	N/A	Like the ConceptName table, the reference term display name table is used to identify human readable display names associated with the reference term.
	ReferenceTermDisplayNameId	UUID	The unique identifier for the reference term.

	ReferenceTermId	UUID	The reference term to which the display name applies.
	LanguageCode	CHAR	The ISO639-2 language code that identifies the language in which the display name is represented.
	DisplayName	VARCHAR	The human readable name for the reference term.
	CreationTime	DATETIME	The time when the display name became active.
	CreatedBy	UUID	Identifies the user that was responsible for the creation of the reference term display name.
	ObsoletionTime	DATETIME	When present, identifies the time when the record should no longer be used.
	ObsoletedBy	UUID	Identifies the user that was responsible for obsoleting the display name.
	ObsoletionReason	VARCHAR	A textual description as to why the display name was obsoleted.
	PhoneticCode	VARCHAR	Represents a phonetic code that can be used in “sounds-like” queries.
	PhoneticAlgorithmId	UUID	Identifies the phonetic algorithm that was used to generate the phonetic code. This allows METAPHONE or SOUNDEX or some other custom language appropriate phonetic algorithm to be used.

8.2.4. OpenIZ Act Model

The act data model describes the tables and fields required for the tracking of acts in the OpenIZ logical model.

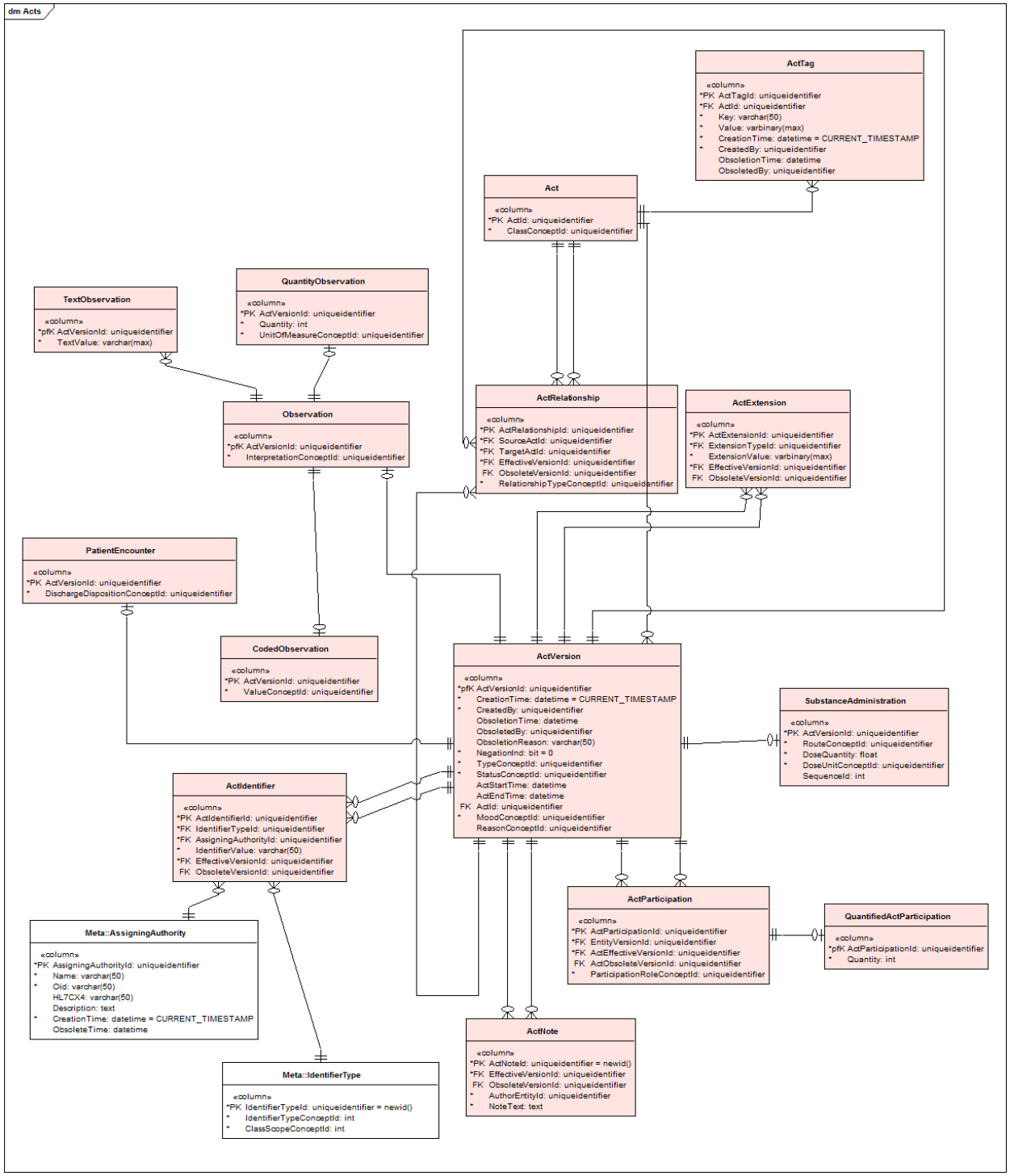


Figure 20 - OpenIZ Act Model

The data dictionary for the OpenIZ act data model is provided in Table 34.

Table 34 - OpenIZ Act Data Model

Table	Column	Type	Description
Act	(None)	N/A	The act table is represents the immutable attributes of an act.
	ActId	UUID	A unique identifier for the act.
	ClassConceptId	UUID	Identifies a concept that classifies the act. This determines the type of act, for example an Observation, PatientEncounter, etc.
	MoodConceptId	UUID	Identifies the mood, or method of the act's performance.
ActTag	(None)	N/A	A table for storing tags related to acts. A tag represents a version independent piece of data attached to a tag.
	ActTagId	UUID	A unique identifier for the tag.
	ActId	UUID	Identifies the act to which the tag is applied.
	Key	VARCHAR	A unique key identifier for the type of tag. A tag's key is used to convey the type of data.
	Value	VARBINARY	Contains the binary data of the tag.
	CreationTime	DATETIME	Identifies the time when the tag became active, or was created.
	CreatedBy	UUID	Identifies the user that was responsible for the creation of the tag.
	ObsoletionTime	DATETIME	When present, identifies the time that the tag data is no longer valid.
	ObsoletedBy	UUID	Identifies the user who obsoleted the act tag.

ActVersion	(None)	N/A	Act events are versioned. The act version table is used to track mutable data.
	ActVersionId	UUID	A unique identifier for the version.
	VersionSequenceId	INT	A sequence identifier for the version which allows for a time independent mechanism for establishing version order.
	ActId	UUID	Identifies the act to which the version data applies.
	CreationTime	DATETIME	Identifies the time when the act version became active (was created)
	CreatedBy	UUID	Identifies the user that was responsible for creating the version.
	ObsoletionTime	DATETIME	When present, identifies the time when the version of the act is no longer active.
	ObsoletedBy	UUID	Identifies the user who was responsible for the obsoletion of the version.
	NegationInd	BIT	When present, indicates that the act's value is not true. For example, when attempting to convey that a vaccine was not given, the negationInd would be set to true.
	TypeConceptId	UUID	Identifies the type of act. This is a type that is a subclass within the major classification. For example, if the class is a substance administration, the type concept may

			represent an Immunization.
	StatusConceptId	UUID	Identifies the status of the act as of the current version.
	ActTime	DATETIME	Identifies the time that the act did occurred, should occur.
	ActStartTime	DATETIME	Identifies the start time of the act.
	ActStopTime	DATETIME	Identifies the stop time of the act.
ActRelationship	(None)	N/A	The act relationship table is used to track the relationship of acts to one another.
	ActRelationshipId	UUID	Uniquely identifies the act relationship.
	SourceActId	UUID	Identifies the source act of the relationship.
	TargetActId	UUID	Identifies the target act of the relationship.
	EffectiveVersionSequenceId	UUID	Identifies the version of the source act where this relationship did become active.
	ObsoleteVersionSequenceId	UUID	Identifies the version of the source act where this relationship is no longer active.
	RelationshipTypeConceptId	UUID	Identifies the type of relationship that the two acts have to one another.
ActParticipation	(None)	N/A	The ActParticipation table is used to track how entities participate in a particular act.
	ActParticipationId	UUID	Uniquely identifies the act participation.
	EntityVersionId	UUID	Identifies the version of the entity that was active when the entity participated in the act participation.

	VersionSequenceId	INT	Identifies the sequence in which the version was created.
	EffectiveVersionSequenceId	UUID	Identifies the version of the act when the participation is active.
	ObsoleteVersionSequenceId	UUID	When present, identifies the version of the act when the participation is no longer active.
	RoleConceptId	UUID	Qualifies what role the entity played in the carrying out of the act.
QuantifiedActParticipation	(None)	N/A	The quantified act participation table is a specialization of an act participation whereby a number of the same entity play the same role in the act.
	ActParticipationId	UUID	Identifies the act participation which this quantified act participation extends.
	Quantity	INT	Identifies the number of entities that are included in the playing of the role.
ActIdentifier	(None)	N/A	The act identifier table is used to store alternate identifiers for the act. This may include vaccine event identifiers, external order identifiers, etc.
	ActIdentifierId	UUID	Uniquely identifies the alternate act identifier.
	IdentifierTypeId	UUID	Identifies the type of identifier this particular identifier instance represents (order #, etc.)
	AssigningAuthorityId	UUID	Identifies the authority that assigned the identifier.

	IdentifierValue	VARCHAR	The actual external identifier value.
	EffectiveVersionSequenceId	UUID	Identifies the version of the act where the alternate identifier became active.
	ObsoleteVersionSequenceId	UUID	When present, identifies the version of the act whereby the alternate identifier is no longer active.
ActExtension	(None)	N/A	The act extension table is used to store extensions attached to acts.
	ActExtensionId	UUID	A unique identifier for the act extension.
	ExtensionTypeId	UUID	Identifies the type of extension represented. This includes information on how the extension should be serialized to/from the ExtensionValue column.
	ExtensionValue	VARBINARY	Carries the value of the extension.
	ExtensionDisplay	VARCHAR	A human comprehendible display value for the extension.
	EffectiveVersionSequenceId	UUID	Indicates the version of the act where this extension became active.
	ObsoleteVersionSequenceId	UUID	When present, indicates the version of the act where the extension is no longer active.
Observation	(None)	N/A	The observation table is used to store extended values related to observation act types.
	ActVersionId	UUID	Identifies the act version to which this extended data applies.

	InterpretationConceptId	UUID	Identifies the concept that represents the interpretation of the observation.
QuantityObservation	(None)	N/A	The quantity observation table is used to store extended information related to the observations that carry quantified values.
	ActVersionId	UUID	Identifies the observation act version to which the quantified observation data applies.
	Quantity	DECIMAL	A decimal value that contains the value of the observation quantity.
	QuantityPrecision	INT	Identifies the precision of the Quantity field.
	UnitOfMeasureConceptId	UUID	Identifies the concept that identifies the units of measurement.
TextObservation	(None)	N/A	The text observation table is used to store additional information related to a text valued observation.
	ActVersionId	UUID	Identifies the observation version id that this text observation data is attached to.
	TextValue	TEXT	A textual field that contains the observation data.
CodedObservation	(None)	N/A	The coded observation table is used to store additional data related to observations that are coded. Problems and Allergies would qualify as coded observations.
	ActVersionId	UUID	Identifies the version of the observation that

			this coded observation value data applies.
	ConceptValueId	UUID	Identifies the concept that represents the value of observation.
SubstanceAdministration	(None)	N/A	The substance administration table is used to store data related to substance administrations to a patient. This include vaccines or any type of Epupin injections for
	ActVersionId	UUID	Identifies the act version to which the substance administration data applies.
	RouteConceptId	UUID	Identifies the concept that describes the route that was taken to administer the substance. This may be drawn from the default route if not supplied.
	DoseQuantity	DECIMAL	Identifies the dosage that was given to the patient.
	DoseQuantityPrecision	INT	Identifies the precision of the dose quantity.
	DoseUnitConceptId	INT	Identifies the dose unit of measure that was given to the patient.
	SequenceId	INT	Identifies the sequence of this dose if it is a part of a sequence of doses.
PatientEncounter	(None)	N/A	The patient encounter table is used to store additional data related to an act that represents a patient encounter.
	ActVersionId	UUID	Identifies the act to which the extended patient encounter data applies.

	DischargeDispositionConceptId	UUID	Identifies the disposition in which the patient left the encounter.
ActNote	(None)	N/A	The act note table is used to store notes associated with an act.
	ActNoteId	UUID	A unique identifier for the note.
	EffectiveVersionSequenceId	UUID	The version whereby the note became effective
	ObsoleteVersionSequenceId	UUID	When populated, indicates the version of the act where the note is no longer active.
	AuthorEntityId	UUID	The identifier of the entity that wrote the note.
	NoteText	TEXT	The textual content of the note.

8.2.5. OpenIZ Security Model

One of the key tenants of the OpenIZ immunization management system is privacy and security by design. To that end, OpenIZ's IMS supports not only external policy enforcement decisions and role providers, but also provides access to internal policy engines (when external policy decision points are not available).

Figure 21 illustrates the relationships between the various security sub systems tables found in the OpenIZ data model.

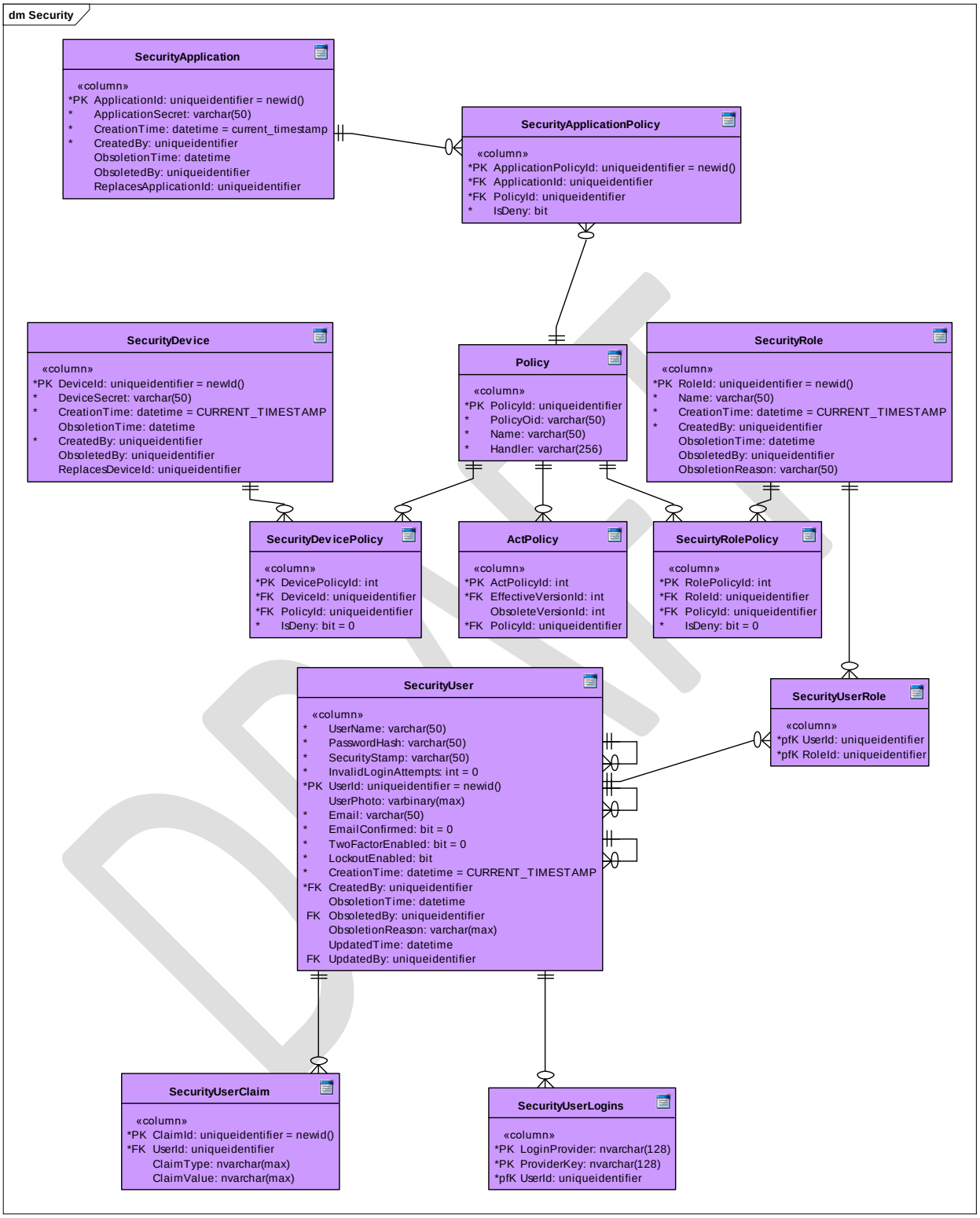


Figure 21 - OpenIZ Security Model

Describes the data dictionary for the OpenIZ security model.

Table	Column	Type	Description

Policy	(None)	N/A	The policy table is a complete dictionary of policies that can be applied to acts within the OpenIZ IMS.
	PolicyId	UUID	Uniquely identifies the policy within the OpenIZ system.
	PolicyOid	VARCHAR	A globally unique identifier in the form of an OID for the policy.
	Name	VARCHAR	A human readable name for the policy.
	Handler	VARCHAR	An assembly qualified name (AQN) of an IPolicyHandler implementation which is triggered when the policy rule fires.
SecurityUser	(None)	N/A	The security user table is used to store a master list of users that have secured access to the OpenIZ IMS functions.
	UserId	UUID	A unique identifier for the user.
	UserName	VARCHAR	A unique identifier for the security user that a human may use to access the OpenIZ IMS system.
	PasswordHash	VARCHAR	A SHA256 hash of the user's password.
	SecurityStamp	VARCHAR	A unique security stamp for the user account. This can include a salt for the user password, or some other security tag for the user.
	InvalidLoginAttempts	INT	Identifies the number of times that a person has attempted to access the OpenIZ IMS with invalid credentials.
	UserPhoto	VARBINARY	An optional photograph for the user.
	Email	VARCHAR	Identifies an electronic mail telecommunications address that can be used to contact the user.
	EmailConfirmed	BIT	Indicates whether the email address of the user has been confirmed.
	TwoFactorEnabled	BIT	Indicates whether the user account requires two-factor

			authentication. The TFA mechanism is enabled by the ITwoFactorAuthenticationService implementation.
	LockoutEnabled	BIT	Indicates whether the user account is in a state of lockout.
	CreationTime	DATETIME	Identifies the time when the user account was created.
	CreatedBy	UUID	Identifies the user who was responsible for the creation of the security user.
	ObsoletionTime	DATETIME	When populated, indicates the time when the user account did or will become obsolete.
	ObsoletedBy	UUID	The identifier of the user who was responsible for obsoleting the record.
	ObsoletionReason	VARCHAR	Identifies the reason why the security user was obsoleted.
	UpdatedTime	DATETIME	Identifies the last timestamp that the user record was updated.
	UpdatedBy	UUID	Identifies the user who was responsible for the last edit of the security user.
SecurityUserClaims	(None)	N/A	The security user claims table is used to store claim tokens associated with a user account/session.
	ClaimId	UUID	A unique identifier of the claim
	UserId	UUID	Identifies the user to which the claim applies.
	ClaimType	VARCHAR	Identifies the type or classification of claim that has been made.
	ClaimValue	VARCHAR	Identifies the value of the claim token
SecurityUserLogins	(None)	N/A	The security user logins table is used to track external authorization providers associated with a user account.
	LoginProvider	VARCHAR	The provider (google, Microsoft, etc.) which holds the external credential.
	ProviderKey	VARCHAR	The key of the user identifier in the provider system.

	UserId	UUID	Identifies the user to which the external login applies.
SecurityRole	(None)	N/A	The security role table is used to store security (user) roles that can be used in policy based decisions.
	RoleId	UUID	Uniquely identifies the security role.
	Name	VARCHAR	A human readable name for the role.
	CreationTime	DATETIME	Identifies the moment in time when the security role was created.
	CreatedBy	UUID	Identifies the user who was responsible for the creation of the role.
	ObsoletionTime	DATETIME	When present, identifies the date/time when the role became obsolete.
	ObsoletedBy	UUID	Identifies the user who was responsible for the obsoletion.
	ObsoletionReason	VARCHAR	Indicates the reason for the obsoletion of the record.
SecurityUserRole	(None)	N/A	An associative entity table between a security user and role.
	UserId	UUID	Identifies the user of the association.
	RoleId	UUID	Identifies the role to which the association applies.
SecurityRolePolicy	(None)	N/A	The security role policy is an associative entity table that links security roles to policies which can be used in a policy decision.
	RolePolicyId	UUID	Uniquely identifies the tuple
	RoleId	UUID	Identifies the role to which the security role policy association applies.
	PolicyId	UUID	Identifies the policy that is being applied to the role.
	IsDeny	INT	When true, indicates that the policy decision process should deny all requests to the policy.
	CanOverride	BIT	When true, indicates that when a policy decision is made, a user

			within the role can override the decision.
PolicyOverride	(None)	N/A	A table that stores data related to policy overrides.
	PolicyOverrideId	UUID	Uniquely identifies the policy override record.
	PolicyId	UUID	Identifies the policy that was overridden.
	UserId	UUID	Identifies the user who was responsible for the override.
	ReasonConceptId	UUID	Identifies the reason why the policy was overridden.
	OverrideTime	DATETIME	The time that the override occurred.
	CreationTime	DATETIME	Identifies the time when the override record was created.
	CreatedBy	UUID	Identifies the user who was responsible for the creation of the override record.
ActPolicy	(None)	N/A	The ActPolicy table is used to associate a policy with an act.
	ActPolicyId	UUID	A unique identifier for the policy identifier.
	EffectiveVersionSequenceId	UUID	Identifies the version of the act whereby the policy is active.
	ObsoleteVersionSequenceId	UUID	Indicates the version of the act where the policy no longer applies.
	PolicyId	UUID	Identifies the policy that is associated with the act.
SecurityDevice	(None)	N/A	The security device table is used to store data related to an authorized device that can access the OpenIZ IMS.
	DeviceId	UUID	Uniquely identifies the device.
	DeviceSecret	VARBINARY	A secret that is used to verify whether the device can connect.
	CreationTime	DATETIME	Indicates the time when the record was created.
	CreatedBy	UUID	Identifies the user responsible for the creation of the record.
	ObsoletionTime	DATETIME	When present, indicates the time when the device record became or will become obsolete.

	ObsoletedBy	UUID	Identifies the user that is responsible for the obsoletion of the device.
	ReplacesDeviceId	UUID	Indicates the old device that the current device would replace.
SecurityDevicePolicy	(None)	N/A	An associated entity that links a security device to a policy.
	DevicePolicyId	UUID	A unique identifier for the device policy association.
	DeviceId	UUID	Identifies the device to which the association applies.
	PolicyId	UUID	Indicates the policy to which the association applies.
	IsDeny	BIT	When true, instructs the decision engine to deny access to an act or policy.
SecurityApplication	(None)	N/A	The security application table is used to store records associated with an application.
	ApplicationId	UUID	Uniquely identifies the application.
	ApplicationSecret	VARBINARY	A secret that is used by the application to authenticate itself.
	CreationTime	DATETIME	The time when the application was created.
	CreatedBy	UUID	The user responsible for registering the application.
	ObsoletionTime	DATETIME	The time that the application record did become or will become obsolete.
	ObsoletedBy	UUID	Indicates the user that the obsoleted the record.
	ReplacesApplicationId	UUID	Identifies the application that this current version of the application record replaces.
SecurityApplicationPolicy	(None)	N/A	An associated entity that links a security application to a policy.
	ApplicationPolicyId	UUID	A unique identifier for the application policy association.
	ApplicationId	UUID	Identifies the application to which the association applies.
	PolicyId	UUID	Indicates the policy to which the association applies.

	IsDeny	BIT	When true, instructs the decision engine to deny access to an act or policy.

8.2.6. OpenIZ Stock Model

The OpenIZ stock model represents a series of stock management tracking tables used by the default stock management implementation within OpenIZ. It is envisioned that alternate stock management interfaces may not require these tables and thus, some OpenIZ installations would not carry these tables.

The stock management tables are used to track the amount of material entities associated, or known to exist, within a place entity.

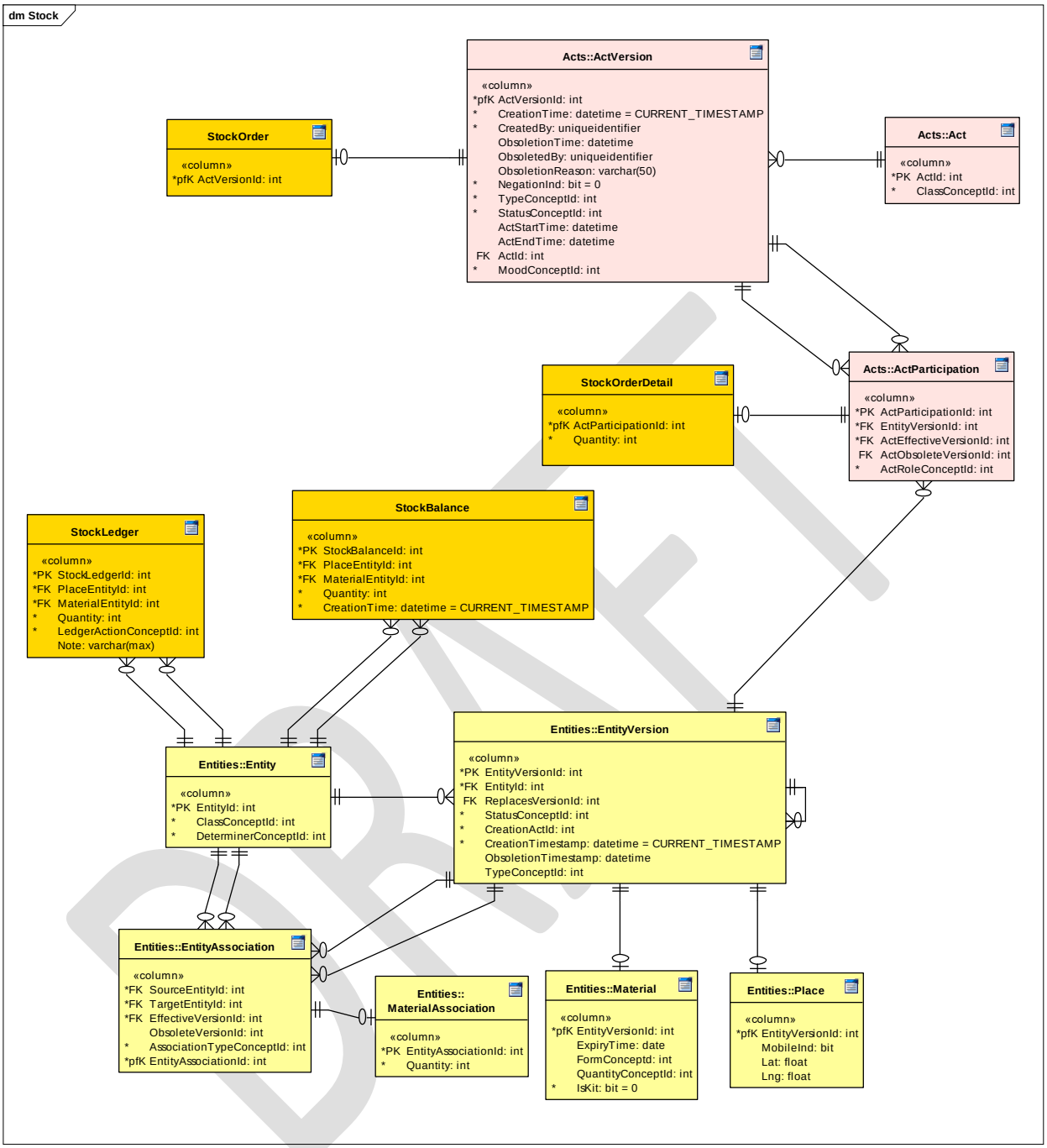


Table	Column	Type	Description
StockBalance	(None)	N/A	The stock balance table represents a working balance table whose primary purpose is the storage of current inventory balance per material entity within a place.
	StockBalanceId	UUID	Uniquely identifies the stock balance tuple.

	PlaceEntityId	UUID	Identifies the place entity to which the stock entry applies.
	MaterialEntityId	UUID	Identifies the material that is represented in the balance tuple.
	LedgerActionConceptId	UUID	Identifies the stock balance type. This may be the balance of allocated items, balance of wasted items, etc.
	Quantity	INT	Identifies the number of the MaterialEntityId currently stocked at the PlaceEntityId.
	CreationTime	DATETIME	Identifies the time that the stock balance entry was created.
	UpdatedTime	DATETIME	Identifies the time that the stock balance was last updated.
	UpdatedBy	UUID	Identifies the user that was responsible for the updating of the balance entry.
StockLedger	(None)	N/A	The stock ledger table is used as backing data to the balance of the facility. It represents the ledger items for stock within a place entity. All immunizations, allocations, transfers, deposits, etc. of stock are tracked in this table.
	StockLedgerId	UUID	Uniquely identifies the stock ledger item.
	PlaceEntityId	UUID	Identifies the place to which the stock ledger action was performed.
	MaterialEntityId	UUID	Identifies the material to which the stock ledger action was performed.
	Quantity	INT	Identifies the number of MaterialEntities to which the stock ledger item was “done”
	LedgerActionConceptId	UUID	Identifies the ledger action (transfer, adjustment, deposit, use, waste, etc.)
	Note	VARCHAR	Identifies a note attached to the stock ledger action entered by the user.
	CreationTime	DATETIME	Identifies the time when the ledger action was created.
	CreatedBy	UUID	Identifies the user that was responsible for the stock ledger action.
StockOrder	(None)	N/A	The StockOrder table represents a sub-class of an Act that is a stock

			order. The act of ordering stock will result in the creation of a stock order act.
	ActVersionId	UUID	Identifies the version of the act to which the stock order applies.

8.2.7. OpenIZ Entity Model

The OpenIZ entity model represents a series of tables which are responsible for the tracking of entities within the OpenIZ data model. Entities represent people, places, organizations, things, etc. and are responsible for participating within acts in some capacity.

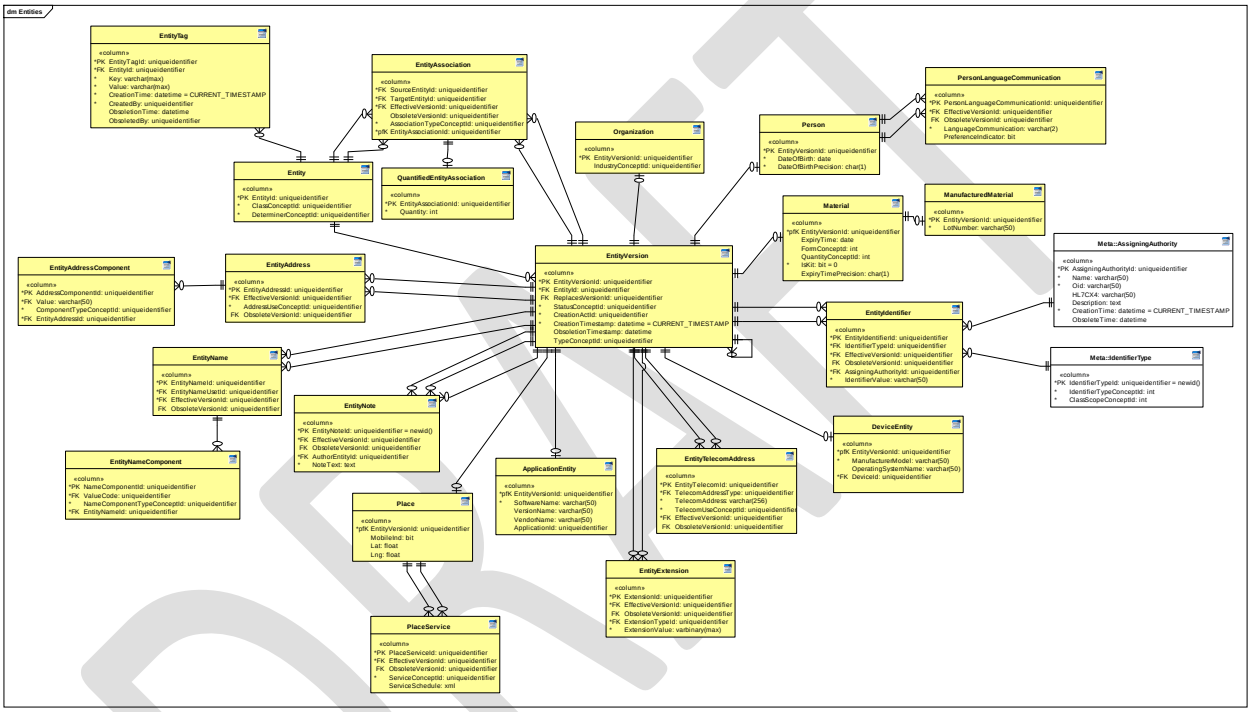


Table	Column	Type	Description
Entity	(None)	N/A	The entity table is responsible for the storage of immutable attributes of an entity.
	EntityId	UUID	Uniquely identifies the entity within the context of the OpenIZ implementation.
	ClassConceptId	UUID	Identifies the concept that classifies the entity

			by a type. The classifier is used to determine “WHAT TYPE” of entity the tuple represents such as a person, material, manufactured material, organization, place, etc.
	DeterminerConceptId	UUID	Identifies the concept that classifies or determines the type of entity. This is either an INSTANCE or CLASS concept identifier.
EntityTag	(None)	N/A	The entity tag table is used to store version independent tags associated with an entity. A tag does not result in new versions of the entity and is used to track additional data related to security and/or workflow related metadata.
	EntityTagId	UUID	Uniquely identifies the entity tag.
	EntityId	UUID	Identifies the entity to which the tag is associated.
	Key	VARCHAR	Qualifies the type of tag associated with the entity. That is to say, type of tag is represented in the tuple of the determiner.
	Value	VARCHAR	A value that carries the data associated with the tag value.

	CreationTime	DATETIME	Indicates the date/time at which time the tag was created.
	CreatedBy	UUID	Identifies the user that was responsible for the creation of the tag.
	ObsoletionTime	DATETIME	When populated, indicates the time when the tag is no longer associated with the entity.
	ObsoletedBy	UUID	Identifies the user that was responsible for obsoleting the tag.
EntityVersion	(None)	N/A	The entity version table is used to store the mutable attributes of an entity, that is to say, any fields associated with an entity that may evolve over the lifespan of the entity are tracked in this table.
	EntityVersionId	UUID	Uniquely identifies the version of the entity represented in the tuple.
	EntityId	UUID	Identifies the entity to which this version applies.
	ReplacesVersionId	UUID	Identifies the version of the entity that the current tuple is responsible for replacing.
	StatusConceptId	UUID	Identifies the status of the entity as of the version represented in the tuple.

	CreationTime	DATETIME	Indicates the time when the entity was created.
	CreatedBy	UUID	Identifies the user that was responsible for the creation of the entity.
	ObsoletionTime	DATETIME	When populated, indicates the time when the entity version became obsolete.
	ObsoletedBy	UUID	Identifies the user that was responsible for the obsoleting of the record.
	TypeConceptId	UUID	Indicates the concept that classifies the subtype of entity. For example, an entity may be a provider; however, the sub-type may be a “physiotherapist”.
EntityRelationship	(none)	N/A	The entity association table is used to associate two or more entities together. An association is made between a source entity and a target entity.
	EntityRelationshipId	UUID	Uniquely identifies the entity association.
	SourceEntityId	UUID	Identifies the source of the entity association.
	TargetEntityId	UUID	Identifies the target of the entity association.
	EffectiveVersionSequenceId	UUID	Indicates the version of the source entity at which time this entity association

			was created or became effective.
	ObsoleteVersionSequenceId	UUID	When populated, indicates that the entity association is no longer active, and indicates the version of the source entity where the association ceased to be applicable.
	AssociationTypeConceptId	UUID	Classifies the relationship between the two entities. Can indicate ownership roles such as “Place OWNS Material”, or relationship “Patient CHILD OF Person”.
QuantifiedEntityRelationship	(None)	N/A	A specialization of the entity association that contains a quantity of target entities within the source entity.
	EntityRelationshipId	UUID	Identifies the EntityRelationship to which the extended information applies.
	Quantity	INT	Indicates the quantity of target entities contained within the source entity.
EntityNote	(None)	N/A	The entity note table is used to store textual notes related to an etity.
	EntityNoteId	UUID	Uniquely identifies the note.
	EffectiveVersionSequenceId	UUID	Identifies the version of the entity to which the note applies.

	ObsoleteVersionSequenceId	UUID	When populated, indicates the version of the entity where the note is no longer relevant.
	AuthorEntityId	UUID	Identifies the entity that was responsible for the authoring of the note.
	NoteText	TEXT	Indicates the textual content of the note.
EntityAddress	(None)	N/A	The entity address table is used to store address information (physical addresses) related to an entity.
	EntityAddressId	UUID	Uniquely identifies the entity address.
	EffectiveVersionSequenceId	UUID	Identifies the version of the entity whereby the address information became active.
	ObsoleteVersionSequenceId	UUID	When populated, indicates the version of the entity whereby the address is no longer applicable.
	AddressUseConceptId	UUID	Indicates the desired use of the address. Examples include physical visit, vacation home, contact, mailing, etc.
EntityAddressComponent	(None)	N/A	The entity address component table is used to store the address components associated with a particular entity address.
	EntityAddressComponentId	UUID	Uniquely identifies the entity address component.

	Value	VARCHAR	Identifies the value of the of the address component
	ComponentTypeConceptId	UUID	Classifies the type of address component represented in the value field. For example: street name, city, country, postal code, etc.
	EntityAddressId	UUID	Identifies the entity address to which the entity address component applies.
EntityName	(None)	N/A	The entity name table is used to store master list of names associated with an entity.
	EntityNameId	UUID	Uniquely identifies the entity name.
	EntityNameUseId	UUID	Classified the intended use of the entity name. Examples: maiden name, legal name, license name, artist name, etc.
	EffectiveVersionSequenceId	UUID	Identifies the version of the entity when this name became active.
	ObsoleteVersionSequenceId	UUID	When populated, identifies the version of the entity where the name is no longer active.
EntityNameComponent	(None)	N/A	The entity name component table is responsible for the storage of name components that comprise an entity name.
	NameComponentId	UUID	Uniquely identifies the name component.

	ValueCode	UUID	Indicates the phonetic value tuple that stores the name value.
	NameComponentTypeConceptId	UUID	Classifies the type of name component represented. Examples: first name, title, family name, etc.
	EntityNameId	UUID	Indicates the entity name to which the name component applies.
EntityIdentifier	(None)	N/A	The entity identifier is table is responsible for the storage of alternate identifies associated with the entity.
	EntityIdentifierId	UUID	Uniquely identifies the entity identifier.
	IdentifierTypeId	UUID	Classifies the type of identifier that is represented by the entity identifier. Examples: business identifier, mrn, primary identifier, etc.
	EffectiveVersionSequenceId	UUID	Indicates the version of the entity when the identifier became active.
	ObsoleteVersionSequenceId	UUID	When populated, indicates the version of the entity where the identifier is no longer active.
	AssigningAuthorityId	UUID	Identifies the authority that was responsible for the assigning of the identifier.
	IdentifierValue	VARCHAR	Indicates the value of the entity identifier.

Place	(None)	N/A	The place table represents a specialization of the Entity table which is used to represent physical places such as clinics, outreach activity sites, etc.
	EntityVersionId	UUID	Identifies the version of the entity to which the place data applies.
	MobileInd	BIT	Indicator that is used to identify that a place is mobile.
	Lat	FLOAT	The latitudinal position of the place expressed in degrees latitude.
	Lng	FLOAT	The longitudinal position of the place expressed in degrees longitude.
PlaceService	(None)	N/A	The place service table is used to identify the services that are provided at a particular place. Services may include stocking, transfer depots, immunization.
	PlaceServiceId	UUID	A unique identifier for the place service.
	EffectiveVersionSequenceId	UUID	The version of the place entity where the service entry is active.
	ObsoleteVersionSequenceId	UUID	When populated, indicates the version of the place entity where the service entry is no longer valid.
	ServiceConceptId	UUID	Indicates a concept that describes the service offered.

	ServiceSchedule	XML	An XML expression of the service schedule.
	ServiceScheduleType	VARCHAR	Identifies the type of data stored in the service schedule column (iCal, GTS, etc.)
ApplicationEntity	(None)	N/A	The application entity table is used to store entity data related to an application. An application is a software program that runs on a device.
	EntityVersionId	UUID	Identifies the version of the entity to which the application data applies.
	SoftwareName	VARCHAR	Identifies the name of the software package (“EMR Package” is an example)
	VersionName	VARCHAR	Identifies the version of the software (example: “1.0”)
	VendorName	VARCHAR	The name of the vendor which distributes the software application (example: “ABC Corp”)
	ApplicationId	UUID	When populated, links the application entity to a security application.
EntityExtension	(None)	N/A	The entity extension table is used to store additional, clinically relevant, versioned data attached to an entity that cannot

			be stored in the native data model.
	EntityExtensionId	UUID	Uniquely identifies the extension.
	EffectiveVersionSequenceId	UUID	Indicates the version of the entity when the extension data did become active.
	ObsoleteVersionSequenceId	UUID	When populated, indicates the version of the entity where the extension value is no longer applicable.
	ExtensionTypeId	UUID	Indicates the type, or handler, for the extension data.
	ExtensionData	VARBINARY	Serialized data that contains the raw value of the extension (serialized and de-serialized by the handler).
	ExtensionDisplay	VARCHAR	A textual, human readable expression of the extension value which can be displayed on reports, etc.
EntityTelecomAddress	(None)	N/A	The entity telecommunications address table is used to store data related to telecommunications addresses (email, fax, phone, etc.) for an entity.
	EntityTelecomId	UUID	Uniquely identifies the telecommunications address.
	TelecomAddressType	UUID	Classifies the type of address represented (example: phone, fax, email, etc.)

	TelecomAddress	VARCHAR	The value of the telecommunications address in RFC-2396 format.
	TelecomUseConceptId	UUID	Identifies the intended use of the telecom address. (Example: home, work, etc.)
	EffectiveVersionSequenceId	UUID	Identifies the version of the entity whereby the telecom address became effective.
	ObsoleteVersionSequenceId	UUID	When populated, identifies the version of the entity where the telecom address is no longer valid.
DeviceEntity	(None)	N/A	The device table is used to store device information.
	EntityVersionId	UUID	Indicates the version of the entity to which the device data applies.
	ManufacturerModel	VARCHAR	Indicates the name of the manufacturer of the device.
	OperatingSystemName	VARCHAR	Indicates the name of the operating system installed on the device.
	DeviceId	UUID	When populated, identifies the security device associated with the device entity.
Material	(None)	N/A	A material represents a physical thing (syringe, drug, etc.) which participates in an act or is assigned to a person.

	EntityVersionId	UUID	Identifies the version of the entity to which the material data applies.
	ExpiryTime	DATETIME	Indicates the time when the material will expire.
	ExpiryTimePrecision	CHAR	Indicates the precision that the expiry time has.
	FormConceptId	UUID	Identifies a concept that denotes the form that the material takes. Examples: capsule, injection, nebulizer, etc. For drugs and vaccines, the form will imply the route of administration.
	QuantityConceptId	UUID	Indicates the unit of measure for a single unit of the material. Examples: dose, mL, etc.
	Quantity	NUMERIC	Indicates the reference quantity in UOM. For example, BCG MMAT is 5 mL of BCG Antigen
	IsKit	BIT	An indicator that is used to identify whether the material is a real material or an administrative material for the purpose of management.
ManufacturedMaterial	(None)	N/A	A manufactured material is a specialization of a material that is manufactured.

	EntityVersionId	UUID	Indicates the version of the material to which the specialized data applies.
	LotNumber	VARCHAR	Indicates the manufacturer lot for the material.
Person	(None)	N/A	Person represents a specialization of Entity representing a person.
	EntityVersionId	N/A	The version of the entity to which the person data applies.
	DateOfBirth	DATE	Indicates the date on which the person entity was born.
	DateOfBirthPrecision	CHAR	Indicates the precision of the date of birth field.
PersonLanguage Communication	(None)	N/A	The person language communication table is used to store information related to the person's language preferences. This can be used by the user interface to determine which language to display, however is also clinically relevant to indicate the language in which a patient wishes to receive communciations.
	PersonLanaugeCommunication Id	UUID	Uniquely identifies the language of communication.
	EffectiveVersionSequenceId	UUID	Indicates the version of the person entity whereby the language of

			communication is effective.
	ObsoleteVersionSequenceId	UUID	When present, indicates the version of the person entity where the language of communication is no longer effective.
	LanguageCommunication	VARCHAR	An ISO-639-2 language code indicating the language preference.
	PreferenceIndicator	BIT	Indicates whether the person prefers the language for communications.
Organization	(None)	N/A	The organization table represents a specialization of an entity representing a logical organization.
	EntityVersionId	UUID	Indicates the version of the entity to which the organization specialization applies.
	IndustryConceptId	UUID	Indicates the industry in which the organization operates. Examples: logistics, healthcare, etc.

8.2.7.1. Relationships between Entities

There are two types of relationships that can exist between entities (quantified and unquantified). An unquantified relationship represents a 1:1 relationship between things and merely identifies that two items are related in some manner. For example, one may say John [Patient] is related to Mary [Person] by way that Mary is John's mother.

Quantified relationships are used for expressing when a certain number of entities are related to a parent. For example, a Box of BCG [Container] contains 25 BCG [Material]. Quantified relationships are only used to represent per relationships and do not take the quantity UOM into consideration (for that use the quantity field on the actual Entity). For example, a box of GSK BCG vial of 5 mL may contain 25 5

ml BCG vials, where each BCG vial is 5 ml of BCG. This more complex relationship is illustrated in Figure 22.

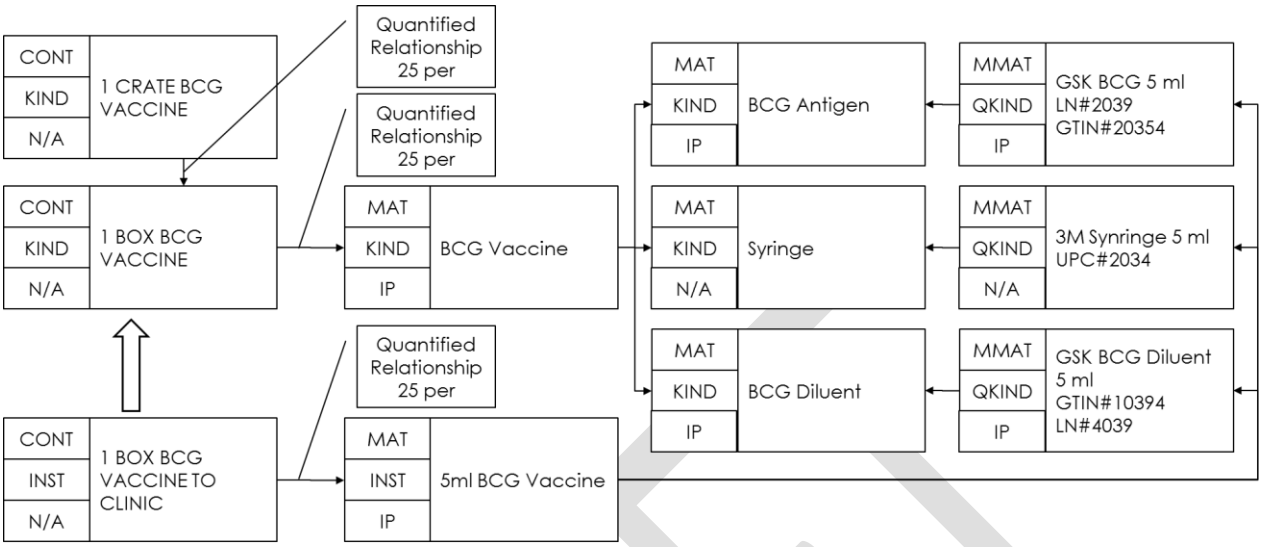


Figure 22 - Entity Relationships

Additionally the association is qualified by the role code in which the target plays in the source. Examples of relationship types are:

Relationship	Mnemonic	Description
Qualified Entity		
Next of Kin / Family Code		
Personal Relationship		
Employee		
Healthcare Provider		
Manufactured Product		
Used Entity		
Therapeutic Agent		
Administrable Material		

In the example above. 3M Syringe 5ml is an manufactured material of Syringe, while Syringe is an part of BCG Dose, BCG Dose is a part for box of BCG and so on.

8.2.8. OpenIZ Protocol Model

The OpenIZ protocol model is used to track the types of clinical protocols and links between an encounter and a clinical protocol. Protocols can be expressed in a variety of manners and are adhered to by their protocol handler. The protocol handler identifies which piece of code should be invoked to handle the particular workflow.

Protocol handlers are pieces of code which run in the backbone IMS application and may schedule future events, analyse prior events and execute tasks, etc. Whenever an Act is associated with an Act protocol any addition, modification, or link made to that Act will trigger the execution of the protocol handler.

Examples of protocols can include vaccination schedules, appointment scheduling, adverse event treatment, etc.

The protocol handler and associative entity tables are illustrated in Figure 23 and described in .

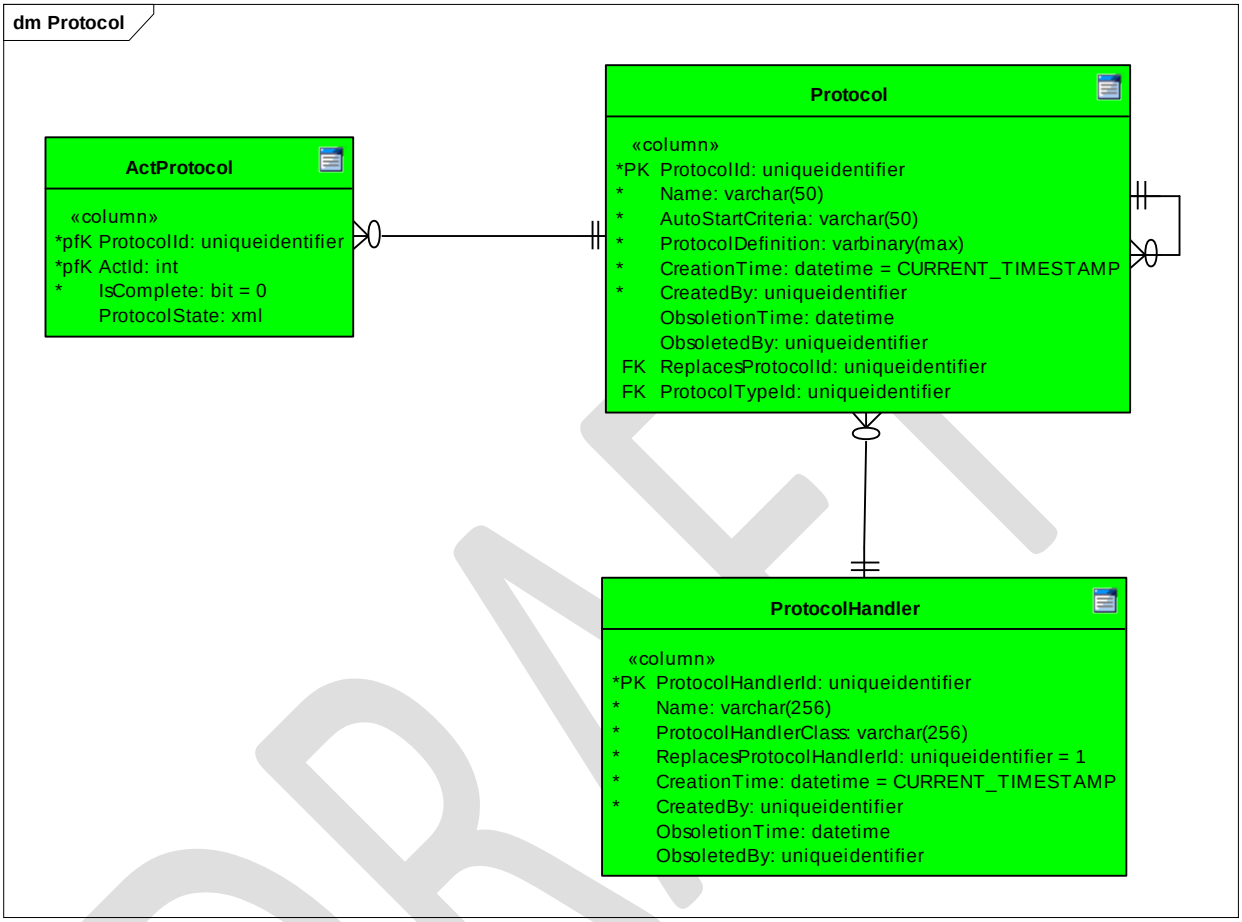


Figure 23 - OpenIZ Protocol Tables

Table	Column	Type	Description
Protocol	(None)	N/A	The protocol table is used to store discrete protocols related to immunization, reporting, etc.
	ProtocolId	UUID	Uniquely identifies the protocol within the OpenIZ data model.
	Name	VARCHAR	A human readable name for the protocol. Example: “BCG Immunization Scheduling”

	AutoStartCriteria	VARCHAR	A predicate which, when true, triggers the automated start of the protocol.
	ProtocolDefinition	VARBINARY	An implementation specific definition of the protocol. This can be BPMN, RulesML, JavaScript, etc. The protocol definition must be understood by the associated handler.
	CreationTime	DATETIME	Identifies the time when the protocol definition was created.
	CreatedBy	UUID	Identifies the user who was responsible for the creation of the protocol definition.
	ObsoletionTime	DATETIME	When present, indicates the time when the protocol definition is or was no longer active.
	ObsoletedBy	UUID	Identifies the user who was responsible for obsoleting the protocol.
	ReplacesProtocolId	UUID	Identifies the protocol which the current tuple replaces.
	ProtocolTypeId	UUID	Identifies the protocol handler which should be used to execute the protocol.
ProtocolHandler	(None)	N/A	The protocol handler table is used maintain a registration of all handlers which are responsible for the execution of protocols.
	ProtocolHandlerId	UUID	Uniquely identifies the protocol handler.

	Name	VARCHAR	A human readable name for the protocol handler type.
	ProtocolHandlerClass	VARCHAR	The assembly qualified name of the handler which executes the protocol. Must implement IProtocolHandler
	CreationTime	DATETIME	Identifies the time when the protocol handler was registered.
	CreatedBy	UUID	Identifies the user which was responsible for the creation of the protocol handler.
	ObsoletionTime	DATETIME	When present, identifies the time when the protocol handler is no longer active.
	ObsoletedBy	UUID	Identifies the user that was responsible for the obsoleting of the protocol handler entry.
	ReplacesProtocolHandlerId	UUID	Identifies the protocol handler registration that the current tuple replaces.
ActProtocol	(None)	N/A	The act protocol table is used to associate a protocol definition with an Act. This association identifies the “why” an act was created (i.e. it was associated with an protocol)
	ProtocolId	UUID	Identifies the protocol definition to which the association applies.
	ActId	UUID	Identifies the act which the protocol is associated with.

	IsComplete	BIT	Identifies whether the act completed the protocol (terminated it).
	ProtocolState	VARBINARY	A handler specific piece of data which can be used to store application specific state related to the instance of the protocol definition.

8.3. Physical Data Design

Because OpenIZ can leverage a variety of data storage mechanisms via the IDataPersistenceService<T> implementations, all OpenIZ plugins and core functions use a business model. IDataPersistenceService implementations are, therefore, responsible for the conversion of queries and data entering/exiting the implementation. This mechanism is illustrated in Figure 24.

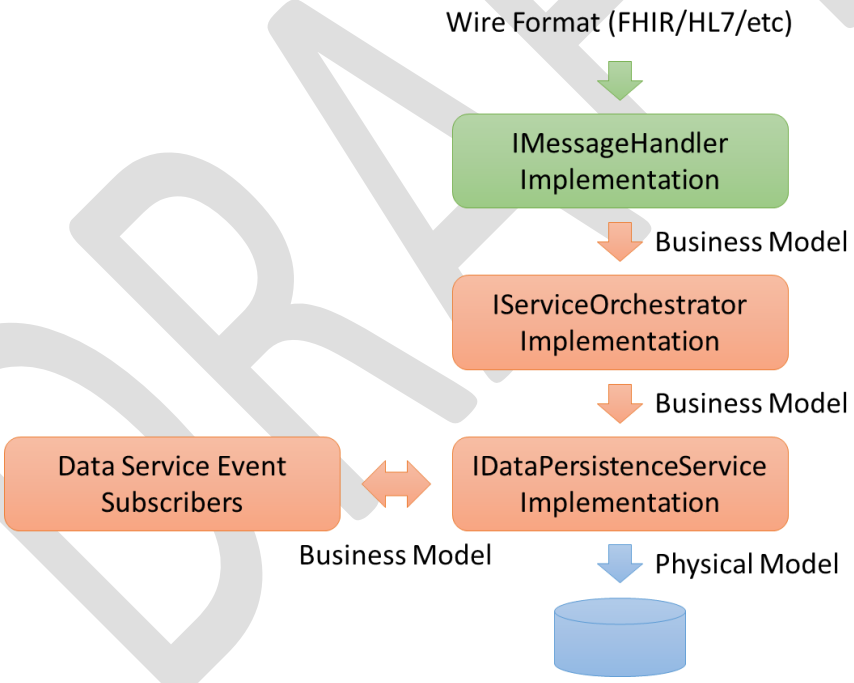


Figure 24 - Business / Physical Model Flow

The OpenIZ business model exposes a series of simpler structures than those of the data structures listed in the Data Model section of this document.

8.3.1. Microsoft SQL Server Data Model

Microsoft SQL Server is one of the two default physical database systems used to implement OpenIZ. The reason for choosing Microsoft SQL Server as the initial database storage medium is the first class support for LINQ and expression trees.

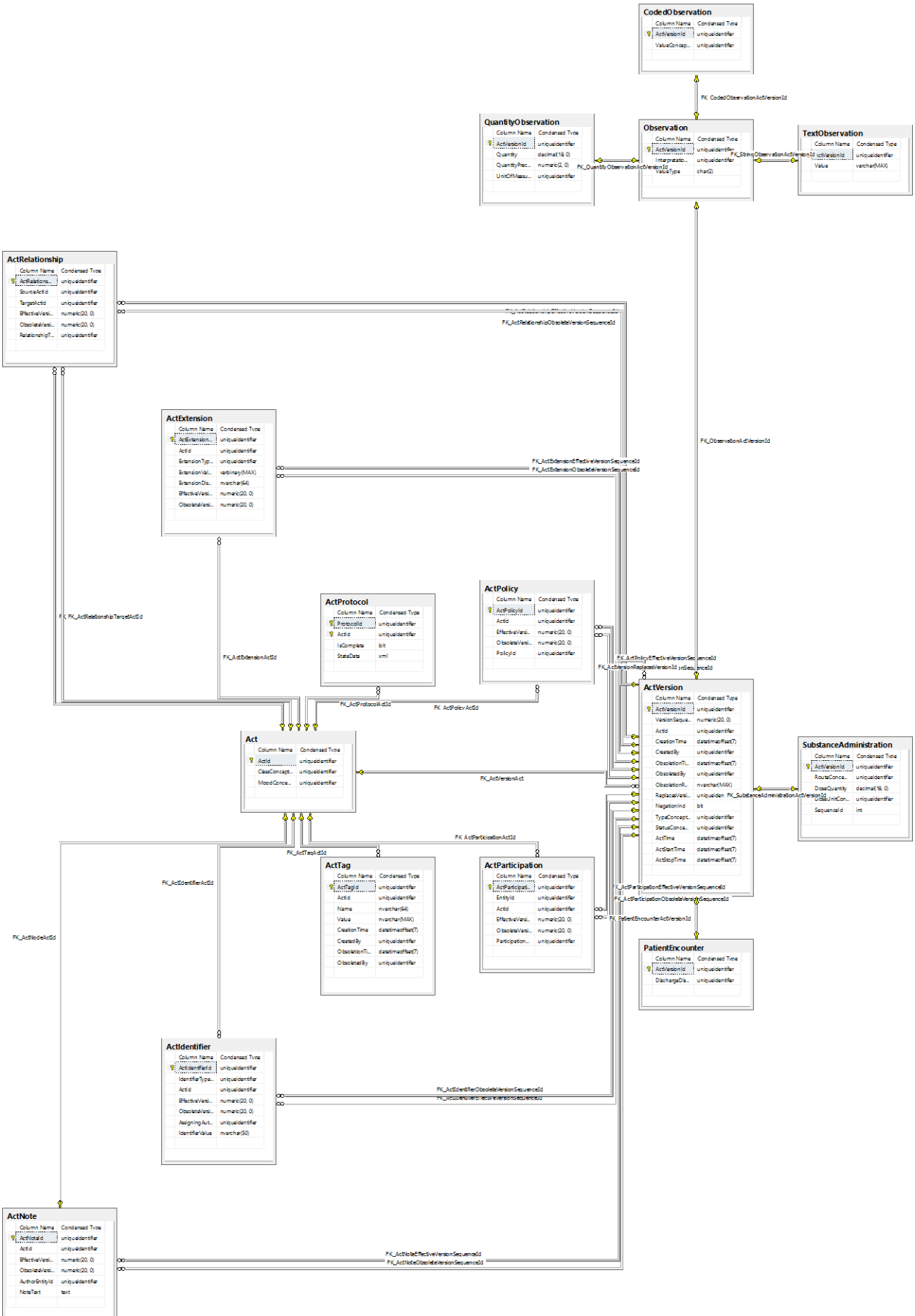
8.3.1.1. General Datatype Mapping

Illustrates the mapping between logical datatypes listed in section 8.2.

Logical Datatype	**Microsoft SQL Server Data Type**
BIT	BIT
VARBINARY	VARBINARY
DATE	DATE
DATETIME	DATETIMEOFFSET
UUID	UNIQUEIDENTIFIER
VARCHAR	NVARCHAR
CHAR	CHAR
INT	INT
FLOAT	FLOAT
XML	XML

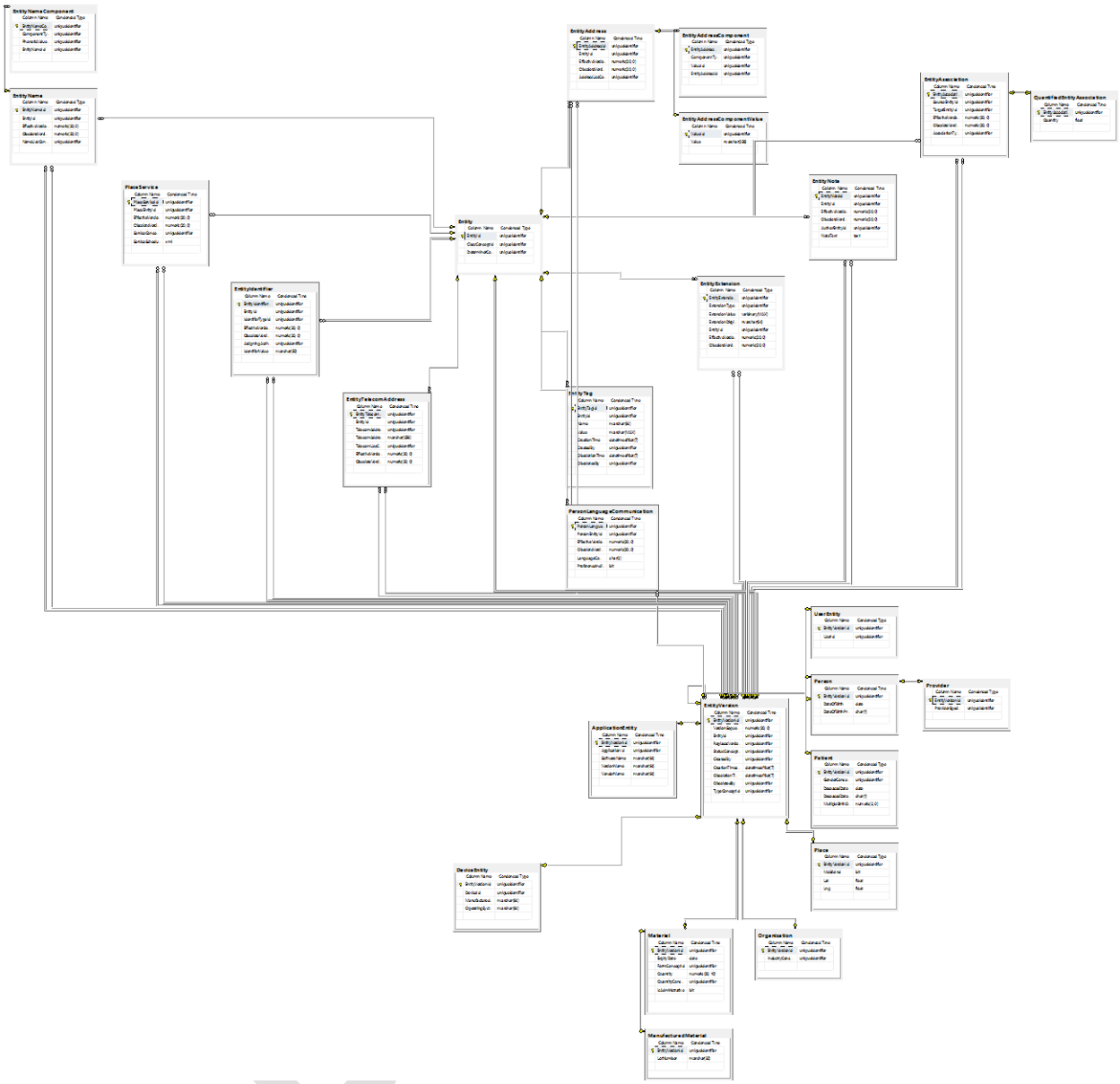
8.3.1.2. SQL Server Act Tables

Illustrates the relationships between the Act series of tables (described in section 8.2.4) as implemented in SQL Server.


CodedObservation
Column Name
Condensed Type
ActVersionId
uniqueidentifier
ValueConcep...
uniqueidentifier
FK_CodedObservationActVersionId
QuantityObservation
Column Name
Condensed Type
ActVersionId
uniqueidentifier
Quantity
decimal(18, 0)
QuantityPrec...
numeric(2, 0)
UnitOfMeasu...
uniqueidentifier
FK_QuantityObservationActVersionId
Observation
Column Name
Condensed Type
ActVersionId
uniqueidentifier
Interpretatio...
uniqueidentifier
ValueType
char(2)
FK_StringObservationActVersionId
TextObservation
Column Name
Condensed Type
ActVersionId
uniqueidentifier
Value
varchar(MAX)
ActRelationship
Column Name
Condensed Type
ActRelations...
uniqueidentifier
SourceActId
uniqueidentifier
TargetActId
uniqueidentifier
EffectiveVersi...
numeric(20, 0)
ObsoleteVersi...
numeric(20, 0)
RelationshipT...
uniqueidentifier
FK_ActRelationshipObsoleteVersionSequenceId
FK_ObservationActVersionId
ActExtension
Column Name
Condensed Type
ActExtension...
uniqueidentifier
ActId
uniqueidentifier
ExtensionTyp...
uniqueidentifier
ExtensionVal...
varbinary(MAX)
ExtensionDis...
nvarchar(64)
EffectiveVersi...
numeric(20, 0)
ObsoleteVersi...
numeric(20, 0)
FK_ActExtensionEffectiveVersionSequenceId
FK_ActExtensionObsoleteVersionSequenceId
FK_ActRelationshipTargetActId
FK_ActExtensionActId
ActProtocol
Column Name
Condensed Type
ProtocolId
uniqueidentifier
ActId
uniqueidentifier
IsComplete
bit
StateData
xml
ActPolicy
Column Name
Condensed Type
ActPolicyId
uniqueidentifier
ActId
uniqueidentifier
EffectiveVersi...
numeric(20, 0)
ObsoleteVersi...
numeric(20, 0)
PolicyId
uniqueidentifier
FK_ActPolicyEffectiveVersionSequenceId
FK_ActVersionReplacesVersionId
FK_ActProtocolActId
FK_ActPolicyActId
ActVersion
Column Name
Condensed Type
ActVersionId
uniqueidentifier
VersionSequ...
numeric(20, 0)
ActId
uniqueidentifier
CreationTime
datetimeoffset(7)
CreatedBy
uniqueidentifier
ObsolescenceTi...
datetimeoffset(7)
ObsoletedBy
uniqueidentifier
ObsolescenceR...
nvarchar(MAX)
ReplacesVersi...
uniqueidentifier
NegationInd
bit
TypeConcept...
uniqueidentifier
StatusConce...
uniqueidentifier
ActTime
datetimeoffset(7)
ActStartTime
datetimeoffset(7)
ActStopTime
datetimeoffset(7)
SubstanceAdministration
Column Name
Condensed Type
ActVersionId
uniqueidentifier
RouteConce...
uniqueidentifier
DoseQuantity
decimal(18, 0)
DoseUnitCon...
uniqueidentifier
SequenceId
int
FK_SubstanceAdministrationActVersionId
Act
Column Name
Condensed Type
ActId
uniqueidentifier
ClassConcept...
uniqueidentifier
MoodConce...
uniqueidentifier
FK_ActVersionAct
FK_ActTagActId
FK_ActParticipationActId
ActTag
Column Name
Condensed Type
ActTagId
uniqueidentifier
ActId
uniqueidentifier
Name
nvarchar(64)
Value
nvarchar(MAX)
CreationTime
datetimeoffset(7)
CreatedBy
uniqueidentifier
ObsolescenceTi...
datetimeoffset(7)
ObsoletedBy
uniqueidentifier
ActParticipation
Column Name
Condensed Type
ActParticipati...
uniqueidentifier
EntityId
uniqueidentifier
ActId
uniqueidentifier
EffectiveVersi...
numeric(20, 0)
ObsoleteVersi...
numeric(20, 0)
Participation...
uniqueidentifier
FK_ActParticipationEffectiveVersionSequenceId
FK_ActParticipationObsoleteVersionSequenceId
FK_PatientEncounterActVersionId
PatientEncounter
Column Name
Condensed Type
ActVersionId
uniqueidentifier
DischargeDis...
uniqueidentifier
FK_ActIdentifierActId
FK_ActNoteActId
ActIdentifier
Column Name
Condensed Type
ActIdentifierId
uniqueidentifier
IdentifierType...
uniqueidentifier
ActId
uniqueidentifier
EffectiveVersi...
numeric(20, 0)
ObsoleteVersi...
numeric(20, 0)
AssigningAut...
uniqueidentifier
IdentifierValue
nvarchar(50)
FK_ActIdentifierObsoleteVersionSequenceId
FK_ActIdentifierEffectiveVersionSequenceId
ActNote
Column Name
Condensed Type
ActNoteId
uniqueidentifier
ActId
uniqueidentifier
EffectiveVersi...
numeric(20, 0)
ObsoleteVersi...
numeric(20, 0)
AuthorEntityId
uniqueidentifier
NoteText
text
FK_ActNoteEffectiveVersionSequenceId
FK_ActNoteObsoleteVersionSequenceId

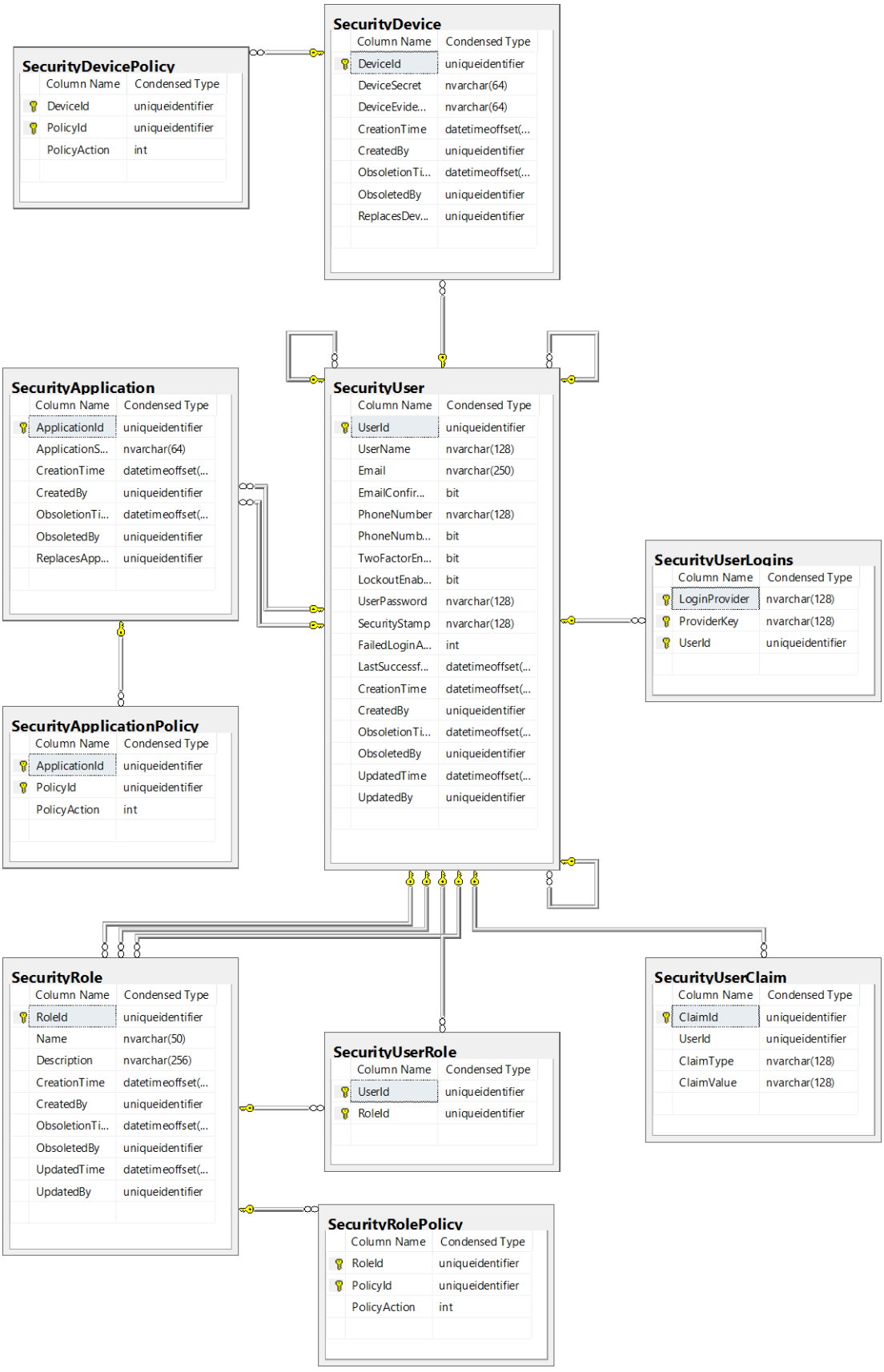
8.3.1.3. SQL Server Entity Tables

Illustrates the relationships between the entity tables in the SQL Server physical data model.



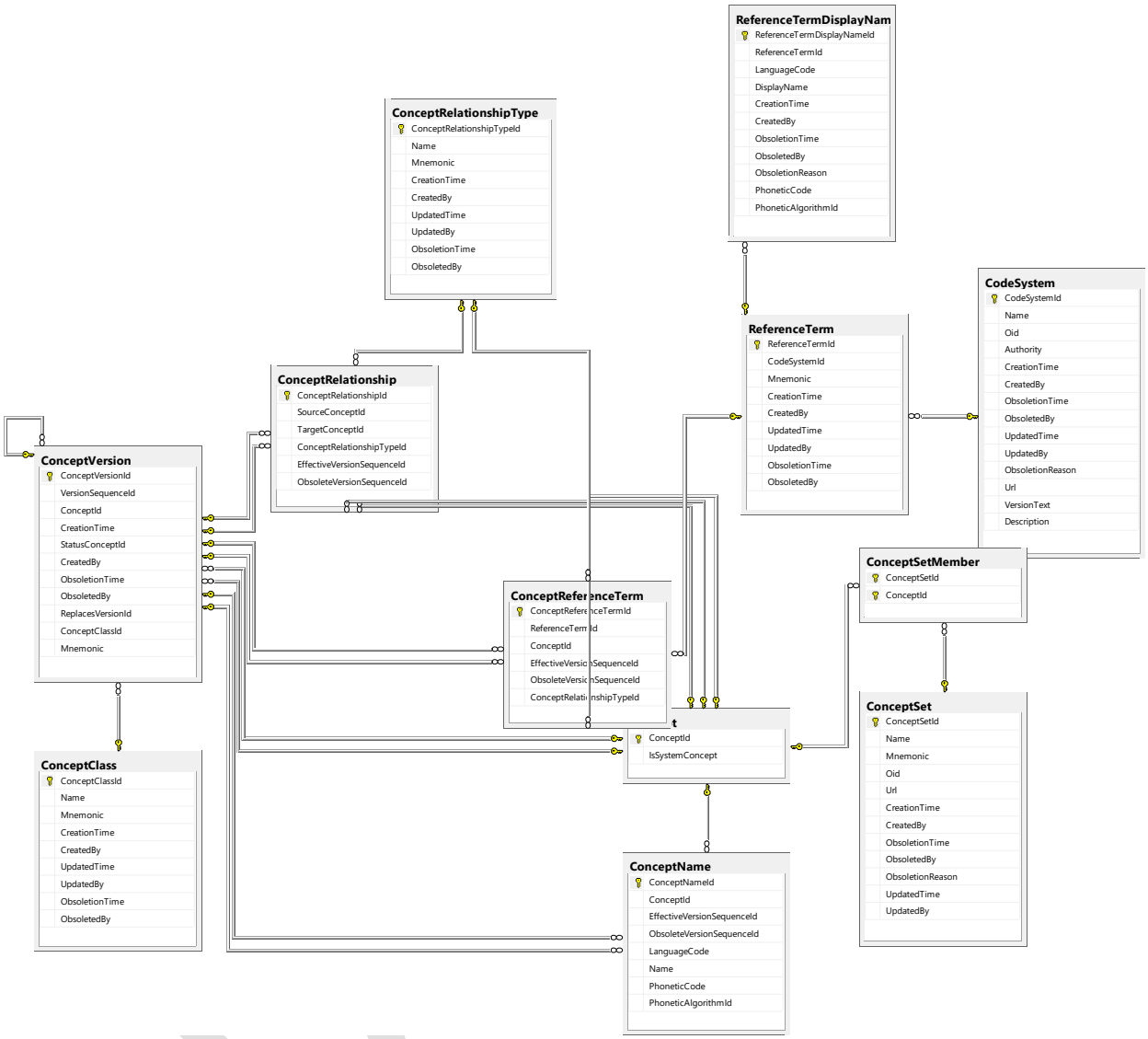
8.3.1.4. SQL Server Security Tables

Illustrates the relationship between tables in the Microsoft SQL Server physical data model between security tables.


SecurityDevice
Column Name Condensed Type
DeviceId uniqueidentifier
DeviceSecret nvarchar(64)
DeviceEvide... nvarchar(64)
CreationTime datetimeoffset(...
CreatedBy uniqueidentifier
ObsoletionTi... datetimeoffset(...
ObsoletedBy uniqueidentifier
ReplacesDev... uniqueidentifier
SecurityDevicePolicy
Column Name Condensed Type
DeviceId uniqueidentifier
PolicyId uniqueidentifier
PolicyAction int
SecurityApplication
Column Name Condensed Type
ApplicationId uniqueidentifier
ApplicationS... nvarchar(64)
CreationTime datetimeoffset(...
CreatedBy uniqueidentifier
ObsoletionTi... datetimeoffset(...
ObsoletedBy uniqueidentifier
ReplacesApp... uniqueidentifier
SecurityUser
Column Name Condensed Type
UserId uniqueidentifier
UserName nvarchar(128)
Email nvarchar(250)
EmailConfir... bit
PhoneNumber nvarchar(128)
PhoneNumb... bit
TwoFactorEn... bit
LockoutEnab... bit
UserPassword nvarchar(128)
SecurityStamp nvarchar(128)
FailedLoginA... int
LastSuccessf... datetimeoffset(...
CreationTime datetimeoffset(...
CreatedBy uniqueidentifier
ObsoletionTi... datetimeoffset(...
ObsoletedBy uniqueidentifier
UpdatedTime datetimeoffset(...
UpdatedBy uniqueidentifier
SecurityUserLogins
Column Name Condensed Type
LoginProvider nvarchar(128)
ProviderKey nvarchar(128)
UserId uniqueidentifier
SecurityApplicationPolicy
Column Name Condensed Type
ApplicationId uniqueidentifier
PolicyId uniqueidentifier
PolicyAction int
SecurityRole
Column Name Condensed Type
RoleId uniqueidentifier
Name nvarchar(50)
Description nvarchar(256)
CreationTime datetimeoffset(...
CreatedBy uniqueidentifier
ObsoletionTi... datetimeoffset(...
ObsoletedBy uniqueidentifier
UpdatedTime datetimeoffset(...
UpdatedBy uniqueidentifier
SecurityUserRole
Column Name Condensed Type
UserId uniqueidentifier
RoleId uniqueidentifier
SecurityUserClaim
Column Name Condensed Type
ClaimId uniqueidentifier
UserId uniqueidentifier
ClaimType nvarchar(128)
ClaimValue nvarchar(128)
SecurityRolePolicy
Column Name Condensed Type
RoleId uniqueidentifier
PolicyId uniqueidentifier
PolicyAction int

8.3.1.5. SQL Server Concept Tables

Illustrates the relationship between physical data model tables in Microsoft SQL Server.



8.3.2. PostgreSQL Data Model

8.4. Business Data Model

The business data model represents a series of classes (rather than database entities) which are used by all application code within the OpenIZ IMS. The business data model is also exposed via the IMSI (Immunization Management Service Interface).

Like the logical data model, the business data model is loosely based on the HL7 RIM. The major difference is that the business data model uses an object hierarchy rather than using relationships. The IDataPersistenceService implementations manage the translation of this paradigm.

8.4.1. Business Data Model Queries

Queries on against the business model are exposed via IDataPersistenceService implementations' Query() method. The Query() method in .NET accepts an expression tree parameter (lambda predicate) which is translated via the ModelMapper class.

In order to map the business model classes to LINQ to SQL classes, it is recommended that third party storage plugins use the ModelMapper class and provide an appropriate mapping file.

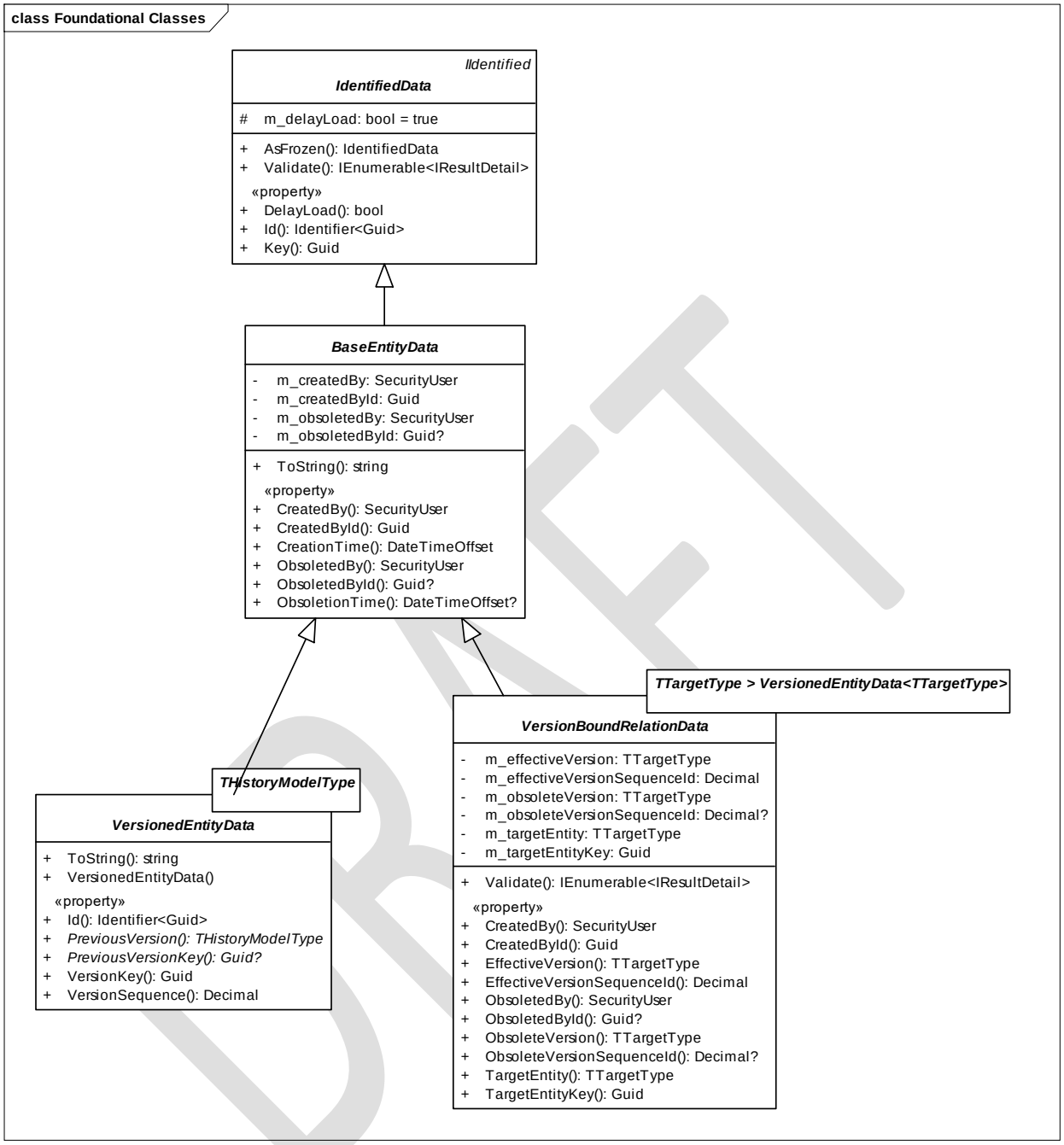
In order to execute a query, a developer should use the following pattern:

```
using (IDataPersistenceService<SecurityUser> persistenceService =
ApplicationContext.Current.GetService<IDataPersistenceService<SecurityUser>>())
{
    var queryResults = persistenceService.Query(u => u.Email.EndsWith("test.com") && u.Roles.Any(r =>
r.Name == "Administrators"), null);
```

Furthermore, additional LINQ expressions such as Count(), Any(), All(), Take(), Skip() can be used to control execution against the datamodel.

8.4.2. Foundational Classes

There are several foundational classes which are used in the business data model. Class associations in the business model are delay loaded meaning that they are loaded upon first access. For example, when loading a *SecurityUser* the *Groups* property is only loaded via the data persistence layer when accessed by a consumer application. This reduces hits to the underlying database.



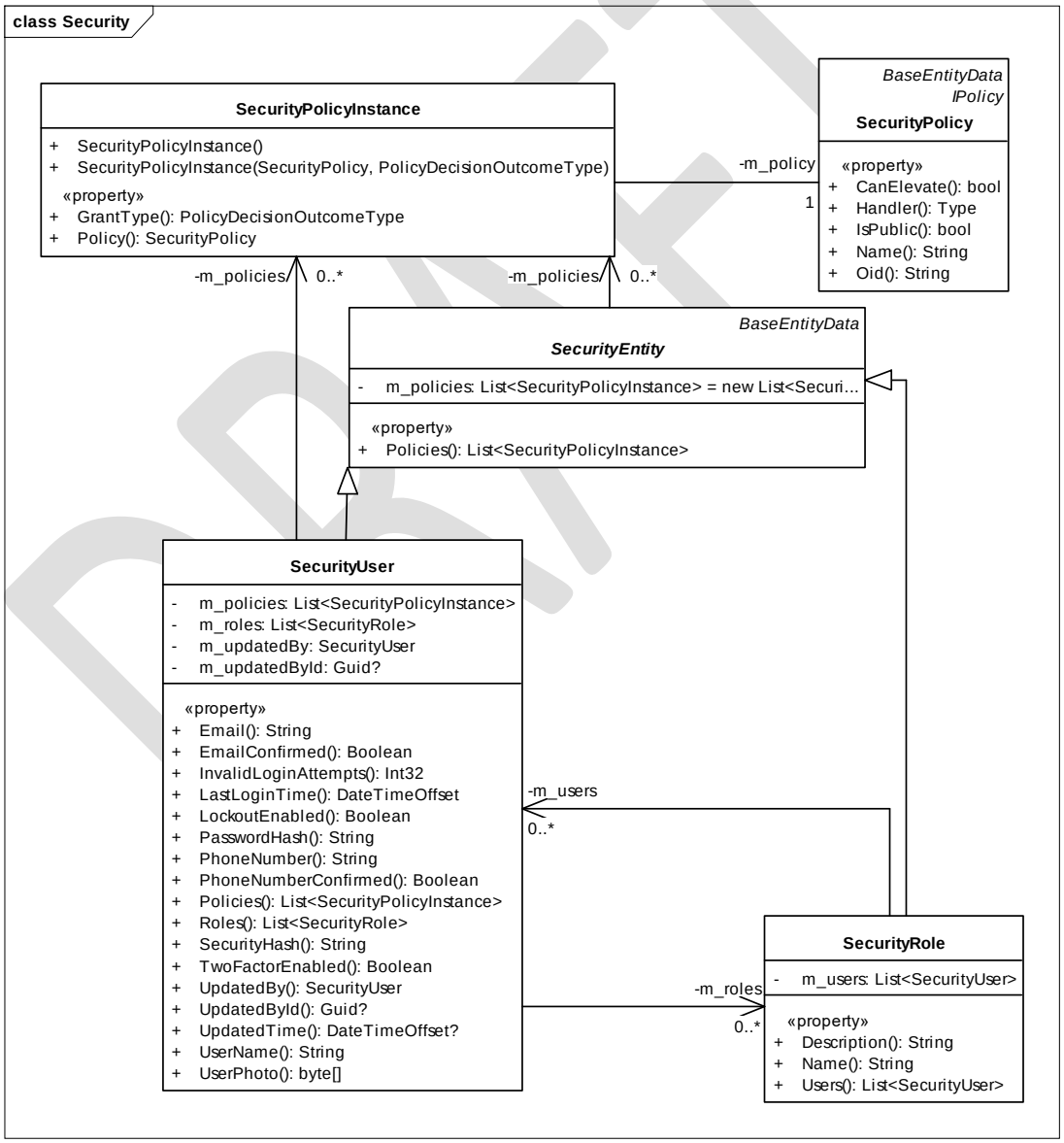
The foundational classes are listed in more detail in .

Class	Purpose
IdentitifedData (abstract)	Encapsulates all business model classes which have a primary key. Controls the delay loading behavior of implementing classes.
BaseEntityData (abstract)	Encapsulates entity data which is audited with a creation/obsoletion time and user.
VersionedEntityData<THistoryModelType> (abstract)	All entities which are versioned are derived from this class. The class contains a property of type

	THistoryModelType which is a pointer to the previous version of the entity instance.
VersionBoundRelationData<TTargetType> (abstract)	All entities which are associated with a version of an entity (those associated with the VersionedEntityData instance) will derive from this class. It contains information such as effective and obsolete version sequence numbers.

8.4.3. Security Classes

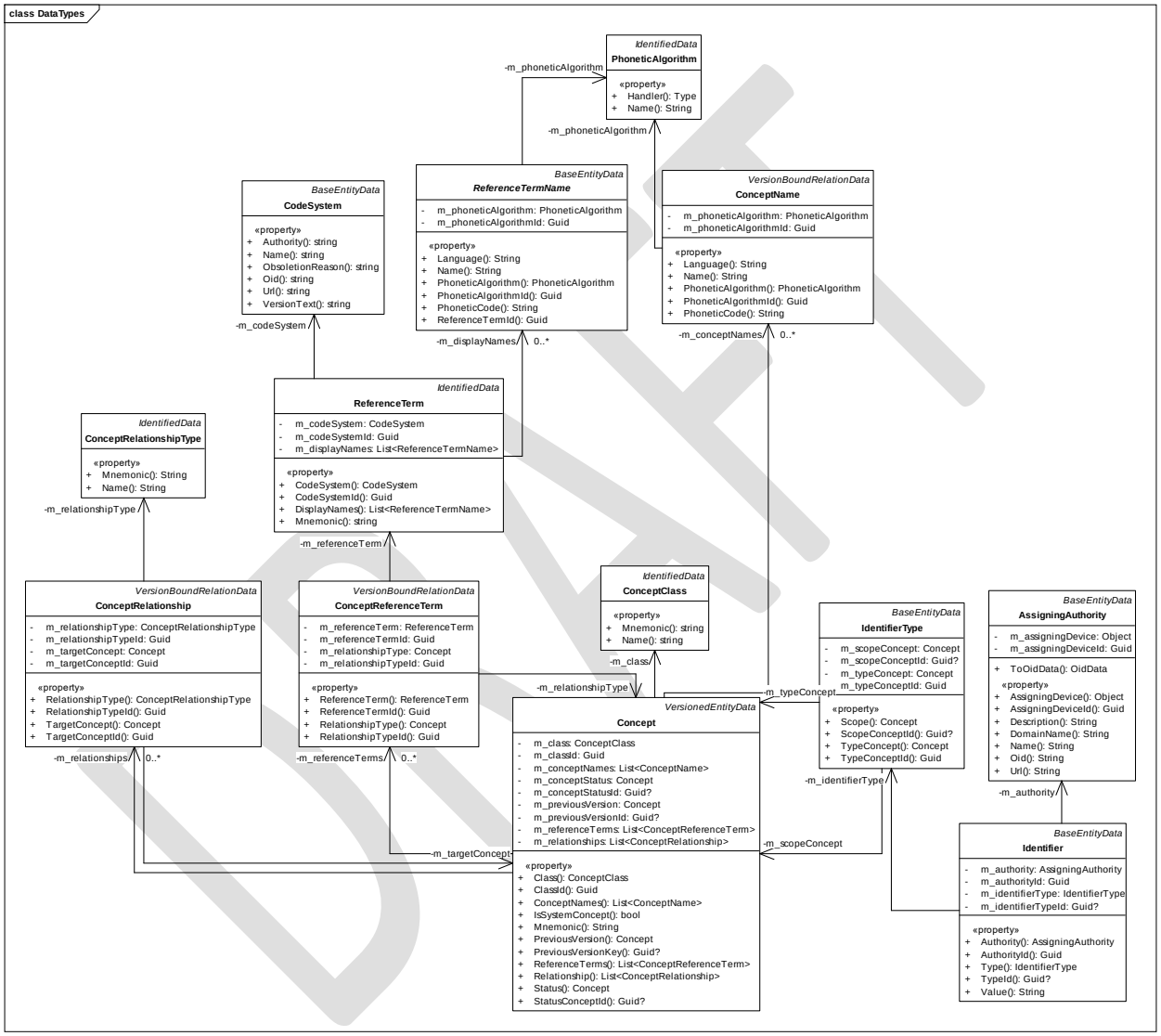
The security business model data classes are not necessarily intended to be used directly by consumers, rather they serve as a series of classes used by the security infrastructure in OpenIZ such as the IPolicyInformationService, IPolicyDecisionService, IIdentityProviderService and IRoleProviderService implementations.



The *SecurityUser* class is used to create *IIdentity* and *IPrincipal* during the course of an authentication request. *SecurityDevice* and *SecurityApplication* are used to populate claims data which indicates the device and application the user is using.

8.4.4. Data Types

The data type business model classes are used to store reusable, common data elements. These include: The concept system, Assigning Authority System, Identifiers, etc.



8.5. Pre-Configured Data Reference

OpenIZ's data model is expected to be populated with a minimum set of data which the core functionality will use. If a data persistence store does not have the required data elements described in this section, key functionality of the OpenIZ system may not function as expected.