OpenEMPI v 2.0

Software Architecture Document

SYSNET International, Inc.
2930 Oak Shadow Drive
Oak Hill, VA 20171
odysseas@sysnetint.com

Revision History

Revision	*Date*	*Name*	*Description*
1.00	1/14/2008	Odysseas Pentakalos	Created the initial version of the architecture template
1.01	1/16/2008	Odysseas Pentakalos	Added use cases
1.02	1/18/2008	Odysseas Pentakalos	Added component diagram
1.03	1/21/2008	Yimin Xie	Added the data model and discussion
1.04	1/27/2008	Yimin Xie	Updates to the data model following discussions
1.05	1/28/2008	Odysseas Pentakalos	Added class and sequence diagrams
1.06	2/2/2008	Yimin Xie	Updated the data model
1.07	2/6/2008	Odysseas Pentakalos	Updates to form first complete draft

Table of Contents

Table of Figures

1. Overview

1.1. Introduction

A Master Patient Index (MPI) is a repository of patient information for a health care provider or group of collaborating health care providers. An Enterprise Master Patient Index (EMPI) is an MPI that stores and indexes patient data across two or more MPIs although the terms EMPI and MPI are sometimes used interchangeably. The exact data elements maintained by a particular MPI vary but generally comprise all known patient information including identifiers from various domains, patient attributes as well as generated attributes, such as phonetic versions of names to support the matching of patients. The MPI serves a very crucial role in a health care organization, Health Information Exchange (HIE)[1] or Regional Health Information Organization (RHIO)[2] because it ensures that clinical and billing data is accurately associated with the correct patient even in the presence of data entry errors.

At a high-level requirements a MPI provides the following services:

- Allows for the accurate associations of persons being registered for care with their MPI record
- Minimizes the presence of duplicate records within an organization or enterprise by matching related records that are all linked to a distinct patient together
- Facilitate the integration of multiple MPIs across an organization or collaborating organizations to form an EMPI
- Facilitate access to longitudinal patient records

1.2. Background

Starting in 1998 the California Health Care Foundation (CHCF) worked with CareScience to develop the Santa Barbara Care Data Exchange (CDE), a robust health information exchange utility. The code was developed largely in the open source Java development language, and is owned by CHCF. In 2007 the code was reviewed and a portion of it, specifically the components that deal with patient record linkage, was refactored, stripped of proprietary elements, and replaced with new or existing open source components. In 2008 CHCF conveyed the source code to Open Health Tools (OHT), a fledgling open source organization, in efforts to leverage CHCF's previous investment and see the technology repurposed and reused. This release formed version 1.0 of OpenEMPI.

---

[1] HIE: “The electronic movement of health-related information among organizations according to nationally recognized standards".

[2] RHIO: "A health information organization [HIO] that brings together health care stakeholders within a defined geographic area and governs health information exchange [HIE] among them for the purpose of improving health and care in that community."

CHCF's goal is to accelerate adoption and implementation of health information exchanges, especially in resource-poor communities. While the initial code base has potential, it needs significant work and revision for it to be useful in any production environment. This architecture document describes the architecture of version 2.0 of OpenEMPI. This version strives to resolve some of the limitations of the 1.0 version including:

- One of the key capabilities of a Master Patient Index (MPI) service is the algorithmic approach used to match requests for patient information against the list of registered patients maintained by the service. The current OpenEMPI code only has support for a simple exact matching algorithm that is not very useful in a real-world implementation of the MPI service.
- Different healthcare domains maintain different sets of attributes about their patients which implies that in order for the OpenEMPI service to be applicable across a large community it needs to have support for a flexible data model that can be customized for a particular domain. The current OpenEMPI code has a fixed data model that is hard-coded into the current source, making it impossible to customize for different domains. Also, the current data model design imposes limitations on the scalability of more powerful matching algorithms.
- The quality of the current source code is rather low. It is very clear that many different developers have worked on the code over time without complying to a common set of coding standards. This makes the source code difficult to understand and imposes a steep learning curve to an open source community member that may be interest to contribute towards the growth of the project.

2. Requirements

2.1. Functional Requirements

This section describes the functional requirements of the OpenMPI server. The section begins by describing the high level requirements and then those requirements are decomposed and described at a more detailed level.

2.1.1. High Level Requirements

The following is a list of the high-level functional requirements for the OpenEMPI server.

1. The OpenEMPI server must be capable of receiving and processing patient identification information from external sources and inserting them into its local repository. The data model of the core server must be capable of capturing a wide variety of patient attributes. The set of attributes must be broad enough that will allow OpenEMPI to be deployed across a wide spectrum of health care providers and organizations. It must also support various external source formats. The supported formats are described in the section on "External Interface Requirements".
2. The OpenEMPI server must be capable of receiving update requests to patient information from external sources. After receiving and processing update requests, the server must apply the matching algorithm to determine whether any links between the modified patient and other patients may need to be adjusted. For example, in the situation where patients are linked together using a combination of first name, last name, and date of birth, a modification to a person's last name and date of birth will cause the patient to be removed from its current group.
3. The OpenEMPI server must be capable of receiving merge and unmerge requests between patients in the system. These requests typically occur when a patient identifier source determines that two patient records were either linked or not linked together incorrectly and a unmerge or merge request is submitted to the MPI server in order to correct the problem, respectively.
4. The data model of the OpenEMPI server must be flexible and extensible so that the software can be easily extended to support additional attributes beyond those supported by the baseline release. It is very likely that although the goal is to cover a broad array of patient attributes, that OpenEMPI may need to be deployed in specialized environments where additional attributes need to be supported.
5. The OpenEMPI server must support query requests for patients. The query requirements need to vary between queries that return all candidate patients based on query criteria specified regardless of links between patients, to queries that return individual patients based on unique identifiers, and to queries that return groups of patients that have been grouped together based on the matching algorithm.

6. The OpenEMPI server must be able of associating with each patient one or more identifiers assigned by distinct identifier domains.
7. The OpenEMPI server must be capable of supporting various matching algorithms for cross referencing patients to one another. It must provide at least one implementation of an exact matching algorithm (deterministic, all-or-none methods) that rely on each patient containing one or more attributes that are universally available, fixed, easily recorded, unique to each individual, and readily verifiable[GILL2001]. The server must also provide at lease one implementation of a probabilistic matching algorithm which is appropriate for cases where the patient data may include errors and omissions in attributes that are used for cross-referencing patients to each other[GILL2001]. The matching algorithm must be pluggable which implies that during installation, a site may select the matching algorithm using configuration rather than development. The behavior of the matching algorithm must also be configurable using parameters provided by each matching algorithm implementation.
8. The OpenEMPI server must be secure and require authentication of all users and support authorization to system capabilities. A given group of users only has access to query the system for person information, another group of users only has access to add person data into the system, and a third group has access to the complete functionality of the system.

2.1.2. Use Cases

This section lists the use cases that summarize the requirements for OpenEMPI and drive the architecture and design of the system. Each use case is documented using the following consistent format and set of attributes[COCA2001].

- A UML Use Case diagram
- Primary Actor: the stakeholder who or which initiates in the behavior of the system
- Scope: Identifies the portion of the system that is considered or participates in the use case
- Level: How high- or low-level is the goal of the use case
- Preconditions and guarantees: what must be true before and after the use case runs
- Description: a brief description of what the use case is all about and of its goal.
- Main success scenario: a case in which nothing goes wrong and the interaction accomplishes the goal of the use case
- Extensions: what can happen differently during the use case

UC.1 Add Person to the Repository

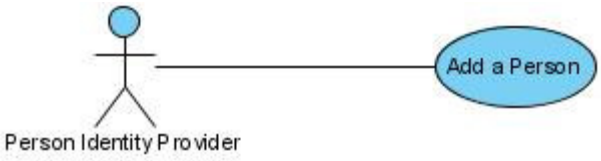

Figure 1 Add a Person to the Repository Use Case

Primary Actor: Person Identifier Provider. A system that provides person data to the system.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The person identity, demographic and other attributes are added to the repository of the OpenEMPI system

Description: A Person Identifier Provider submits a person's data for insertion into the OpenEMPI repository. The Person Identifier Provider is any entity that participates in the federation of organizations whose person's identities are managed by a single EMPI instance. A patient may be associated with multiple identifiers in a single request to support the case where a single Person Identifier Provider is supporting multiple identity domains (this is an Enterprise MPI so it needs to support the case where the Person Identifier Provider is an MPI server).

Main Success Scenario:

1. A Person Identifier Provider submits a person's data for insertion into the OpenEMPI repository.
2. The system validates the attributes provided as part of the request to ensure that the minimally acceptable dataset is present and valid. If the request is invalid, an error is reported to the actor and processing of the request terminates.
3. The person's identifier is used to determine if the person already exists in the system. No such match is found in the system.
4. The person's attributes are used by the matching algorithm to determine if the person already exists in the system (in exact or similar form based on the configured matching algorithm). No matches are found in the system.
5. The person's attributes are stored in the repository.

Extensions:

Person with same identifier in the system

3a. The person's identifier is used to determine if the person already exists in the system and a match is found. Since the person is already known to the system, processing of the request terminates.

Matching person in the system

4a. The person's attributes are used by the matching algorithm to determine if the person already exists in the system and one or more matches are found.

5a. The persons attributes are stored in the system and the patient's record is linked to the other matching patients.

UC.2 Update a Person's Attributes

Figure 2 Update a Person's Attributes Use Case

Primary Actor: Person Identifier Provider. A system that provides person data to the system.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The person identity, demographic and other attributes are updated in the repository of the OpenEMPI system

Description: A Person Identifier Provider submits an update to a known person's attributes. This is usually the result of correcting a typographic or data entry error done when the person was first added or as the result of a follow-up visit by the person, indicating that some of their attributes have changed (person got married, moved to a different location, etc.)

Main Success Scenario:

6. A Person Identifier Provider submits an update to a known person's attributes.
7. The system validates the attributes provided as part of the request to ensure that the minimally acceptable dataset is present and valid. If the request is invalid, an error is reported to the actor and processing of the request terminates.
8. The person's identifier is used to determine if the person already exists in the system. If no match is found, processing of the request terminates.
9. The person's attributes are updated according to the request and the information is stored in the repository.
10. The matching algorithm is used to identify any other patients that are linked with the updated patient and the patient's links are updated accordingly

UC.3 Merge a Person's Identities

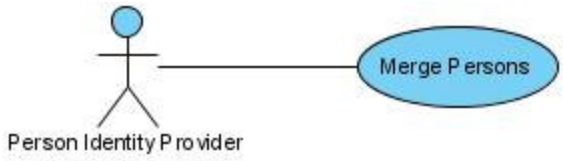

Figure 3 Merge two or more Person's Identities Use Case

Primary Actor: Person Identifier Provider. A system that provides person data to the system.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The identifiers of two or more persons are retired and a single person identified by the surviving identifier in the request is preserved as active in the system.

Description: Person Identifier Provider submits one or more identifiers that were in the past used to identify distinct patients and a surviving patient identifier and requests that the identifiers are merged into the one surviving patient identifier. The following scenario is an example of how this may occur. A patient name Mary Washington visits a provider for services and registers, obtaining an identifier at that provider. Mary gets married taking on the name Lincoln and moving to a new home. Mary then attends the same provider for services, registering again and obtaining a new identifier. Someone recognizes that Mary Washington and Mary Lincoln refer to the same person and a merge request is submitted to preserve the identity associated with Mary Lincoln and to retire the identifier associated with Mary Washington.

Main Success Scenario:

1. A Person Identifier Provider submits one or more identifiers that were in the past used to identify distinct patients and a single surviving patient identifier in a merge request.
2. The system validates the attributes provided as part of the request to ensure that the minimally acceptable dataset is present and valid. If the request is invalid, an error is reported to the actor and processing of the request terminates.
3. Search the repository for a patient with an exact match on the surviving identifier. If not found, then return an appropriate error message to the actor and terminate processing of the request.
4. Search the repository for an exact match of the identifiers that will be retired. If none are found then return an appropriate error message to the actor and terminate processing of the request.
5. Deactivate the identifiers that were retired. If any of the retired records were linked to other persons, evaluate a match with the surviving record and if it persists linked them together.

UC.4 Delete a Person

Figure 4 Delete a Person Use Case

Primary Actor: Person Identifier Provider. A system that provides person data to the system.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The person identified by the request is deleted by the system

Description: Person Identifier Provider submits a request to delete a person from the system. The person must be identified by a unique identifier. The record is found and is deactivated so that it is no longer included in subsequent query requests.

Main Success Scenario:

6. A Person Identifier Provider submits a request to have a person deleted by the system
7. The system validates the attributes provided as part of the request to ensure that the minimally acceptable dataset is present and valid (the only required attribute in this case is a unique patient identifier). If the request is invalid, an error is reported to the actor and processing of the request terminates.
8. The repository is searched for an exact match on the patient identifier. If no match is found an appropriate error message is returned to the actor
9. The person entry in the repository is deactivated.

UC.5 Query Persons by Identifier

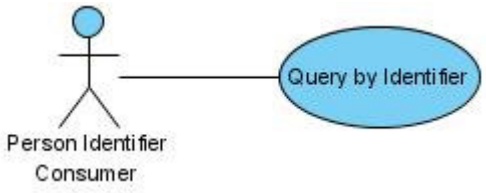

Figure 5 Query Persons by Identifier Use Case

Primary Actor: Person Identifier Consumer. A system that needs to acquire patient information from the EMPI.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The system will return zero or more persons that match the query criteria.

Description: Person Identifier Provider submits a query request to retrieve persons from the system that match the query criteria. The query criteria includes one or more partially or fully specified person identifiers.

Main Success Scenario:

1. A Person Identifier Consumer submits a query request consisting of one or more person identifiers
2. The system validates the attributes provided as part of the request to ensure that the minimally acceptable dataset is present and valid
3. The repository is searched for person's that match the specified query criteria. The matching algorithm is not applied for this query and the repository is searched for an exact match on the components of each identifier specified
4. The collection of matching patients is returned to the consumer.

UC.6 Query Persons by Attributes

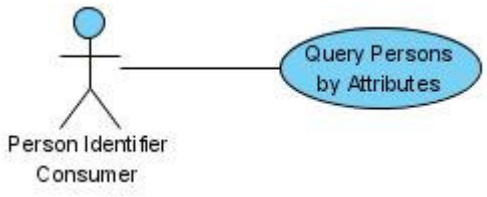

Figure 6 Query Persons by Attributes Use Case

Primary Actor: Person Identifier Consumer. A system that needs to acquire patient information from the EMPI.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The system will return zero or more persons that match the query criteria.

Description: Person Identifier Provider submits a query request to retrieve persons from the system that match the query criteria. The query criteria include a subset of the attributes maintained on behalf of each person by the EMPI.

Main Success Scenario:

1. A Person Identifier Consumer submits a query request consisting of one or more person attributes
2. The system validates the attributes provided as part of the request to ensure that they are valid
3. The repository is searched for person's that match the specified person attribute query criteria. The matching algorithm is not applied for this query and the repository is searched for an exact match on the attributes specified
4. The collection of matching patients is returned to the consumer.

UC.7 Query Person Identifiers by Identifier

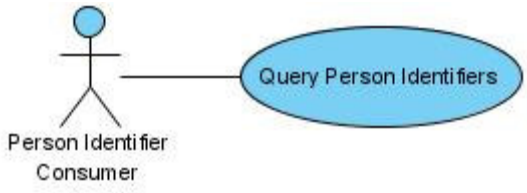

Figure 7 Query Person Identifiers Use Case

Primary Actor: Person Identifier Consumer. A system that needs to acquire patient information from the EMPI.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The system will return zero or more persons that match the query criteria.

Description: Person Identifier Provider submits a query request to retrieve persons from the system that match the query criteria. The query criteria include a subset of the attributes maintained on behalf of each person by the EMPI.

Main Success Scenario:

1. A Person Identifier Consumer submits a query request consisting of a person identifier
2. The system validates the attributes provided as part of the request to ensure that the minimal subset is provided and that they are valid
3. The repository is searched for a person that has an exact match on the identifier provided as part of the query criteria
4. The identifiers of any linked patients are retrieved from the repository and returned to the consumer

Extension:

Consumer supplies a subset of identifier domains that will be used to filter the identifiers returned

4a. The identifiers of any linked patients are retrieved from the repository.

5a. The list of associated identifiers is filtered to only those identifiers assigned by the assigning authority domains supplied by the consumer. The filtered list of identifiers is returned to the consumer

UC.8 Notification Registration

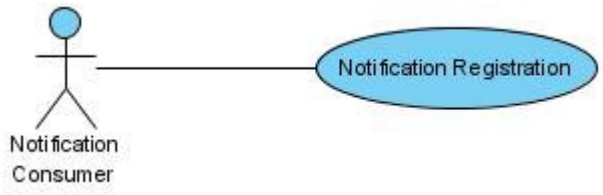

Figure 8 Notification Registration Use Case

Primary Actor: Notification Consumer. A system that needs to be informed when changes take place to persons with identifiers assigned by domains managed by the consumer

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The system register the interest of the particular consumer to receive notifications when patient information changes

Description: Person Identifier Provider submits a query request to retrieve persons from the system that match the query criteria. The query criteria include a subset of the attributes maintained on behalf of each person by the EMPI.

Main Success Scenario:

1. A Notification Consumer submits a request to be registered to receive notifications about person changes from specific domains
2. The notification request is validated to ensure that the consumer has the privilege to be receive notifications on the requested assigning authority domains
3. The registration request is recorded

UC.9 Notification Distribution

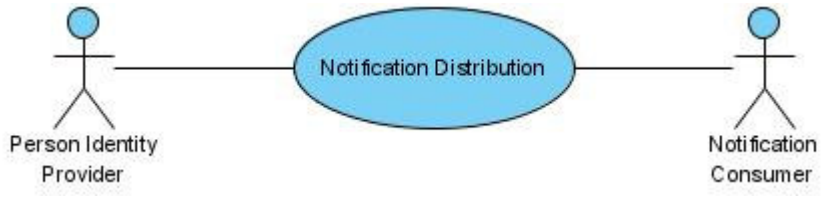

Figure 9 Notification Distribution Use Case

Primary Actors: Notification Consumer. A system that needs to be informed when changes take place to persons with identifiers assigned by domains managed by the consumer

Actor: Person Identity Provider

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The system sends out notifications to every registered notification consumer when an even of interest takes place.

Description: Person Identifier Provider generates a request that results in changes to one or more person records (add new person, update a person's record, delete a person's record, etc.). The system sends out notifications to every registered notification consumer to inform them of the occurrence of the event

Main Success Scenario:

1. A Person Identifier Provider adds a new person to the system. The request is processed according to the appropriate steps for the use case corresponding to the event
2. The assigning authority domains of all person records affected by the event are identified and notification consumers that have registered an interest to received notifications for those particular assigning authority domains are sent notifications

UC.10 Audit Events

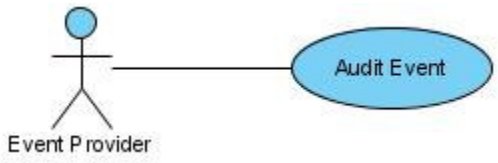

Figure 10 Audit Event Use Case

Primary Actors: Event Provider. Any agent that generates a request that causes an auditable event to be generated. ***A list of auditable events needs to be determined.***

Actor: Event Provider

Scope: Application Programming Interface

Level: System requirement

Guarantees: When an operation that needs to be audited by the system is completed successfully, an audit event is generated and persisted by the system

Description: Various events generated by the system must be audited within an EMPI. For example, the addition of a new patient, the update of an existing patient, or the merging of two patients are all auditable events. When such an event occurs, the system must transactionally persist the audit event along with any other information that is persisted as part of the request.

Main Success Scenario:

1. An Event Provider generates an auditable event.
2. The system persists the event in the repository

UC.11 Query Audit Events

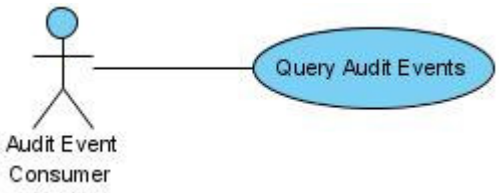

Figure 11 Query Audit Event Use Case

Primary Actor: Audit Event Consumer. A consumer that needs to acquire audit events from the system.

Scope: Application Programming Interface

Level: End-user requirement

Guarantees: The system will return zero or more audit events that match the query criteria.

Description: Audit Event Consumer submits a query request to audit events from the system that match the query criteria. The query criteria include a date range, event type, and person identifier.

Main Success Scenario:

1. An Audit Event Consumer submits a query request of audit events specifying one or more query criteria.
2. The system validates the attributes provided as part of the query request to ensure that the minimal subset is provided and that they are valid
3. The audit event repository is searched for matching audit events
4. The collection of matching audit events is returned to the consumer

2.1.3. Other Requirements

OpenEMPI 2.0 must support all the implied requirements of a IHE PIX/PDQ server on the EMPI, which is referred to as the Patient Identifier Cross Reference Manager in the IHE IT Infrastructure Technical Reference[IHEITI1],[IHEITI2].

2.2. External Interface Requirements

Initially OpenEMPI exposes its interface in the form of Java J2EE beans. During deployment, the user will also have the option of using OpenEMPI through the Spring interface. Subsequent enhancements that planned for the future include exposing Web Services interfaces, in the form of both SOAP- and REST-based interfaces.

2.3. Non-functional Requirements

2.3.1. Performance Requirements

This section identifies and quantifies the performance requirements for OpenEMPI. These requirements will be established after surveying existing MPI environments and collecting information as to what specific values are acceptable and desirable by the users.

The following performance metrics will be documented here:

- Define the typical workload that is imposed on an EMPI server.
- Define the maximum satisfactory response time to be experienced by the users most of the time for each of the critical workload components
- Define the desired throughput required for each of the critical workload components

- Assumptions made about the hardware that OpenEMPI will be deployed on and expectations of the capabilities of that hardware

2.3.2. Security Requirements

This section identifies the security requirements for OpenEMPI.

- User Data Protection: OpenEMPI will capture and manage sensitive person data attributes. It is critical that all person data managed by the system both flows through the system and is persisted in a secure way.
- Identification and authentication: Before a user can utilize the functionality available by the OpenEMPI, they need to verify their identity, in a process called authentication. Successful authentication will result in a security token passed back to the user which they must use in subsequent requests against OpenEMPI. The security token must have an expiration time.
- Security audit: All security-relevant activities as well as operations against sensitive person attributes must be recorded, linking the operation to the actor that requested it.
- Role-based access control: Users will be assigned roles during the registration process and those roles will determine what operations they are permitted to perform against the system. Every request against the system will be checked against the access control rules and denied if the caller does not have the appropriate role.

2.3.3. Data Elements

The data model for OpenEMPI 2.0 needs to support at the very least, all the attributes that are required by the PIX/PDQ integration profiles of the IHE IT Infrastructure Technical Framework[IHEITI1], [IHEITI2]. According to the IHE IT Infrastructure Technical Framework specification:

> "*the Patient Identifier Cross-reference Manager (or EMPI) shall be capable of accepting attributes in the PID segment as specified in Table 3.8-2. This is to ensure that the Patient Identifier Cross-reference Manager can handle a sufficient set of corroborating information in order to perform its cross-referencing function*"

OpenEMPI needs to be able to form the core of a PIX/PDQ server and as such must at least provide support for the data elements listed below, which may be present in an HL7 PID segment.

Table 1 List of Patient Attributes Required by PIX/PDQ Specification

Seq	Field Name
1	**Patient Identifier(s)**
2	**Patient Name**

3	Mother's Maiden Name
4	Date/Time of Birth
5	Administrative Sex
6	Patient Alias
7	Race
8	Patient Address
9	County Code
10	Phone Number
11	Primary Language
12	Marital Status
13	Religion
14	Patient Account Number
15	Patient SSN Number
16	Patient's Driver's License Number
17	Mother's Identifier
18	Ethnic Group
19	Birth Place
20	Multiple Birth Indicator
21	Birth Order
22	Citizenship
23	Veteran Military Status
24	Nationality
25	Patient Death Date and Time
26	Patient Death Indicator

3. Functional Viewpoint

3.1. Component Diagram

This section presents the decomposition of the system architecture into components. The component diagram shown in Figure 12 describes the high-level pieces that comprise the architecture of OpenEMPI along with the dependencies between the components and the services exposed to the clients of the system.

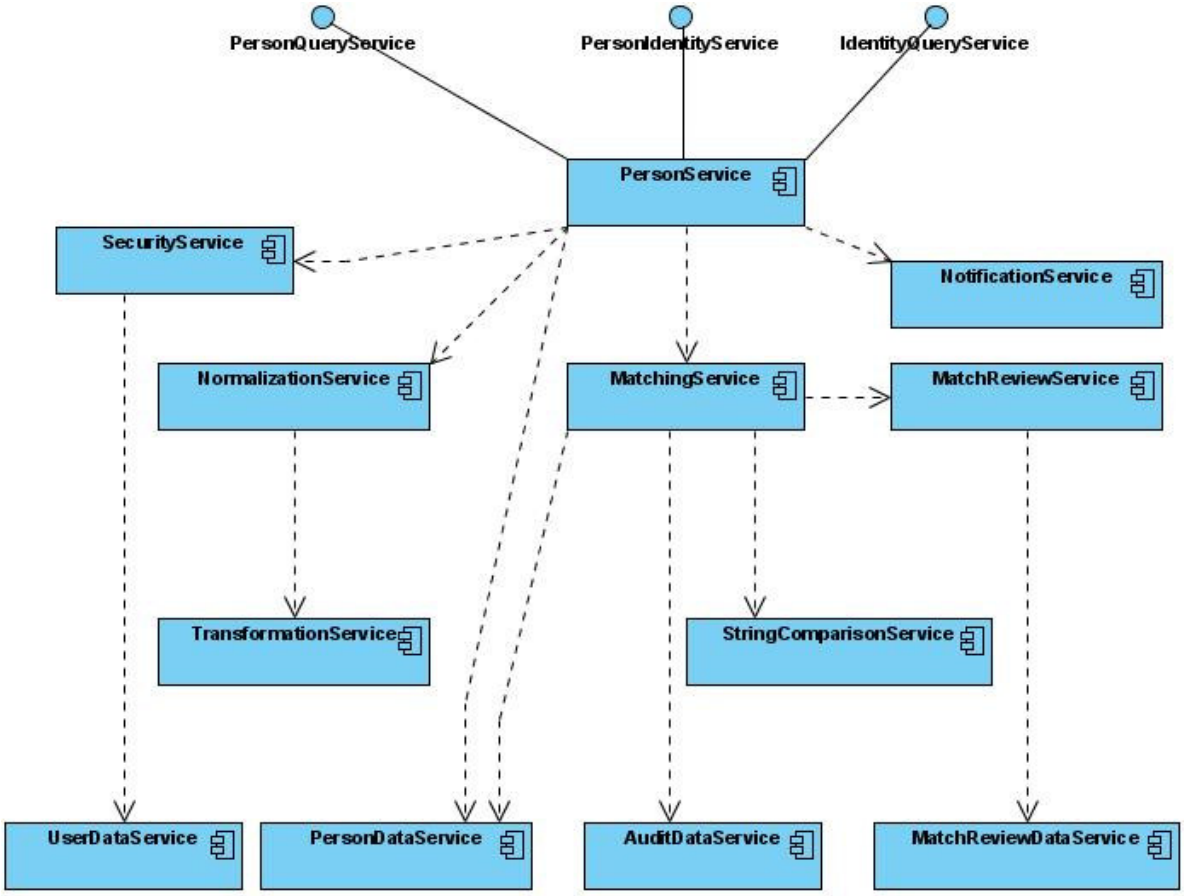

Figure 12 Component Diagram

Each component of the system illustrated in the component diagram provides a service that is needed by the system. The principle of loose coupling is followed in the architecture of the OpenEMPI to reduce the complexity of the overall architecture by decomposing functionality across components, reduce unnecessary coupling between components by reducing interaction between components only through well-defined interfaces implemented by each component, and allowing for the flexible evolution of the system through the eventual development of enhanced versions of each component.

Person Service: The Person Service component is the only component that exposes interfaces to external clients. The functionality of the Person Service is exposed through three separate interfaces that form the Application Programming Interface (API) of OpenEMPI. The PersonQueryService interface exposes methods that allow the users to query the system for person information, the IdentityQueryService exposes methods for querying the system for person identifier information, and the PersonIdentityService exposes methods for manipulating person information.

SecurityService: The Security Service handles the authorization, authentication, and access control requirements of the system. Before an client application uses the services of the OpenEMPI they need to authenticate themselves and acquire a session identifier that they need to provide in every request to be authorized for access to various functions provided by the system. Security services are cross-cutting concerns that will be provided to the individual components using Aspect Oriented Programming techniques in order to eliminate duplicating code across the entire application.

MatchingService: The Matching Service component encapsulates the matching functionality that cross references the person records entered into the system to one another. There are determinist (exact) matching algorithms, that determine whether two records refer to the same person or not, as well as probabilistic matching algorithm that determine the probability that two records refer to the same person and depending on the value of that probability, two records are either linked or assumed to refer to two distinct persons. This component abstracts the matching algorithms and makes them available to the rest of the components through a well defined interface[GILL1997].

MatchReviewService: In some cases, certain matching algorithms cannot make a conclusive decision as to whether two records are linked or not and the decision is deferred to human review. The Match Review Service provides access to match evaluations that were deferred to human review.

NotificationService: The notification service provides support for informing an assigning authority domain that information about a person that has an identifier assigned by it have changed as a result of a request. Entities that have the privileges can register to received notifications regarding changes to persons from a particular assigning authority domain or any assigning authority domain. During the registration, the registrant can also express interest in notification about specific operations.

NormalizationService: Names and addresses are often captured in non-standard and varying formats, usually with some degree of spelling and typographical errors. The precision of the person matching process is dependent on some standardization or preprocessing of the attributes that describe a person[CHRI2002],[RAHM]. This service provides a comprehensive set of normalization routines that can be configured at a site level to ensure that the attributes for all persons in the repository are stored in a consistent format. All standardization capabilities are encapsulated and exposed by this service.

TransformationService: This service encapsulates a comprehensive set of transformation routines for various data types such as strings and dates. The capabilities of this service are utilized by the NormalizationService as well as the MatchingService as needed.

StringComparisonService: Matching algorithm calculate how closely two person records match one another based on comparisons of individual person record attributes. Because pairs of strings often exhibit typographical errors, matching algorithms make use of approximate string comparators that deal with typographical variations[PORT97][YANC]. This component encapsulates all string comparator algorithms including the Jaro, Jaro-Winkler, and bigram algorithms.

Data Services Layer: The Data Services layer provides the abstract interface through which the rest of the components access data stored in the repository and other relevant sources. Layering is a very well known concept in software architecture that provides many benefits including loose coupling, extensibility, and performance. The **UserDataService** support access and persistence for user information, the **PersonDataService** for person data, the **AuditDataService** for audit data, and the **MatchReviewDataService** for reviewing inconclusive matches.

3.2. Class Diagrams

This section includes the high level class diagrams for the key components of the OpenEMPI. The class diagrams are broken down by component package.

3.2.1. Model Package

The model package include the main data objects shared by the application.

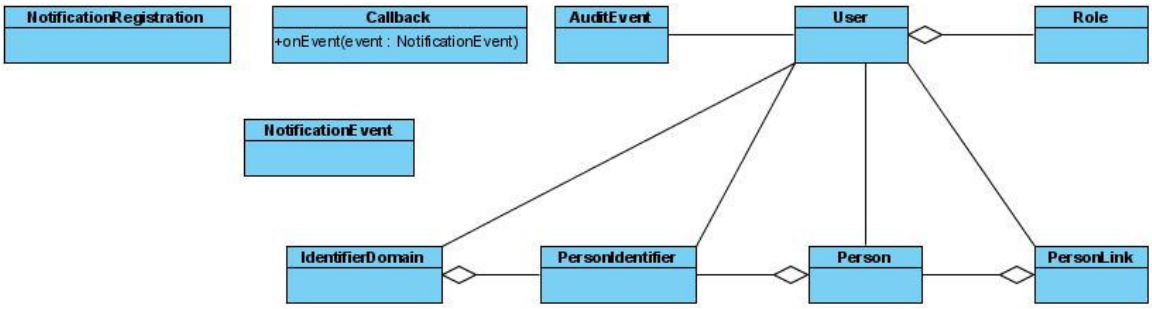

Figure 13 Model Class Diagram

3.2.2. Person Package

The Person package includes all the classes that requests from users.

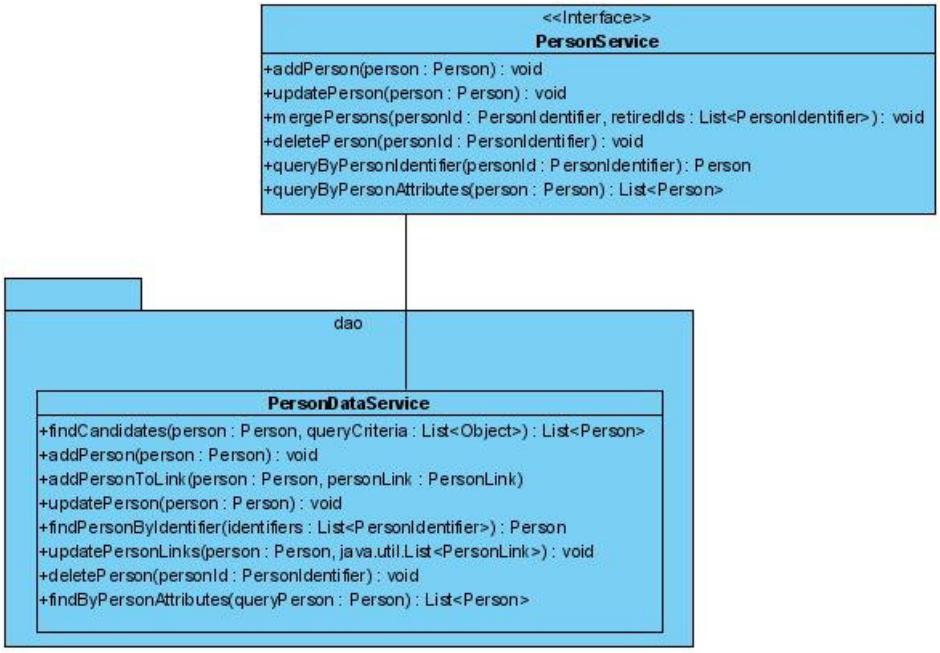

Figure 14 Person Package Class Diagram

3.2.3. Security Package

The security package includes all the classes that relate to the security requirements of OpenEMPI.

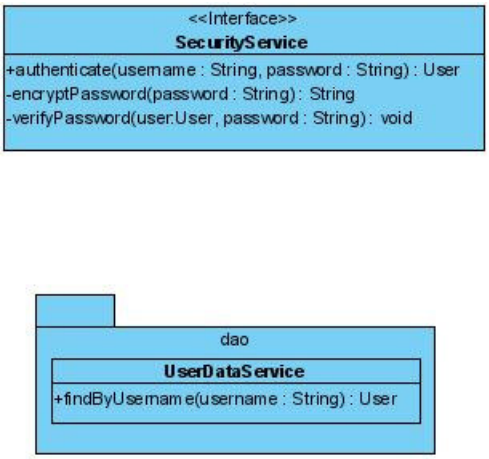

Figure 15 Security Class Package

3.2.4. Matching Package

The matching package includes the classes that encapsulate the person matching algorithms. Individual implementation of different matching algorithms will use sub-packages of the Matching package.

Figure 16 Matching Class Package

3.2.5. Notification Package

The notification package includes classes that manage the registration of callbacks and notification distribution.

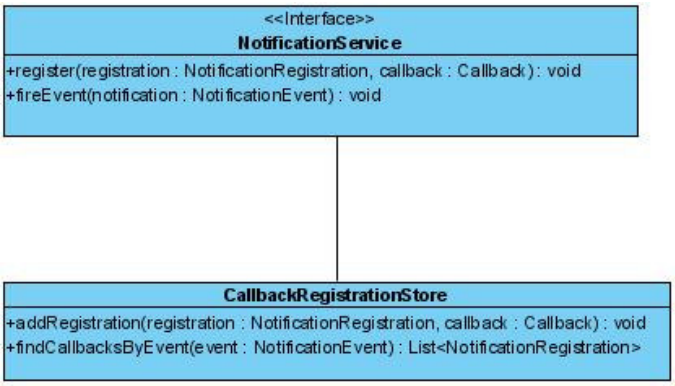

Figure 17 Notification Class Diagram

3.2.6. Audit Package

The audit package includes classes that relate to auditing of events.

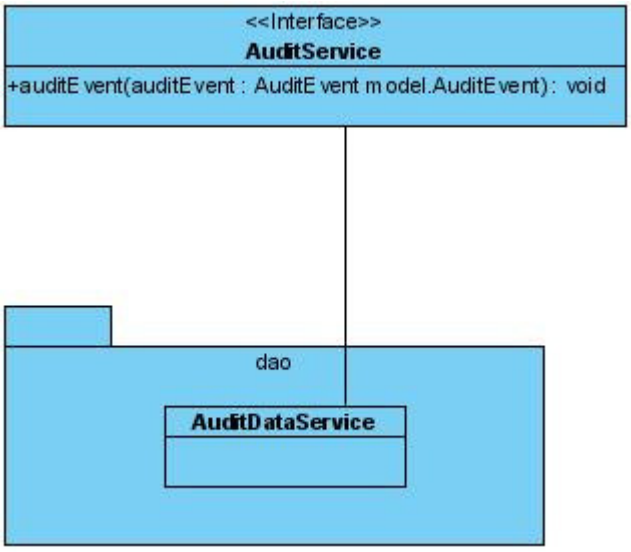

Figure 18 Audit Class Diagram

3.3. Sequence Diagrams

This section includes sequence diagrams for all the use cases described in the Use Case section as well as other sequences that are derived from those requirements.

3.3.1. Authenticate User

Before users can make use of the services of the OpenEMPI, they must first authenticate themselves with the SecurityService and establish a Context that is used for all subsequent operations.

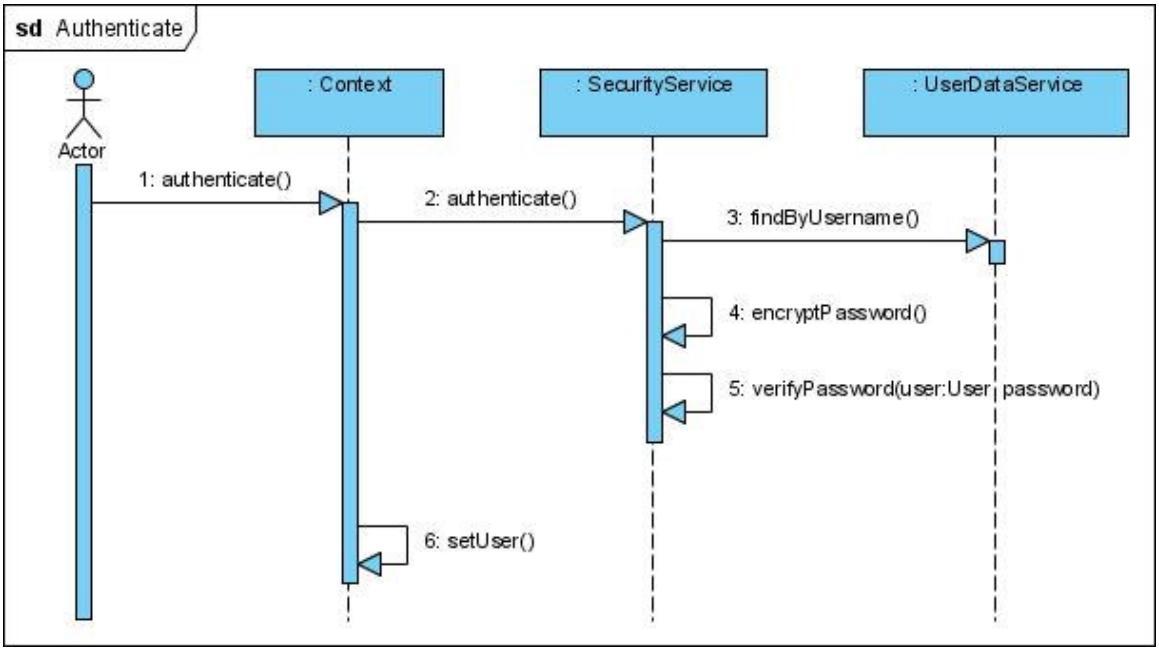

Figure 19 Authenticate User Sequence Diagram

3.3.2. Add Person to Repository

This sequence diagram illustrates the scenario where a person is added to the repository. Before adding a person, we need to make sure that the person is not already in the repository and need to take in consideration that a person may exist in the repository using similar but not the exactly identical attributes to describe the person. This implies that the BlockingService must be used to get a list of candidate matches, and then the MatchingService must be used to assess whether the new person is the same with one of the candidates. If the person isn't already in the system, then the person should be added with an association to other close matches if such matches are found.

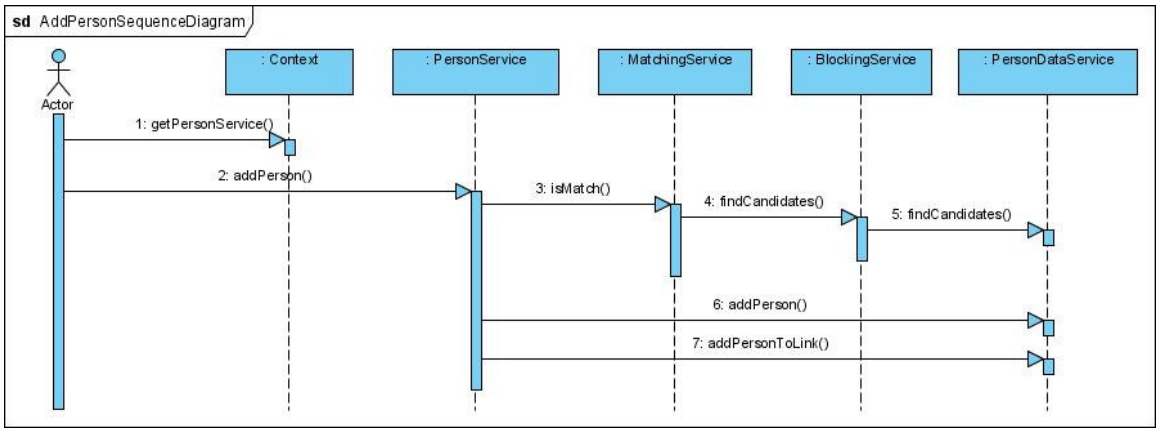

Figure 20 Add Person to Repository Sequence Diagram

3.3.3. Update Person's Attributes

This sequence diagram illustrates the scenario where a person's attributes need to be updated. This requires that the person is located in the repository first, using a person identifier, the attributes are updated as requested, and then the person's is stored back. The matching algorithm needs to be also invoked to update the links between associated person's based on the attributes that were updated.

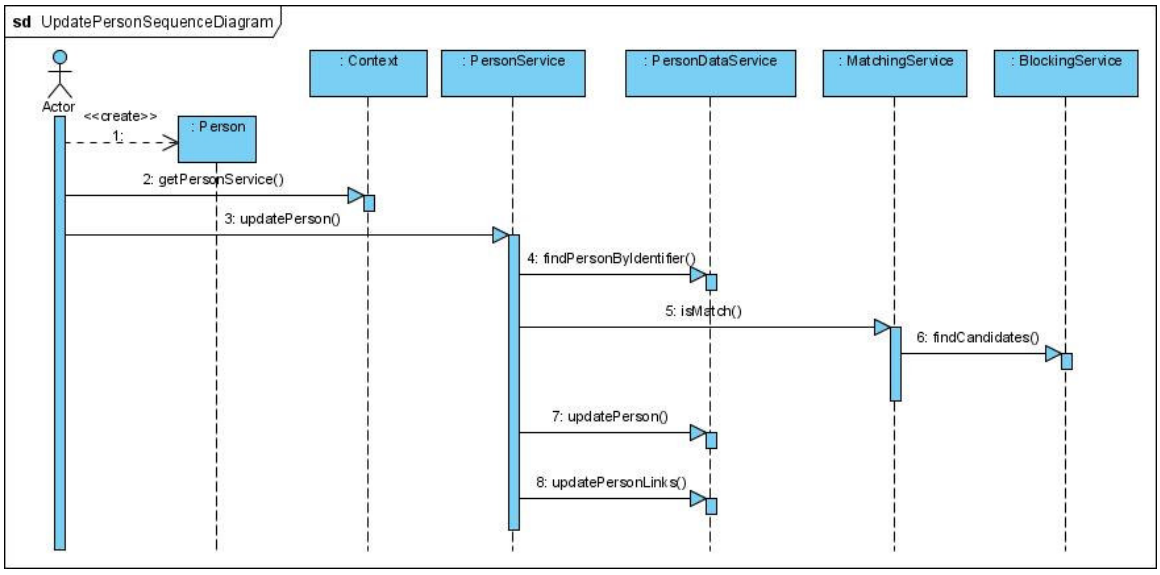

Figure 21 Update Person in Repository Sequence Diagram

3.3.4. Merge Persons

This sequence diagram captures the processing of the merge persons operation. The called specifies the surviving identifier and one or more retired identifiers.

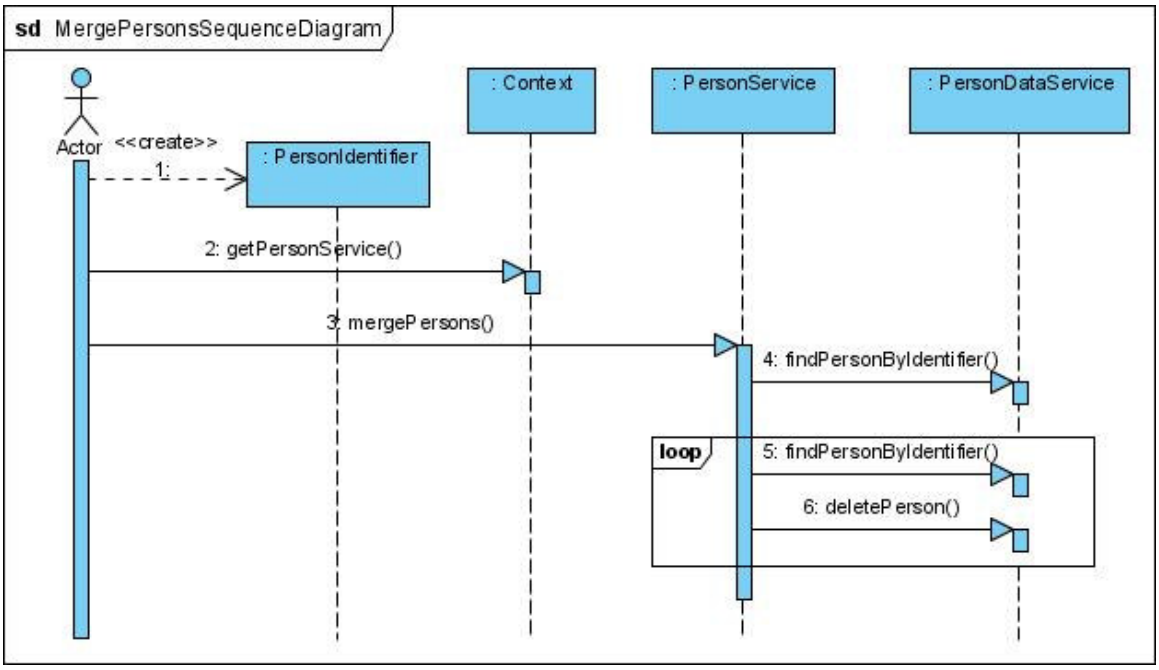

Figure 22 Merge Persons in Repository Sequence Diagram

3.3.5. Delete a Person

The following sequence diagram illustrates the processing logic for deleting a person. A person is never deleted from the repository but rather the record is inactivated.

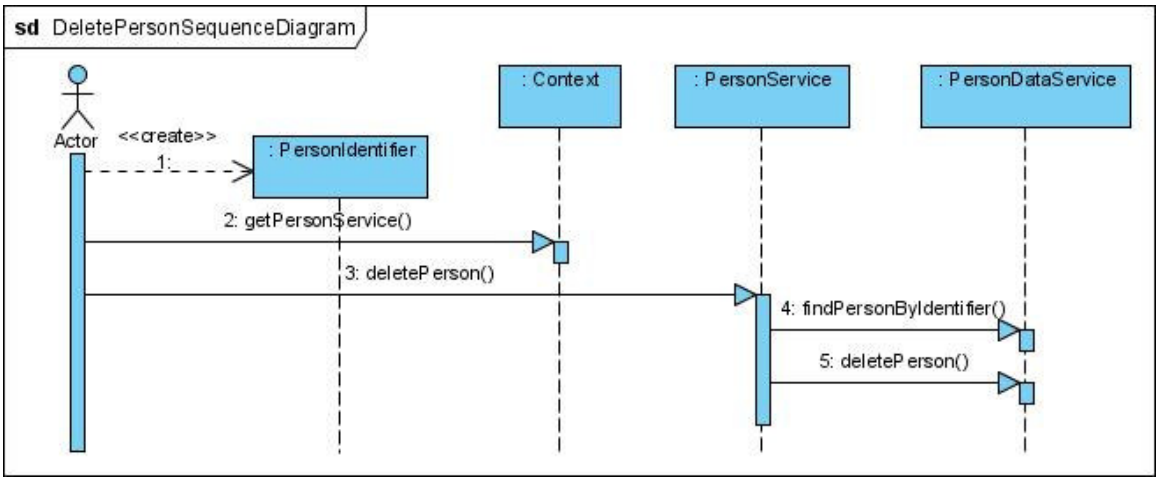

Figure 23 Delete Persons in Repository Sequence Diagram

3.3.6. Query Persons by Identifier

This sequence covers the scenario where a user needs to locate persons using an identifier. There are two variations of this request: one variation returns the single matching person, if such a person is found in the repository, the other variation returns a list of all the

person records linked to the person record with the matching identifier. Only the first variation is illustrated here with a sequence diagram.

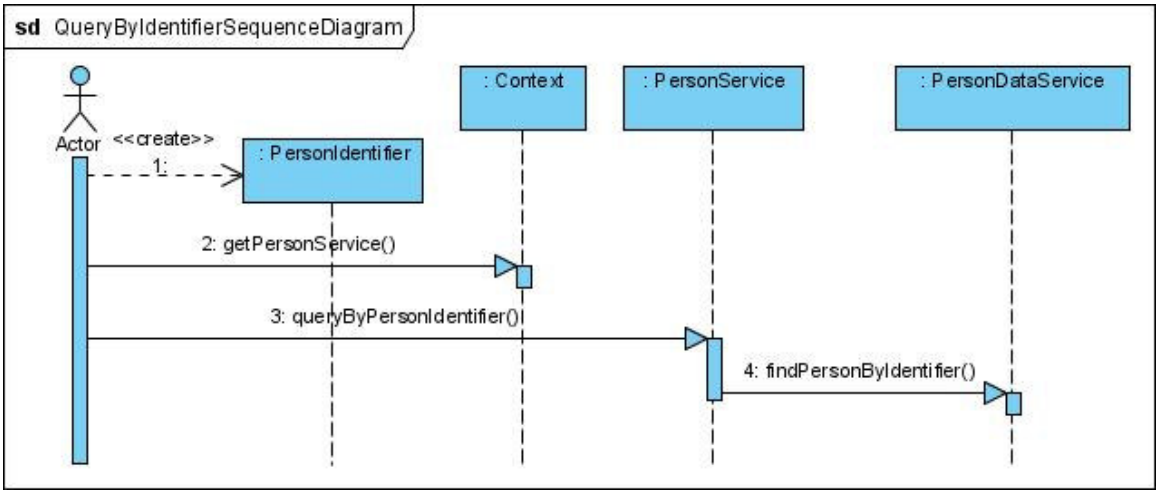

Figure 24 Query By Identifier Sequence Diagram

3.3.7. Query Persons by Attributes

This sequence covers the scenario where a user is trying to locate person records using one or more attributes as query criteria. For example the user may know the person's first name and date of birth. There are two variations of this operation that will need to be implemented. The first one returns only the person records that matched the query criteria whereas the other variation returns the matching person records and their associated records based on the links established by the matching algorithm.

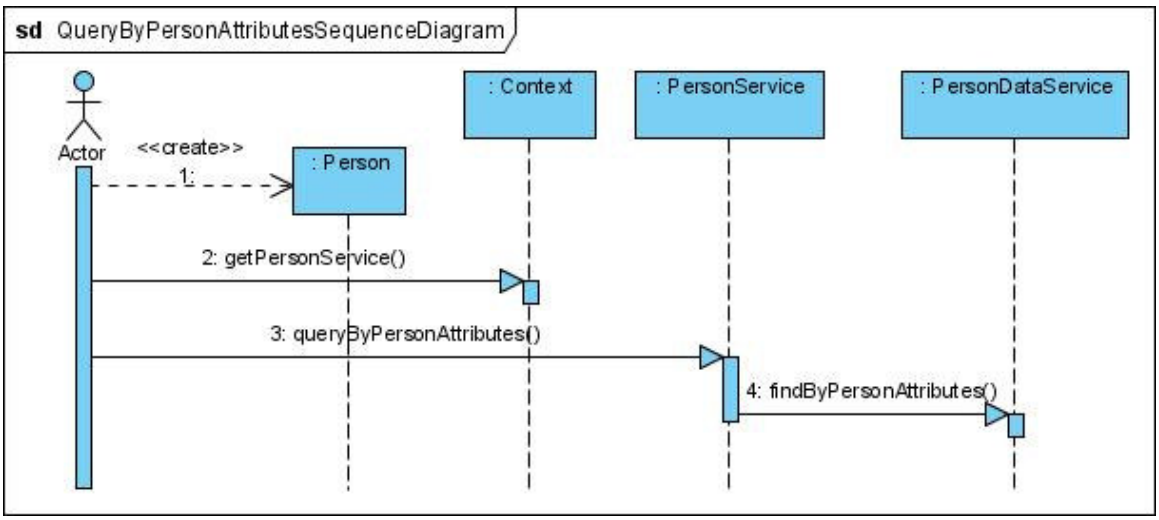

Figure 25 Query by Person Attributes Sequence Diagram

3.3.8. Notification Registration and Notification Distribution

This sequence diagram corresponds to Use Cases UC.8 and UC.9. A user registers interest in certain notification events. Once the user is registered, when an event of

interest takes place against a person that matches the interest criteria of the registered user, a notification event is sent to the user.

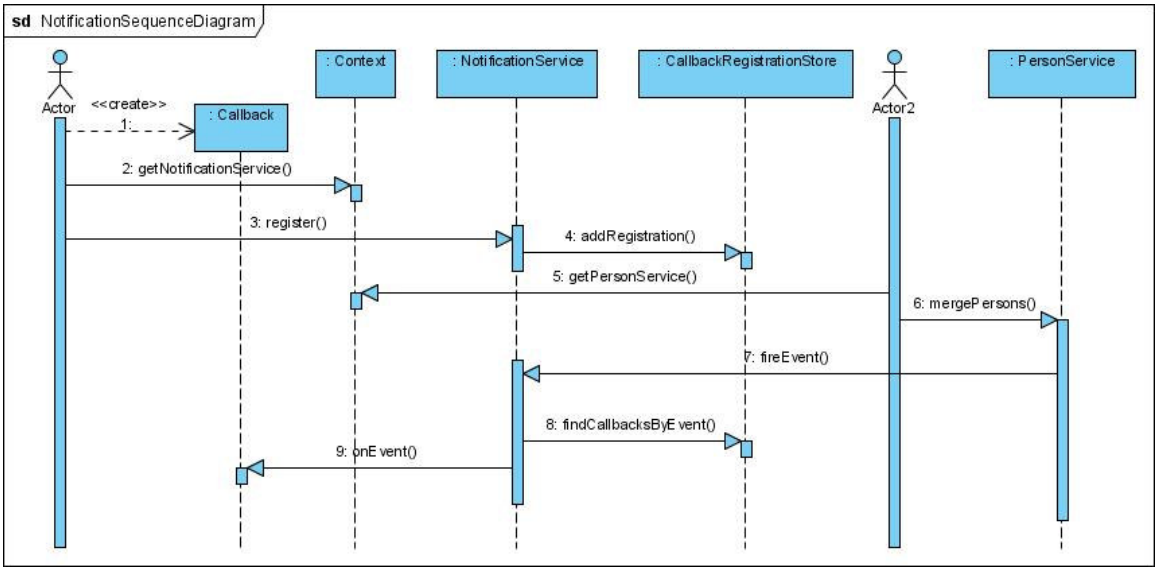

Figure 26 Notification Sequence Diagram

4. Information Viewpoint

4.1. Database Model

This section describes the data model for persistence storage of all data elements of the OpenEMPI. The primarily goals of the data model design are accurate representation of the attributes that describe a person and scalability in terms of throughput and response time to matching algorithms and query requests. The performance requirements are taken in consideration throughout the design since OpenEMPI should be scalable and applicable to environments with millions of persons. As a result of these design goals, some of the person attributes were stored in non-normal form.

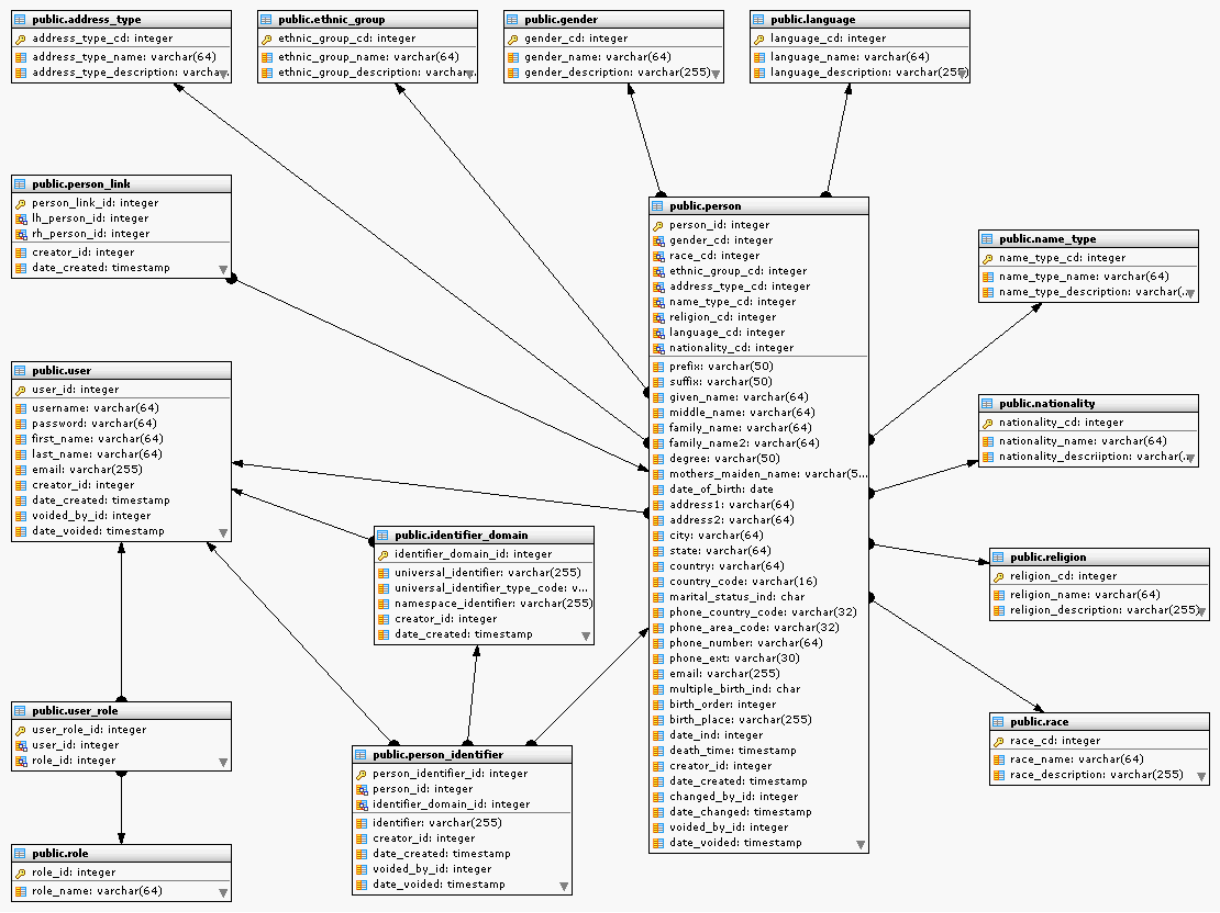

Figure 27 Entity Relationship Diagram of OpenEMPI

5. References

This section includes references to sources of information for this architecture document.

[COCA2001] Cockburn, Alistair, "*Writing Effective Use Cases*", Addison-Wesley, 2001.

[CHRI2002] Christen, Peter, Churches, Tim and Zhu, Justin Xi, "*Probabilistic Name and Address Cleaning and Standardisation"*, The Australian Data Mining Workshop, 2002.

[FOG2006] Fogel, David, "*Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*", IEEE Series on Computational Intelligence, 2006.

[GILL1997] Gill, Leicester, "*OX-LINK: The Oxford Medical Record Linkage System*", in Record Linkage Techniques, http://www.fcsm.gov/working-papers/gill.pdf.

[GILL2001] Gill, Leicester, "*Methods for Automatic Record Matching and Linkage and their Use in National Statistics*", National Statistics Methodological Series, No. 25, Oxford University, 2001.

[IHEITI1] IHE International, "*IHE IT Infrastructure Technical Framework: Integration Profiles*", Volume 1, Revision 5.1, November 14th, 2008.

[IHEITI2] IHE International, "*IHE IT Infrastructure Technical Framework: Transactions*", Volume 2, Revision 5.1, November 14th, 2008.

[PORT97] Porter, H., Edward and Winkler, William, "*Approximate String Comparison and its Effect on an Advanced Record Linkage System*", Research Report, U.S. Census Bureau.

[RAHM] Rahm, Erhard and Do, Hong Hai, "*Data Cleaning: Problems and Current Approaches*", IEEE Data Engineering Bulletin, http://www.sigmod.org/disc/disc01/out/websites/deb_december/rahm.pdf, 2000.

[YANC] Yancey, E., William, "*Evaluating String Comparator Performance for Record Linkage*", Research Report Series: Statistics #2005-05, Statistical Research Division, U.S. Census Bureau, 2005.