**OHIE Client Registry Component Requirements - DRAFT**

**Overview:** Accurate and efficient unique identification of patients is an essential function for a fully realized eHealth architecture. A client registry (CR) is designed to support patient identity management. The OpenHIE CR community seeks to foster innovative technology that provides accurate, reliable and stable identification and de-duplication of individuals and other entities in a variety of contexts, particularly resource-constrained settings.

## 1. OHIE Workflow Requirements

To be OHIE  the system must support one or more of the OHIE workflows listed below:

- **a.** **Create patient demographic record workflow**
- **b.** **Update patient demographic record workflow**
- **c.** **Query patient demographic records by identifier workflow**
- **d.** **Query patient demographic records by demographics workflow**

## 2. Recommended functional requirements:

Depending upon the desired use case(s), the system may support many or all of these functional features.

a. **Configurable Entity matching -** A service to assist in identifying duplicate patients
   i. The rules for determining whether two records match each other should be configurable.*(e.g., ability to use both statistical and/or rules based, etc.)*
   ii. The blocking strategy for loading potential matches before the matching rules are applied should be configurable.
   iii. Any configurable component should have an interface so that advanced users can write their own implementation from scratch if desired.
   iv. Any interface should have at least one default implementation.
   v. The default implementation should be flexible and configurable so that non-programmers can adjust it to meet their needs.
   vi. To the extent possible, CR system configuration information should be managed using consistent and easy to access methods, such as a  database, properties files, or XML files).

b. **Patient Linking and De-duplication**
   1. The system should implement accurate and efficient patient linking and de-duplication methods.

c. **Configure and monitor inbound/outbound transactions.**
   **i.** The system must have the capacity to record receipt and transmission of transactions.

d. **Synchronize client IDs with a SHR. (Support patient-level clinical data OHIE workflow)**

e. **UI to search patients, manually edit (e.g., create, update, merge, split, and deprecate)**
f. **UI to review and manually adjudicate uncertain ("potential") matches, and override incorrect matches.**
g. **Configurable Attributes** -
    i. The attributes that form a patient record and are used for matching should be configurable.
    ii. The implementation can include an example/default patient schema.
    iii. It should be easy to add attributes to the schema.
    iv. It should also be easy to remove attributes from the default model (or start over from scratch).
h. **Error Management**: Ensure that error handling comprehensively captures and logs all related exceptions, and to the extent possible, shows relationships between exceptions.
i. **Logging:** Logging should be consistent; it should be easy to find information in the log.
j. **Privacy/Security:** The system should have functions including user management and access controls.
k. **Pediatric Option**: it is mandatory for an OpenHIE-conformant CR to support the PIX "Pediatric Option"

3. The following are the recommended core software architectural characteristics of a CR:
    a. **System configuration:** Defines entity features, identity sources, decision models, business process rules.
    b. **Data persistence:** Supports reliable low latency, high bandwidth access to potentially large volumes of patient identity information.
    ***c.*** **Object Representations of Patients**
        i. An incoming patient record should not need to be converted into many different formats prior to storing in a database.
        ii. A process like this should be sufficient:
            1. An HL7 message is received, representing the patient as a PID segment within the text
            2. An HL7 library (like HAPI) parses the message, creating an instance of a PID Java object
            3. The Client Registry converts the PID object into an instance of its own patient/entity/whatever class
            4. That object is loaded into the database

4. The following are recommended non-functional requirements.

a. **Well Documented:** A Client Registry system should include appropriate background, design, installation, configuration, and operational documentation to ensure it is easy to understand, maintain, and debug.
    i. *Source code should have comments so that developers do not need to look anywhere else to understand the code.*
    ii. *Configuration files should have embedded comments explaining the different options.*
    iii. *Installation, configuration, and operational activities should be described.*
b. **Easy to implement for common use cases:** While a CR may support a variety of bespoke, tailored clinical workflows, it commonly supports well-known care patterns including registering, updating, and linking identities across multiple registration sources.
c. **Built using open source tools and technology:** The CR should be built using widely available open-source technology (including development environments and languages).
d. **Open, easy access to source code:** A standard version control system (e.g., GitHub) should be used to ensure that source code access is fast, easy to download, compile, and execute code.
e. **Standards-based:** The software should use broadly adopted standards that enable interoperability among systems.
f. **Scales to millions of patients:** Client registries are increasingly expected to support unique at edification of large patient populations. The CR design must support efficient operation (sub-second response time to identity queries) when managing millions of patients.
g. **Reliable and easy-to-use User Interface:** Common identity management workflows must be supported by the CR user interface, including initial system configuration, and routine workflows.
h. **Minimal software library dependencies:** A CR should minimize dependencies on 3rd-party libraries.
i. **Minimal abstraction:** A CR should not have more layers of abstraction than necessary, and seek to minimize abstraction that confounds design.
j. **Easy Initiation:** When properly installed and configured, CR administrators should be able to initiate the CR and any associated supporting processes with a single step.
k. **Build on commonly used technology:**
    i. In order to make it easy to run/configure/debug, the Client Registry should be built on popular technologies that developers like to use.
    ii. Any 3rd party libraries used by the Client Registry should be easy for a typical developer to use.
    iii. Any external software/systems (like the database) should also be easy to use.
    iv. It should be easy to view the contents of the database.
    v. If a traditional SQL database is used, then multiple databases should be supported (MySQL/PostgreSQL/Oracle).
l. **Unit Tests:**

i. The source code should include unit tests that are based on the specific requirements of OpenHIE.

*n.* **License:** The CR would ideally be distributed under an open-source license that minimizes complexity and enables an implementer community to leverage the software in a broad variety of sustainability contexts.

i. The Client Registry should have a clear and standard license so that it is easy to understand what kinds of usage are allowed.

*o.* **Accessible Code:**

ii. The code should be hosted somewhere that developers like to use.

p. **GUI**

*i. The CR should have an easy to use, well thought out and well implemented front end*

*ii. It should allow "wizard-based" or "guided" setup of matching rules*

*iii. It should provide an easy to use and intuitive way to see merge/linkage operations*

*iv. It should allow an easy to use and intuitive way of manually accepting or rejecting merge suggestions, with the ability to choose fields from either record to be merged*