Nationwide Health Information Network (NHIN)

Trial Implementations

Service Interface Specifications

NHIE Service Registry

V 1.1

1/30/2009

Contributors

Name	Organization	Area
Craig Miller	NHIN-C	Specification
Neel Phadke	CareSpark	Specification
Erik Rolf	CareSpark	Specification
Matt Weaver	CareSpark	Specification
Richard Doyle	CareSpark	Specification
Ravi Nistala	NHIN-C	Specification
Karen Witting	IBM/IHE	Specification
Eric Heflin	Delaware	Specification

Document Change History

Version	Date	Changed By	Items Changed Since Previous Version
0.1		Craig Miller	Initial Draft
1.0		Craig Miller	Added sections on use cases and future directions; governance preconditions; provided additional documentation on UDDI tModel, registry authentication and replication; editing for grammar, clarity and formatting
1.1	01/30/2009	David L. Riley	Minor edits to prepare for publication.

Document Approval

Version	Date	Approved By	Role

Table of Contents

1 Preface

1.1 Introduction

The NHIN Trial Implementations Service Interface Specifications constitute the core services of an operational Nationwide Health Information Network. They are intended to provide a standard set of service interfaces that enable Nationwide Health Information Exchange (NHIE) to NHIE exchange of interoperable health information. These services provide such functional capabilities as patient look-up, document query and retrieve, notification of consumer preferences, and access to logs for determining who has accessed what records and for what purpose for use. These functional services rest on a foundational set of messaging and security services. The current set of defined core services includes the following:

1. NHIN Trial Implementations Message Platform Service Interface Specification,
2. NHIN Trial Implementations Authorization Framework Service Interface Specification,
3. NHIN Trial Implementations Subject Discovery Service Interface Specification,
4. NHIN Trial Implementations Query for Documents Service Interface Specification,
5. NHIN Trial Implementations Document Retrieve Service Interface Specification,
6. NHIN Trial Implementations Audit Log Query Service Interface Specification,
7. NHIN Trial Implementations Consumer Preferences Service Interface Specification
8. NHIN Trial Implementations Health Information Event Messaging Service Interface Specification
9. NHIN Trial Implementations NHIE Service Registry Interface Specification
10. NHIN Trial Implementations Authorized Case Follow-Up Service Interface Specification

It is expected that these core services will be implemented together as a suite since the functional level services are dependent on the foundational services. Specifications #1 through #7 were the focus of the August 2008 testing event and September AHIC demonstrations. Specifications #1 through #9 were included in the November testing and demonstrations during the December 2008 NHIN Trial Implementations Forum.

1.2 Intended Audience

The primary audience for the NHIN Trial Implementations Service Interface Specifications is the individuals responsible for implementing software solutions that realize these interfaces for a NHIE. After reading this specification, one should have an understanding of the context in which the service interface is meant to be used, the behavior of the interface, the Web Services Description Language (WSDLs) used to define the service, any Extensible Markup Language (XML) schemas used to define the content and what “compliance” means from an implementation testing perspective.

1.3 Focus of this Specification

This document presents the NHIN Trial Implementations Query for Documents Service Interface Specification. The purpose of this specification is to provide the ability to exchange patient specific clinical documents between NHIEs.

2 NHIE Interface

The NHIE Service Registry specification defines the capabilities and interfaces for one or more Service Registries. These registries maintain the information required for one NHIE to discover the existence of other NHIEs within the NHIN, and the associated information that enables one NHIE to establish a connection to another NHIE. The selected platform for the NHIE Service Registry is based on the Universal Description Discovery Interface (UDDI) version 3.0.2 specification, which is available for download at http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3.

NHIN Cooperative WG name
Security and Technical Working Group

3 Interface Between

This specification document specifies the interface between an NHIE and one or more Service Registries that are maintained by the NHIN. It also specifies the replication interface between two or more instances of NHIE Service Registries.

4 NHIE Core Services and use cases supported

This specification supports the Messaging Platform Core Service. It does not directly support any identified AHIC use case. However, the NHIE Service Registry specification does address underlying infrastructure requirements for connection management among NHIEs. There are four important use cases that have been identified thus far:

4.1 Get all NHIE data

In this use case, an NHIE has been approved for participation within the NHIN and chooses to retrieve all information available in the Service Registry. It requests a list of all other NHIE's currently registered, and all the services they support. This new NHIE will maintain its local cache of this information for use in responding to local requests to share and retrieve data. This NHIE will need to be notified when new NHIE's are added to the Service Registry or when an existing NHIE entry is changed. To do this the new NHIE will provide a subscription to the Service Registry requesting notification of updates.

4.2 Get data about NHIE's by state

In this use case, an NHIE has been approved for participation within the NHIN. Given the scope of this NHIE it does not request all NHIE's registered, but instead desires a subset of all NHIE's in a particular region or state. The new NHIE queries the Service Registry and specifies a list of states to restrict the results.

A similar use case comes up when a patient requests that records from an NHIE in a particular state be retrieved and the local NHIE has not previously searched for NHIE's in that state. In both these cases the requesting NHIE will want to subscribe for updates to its locally cached information, or new NHIE entries matching its original query.

4.3 Getting data by homeCommunityId

There are several potential use cases which would require the ability to query the Service Registry by a specific homeCommunityId to retrieve the service connection information for the corresponding NHIE. For instance, the patient may have a printed paper, generated by an NHIE, which includes a homeCommunityId, perhaps even in a form that could be scanned. Another case is a notification may have been received which identifies the homeCommunityId which contains the record of interest and the NHIE receiving the notification may not have previously retrieved the service connection information for that NHIE. For these, and potentially others, the NHIE submits a query to the Service Registry specifying the homeCommunityId and receives the details about that community that have been saved in the Service Registry.

4.4 Service Registry backup

In the case of a physical malfunction of the site of the Service Registry, or a software failure, the data in the service registry must still be available to communicating NHIE's, requiring a backup and replication function to support the malfunction of the primary.

5 NHIE context for use

A Service Registry provides the mechanism for NHIEs to discover and initially connect to each other within the context of the network.

More specifically, for each NHIE within the NHIN, the Registry maintains the following information:

- The name of the NHIE
- The unique network identifier (Home Community ID) of the NHIE
- A URI where the public key of the NHIE x.509 security certificate can be accessed
- A URI where the WSDL interface definitions for the NHIE can be accessed
- Contact information for the NHIE's technical point of contact

In UDDI terminology, the structure of a registry entry is described using a tModel, which is articulated below in this document. With this information, one NHIE can establish a secure connection to another (using the x.509 public key), locate and invoke the services of other NHIEs (based on the endpoints defined in the WSDLs), and uniquely identify and direct messages to other NHIEs (using their Home Community ID).

The Service Registries are a necessary prerequisite to establish a scalable, national network where the entities participating and the services they offer are dynamic. It is envisioned that NHIEs will query a Registry to obtain connection information for another NHIE, but then cache that information locally for subsequent communication with the other NHIEs, in order to improve performance and minimize network bandwidth. However, the means and frequency of local caching of this information is within the discretion of individual NHIEs and outside the scope of this specification. This specification does define a Subscription interface that NHIEs can use to be notified when changes are made to the Registry which should cause them to flush their local cache and then re-query the Registry to obtain updated information.

The content of NHIE Service Registries will be entered and maintained by the governance body established by the NHIN, not by individual NHIEs. Multiple Registry servers will be maintained by this body in order to provide continuous availability; this specification defines a replication interface to ensure that all copies of the Registry are synchronized with each other.

Access to the Registry is not available to the public. It will require NHIEs to authenticate themselves to a Registry server using the NHIE's x.509 certificate information. This is described in more detail below.

6 Description of Interface

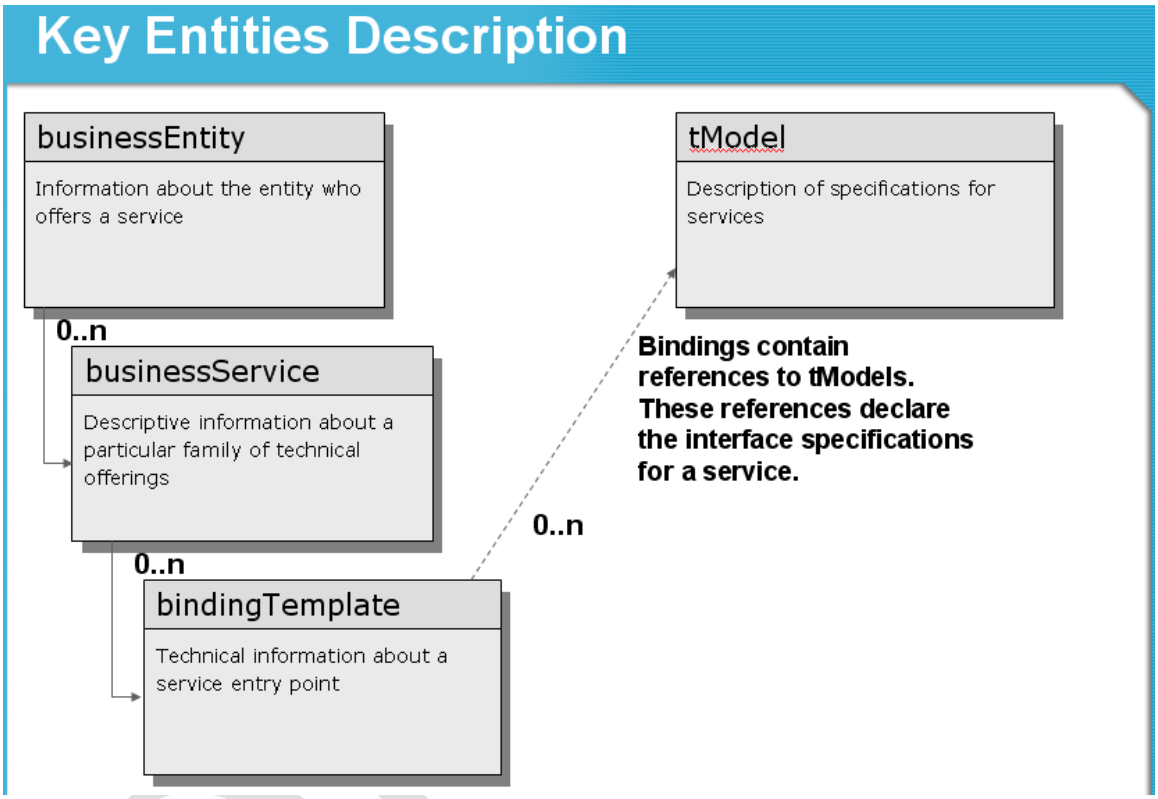
This section of the document is organized into five subsections:

- **Registry Data Model**, which describes the structure of the Registry entries for each NHIE
- **Inquiry API**, which specifies how an NHIE can query a registry to obtain connection information for another NHIE
- **Subscription API**, which specifies how an NHIE can subscribe to a Registry to be notified of changes to Registry content that should prompt a re-query of the Registry
- **Security**, which describes how NHIEs must authenticate themselves to a Registry

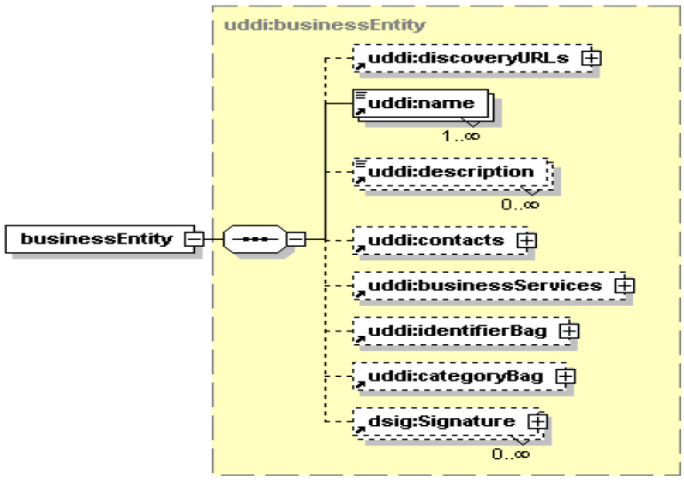
- **Replication**, which defines the specification for ensuring that multiple copies of the registry are synchronized with each other

6.1 Registry Data Model (UDDI tModel)

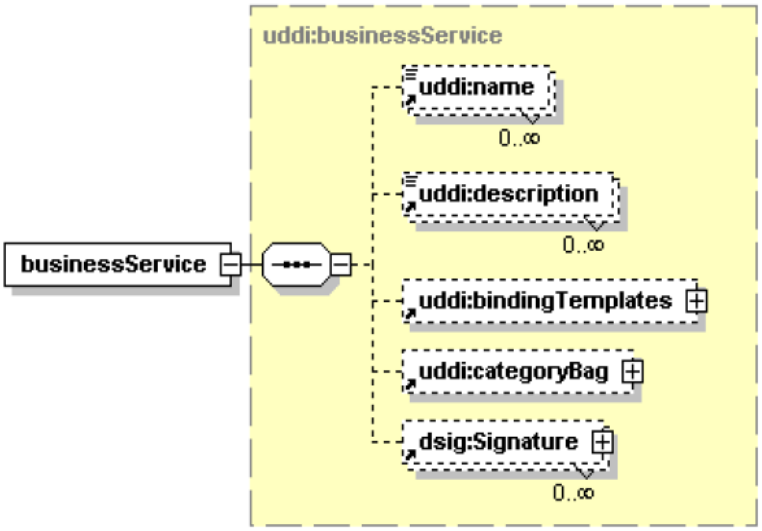
The UDDI data model is composed of four primary "top-level" entities each identified by a unique identifier (UUID).



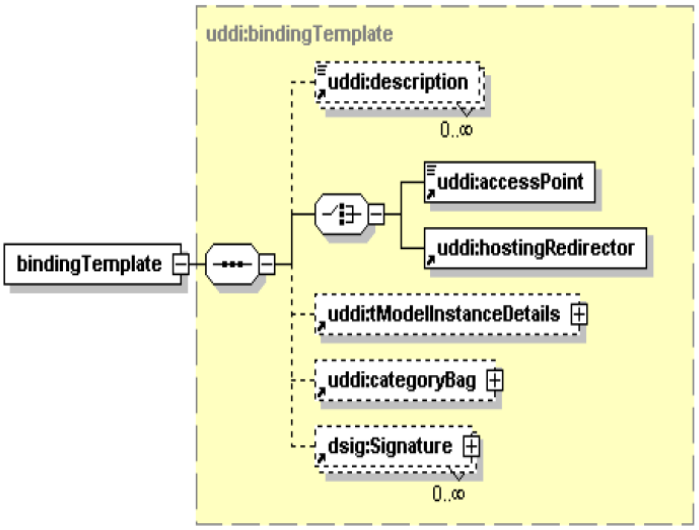
BusinessEntity - identifies a business or an organization providing the services. It captures the following identifiable information about the business / organization. (E.g. information like Name of the NHIE, Point of contacts can be captured here)



BusinessServic lowing classification information. (E.g. Information regarding services implemented by a given NHIE can be captured here)



BindingTemplate – provides the technical information required to access a service. (E.g. WSDL specific information i.e. porttypes, etc. can be captured here)



tModel – is a co letailed information that a UDDI registry can provide about any specification. A *BusinessEntity* references *tModels* to define value sets (e.g. organization categories, system of values categorization, etc.), to define technical fingerprints (e.g. represent a spec that defines a soap protocol, etc.)

Assumptions

- UDDI queries based on specific WSDL artifacts are beyond the current scope of requirements
- Assumed that nothing is known about the WSDL other than its URL

6.1.1 NHIN Taxonomies

Defining categorization scheme(s) is a key to reuse and enables flexible UDDI queries based on the categorization. For NHIN service registry, the following custom categorizations are defined:

Name of Taxonomy	Taxonomy Type	Description	Compatibility	Current Valid values
uddi:nhin:nhie:state	CategoryBag	The geographic location of the NHIE	BusinessEntity	2 letter state codes (e.g. VA, TN, etc.)
uddi:nhin:nhie:federalhie	CategoryBag	Is it a federal NHIE	BusinessService	YES, NO
uddi:nhin:nhie:homecommunityid	IdentifierBag	A unique identifier for the NHIE	BusinessEntity	Value defined in the urn:oid format
uddi:nhin:nhie:publickeyuri	CategoryBag	URI to the public key of the NHIE	BusinessEntity	URI to the public key
uddi:nhin:uniformservicename	CategoryBag	A uniform (standardized) name for the NHIN business service	BusinessService	AuditLogQueryService, DocumentQueryService, DocumentRetrieveService, SubjectDiscoveryService
uddi:nhin:versionofser	CategoryBag	Indicates the current	BusinessService	Value is of

vice		version of the service		the form Integer.Integer (e.g. 1.0, 1.1, 2.0, etc.)

6.1.2 Sample NHIN Service Registry Entry

The following sample business entity is modeled Based on the above taxonomies. The sample also outlines the required and optional elements for the NHIN service registry:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <businessEntity businessKey="uddi:caresparknode:aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa">
      <!-- REQUIRED: Name of the HIE -->
      <name>CareSpark</name>

      <!-- OPTIONAL: alternate name for the HIE -->
      <name>CareSpark RHIO</name>

      <!-- REQUIRED: HIE's website home page url -->
      <discoveryURLs>
        <discoveryURL useType="homepage">
          http://www.carespark.com
        </discoveryURL>
      </discoveryURLs>

      <!-- REQUIRED: Brief description about the HIE -->
      <description>Health Information Exchange in the Tennessee region</description>


      <!-- REQUIRED: HIE Contact details. Atleast ONE contact element is required with
personName, phone, email & address elements defined -->
      <contacts>
        <contact>
          <personName>John Doe</personName>
          <phone>111-111-1111</phone>
          <email>john.doe@carespark.com</email>
          <address>
            <addressLine>112 W. Main Street</addressLine>
            <addressLine>Kingsport, TN 37662</addressLine>
          </address>
        </contact>
      </contacts>

      <!-- REQUIRED: The custom taxonomies defined for the business entity -->
      <identifierBag>
        <keyedReference
          tModelKey="uddi:nhin:nhie:homecommunityid"
          keyValue="urn:oid:2.16.840.1.113883.3.166.4"/>
      </identifierBag>

      <!-- REQUIRED: The custom taxonomies defined for the business entity -->
      <categoryBag>
        <!-- Only ONE reference to the uddi:nhin:nhie:state is mandatory. There could be
multiple references -->
        <keyedReference
          tModelKey="uddi:nhin:nhie:state"
          keyValue="TN"/>
        <keyedReference
          tModelKey="uddi:nhin:nhie:state"
          keyValue="VA"/>

        <keyedReference
          tModelKey="uddi:nhin:nhie:federalhie"
```

```
            keyValue="NO"/>
          <keyedReference
            tModelKey="uddi:nhin:nhie:publickeyuri"
            keyValue=" https://www.NhinGovernanceServer/pki/id48.cer"/>
        </categoryBag>
        <businessServices>
          <businessService
            businessKey="uddi:caresparknode:aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
            serviceKey="uddi:caresparknode:bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb">

            <!-- REQUIRED: Brief name of the business service -->
            <name>Audit Log Query Service</name>

            <!-- REQUIRED: Brief description about the service -->
            <description>Service to retrieve audit events for a given date
range</description>

            <!-- REQUIRED: The custom taxonomies defined for the business service -->
            <categoryBag>
              <keyedReference
                tModelKey="uddi:nhin:uniformservicename"
                keyValue="AuditLogQueryService"/>
              <keyedReference
                tModelKey="uddi:nhin:versionofservice"
                keyValue="1.0"/>
            </categoryBag>
            <bindingTemplates>
              <bindingTemplate
                serviceKey="uddi:caresparknode:bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb"
                bindingKey="uddi:caresparknode:cccccccc-cccc-cccc-cccc-cccccccccccc">

                <!-- REQUIRED: Brief about the service's technical information -->
                <description>AuditLogQuery service implemented as a web-service. Supports
saml over 2way ssl </description>

                <!-- REQUIRED: location of the the service endpoint -->
                <accessPoint useType="endPoint">

http://csnhintiapps03.cgifederal.com:5050/findAuditEvents/AuditLogQueryService
                </accessPoint>

                <!-- REQUIRED: tModels for service WSDL and transport -->
                <tModelInstanceDetails>
                  <tModelInstanceInfo tModelKey="uddi:nhin:auditlogquery_interface"/>
                </tModelInstanceDetails>
              </bindingTemplate>
            </bindingTemplates>
            </businessService>
            <businessService
              businessKey="uddi:caresparknode:aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
              serviceKey="uddi:caresparknode:dddddddd-dddd-dddd-dddd-dddddddddddd">

            <!-- REQUIRED: Brief name of the business service -->
              <name>Subject Discovery Service</name>

            <!-- REQUIRED: Brief description about the service -->
              <description>Service to perform subject discovery over NHIN</description>


            <!-- REQUIRED: The custom taxonomies defined for the business service -->
              <categoryBag>
                <keyedReference
                  tModelKey="uddi:nhin:uniformservicename"
```

```
                keyValue="SubjectDiscoveryService"/>
              <keyedReference
                tModelKey="uddi:nhin:versionofservice"
                keyValue="2.0"/>
            </categoryBag>
            <bindingTemplates>
              <bindingTemplate
                serviceKey="uddi:caresparknode:dddddddd-dddd-dddd-dddd-dddddddddddd"
                bindingKey="uddi:caresparknode:eeeeeeee-eeee-eeee-eeee-eeeeeeeeeeee">

              <!-- REQUIRED: Brief about the service's technical information -->
                <description>Subject discovery service implemented as a web-service.
Supports saml over 2way ssl </description>

              <!-- REQUIRED: location of the the service endpoint -->
                <accessPoint useType="endPoint">

https://csnhintiapps03.cgifederal.com:6151/SubjectDiscoveryRespondingGatewayService/Subject
DiscoveryRespondingGatewayPort
                </accessPoint>

              <!-- REQUIRED: tModels for service WSDL and transport -->
                <tModelInstanceDetails>
                  <tModelInstanceInfo tModelKey="uddi:nhin:subjectdiscovery_interface"/>
                </tModelInstanceDetails>
              </bindingTemplate>
            </bindingTemplates>
          </businessService>
        </businessServices>
      </businessEntity>
```

The above sample references the following common custom *tModels* defined for NHIN Service registry:

```
      <!-- Home community id identifier -->
      <tModel tModelKey="uddi:nhin:nhie:homecommunityid">
            <name>nhin-nhie-homecommunityid</name>
            <description>Home community ID</description>
            <categoryBag>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:identifier" keyValue="identifier"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:unchecked" keyValue="unchecked"/>
            </categoryBag>
      </tModel>

      <!-- State Category -->
      <tModel tModelKey="uddi:nhin:nhie:state">
            <name>nhin-nhie-state</name>
            <description>NHIE State</description>
            <categoryBag>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:categorization" keyValue="categorization"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:unchecked" keyValue="unchecked"/>
            </categoryBag>
```

```
	</tModel>

	<!-- NHIE Federal HIE Category -->
	<tModel tModelKey="uddi:nhin:nhie:federalhie">
		<name>nhin-nhie-federalhie</name>
		<description>Is HIE Federal HIE</description>
		<categoryBag>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:categorization" keyValue="categorization"/>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:unchecked" keyValue="unchecked"/>
		</categoryBag>
	</tModel>

<!-- State Category -->
	<tModel tModelKey="uddi:nhin:nhie:publickeyuri">
		<name>nhin-nhie-publickeyuri</name>
		<description>URI to the public key</description>
		<categoryBag>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:categorization" keyValue="categorization"/>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:unchecked" keyValue="unchecked"/>
		</categoryBag>
	</tModel>


	<!-- NHIN Uniform Service Name Category -->
	<tModel tModelKey="uddi:nhin:uniformservicename">
		<name>nhin-uniformservicename</name>
		<description>Uniform Service Name</description>
		<categoryBag>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:categorization" keyValue="categorization"/>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:unchecked" keyValue="unchecked"/>
		</categoryBag>
	</tModel>

	<!-- NHIN Version of Service Category -->
	<tModel tModelKey="uddi:nhin:versionofservice">
		<name>nhin-versionofservice</name>
		<description>Service Version Number</description>
		<categoryBag>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:categorization" keyValue="categorization"/>
			<keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:unchecked" keyValue="unchecked"/>
		</categoryBag>
	</tModel>

	<!—tModel for the AuditLogQueryService -->
	<tModel tModelKey="uddi:nhin:auditlogquery_interface">
		<name>nhin:auditlogquery_interface</name>
```

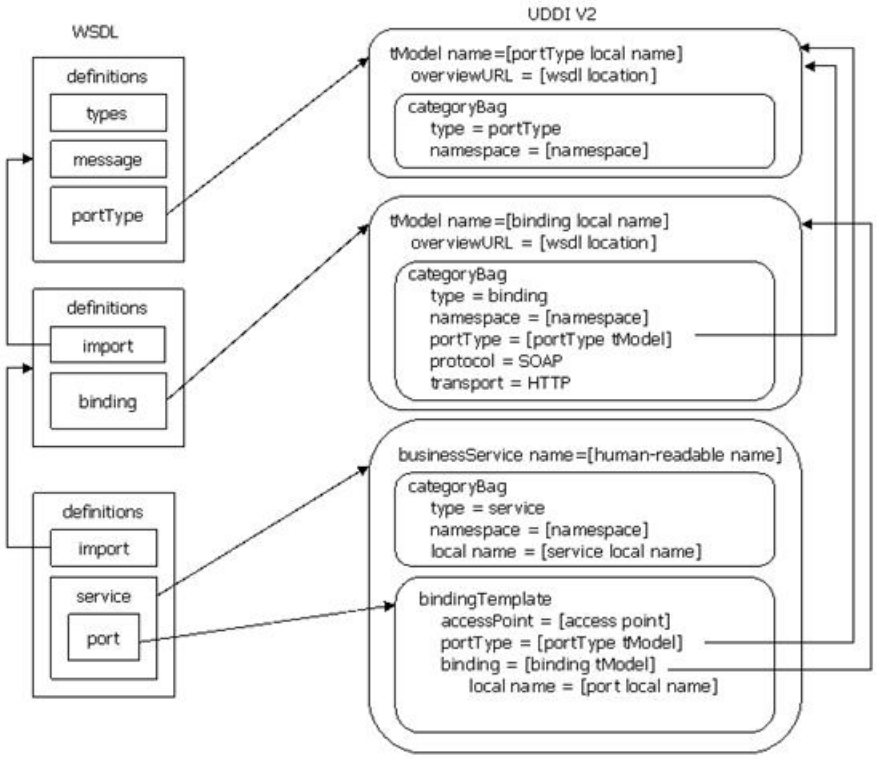
```
            <description>This tModel represents the Audit Log Query Service
WSDL as defined by the NHIN cooperative.</description>
            <overviewDoc>
                  <overviewURL
useType="wsdlInterface">http://NhinGovernanceServer
/findAuditEvents/AuditLogQuery.wsdl</overviewURL>
            </overviewDoc>
            <categoryBag>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-org:types:wsdl"
keyValue="wsdlSpec"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-org:types:soap"
keyValue="soapSpec"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-org:types:xml"
keyValue="xmlSpec"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:specification" keyValue="specification"/>
            </categoryBag>
      </tModel>

      <!—tModel for the SubjectDiscoveryService -->
      <tModel tModelKey="uddi:nhin:subjectdiscovery_interface">
            <name>nhin:subjectdiscovery_interface</name>
            <description>This tModel represents the Subject Discovery Service
WSDL as defined by the NHIN cooperative </description>
            <overviewDoc>
                  <overviewURL
useType="wsdlInterface">http://NhinGovernanceServer
/someFolder/SubjectDiscovery.wsdl</overviewURL>
            </overviewDoc>
            <categoryBag>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-org:types:wsdl"
keyValue="wsdlSpec"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-org:types:soap"
keyValue="soapSpec"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-org:types:xml"
keyValue="xmlSpec"/>
                  <keyedReference
tModelKey="uddi:uddi.org:categorization:types" keyName="uddi-
org:types:specification" keyValue="specification"/>
            </categoryBag>
      </tModel>
```

6.1.3 WSDL to UDDI Mapping *(currently out of scope)*

WSDL to UDDI mapping enables precise and flexible UDDI queries based on specific WSDL artifacts and metadata. The OASIS UDDI spec technical note (http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm) describes the mapping methodology for the UDDI. The sections below have been extracted from the OASIS specification.

Mapping Overview -

The diagram below describes the methodology for mapping WSDL 1.1 to UDDI V2/V3.



Mapping Table

The table below has been derived from the OASIS UDDI recommendation.

WSDL artifact	Corresponding UDDI element	Rules
wsdl:service	uddi:businessService	Only one wsdl:service can be modeled by an individual uddi:businessService. If a new businessService is created, the uddi:name elements of this businessService SHOULD be human readable names, although if no human readable names are specified, exactly one uddi:name MUST be added, containing the value of the name attribute of the wsdl:service

		The businessService MUST contain a categoryBag categoryBag MUST contain A keyedReference with a tModelKey of the WSDL Entity Type category system and a keyValue of "service" AND/OR A keyedReference with a tModelKey of the XML Local Name category system and a keyValue that is the value of the name attribute of the wsdl:service If the wsdl:service has a targetNamespace then the categoryBag MUST also contain an additional keyedReference with a tModelKey of the XML Namespace category system and a keyValue of the target namespace of the wsdl:definitions element that contains the wsdl:service The bindingTemplates element of the businessService MUST include bindingTemplate elements that model the ports of the service
wsdl:port	uddi:bindingTemplate	A wsdl:port MUST be modeled as a uddi:bindingTemplate tModelInstanceDetails element MUST be present The bindingTemplate tModelInstanceDetails element MUST contain at least the following tModelInstanceInfo elements: A tModelInstanceInfo element MUST be present with a tModelKey of the tModel that models the wsdl:binding that this port implements. The instanceParms of this tModelInstanceInfo MUST contain the wsdl:port local name A tModelInstanceInfo element MUST be present with a tModelKey of the tModel that models the wsdl:portType
soap:address	uddi:accessPoint	A soap:address MUST be modeled as a uddi:accessPoint in the uddi:bindingTemplate that models the wsdl:port that contains the soap:address The accessPoint value MUST be the value of the location attribute of the soap:address element The useType attribute of the accessPoint MUST correspond to the transport specified by the soap:binding, or "other" if no correspondence exists. In the case of the HTTP transport, for example, the URLType attribute MUST be "http"
wsdl:binding	uddi:tModel	The uddi:name element of the tModel MUST be the

		value of the name attribute of the wsdl:binding The tModel MUST contain a categoryBag categoryBag MUST contain a keyedReference with a tModelKey of the WSDL Entity Type category system and a keyValue of "binding" categoryBag MUST contain a keyedReference with a tModelKey of the WSDL portType Reference category system and a keyValue of the tModelKey that models the wsdl:portType to which the wsdl:binding relates categoryBag MUST contain a keyedReference with a tModelKey of the UDDI Types category system and a keyValue of "wsdlSpec" for backward compatibility categoryBag MUST contain a one or two keyedReferences as required to capture the protocol and optionally the transport information If the wsdl:binding has a targetNamespace then the categoryBag MUST also contain an additional keyedReference with a tModelKey of the XML Namespace category system and a keyValue of the target namespace of the wsdl:definitions element that contains the wsdl:binding If the targetNamespace is absent from the portType, a categoryBag MUST NOT contain a keyedReference to the XML Namespace category system The tModel MUST contain an overviewDoc with an overviewURL containing the location of the WSDL document that describes the wsdl:binding.
wsdl:portType	uddi:tModel	The uddi:name element of the tModel MUST be the value of the name attribute of the wsdl:portType The tModel MUST contain a categoryBag, and the categoryBag MUST contain a keyedReference with a tModelKey of the WSDL Entity Type category system and a keyValue of "portType" If the wsdl:portType has a targetNamespace then the categoryBag MUST also contain an additional keyedReference with a tModelKey of the XML Namespace category system and a keyValue of the target namespace of the wsdl:definitions element

		that contains the wsdl:portType If the targetNamespace is absent from the portType, a categoryBag MUST NOT contain a keyedReference to the XML Namespace category system The tModel MUST contain an overviewDoc with an overviewURL containing the location of the WSDL document that describes the wsdl:portType.

Future work – Based on the above table, it would be required to define custom taxonomies. The NHIN service registry business entities would have to be re-defined using the custom taxonomies and standard (WSDL specific) *tModels*.

6.2 Inquiry API (Client discovery API)

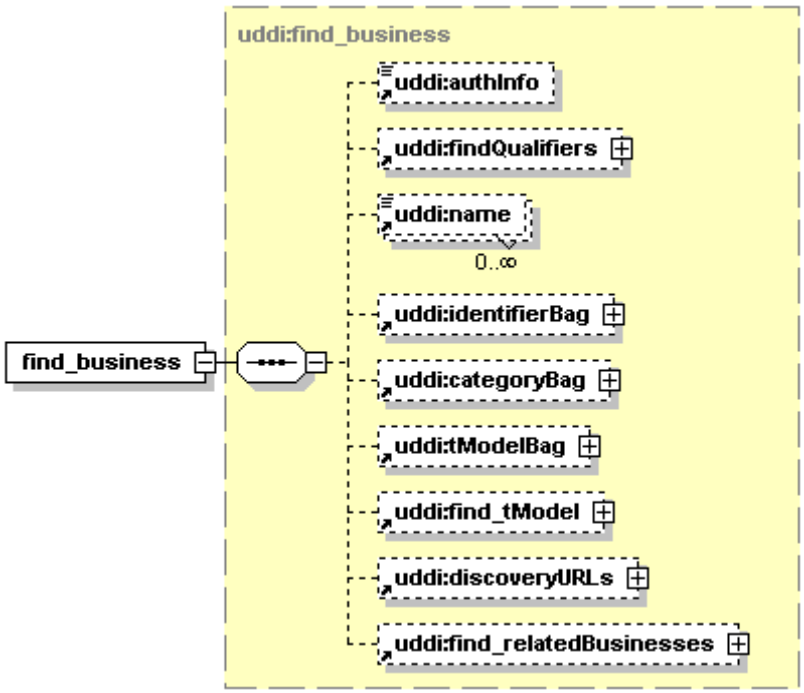
The UDDI V3.0.2 inquiry APIs provide a simple and complete set of programming interfaces, these interfaces can be used to:

- Search the UDDI registry to locate registry entries (business / technical) pertaining to the given search criteria
- Retrieve further details of a given registry entry.

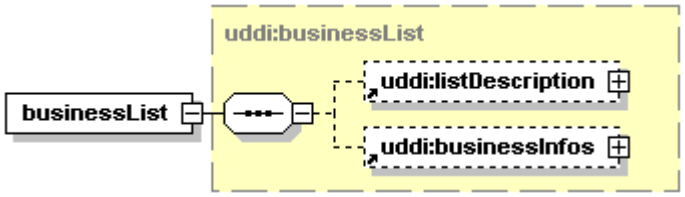
6.2.1 Details

A summary of all the inquiry API's supported by UDDI v3 specification are defined in **Appendix F.** Details of all the API's can be found in sections 5.1.9 to 5.1.18 of the UDDI specification.
Below is a detailed description of subset of inquiry APIs that would be required to perform searches on the NHIN service registry:

1) ***find_business*** – This API is used to locate which of the organization are registered with the UDDI. This will return the list of the organizations (both federal and non federal) which matches the criteria.

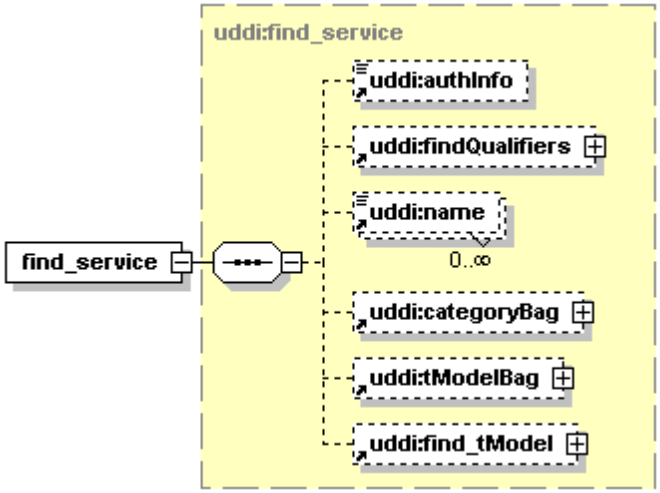


The syntax of the return list is:

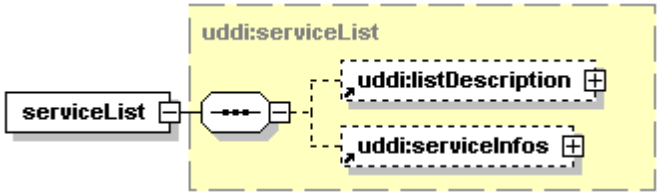


See section 5.1.10 of the UDDI specification for more details.

2) ***find_service*** – The *find_service* API is used to find UDDI *businessService* elements. The *find_service* API call returns a *serviceList* structure that matches the conditions specified in the arguments. In the case of NHIE these will be the services which will get you the URL to get to the core services WSDL's.

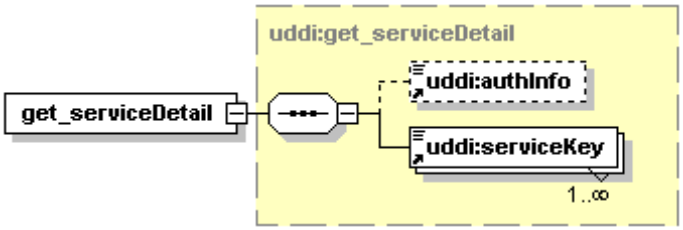


This API call returns a *serviceList* on success

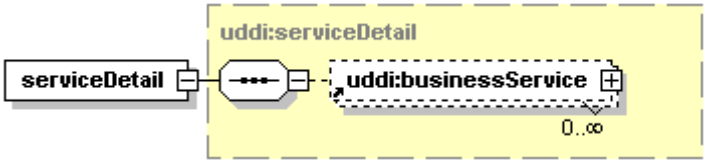


See section 5.1.12 of the UDDI specification for more details.

3) ***get_serviceDetail*** – This API is used to get details of business services implemented at a particular NHIE gateway.

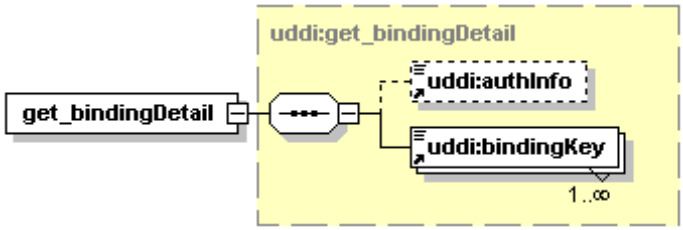


This API call returns a *serviceDetail* on successful match of the specified *serviceKey* values.

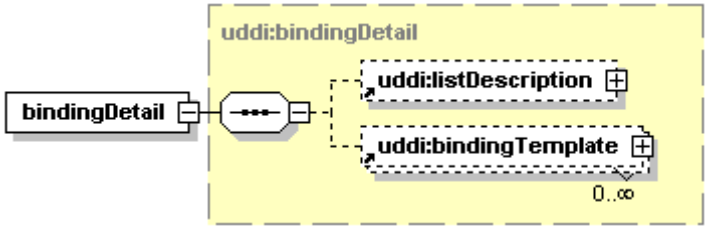


See section 5.1.17 of the UDDI specification for more details.

4) ***get_bindingDetail*** **-** This API is needed if we need to get the location of the WSDL. The required parameters are shown below.

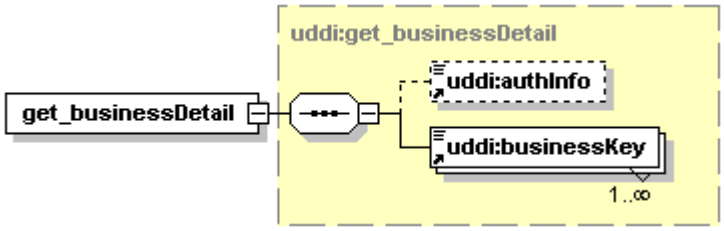


This API returns the binding detail upon success. The binding detail contains the location of the WSDL.The location is defined at “overviewURL”



See section 5.1.14 of the UDDI specification for more details.

5) ***get_businessDetail*** – This API is needed if we need to get to the public key. The input parameter required for this API is business key which we can from either *find_business* or from *find_service*. The structure of the input is as specified below.



The response contains the business details which contains the category bag for public key.

The ***find_XX*** API does contain an argument called "***maxRows***" where you can specify how many rows to return. ***Get_XX*** APIs do not have this attribute. In the event of a large number of matches (this can be configured at the implementation node), or if the number of matches exceeds the value of the *maxRows* attribute, the node MAY truncate the result set. When this occurs, the returned list contains the attribute "**truncated***"* with the value of this attribute set to "**true***"*.

Find qualifiers are used with *find_XX* API's. Each of the *find_xx* API accepts an optional *findQualifier* values. Find qualifiers may be either *tModelKeys* or may be referenced by a string containing a "short name". Each of the pre-defined *findQualifiers* in UDDI can be referenced using either the appropriate *tModelKey*, or by its short name. Registries MUST support both forms, and MUST accept the find qualifiers case-insensitively. There are some invalid combinations of *findQualifier's*. Details can be found in section 5.1.4.

Note that the APIs can be invoked over a SSL transport layer and the authentication/authorization mechanism is based on PKI/X509. More details can be found under section 5.4.

6.2.2 Sample – Using Inquiry APIs for NHIN Service Registry search

This section describes the usage of Inquiry APIs for searching the NHIN service registry.

Assume that we have the following *BusinessEntity* as described by the sample below.

```
<businessEntity businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
xmlns="urn:uddi-org:api ">
<!-- REQUIRED: Name of the HIE -->
<name>NHIEGateway</name>
<!-- REQUIRED: HIE's website home page url -->
<discoveryURLs>
    <discoveryURL useType="homepage">
        http://nhie.fedsconnect.org
    </discoveryURL>
  </discoveryURLs>
<!-- REQUIRED: HIE's website home page url -->
  <description>Health Information Exchange for the Federal Agency</description>
<!-- REQUIRED: HIE Contact details. Atleast ONE contact element is required with
personName, phone, email & address elements defined -->
  <contacts>
    <contact>
      <personName>Manager</personName>
      <phone>111-111-1111</phone>
      <email>manager@fedsconnect.org </email>
      <address>
        <addressLine>112 W. Main Street</addressLine>
        <addressLine>Kingsport, TN 37662</addressLine>
      </address>
    </contact>
  </contacts>
```

```
<!-- REQUIRED: The custom taxonomies defined for the business entity -->
  <identifierBag>
    <keyedReference
              tModelKey="uddi:nhin:nhie:homecommunityid"
              keyValue="urn:oid:2.16.840.1.113883.3.166.4"/>
  </identifierBag>
<!-- REQUIRED: The custom taxonomies defined for the business entity -->
  <categoryBag>
<!-- Only ONE reference to the uddi:nhin:nhie:state is mandatory. There could be multiple
references -->
    <keyedReference
              tModelKey="uddi:nhin:nhie:state"
              keyValue="TN"/>
       <keyedReference
              tModelKey="uddi:nhin:nhie:state"
       keyValue="VA"/>
       <keyedReference
              tModelKey="uddi:nhin:nhie:federalhie"
              keyValue="YES"/>
              <keyedReference
              tModelKey="uddi:nhin:nhie:publickeyuri"
              keyValue=" https://www.NhinGovernanceServer/pki/id48.cer"/>

  </categoryBag>
  <businessServices>
    <businessService
              businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
              serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 ">
              <!-- REQUIRED: Brief name of the business service -->
       <name>Audit Log Query Service</name>
<!-- REQUIRED: Brief description about the service -->
      <description>Service to retrieve audit events for a given date range</description>
<!-- REQUIRED: The custom taxonomies defined for the business service -->
      <categoryBag>
        <keyedReference
              tModelKey="uddi:nhin:uniformservicename"
              keyValue="AuditLogService"/>
        <keyedReference
              tModelKey="uddi:nhin:versionofservice"
              keyValue="1.0"/>
      </categoryBag>
      <bindingTemplates>
        <bindingTemplate
              serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 "
              bindingKey=" uddi:nhiedomain.com:b6c2a9f8-a88b-48bc-98a3-b160dabc4232 ">
              <!-- REQUIRED: Brief about the service's technical information -->
          <description>AuditLogQuery service implemented as a web-service with soap over
http</description>
<!-- REQUIRED: location of the WSDL -->
          <accessPoint useType="endPoint">
              http:// nhie.fedsconnect.org:18181/findAuditEvents/AuditLogQuery
           </accessPoint>
        <!-- REQUIRED: tModels for service WSDL and transport -->
        <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="uddi:nhin:auditlogquery_interface"/>
        </tModelInstanceDetails>
            </bindingTemplate>
      </bindingTemplates>
      </businessService>
      <businessService
              businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
              serviceKey=" uddi:nhiedomain.com:242c7a13-7914-4284-9212-9e551849f7fd ">
              <!-- REQUIRED: Brief name of the business service -->
        <name>Subject Discovery Service</name>
<!-- REQUIRED: Brief description about the service -->
        <description>Service to perform subject discovery over NHIN</description>
<!-- REQUIRED: The custom taxonomies defined for the business service -->
        <categoryBag>
          <keyedReference
              tModelKey="uddi:nhin:uniformservicename"
```

```
              keyValue=" SubjectDiscoveryService "/>
          <keyedReference
              tModelKey="uddi:nhin:versionofservice"
              keyValue="2.0"/>
         </categoryBag>
         <bindingTemplates>
           <bindingTemplate
              serviceKey=" uddi:nhiedomain.com:242c7a13-7914-4284-9212-9e551849f7fd "
              bindingKey=" uddi:nhiedomain.com:4a3c042d-9687-484e-8857-c6d994bfed60 ">
              <!-- REQUIRED: Brief about the service's technical information -->
            <description>Subject discovery service implemented as a web-service with soap
over http</description>
<!-- REQUIRED: location of the WSDL -->
            <accessPoint useType="endPoint">
                http:// nhie.fedsconnect.org:18181/PIXConsumer_Soap11
            </accessPoint>
<!-- REQUIRED: tModels for service WSDL and transport -->
<tModelInstanceDetails>
<tModelInstanceInfo tModelKey="uddi:nhin:subjectdiscovery_interface"/>
      </tModelInstanceDetails>
             </bindingTemplate>
       </bindingTemplates>
     </businessService>
  </businessServices>
</businessEntity>
```

1) a) If the requester knows the name of the business then a call to the ***find_business*** API is made which returns business details. This business details contains *businessKeys*, service keys implemented by the business entities. This scenario is normally used when the NHIE wants to find for a particular business entity.

   Fig2 below shows the query and the result of *find_business* API call.

FIG 2:

Query 1

```
<find_business>
  <name> NHIEGateway </name>
</find_business>
```

Result 1

```
<businessList>
  <businessInfos>
    <businessInfo
       businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 ">
      <name> NHIEGateway </name>
      <description>
       Health Information Exchange for the Federal Agency
      </description>
      <serviceInfos>
        <serviceInfo
              businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
              serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 ">
          <name> Audit Log Query Service </name>
        </serviceInfo>
        <serviceInfo
              businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11"
              serviceKey=" uddi:nhiedomain.com:242c7a13-7914-4284-9212-9e551849f7fd ">
          <name> Subject Discovery Service </name>
        </serviceInfo>
      </serviceInfos>
    </businessInfo>
  <businessInfos>
```

```
</businessList>
```

b) If the requester knows the service name then "**find_service**" API is called which returns all the services which match the criteria from multiple business entities. The sample below shows how it could be implemented.

Query 2

```
<find_service>
  <name> Audit Log Query Service </name>
  <categoryBag>
            <keyedReference
                  tModelKey="uddi:nhin:uniformservicename"
                  keyValue="AuditLogService"/>
        <keyedReference
                  tModelKey="uddi:nhin:versionofservice"
                  keyValue="1.0"/>
  </categoryBag>
</find_service>
```

Result 2

```
<serviceList>
  <serviceInfos>
    <serviceInfo
    businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
           serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 ">
      <name> Audit Log Query Service </name>
    </serviceInfo>
  </serviceInfos>
</serviceList>
```

2) From both a and b above we can get the service key. Then using the service key a call to ***get_serviceDetail*** API. This API call returns a *serviceDetail* on successful match of the specified *serviceKey* values. The service details contain business services. Each business services contain multiple binding templates. A sample is provided below.

Query 3

```
<get_serviceDetail>
  <serviceKey> uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 </serviceKey>
</get_serviceDetail>
```

Result 3

```
<serviceDetail>
  <businessServices>

    <businessService
              businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
              serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 ">
              <!-- REQUIRED: Brief name of the business service -->
      <name>Audit Log Query Service</name>
<!-- REQUIRED: Brief description about the service -->
      <description>Service to retrieve audit events for a given date range</description>
<!-- REQUIRED: The custom taxonomies defined for the business service -->
      <categoryBag>
        <keyedReference
              tModelKey="uddi:nhin:uniformservicename"
              keyValue="AuditLogService"/>
        <keyedReference
              tModelKey="uddi:nhin:versionofservice"
```

```
               keyValue="1.0"/>
      </categoryBag>
      <bindingTemplates>
        <bindingTemplate
               serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 "
               bindingKey=" uddi:nhiedomain.com:b6c2a9f8-a88b-48bc-98a3-b160dabc4232 ">
               <!-- REQUIRED: Brief about the service's technical information -->
          <description>AuditLogQuery service implemented as a web-service with soap over
http</description>
<!-- REQUIRED: location of the WSDL -->
          <accessPoint useType="endPoint">
               http:// nhie.fedsconnect.org:18181/findAuditEvents/AuditLogQuery
           </accessPoint>
        <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="uddi:nhin:auditlogquery_interface"/>
        </tModelInstanceDetails>
           </bindingTemplate>
      </bindingTemplates>
      </businessService>
</businessServices>
</serviceDetail>
```

3) Each binding template contains an attribute called "**access point**" whose value we will give the URL to the service endpoint. This end point is needed to invoke the service.
4) If we need to get to the WSDL we need to invoke the *get_bindingDetail* API and pass the binding key which can obtained from the *get_ServiceDetails* API.
5) If the public key is needed from the business entity the *get_businessDetail* API can be made with the *businessKey* as the parameter. The business key can be found in response from the *find_business* API.

Query 5

```
<get_businessDetail>
  <businessKey> uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 </businessKey>
</get_businessDetail>
```

Result 5

```
<businessDetail>
  <businessEntity businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11>
<!-- REQUIRED: Name of the HIE -->
<name>NHIEGateway</name>
<!-- REQUIRED: HIE's website home page url -->
<discoveryURLs>
    <discoveryURL useType="homepage">
       http://nhie.fedsconnect.org
    </discoveryURL>
  </discoveryURLs>
<!-- REQUIRED: HIE's website home page url -->
  <description>Health Information Exchange for the Federal Agency</description>
<!-- REQUIRED: HIE Contact details. Atleast ONE contact element is required with personName,
phone, email & address elements defined -->
  <contacts>
    <contact>
      <personName>Manager</personName>
      <phone>111-111-1111</phone>
      <email>manager@fedsconnect.org </email>
      <address>
        <addressLine>112 W. Main Street</addressLine>
        <addressLine>Kingsport, TN 37662</addressLine>
      </address>
    </contact>
  </contacts>
<!-- REQUIRED: The custom taxonomies defined for the business entity -->
  <identifierBag>
    <keyedReference
               tModelKey="uddi:nhin:nhie:homecommunityid"
               keyValue="urn:oid:2.16.840.1.113883.3.166.4"/>
  </identifierBag>
<!-- REQUIRED: The custom taxonomies defined for the business entity -->
  <categoryBag>
<!-- Only ONE reference to the uddi:nhin:nhie:state is mandatory. There could be multiple
references -->
    <keyedReference
               tModelKey="uddi:nhin:nhie:state"
               keyValue="TN"/>
       <keyedReference
               tModelKey="uddi:nhin:nhie:state"
       keyValue="VA"/>
       <keyedReference
               tModelKey="uddi:nhin:nhie:federalhie"
               keyValue="YES"/>
               <keyedReference
               tModelKey="uddi:nhin:nhie:publickeyuri"
               keyValue=" https://www.NhinGovernanceServer/pki/id48.cer"/>

  </categoryBag>
  <businessServices>
    <businessService
               businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
               serviceKey=" uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546 ">
</businessService>
<businessService
               businessKey=" uddi:nhiedomain.com:155463c5-b027-41a7-a7c9-b99f0b0bcb11 "
               serviceKey=" uddi:nhiedomain.com:242c7a13-7914-4284-9212-9e551849f7fd ">
</businessService>

</businessEntity>
</businessDetail>
```

NOTE: The NHIN UDDI registry will not store the service WSDL but would have reference to the location where the WSDL is stored.

There are typically 2 ways to invoke these API's.

a) Using the Programming API. One should have the client API files to support this. The Java implementation can be downloaded from here: http://sourceforge.net/project/showfiles.php?group_id=220485&package_id=266239

b) Using UDDI browsers – these browsers provide a UI to interact with UDDI registry nodes.

List of Open source Projects which are UDDI V3 compliant

- JOrbit http://sourceforge.net/projects/jorbit
- SCOUD http://sourceforge.net/projects/scoud
- OpenUDDI – http://openuddi.sourceforge.net/
- ruddi http://www.ruddi.org/ (Note: The license is free only if you want to contribute back.)
- Novell NSure Browser – compatible with OpenUDDI (http://developer.novell.com/wiki/index.php/Special:Downloads/uddiserver/Novell~~~Nsure~~~UDDI~~~Server/1.1.3/)

6.3 Subscription API

Subscription provides clients, known as subscribers, with the ability to register their interest in receiving information concerning changes made in a UDDI registry.

The *tModel*(s) for the subscription are:

- uddi-org:subscription_v3
- uddiorg:subscriptionListener_v3

More details regarding *tModels* are described in 11.2.10 of the UDDI specification.
The security mechanisms including integrity and confidentiality are implemented as described in the section 5.4 of this specification. Subscribers referred to in this document refer to NHIE gateways. NHIE Gateways subscribe to UDDI node service registry.

In general, according to the UDDI v3 specification subscriptions can be made any changes to entities. But the scope of this specification only pertains to changes in *businessEntity*, *businessService*, *bindingTemplate*, and *tModel*. These entities are considered to be changed if the modified *IncludingChildren* elements of the *operationalInfo* element of the entities have been changed.

Subscription allows subscribers to "monitor" a particular subset of data within a registry. Two patterns are defined. UDDI nodes MAY support either or both:

- **Asynchronous notification** – subscribers choose to be asynchronously notified by the node when registry data of interest changes via calls to the *notify_subscriptionListener* API, which they implement as a "subscription listener" service.
- **Synchronous change tracking** – subscribers issue a synchronous request using the *get_subscriptionResults* API to obtain information on activity in the registry which matches their subscription preferences.

In case of an NHIE, it is recommended to use asynchronous notification from a performance point of view.

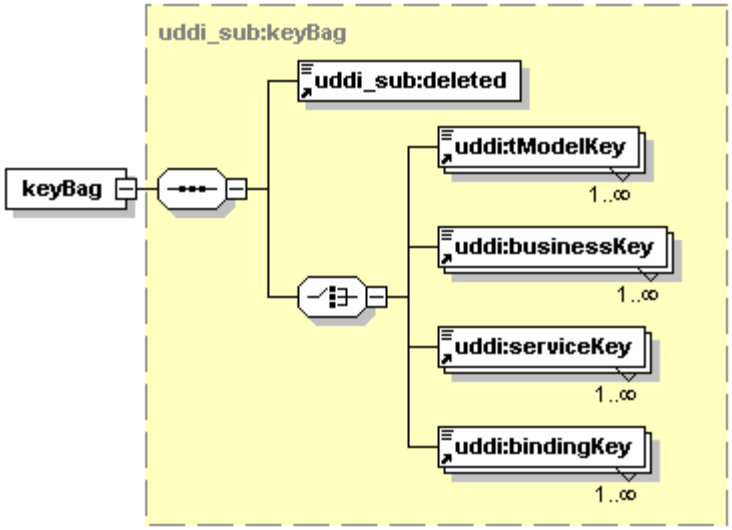
Any of the existing standard inquiry APIs (find_xx and get_xx) defined in UDDI v3 specification, may be used within a subscription request to define the criteria. Please refer to the Inquiry API provided in section 5.2.

The duration, or life of a subscription is also a matter of node policy, but subscribers can renew existing subscriptions periodically instead of having to create new ones. Subscribers may also create multiple subscriptions. Each subscription request is treated independently. The level of detail provided for the data returned is controlled by the subscription request.

NHIE gateways will implement an HTTP/SOAP based web services as subscription listeners. The UDDI registry will send a notification to this web service endpoint.

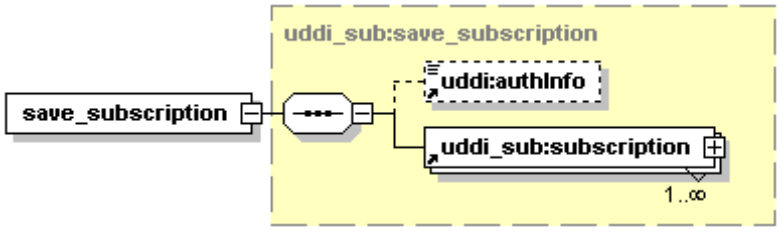
An NHIE will use *keyBag* while returning the results from the subscription. A *keyBag* contains a list of entity keys, which correspond to any of the core data structures (*businessEntity*, *businessService*, *bindingTemplate* or *tModel*). The *keyBag* has two uses:

- Returning results when a "brief" format is selected, this minimizes returned information.
- Indicating entities which have been deleted, or which no longer match the subscription criteria provided with the subscription.



In general there are 5 subscription API's provided by the UDDI v3 specification. But the subscription API's implemented by NHIE is a subset of those and are as follows:

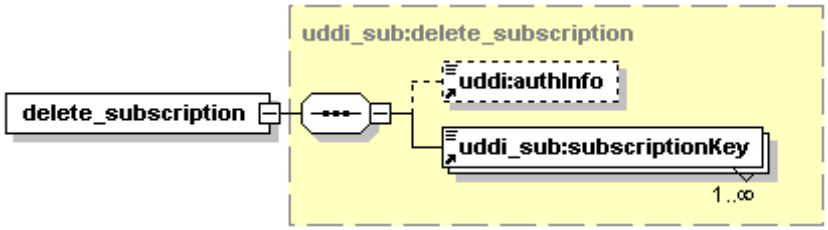
1) **save_subscription:** Establishes a new subscription or changes an existing one. Also used to renew existing subscriptions. The attribute "Brief" will be set to true in order to get the return in *keyBag*.



The subscription contains data which is automatically generated by the implementation. It also contains subscription filter which need to be filled .It is recommended to specify only *uddi:get_serviceDetail* as the subscription filter.

See section 5.5.8.1 of the UDDI v3 specification for more details.

2) **delete_subscription:** Cancels one or more specified subscriptions



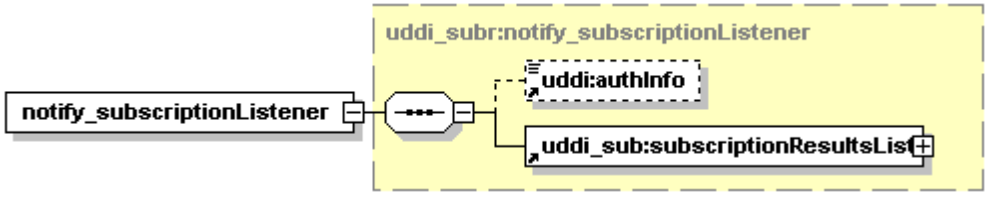
See section 5.5.9.1 of the UDDI v3 specification for more details.

3) **notify_subscriptionListener**: This API is invoked by UDDI registry. The subscriber implements as a subscription listener service to accept notifications containing the data that changed since *notify_subscriptionListener* was last invoked for a particular subscription.

This is an optional API from a UDDI perspective but it will be useful for our implementation. This API allows us to implement asynchronous notification. It enables the node to deliver notifications to subscription listeners by invoking a Web service.

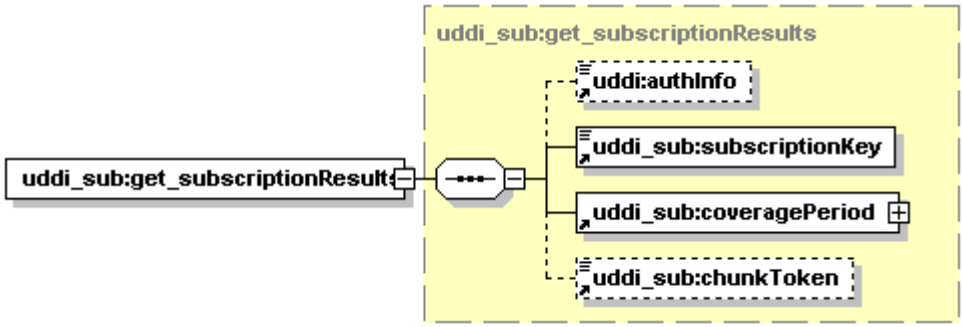
Upon successful completion, *notify_subscriptionListener* returns an empty message. Note that this is being returned by the NHIE gateway. The “brief” attribute value will be set to true in order to get result as a *keyBag*.

To allow subscribers to determine whether a notification has been lost, the coverage period of the notification is included. A date/time indicating the date/time values corresponding to the start and end points of this is provided. The start date/time used in this call SHOULD align with the end date/time of the previous call and so fourth.



See section 5.5.12.1 of the UDDI v3 specification for more details.

3) **get_subscriptionResults:** This API allows the client to call synchronously to the UDDI node. The syntax of sending this request is as described below.



The response is a *subscriptionList*. The sample 3 and 4 described how the request is sent and response is received.

Please look at section 5.5.11 of the UDDI specification for more details.

6.3.1 Steps for subscription and notification configuration with a sample scenario

The sample scenario shows the sequence of steps which take place in subscription. Refer to the Inquiry API sample for some keys defined here.

1) Setting up the Notification web service.
This is done by registering a binding template. To do this we need to register first the business entity and a business service for that entity and then the binding template for that business service. This can be done either using programming API or using some kind of UI which supports the publication API's as specified in the UDDI specification (section 5.2). The important thing is that we need the binding key when we create the subscription in the next step. Usually the keys are generated by the underlying software which implements the UDDI specification.

The following is a typical *bindingTemplate* for a Web Service that implements the UDDI Version 3 Subscription Listener API. This Web service is registered by a subscriber of asynchronous updates to UDDI entities:

```
<save_binding xmlns="urn:uddi-org:api_v3">
         <bindingTemplates>
           <bindingTemplate serviceKey="uddi:nhiedomain.com:681dba36-b47c-
454d-808b-042978f6d7ce" bindingKey="uddi:nhiedomain.com:2a45f809-b313-4b47-
a215-665c40533149">
              <description>UDDI Subscription Listener API V3</description>
              <accessPoint
URLType="http">http://localhost:18181/subscriptionListener</accessPoint>
           <tModelInstanceDetails>
                 <tModelInstanceInfo
tModelKey="uddi:uddi.org:v3_subscriptionlistener" />
           </tModelInstanceDetails>
           </bindingTemplate>
        </bindingTemplates>
     </save_binding>
```

2) Create subscription to get service details.
Again this can be done either using programming API or using some kind of UI which supports the subscription API's as described above.

```
<save_subscription xmlns="urn:uddi-org:sub_v3">
<subscriptions>
<subscription brief="true" xmlns="urn:uddi-org:sub_v3">
    <subscriptionKey>uddi:4f0d7450-a578-11d8-91cd-
5c1d367091cd</subscriptionKey>
    <subscriptionFilter>
        <serviceDetail xmlns="urn:uddi-org:api_v3">
            <servicekey> uddi:nhiedomain.com:a4dc6e0f-f3a6-4dee-a6ac-
c274e4cc6d87</servicekey>
     </subscriptionFilter>
    <maxEntities>10</maxEntities>
    <expiresAfter></expiresAfter>
</subscription>
</subscriptions>
</save_subscription>
```

3) Get notification for subscriptions

This API is invoked by the UDDI node which implements the UDDI specification.
The sample below shows when the “brief” is set to true.

```
<notify_subscriptionListener>
<subscriptionResultsList xmlns="urn:uddi-org:sub_v3">
    <chunkToken>0</chunkToken>
    <coveragePeriod>
        < startPoint>2008-09-01T00:00:00.000</startPoint>
        < endPoint>2008-10-01T00:00:00.000</endPoint>
    </coveragePeriod>
    < subscription brief="true">
        < subscriptionFilter>
          <categoryBag>
        <keyedReference
            tModelKey="uddi:nhin:uniformservicename"
            keyValue="AuditLogService"/>
        <keyedReference
            tModelKey="uddi:nhin:versionofservice"
            keyValue="1.0"/>
      </categoryBag>
        </subscriptionFilter>
        </ maxEntities>
        </expiresAfter>
    </subscription>
    <keyBag>
<deleted>false</deleted>
<serviceKey> uddi:nhiedomain.com:93afd501-12d5-4023-b733-f3036a10a546
</serviceKey>
</keyBag>
</subscriptionResultsList>
</notify_subscriptionListener>
```

4) Get changed subscriptions

This API is called by the client to the UDDI node. The response to this request is very much similar the sample described above (3) except that it is without the "**notify_subscriptionListener**" tags.The request is generally made using the programming API and is a part of the client side API.

```
<get_subscriptionResults>
<subscriptionKey> uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd
</subscriptionKey>
<coveragePeriod>
        < startPoint>2008-09-01T00:00:00.000</startPoint>
        < endPoint>2008-10-01T00:00:00.000</endPoint>
    </coveragePeriod>
</get_subscriptionResults>
```

5) Delete subscription
Again this can be done either using programming API or using some kind of UI which supports the subscription API's as described above.

```
<delete_subscription>
      <subscriptionKey>uddi:4f0d7450-a578-11d8-91cd-
5c1d367091cd</subscriptionKey>
</delete_subscription>
```

Some more examples are found at C.2.2 section of the UDDI v3 specification.

6.4 NHIE Service Registry Security Model:

The basis of authentication and non-repudiation for all NHIE Service Registry participants shall be X.509 digital certificates, which are commonly deployed in conjunction with a Public Key Infrastructure (PKI) and are also associated with Transport Layer Security (TLS). (The use of TLS in the NHIN is further described in the Messaging Platform Specification). A combination of the above technologies within the NHIN works to ensure that only reliably authenticated NHIE participants can communicate within the NHIN, and to maintain data confidentiality and integrity while data is in transit.

For the December 2008 Trial Implementation demonstrations, a single Certificate Authority (Trial CA) will be utilized for the purpose of issuing the necessary X.509 certificates to all NHIEs, which shall be used to authenticate communication with the Service Registry. This Trial CA will therefore serve as the "trust root" or "trust anchor" of a limited PKI during the authentication process described below as well as digital signature verification.

In order to reduce the complexity of this demonstration, traditional PKI functionality such as chaining of certificate authorities, bridges, revocation list processes and registration authorities will not be implemented. However, for scalability and security reasons, true PKI functionality will clearly need to be implemented after the Trial Implementation period. So, for 2009 and beyond, a master NHIN Certificate Authority and PKI infrastructure as well as an identity proofing and credentialing provider should be deployed and maintained by the future NHIN governance body.

Basis of Trust and Access Control:
Given that the common Trial CA issues and signs all Trial Implementation X.509 certificates, these certificates will serve as the basis of trust and authentication between NHIEs and shall be used to control access to the service registry. All NHIE to Service Registry communication must be authenticated and digitally signed via these certificates to ensure only authorized and properly authenticated NHIEs are allowed to communicate with the Service Registry. Access to the discrete NHIE registry entries (WSDL

endpoints and other sensitive information) shall be restricted to other accredited NHIEs who themselves posses valid Trial Implementation X.509 certificates. The public key of the NHIE X.509 certificate will also be included in the registry if technically feasible.

By leveraging even limited PKI and CA functionality for the Trial Implementation, NHIE participants can engage in secure transactions and will have reasonable assurance that:

1. Both the sender and recipient of NHIN data will be identified uniquely so the parties know where the information is flowing from and to (providing identification and authentication);
2. Data transmissions were not altered deliberately or inadvertently (establishing data integrity);
3. Both sender and recipient identity is inextricably tied to the data transmitted (technical non-repudiation);
4. NHIN data is protected from unauthorized access (guaranteeing confidentiality or privacy).

Client and Server Authentication:
When X.509 certificates are utilized to provide mutual client and server authentication, the Trial PKI infrastructure described above and TLS functionality outlined in the Messaging Platform are combined to ensure strong authentication of the communicating parties. For example, when an NHIE client needs to be authenticated to a particular server (or service), a valid certification path to a trust root, (in this case, the Trial CA), is required in order for the X.509 certificate presented to the server or client to be trusted and authentication performed.

As detailed in NIST *SP800-52*, clients play an important role in the overall security posture of this implementation, especially with regards to authentication. The protocol version and cipher suite (see Messaging Platform Specification for required TI standards) are negotiated by the client with the server, and are presented in the “ClientHello” message to the server. This message forms the foundation for the server’s negotiation of the strongest possible security in the communication, and is the first message to be sent as the client establishes a TLS connection to the server. In this way, the client can remain connected, re-establish a preexisting connection or establish additional secure sessions with the server without having to repeat the entire handshake process.

The handshake process also accomplishes client authentication in the following manner. The server requests the client’s X.509 certificate and negotiates the necessary communication parameters. The client responds with its certificate combined with a signed hash of the initial handshake to establish proof of possession to the server of the corresponding key. However, in the event that the client does not have a valid X.509 certificate, (i.e. one not issued by the Trial CA), the server shall terminate the connection and issue a “handshake failure” alert, as trust cannot be established between the client and the server.

Servers shall also use their Trial CA issued X.509 certificate and corresponding key to authenticate themselves to NHIE clients. Server authentication is performed via the TLS protocol and therefore provides assurance to the client that the server has been authorized to communicate within the NHIN (since the server’s network address matches the address detailed in the valid Trial CA issued certificate), that the server is in control of the corresponding public and private keys, and is therefore in actuality the server it claims to be. Server authentication follows a handshake process similar to that of client authentication with some minor differences. The server generates a session id that is sent to the client in a “ServerHello” message. (This server-generated session id and related key and cipher suite material is stored for later use to resume sessions with the client).

In summary, by requiring the NHIN clients and servers to mutually authenticate one another, (i.e. both server and client authentication are performed) all data is protected via TLS and is also tied to the keys established during the authentication process. As a result, the overall security of all NHIN service registry communication is reasonably assured since public key technology and X.509 digital certificates (which bind the identity of a the client or server to its public key) are used to assure strong authentication, reliable encryption, nonrepudiation, and data integrity.

References:

RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
http://www.ietf.org/rfc/rfc2459.txt
WS-I Security Profile 1.1, X.509 Token Profile 1.0
http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html
Minimum Interoperability Specification for PKI Components
http://csrc.nist.gov/publications/nistpubs/800-15/SP800-15.PDF
Federal Agency Use of Public Key Technology for Digital Signatures and Authentication
http://csrc.nist.gov/publications/nistpubs/800-25/sp800-25.pdf
Introduction to Public Key Technology and the Federal PKI Infrastructure
http://csrc.nist.gov/publications/nistpubs/800-32/sp800-32.pdf
Guidelines for the Selection and Use of Transport Layer Security (TLS) Implementations
http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf
The X.509 certificate standard [ISO94-8]
http://www.ietf.org/html.charters/pkix-charter.html

6.5 UDDI Replication

This part of the specification describes the inter node replication between two or more nodes of the NHIN UDDI service registry. The replication is a server to server communication with no client calls.

6.5.1 Replication Concepts

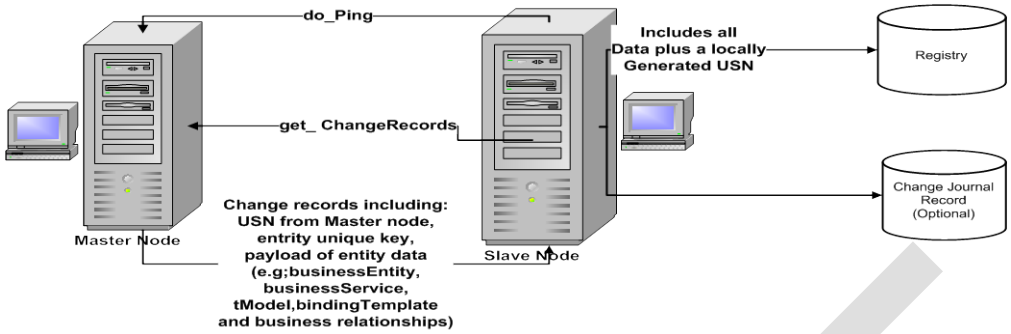
The goal of replication is to facilitate the establishment and maintenance of a single consistent shared set of registry data. Per the UDDI specification, there are two possible options to configure replication:

a) **Push like model** – In this model there is a replication communication graph node at each node which has information of all of its neighbors. When that node receives a change it notifies its neighbors about the change using the "**notify_ChangeRecordsAvailable**". Then the receiving nodes who want to get the changes will call the API "**get_ChangeRecords**" to get the records. This is new to UDDI v3.
b) **Pull like model** – In this model works on the principle of Master and Slave nodes. All updates are made to the Master node and the slave nodes get updated via replication. The slave node, on regular intervals, invokes call to the Master node and acquires the changed data.

For NHIN service registry, based on the requirements, the Pull model would be implemented. For the demo purposes, the federal agencies would host the master node and a slave node while CareSpark would implement another slave node. The NHIN governance body would facilitate the registration of businesses to the NHIN service registry. The governance body would maintain the Master node and assist with setup of replication and Slave nodes.The assumptions is that the HIE's will be sending the updates to the governance body to update the master node. The governance body will be maintaining the master node and the slave node.

The WSDL which defines the replication API are in "**uddi_repl_v3_portType.wsdl**". The binding for this WSDL is defined in "**uddi_repl_v3_binding.wsdl**". These WSDL's and related schemas are in the zip file embedded in this document. They are in Appendix B of this specification.

6.5.1.1 Replication Process



Following are the steps which occur during replication process:

- **Step 1** – The slave node does a "***do_ping***" to the master nodes to see if it is available.
- **Step 2**- If the master node is available then it makes a call to the master node with the
- "***get_ChangeRecords***" API call.
- **Step 3** - Then the master node will send the changed records, which includes USN of the master node, entity unique key, payload of entity data (e.g. *businessEntity*, *businessServices*, *tModel*, *bindingTemplates* and *businessRelationships*).

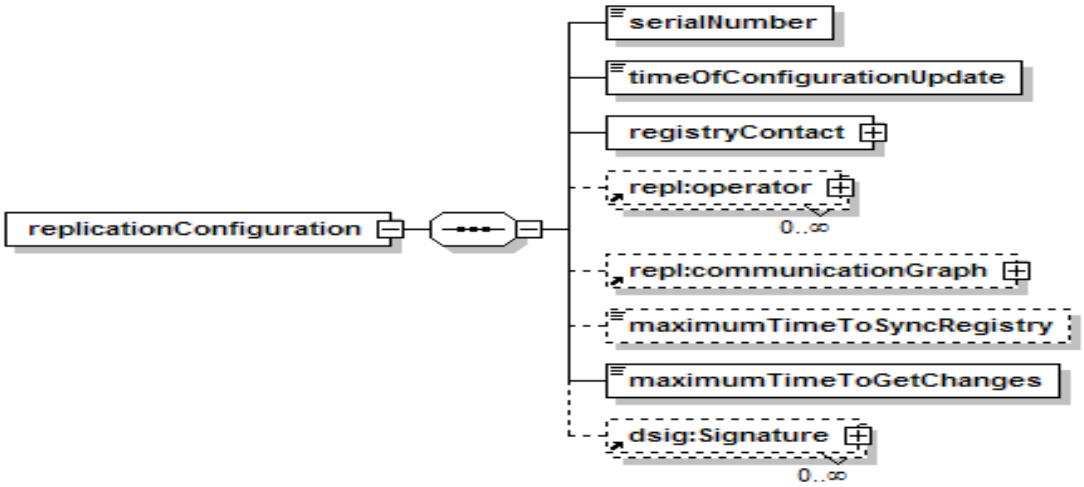
USN – This is called updated sequence number. This is a strictly increasing sequence number maintain by registries at each node. An originating USN is *assigned to a change record at its creation. No change record may have a USN equal to 0.UDDI* nodes must not rely on an originating sequence increasing monotonically by a value of "1".

The node which is updating the change records should maintain a "**Change record Journal**" as a part of their internal implementation. This journal maintains the before and after state of the changed records at the receiving node. It is recommended to have a change record Journal based on the implementation selected.

The UDDI implementation should be able to add node and remove node using replication. Refer to sections 7.8 and 7.9 of the UDDI specification.

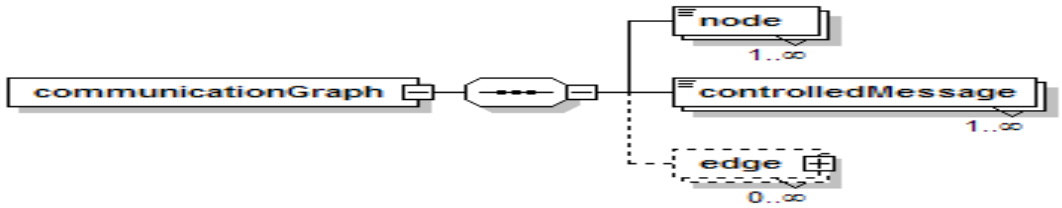
6.5.1.2 UDDI Replication Configuration

In case of NHIE the replication of UDDI data within a registry should be governed by information maintained within a Replication Configuration Structure. The structure includes sufficient information to uniquely identify each node within the UDDI registry. Each current UDDI node within the Registry is identified with an operator element in the *replicationConfiguration*



More detail on Replication Configuration Structure is described at 7.5.1 and 7.5.2 sections of the specification.

Replication *communicationGraph* is used to administer and control the replication configuration structure. The replication configuration should be stored in a format easily accessible for administration. This may be a in a file, for example.



More detail on the Replication *communicationGraph* is described at 7.5.3 section of the UDDI specification.

Replication configuration consists of two parts:

1) **Master Node configuration** – This involves creation of two subscriptions; one with filter as *find_business* and the other with filter as *find_tModel*. These subscriptions are set for receiving changed subscriptions as *KeyBag*.

   Refer to the section 5.5 of the UDDI specification for more details on how to create subscription.

2) **Slave Node configurations** –This involves registration of subscriptions created in step 1). The Master node governance body would distribute the subscription keys (generated in step 1) to the Slave node administrators. The parameters used to configure this node are
   i. Period = 60
      1. Period is how often the master is polled for changes, in seconds.
   ii. Name = find *tModels*
   iii. Username = valid username on master server node
   iv. Password = valid password for user above
   v. subscriptionKey = subscription key for tModel or business subscription that was created in step 1 above.
   vi. inquiryURL = URL for inquiry service on master server node
   vii. subscriptionURL = URL for subscription service on master server node

viii. securityURL = URL for security service on master server node
ix. disabled = false

6.5.1.3 Security configuration

The communication in between the nodes for replication will use SSL as transport security and public keys as defined in the security section of this specification. SSL 3.0 with mutual authentication is represented by the *tModel* ***uddiorg:mutualAuthenticatedSSL3*** as described within Section 11.3.2 Secure Sockets Layer Version 3 with Mutual Authentication. More details on the security are described on section 5.4 of this specification.

6.5.1.4 UDDI Replication API

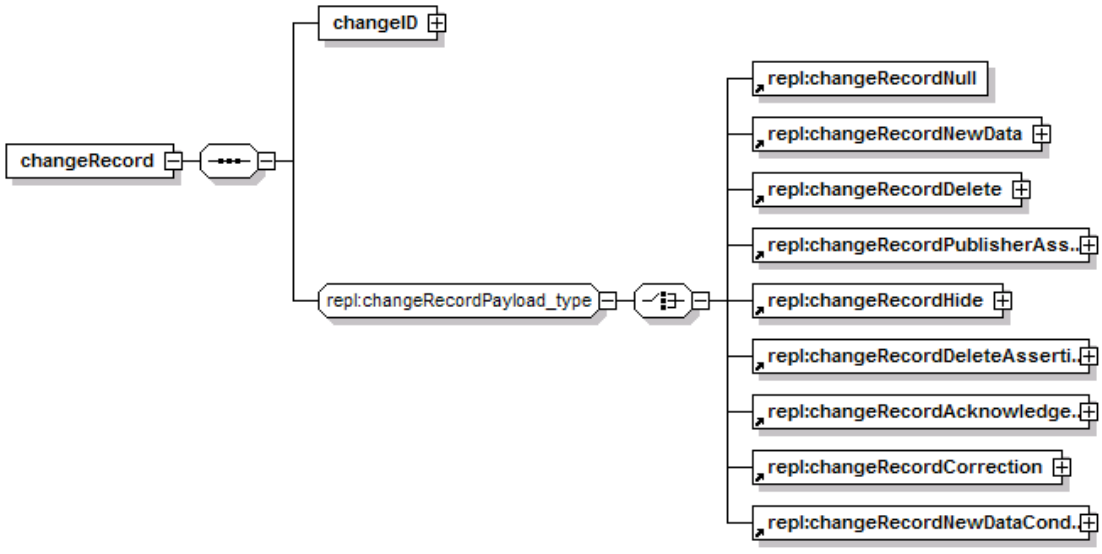
UDDI Replication defines four APIs. From a NHIN perspective we will be using only the 2 API's mentioned below.

- ***get_changeRecords*** - This message is used to initiate the replication of change records from one node to another. The invoking node, wishing to receive new change records, provides as part of the message a high water mark vector. This is used by the replication source node to determine what change records satisfy the caller's request.

- ***do_ping*** - This UDDI API message provides the means by which the current existence and replication readiness of a node may be obtained.

More information on the API is available in section 7.4 of the UDDI specification

FIG 1.**Changed Record Structures –** This is the structure which is returned back to the slave node. Each of the change record contains the details of the changes which will be updated on the slave node.



Each change record contains a *changeID* that identifies the node on which the change originated and the originating USN of the change within that node. It then contains one of several allowed forms of change indication.

More details regarding the change indications are described in sections 7.3.1 to 7.3.10 in the UDDI specification.

In UDDI node-to-node replication communication MUST be carried out by means of SOAP messages and responses. The *soapReplicationURL* element of the operator element indicates where such messages should be sent to communicate with a given node. Refer to section 7.5.2 for more detail on *soapReplicationURL*. The replication API should use SOAP 1.1 to make replication calls.

6.5.2 *tModel* for UDDI replication

The *tModel* is defined in the section 11.2.4 of the UDDI specification. More examples are available in **Appendix J** of the UDDI specification.

7 Other Standards

None.

8 Mapping to HITSP and IHE Technical Standards

HITSP and IHE have not addressed the subject of service registries.

9 Technical Pre-conditions

This specification assumes a centralized governing agency which, prior to any sharing of patient data, validates an NHIE. The governing agency is expected to define a process whereby a NHIE can apply to share data through the NHIN interfaces, the NHIE is verified as a qualified organization and the NHIE has been given a certificate whose root includes a root from the governing agency. Thus certificates which bear the certification from this governing agency prove that the providing organization has met the standards required to sharing data on the NHIN. The standards enforced by the governing agency are expected to include, but are not limited to:

- Ensuring the organization is a valid organization and its sole intention is in line with the purposes of the NHIN
- Ensuring the organization has appropriate policies and governance agreements to protect the data it is given
- Ensuring the organization is able to reliably and securely provide the minimum NHIN services

Only through this bootstrapping process provided by the governing agency can a NHIE have its information saved in the Service Registry. Thus the users of the Service Registry can be assured that organizations listed within it have satisfied the requirements and are generally safe to share data with.

In addition to verification of new NHIE's, the governing agency will need to also provide a repository for saving common WSDL's and public key certificates. This is further addressed in section 10.1, below.

10 Technical Post-conditions

None

11 Future Directions

11.1 Moving from a registry to a repository

The use of UDDI is a component of compliance with the WS-I Basic Profile Version 1.1, which has been adopted by the NHIN as an underlying profile for the NHIN *Messaging Platform* specification. More specifically, the OASIS WS-I WS-BasicProfile V 1.1 Final (http://www.ws-i.org/Profiles/BasicProfile-1.1.html) states:

- In Section 2.4 (Claiming Conformance) UDDI seems to be required, although the standard falls short of actually stating UDDI is a requirement.
- Section 4.1 (Required Description) requires either a WSDL 1.1 description or a UDDI binding template for end points.
- Section 5.0 (Service Publication and Discovery) states:
  "When publication or discovery of Web services is required, UDDI is the mechanism the Profile has adopted to describe Web service providers and the Web services they provide. ... Registration of Web service instances in UDDI registries is optional. By no means do all usage scenarios require the kind of metadata and discovery UDDI provides, but where such capability is needed, UDDI is the sanctioned mechanism."
- The NHIN Implementation Work Group also attempted to anticipate the future direction of WS-BasicProfile by reviewing the WS-BasicProfile Draft 2.0. Analysis of this draft standard reveals the same UDDI references in 2.0 as in 1.1. This cannot be considered proof that WS-BasicProfile 2.0 will support UDDI, but does seem to indicate that direction is a likely outcome.
- No other registry specifications are mentioned by name in the WS-BasicProfile documents 1.1 Final or 2.0 Draft other than UDDI.

However, there are some inherent limitations in the use of UDDI for the NHIN services registry component. Because UDDI is a registry rather than a repository, its primary purpose is to maintain pointers to artifacts, not copies of the artifacts themselves. This limits the ability of the service registry to store and thus govern the artifacts, which is a desirable goal.

An example of this limit in practice occurs with respect to WSDL files, and X.509 certificates. It may be desirable to store both in the services registry but there is no currently known mechanism for accomplishing such. As a result, UDDI can only store a link to these artifacts (in the form of a Uniform Resource Identifier (URI)). Beyond governance, support for ontologies, object relationship representations, role-based access control, federated identity management and extensible metadata may all be desirable. Alternatives such as ebRS or other technologies may be worth considering beyond the 2008 NHIN Trial Implementations period if they can achieve broad industry support and integration with the WS-I Basic Profile.

11.2 10.2 Entity directory services

It may be desirable in the future to create directories of entities (such providers, labs, pharmacies, etc.) that participate within a given NHIE. For example, a patient may have knowledge of the provider who holds their records but not the details of which NHIE that provider participates in. A mapping from provider to NHIE would be useful for finding the records for this patient. At this time this use case is declared out of scope as it requires some detailed discussion regarding its appropriateness, security and privacy of providers and what the best approach to implementation might be.