

nehta

Service Instance Locator

Implementation Guide

Version 1.1 — 1 December 2008

Release

National E-Health Transition Authority Ltd

Level 25

56 Pitt Street

Sydney, NSW, 2000

Australia.

www.nehta.gov.au**Disclaimer**

NEHTA makes the information and other material (“Information”) in this document available in good faith but without any representation or warranty as to its accuracy or completeness. NEHTA cannot accept any responsibility for the consequences of any use of the Information. As the Information is of a general nature only, it is up to any person using or relying on the Information to ensure that it is accurate, complete and suitable for the circumstances of its use.

Document Control

This document is maintained in electronic form. The current revision of this document is located on the NEHTA Web site and is uncontrolled in printed form. It is the responsibility of the user to verify that this copy is of the latest revision.

Copyright © 2008, NEHTA.

This document contains information which is protected by copyright. All Rights Reserved. No part of this work may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without the permission of NEHTA. All copies of this document must include the copyright and other information contained on this page.

Table of contents

Table of contents	iii
Document information	v
Change history	v
1 Introduction	1
1.1 Document purpose	1
1.2 Intended audience	1
1.3 Definitions, acronyms, abbreviations	1
1.4 Normative references	2
1.5 Overview	3
1.5.1 Alternate and Complementary Interfaces	3
1.5.2 Web Service Environments	3
1.5.3 SIL Deployment	3
2 National Infrastructure	4
2.1 Unique Healthcare Identifier Service	4
2.1.1 Identifiers	4
2.1.2 SIL Endpoints	4
2.2 NASH	5
2.2.1 Certification Authority	5
2.2.2 Certificate References	5
2.3 NEHTA SIL	6
3 Data Model	7
3.1 Database Schema	7
3.1.1 Entities	7
3.1.2 Constant URIs	8
3.1.3 Intersection Tables	8
3.2 DAO or Alternative	9
3.3 Query for listInteractions	9
3.3.1 SQL Select Example	10
3.4 Standard Update Operations	12
3.4.1 addInteraction	12
3.4.2 removeInteraction	16
4 Trust and Security	20
4.1 SIL Certificates	20
4.1.1 SIL Associations	20
4.2 Lookup Operations	20
4.3 Update Operations	20
4.3.1 Updates by Owners	20
4.3.2 Updates by Non-owners	20
5 Maintenance	23
5.1 Extensions to the Standard Interface	23
5.1.1 Associating Primary Identifiers	23
5.1.2 Lower-level Updates	23
5.1.3 Bulk Interfaces	23
5.1.4 Non-Web service Updates	24
5.2 Update Responsibility	24
5.2.1 Owner and SIL Updates	24
5.2.2 Changes Affecting HPIO	25
6 Reliability	27
6.1 System Down-time	27
6.1.1 Clustered Application Server Environment	27

6.1.2	Failover Databases	27
7	Network Configuration	28
8	Existing Provider Directories	29
	Appendix A: Informative references.....	30

Document information

Change history

Version	Date	Comments
1.1	2008-12-01	Release

This page is intentionally left blank.

1 Introduction

1.1 Document purpose

This document provides guidance on how to design and implement a service instance locator (SIL). Some of the issues that may arise through deployment of a SIL are also covered.

This is a non-normative document. While implementers are obliged to conform to the SIL specification, they are not required to follow the advice contained in this document.

1.2 Intended audience

This is a technical document.

This document is intended for:

- Enterprise architects who develop strategic plans for solutions that involve connectivity.
- Solution architects who develop solutions that implement or interact with the connectivity architecture.
- Software developers who implement the solutions designed by the solution architects.

The reader is expected to be familiar with the SIL Architecture [SA2008].

1.3 Definitions, acronyms, abbreviations

Definitions are reproduced from [SA2008] with a few additions.

CA	Certification Authority—a trusted entity that establishes healthcare provider membership in the e-health community by signing the providers X.509 certificate with their own (CA) private key. The certificate containing the corresponding public key would be stored by e-health clients.
CRUD	Create, read, update and delete
CUD	Create, update and delete
Document	A clinical document or ancillary message, such as a notification or acknowledgement for a clinical document. Documents are usually represented in XML. Elements within such documents may contain non-XML data, e.g. formatted according to HL7.
Endpoint	A URI including network protocol and address. It provides the binding of an interface to an implementation.
FTPS	File transfer protocol over SSL (Secure sockets layer)
HPII	Healthcare Provider Identifier for an Individual
HPIO	Healthcare Provider Identifier for an Organisation
IHI	Individual Healthcare Identifier
Interaction	Pattern of communication whereby a target obtains its document – corresponds to the patterns outlined in [CPIS2008].

Interaction Role	Function within an interaction. Roles are played by healthcare providers or intermediaries.
NASH	National Authentication Service for Health—NASH will be a CA for e-health technical service providers.
Service	In this document the term service generally refers to a technical service as per [IF2007]. A technical service is usually a Web service.
Service Category	Service types encompassed by a medical realm. Categories are initially expected to be document types specified by the NEHTA work packages. Each service category will be identified by a URI.
Service Interface	URI-based definition of a service offered by a role. Initially these interfaces will describe a Web service.
Service Provider	An organisation that hosts a Web service. This could be the target, source, or a third party.
SFTP	File transfer protocol using secure shell (SSH)
SIL	Service Instance Locator
Source	Document suppliers / compilers. For clinical documents a source is a healthcare organisation, e.g. pathology laboratory.
Target	The final destination (intended recipient) of a document. For clinical documents a target is a healthcare provider organisation, e.g. medical clinic.
UHI	Unique Healthcare Identifier (HPIO, HPII, or IHI)
UHI Service	Proposed National UHI Web service

1.4 Normative references

The following NEHTA specifications and other references contain provisions which, through not referenced in this text, constitute provisions of this Specification. At the time of publication, the editions indicated were valid. All Specification and other references are subject to revision: all users of this Specification are therefore encouraged to investigate the possibility of applying the most recent edition of the Specification and other references listed below.

- [IF2007] NEHTA, *Interoperability Framework v2.0*, 17 August 2007.
- [CPIS2008] NEHTA, *Concepts and Patterns for Implementing Services, V2.0*, 1 December 2008.
- [WSP2008] NEHTA, *Web Services Profile v3.0*, 1 December 2008.
- [SA2008] NEHTA, *SIL Architecture v1.1*, 1 December 2008.

1.5 Overview

SIL implementations are responsible for maintaining the interaction patterns, roles, and associated endpoints described in [SA2008]. Some of the issues dealt with in this document are:

- Data store requirements,
- Tie-in with HPIO records,
- HPIO Identifiers,
- Certificates,
- Trust of/by clients,
- Semantics Identifiers.

Note that SIL must be implemented as a Web service. Details of how to create and deploy Web service applications are not covered.

1.5.1 Alternate and Complementary Interfaces

A SIL instance must provide the interfaces dictated by [SA2008]. However the specification does not preclude implementing other interfaces. As outlined in 5.1, alternate mechanisms, particularly for updates, may be more convenient and easier to implement. One such approach is to use Web applications.

[SA2008] does not specify operations for all the updates which a SIL implementation will have to encompass. Therefore, many design decisions, even at the interface level, are left to the discretion of implementers. For ease of use, particularly for staff of healthcare organisations, it is highly desirable that a user-friendly GUI is provided, at least for updates.

1.5.2 Web Service Environments

There are several choices for building Web service applications, the most popular of which are Microsoft's .NET environment and various Java application servers. Both choices allow for integration of Web service and Web applications if this is desired.

1.5.3 SIL Deployment

According to [SILR2008] there is a 1:N relationship between HPIOs and their SILs. As stated in [SA2008], there could be a single centrally administered SIL associated with every HPIO. That would require significant resources from one provider agent while preventing others from value adding their own features. Consequently, such a solution may be unacceptable to many healthcare providers in the long term. Other deployment scenarios include distributing several large SILs based on regions, or distributing a large number of small SILs, where each is associated with only one or two HPIOs. In all cases it is anticipated that a HPIO record will contain a SIL address and associated X.509 certificate(s). Those records will eventually be obtained from the HPIO service.

1.5.3.1 Data Stores

Depending on the number of associated HPIOs, different SIL implementations will need data stores of widely varied capability. Except in cases where the backing data store is very small, a relational database will be required. This document contains suggestions for implementing the data model and base access/update statements. Actual database access will most likely be accomplished through the libraries and/or transaction managers available in the deployment environment.

2 National Infrastructure

2.1 Unique Healthcare Identifier Service

2.1.1 Identifiers

SIL lookups require a URI representing a target's HPIO as input. This HPIO identifies the target healthcare provider to whom a source transmits a document. It is envisaged that sources will locate correct SIL endpoints by first consulting the HPIO service.

It is likely that SIL implementations will be needed before the HPIO service is available and before any HPIOs are issued. If so, SIL clients will need an identifier to substitute for the HPIO in the short term. According to [SA2008] this identifier can be any URI. A SIL provider could publish these de-facto identifiers on a Web site. Alternatively, if the potential client organisations were few and well known, the information could be exchanged by telephone.

For SILs with a large number of associated healthcare providers, it would be beneficial if some consistent scheme were used. One such scheme would be to formulate the URL in a standard way using an Australian Business Number (ABN) in place of the HPI. The result would be similar to:

- <http://ns.nehta.gov.au/Id/Const/Abn/1.0#53004085616>

Australian Business Numbers can be searched from the registry at <http://www.abr.business.gov.au/>. Such a scheme would ensure uniqueness of identifiers, although it does not allow for full automation. If desired, it is possible for an entity to register with the Australian Business Registry to use Web services to obtain ABN records. After registration, clients can automate workflows to exchange messages.

2.1.2 SIL Endpoints

Lack of a HPIO service raises the issue of how to resolve SIL endpoints. The most likely short term scenario will be for cooperating source and target pairs to obtain addresses through out-of-band means (eg by telephone, email).

Although the SIL specifications discuss document exchange scenarios in terms of healthcare providers and clinical documents, the interfaces themselves are generic. This means it is possible for a special SIL to exist that could provide the endpoints for other SILs. In effect, such a SIL would function in place of the UHI service for the purposes of obtaining SIL endpoints.

2.1.2.1 SIL of SILs

In order to substitute for the UHI service, an Interaction data structure would contain values similar to the example below.

Interaction:

```
Interaction: "Retrieve"  
Target: <HPIO number>  
Service Category: "SIL"  
Role:  
    Role Name: "SIL Service"  
    Service Interface: <SIL lookup interface>  
    Service Provider: <HPIO of SIL provider>  
    Service Endpoint: <SIL service endpoint>  
    Qualified Cert Ref: <ref to SIL response signing cert>
```

2.2 NASH

2.2.1 Certification Authority

In the medium term, NEHTA will become an intermediate CA, issuing trusted certificates to HPIO organisations. It is likely that SIL implementations will be needed before NASH is available or any trusted certificates have been issued. However, early implementations may choose to trust certificates issued by Medicare Australia PKI.

2.2.2 Certificate References

In the absence of NASH, there is no standard way of resolving certificate references. However, SIL clients need to obtain certificates to use for services returned by the SIL for the purposes of encrypting symmetric keys and checking response signatures.

To get around this problem, the qualified certificate reference data structure is flexible enough to contain the certificate itself. To accomplish this, values for the `QualifiedCertRef` attributes could be used similar to those shown below.

`useQualifier:`

`http://ns.nehta.gov.au/Qcr/Use/PayloadTransport/Const/SignKeyEnc/1.0`

`typeQualifier:`

`http://ns.nehta.gov.au/Qcr/Ref/Const/Direct/PEM/1.0`

`value:`

```
-----BEGIN CERTIFICATE-----
MIIDHjCCAtugAwIBAgIESOxRITALBgcqhkJ0OAQDBQAwcjELMAkGA1UEBhMCQVUx
DDAKBgNVBAGTA1FMRDERMA8GA1UEBxMIQnJpc2JhbmUxDjAMBgNVBAoTBUE5FSFRB
MRkwFwYDVQQLExBTZWNN1cmUgTWVzc2FnaW5nMRcwFQYDVQQDEw5TYW1wbGUU2Vy
dmljZTAeFw0wODEwMDgwNjIwMTdaFw0wOTAxMDYwNjIwMTdaMHExCzAJBgNVBAYT
AkFVMQwwCgYDVQQIEwNRTEQxETAPBgNVBACtCEJyaXNiYW5lMQ4wDAYDVQQKEwVO
RUhUQTEZMbcGA1UECXMQU2VjdXJlIE1lc3NhZ2luZzEXMBUGA1UEAxiMOU2FtcGxl
IFNlcnZpY2UwgG3MIIBLAYHkoZIZjgEATCCAR8CgYEA/X9TgR11EilS30qcLuzk
5/YRt1I870QAwX4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuAHTRv8mZg
t2uZUKWkn5/oBHSQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVUE1oWkTL2
dfOuK2HXKu/yIgmZndFIAccCFQCXYFCPFsMLzLKSuYKi64QL8Fgc9QKBgQD34aCF
1ps93su8q1w2uFe5eZSvu/o66oL5V0wLPQeCZ1FZV4661F1P5nEHEIGAtEkWcSPo
TCgWE7fPCTKMyKbhPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMppG+
qFGQiaid3+Fa5Z8GkotmXoB7VSVkAUw7/s9JKgOBhAACgYAvKwhiCES/MuDBAm0c
8JEypfVBB17ptjtEQU42P/3ILOsRyHbG2+yNg3tteQmyMDDfm44+zFiNL4aPMWx4
/WZm0u8URfLgHkgFam9q5Lb1jYs7SWeponYeGz5okoBzF4D3oAxXjIrMRTEt34nQ
5kZvOcmr6WDrLKBppjkrqVBA1DALBgcqhkJ0OAQDBQADMAAwLQIURep/PTyiwoMf
p6WjUgIuV5nrWhYCFQCPy85CQOrPDnAWM6D80MJAE4p0nQ==
-----END CERTIFICATE-----
```

2.3 NEHTA SIL

NEHTA will implement and operate a SIL in the near future. It will allow publishing and lookup of the *interactions* and associated technical *services* offered by and on behalf of healthcare providers.

The address of this SIL will be published on NEHTA's Website. Healthcare providers will not be obliged to utilize this implementation in either the short or long term. However, they may wish to take advantage of it until other instances become available.

NEHTA will implement a Web application as an interface to the backing data repository in addition to the standard Web service interfaces of [SA2008]. NEHTA undertakes to publish the source code as an example for future implementers.

NEHTA will also operate a SIL of SILs for the benefit of early adopters until the UHI Service becomes available. The address of the SIL of SILs will be published on NEHTA's website.

3 Data Model

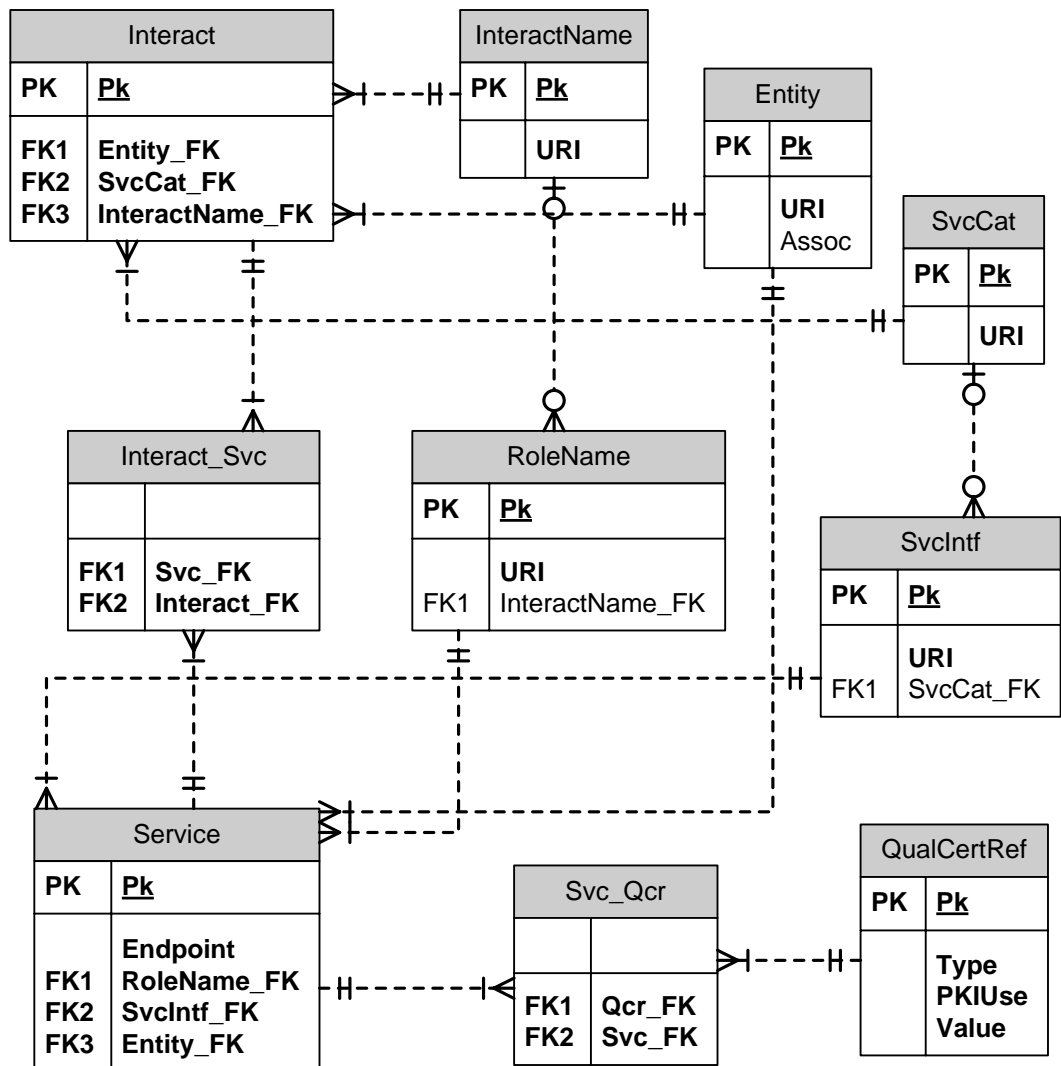


Figure 1 Possible Data Model

SIL data structures follow a relatively straightforward hierarchy which is a little complicated to represent using an Entity-Relationship model. Difficulty arises due to aggregations inherent in the data structures along with the potential for multiple M:N mappings.

For example, an interaction consists of one or more roles, which essentially describe a service. However, a different interaction could reference the same service(s), requiring an M:N storage relationship. Similarly, a role has one or more qualified certificate references but the same certificate reference could apply to any service hosted by the same provider.

3.1 Database Schema

Figure 1 illustrates one potential relational data model. This schema is sufficient to implement a basic SIL that complies with the specifications. It does not take account of any enhancements or options which a SIL implementer may wish to adopt.

3.1.1 Entities

Table `Entity` in the model is used to contain URIs with embedded HPIOs. Entities associated with a SIL are healthcare providers, but the table can also

contain organisations that provide technical services, e.g. a *deliver* Web service. It is a requirement of [SA2008] that standard operations return a *SILError* fault if an entity referenced by an input *InteractionRequest* or *Interaction* contains a HPIO with no SIL association. *Entity.Assoc* is a column of type Boolean, indicating whether the row describes an associated healthcare provider or a technical service provider. If the value of *Assoc* is false or NULL, column *URI* references a non-associated organisation.

3.1.2 Constant URIs

The SIL model consists of several constant URIs. These are:

- Interaction URIs (modelled in Figure 1 by the *InteractName* table);
- Role URIs (modelled in Figure 1 by the *RoleName* table);
- Service Interface URIs (modelled in Figure 1 by the *SvcIntf* table);
- Service Category URIs (modelled in Figure 1 by the *SvcCat* table).

These URIs are used for identification. There are relatively few in each category, and queries will typically begin by comparing them against the input.

There are optional foreign keys (i.e. may be null) as follows.

- *RoleName.InteractName_FK* -> *InteractName_PK*

Certain roles identifiers (URIs) are defined by NEHTA packages to be compatible with certain interaction identifiers. If implemented, this relation could provide a check to ensure input roles matched interactions, or could be used internally to determine roles aligning with interactions.

- *SvcIntf.SvcCat_fk* -> *SvcCat_PK*.

Certain interfaces correspond to particular service categories. If implemented, this relation could provide a check to ensure a service interface corresponds to a category. It could also be used in internal queries to determine interfaces aligning with categories.

3.1.3 Intersection Tables

There are two intersection tables, *Interact_Svc* and *Svc_Qcr* to account for M:N relationships of interactions to roles (services) and services to certificate references, respectively.

These are not strictly necessary for a lookup. For example, since data need be traversed only in the direction from interaction to services, table *interact* could be modelled as columns *Entity_FK*, *SvcCat_FK*, *InteractName_FK*, along with another foreign key, e.g. *Svc_FK*, pointing to the primary key of the service table. An interaction with two roles would have two records in table *Interact*; there would be no need for a unique primary key.

If this approach is taken it would be less efficient to match interactions from services. Depending on how many records were maintained, it may also lead to a higher storage volume. Neither consideration would be important for a SIL associated with only a few HPIOs. Eliminating the intersection tables would simplify queries to an extent.

Similar considerations apply to table *Svc_Qcr*. A service hosted by a third party on behalf of a healthcare provider will have at least two certificates. One is required to encrypt content for the healthcare provider. Another is needed to encrypt content for the service provider. The same or another certificate would be used to check any signature on the response. Consequently, many largely redundant records would be present in table *Service* if *Svc_Qcr* were dropped. Since there are likely to be significantly less certificates than services, it is probably a good idea to keep *Svc_Qcr*.

3.2 DAO or Alternative

It is expected that implementers will provide or choose their own data access object (DAO) layer as a façade to the database, which is relevant to the platform and programming language of choice. The most obvious way of doing this is to use ODBC/JDBC libraries.

There are alternative means to accomplish data access from code such as object-relational mapping technologies (e.g. hibernate/nhibernate, TopLink, Kodo, etc).

Insertions, updates, and deletions may require substantial consistency checking, which is much easier to implement in a programming language like Java, rather than exclusively through stored procedures, etc. No code examples are provided in this document, but some SQL examples are given in the following sections to provide some idea of the complexities involved.

SIL is a relatively simple specification in its own right, but implementers may choose to extend the base functionality at the expense of additional complexity. Subsequent sections outline some of the issues that could come into play for early adopters.

3.3 Query for *listInteractions*

Operation *listInteractions* returns a set of interaction data structures. Each interaction contains a set of roles, which in turn contain a set of qualified certificate references. This is a tree structure which can be built from a data model by obtaining the high level interactions, then obtaining any associated roles (modelling services), then obtaining any associated certificate references.

Input to *listInteractions* is a target organisation, a set of service category URIs, and optionally, a set of interaction URIs. The starting point is to match all the interaction rows by comparing the input target, input service categories and interaction URIs, if present.

3.3.1 SQL Select Example

Below is an example matched on a HPIO, specified in the variable *org*, and two categories, specified in variables *cat* and *cat2*.

```
# input is of class InteractionRequest as per [SA2008]
set @org = <input.target>;
set @cat = <input.serviceCategory[1]>;
set @cat2 = <input.serviceCategory[2]>;
select
    iname.uri as interaction,
    r.uri as role,
    svc.endpoint,
    intf.uri as interface,
    prov.uri as provider,
    qcr.type,
    qcr.pkiuse,
    qcr.value
from Entity
join Interact i on entity.pk = i.entity_fk
join SvcCat cat on i.svccat_fk = cat.pk
join InteractName iname on i.interactname_fk = iname.pk
join Interact_Svc i_s on i.pk = i_s.interact_fk
join Service svc on i_s.svc_fk = svc.pk
join RoleName r on svc.rolename_fk = r.pk
join SvcIntf intf on svc.svcintf_fk = intf.pk
join Entity as prov on svc.entity_fk = prov.pk
join svc_qcr s_c on svc.pk = s_c.svc_fk
join QualCertRef qcr on s_c.qcr_fk = qcr.pk
where
    Entity.uri = @org
    and (
        cat.uri = @cat
        or cat.uri = @cat2
    );
```

If a set of interaction names were present, the input(s) would have to be checked against the URI in table *InteractName*. It is expected that most queries will specify an HPIO and one category URI.

Section 3.3.1.1 contains a table which is the output from a query. Note that the HPIOs, URIs, and other fields are not in standard format; they have been shortened to fit the page.

3.3.1.1 Sample output

interaction	role	endpoint	intf	provider	type	pkiuse	value
retrieve	retrieve	http://o2.org/med_ret	http://retrieve	8036000000000002	hdi	pl:sign+enc	cert5
retrieve	retrieve	http://o2.org/med_ret	http://retrieve	8036000000000002	hdi	tr:sign+enc	cert3
notifyAndRetrieve	notify	http://org3.org/ds_not	http://notify	8036000000000003	hdi	pt:sign+enc	cert6

The table is a hierarchical representation, so it is easy to adapt from the result columns to the interaction structure required by the SIL specification. There is a fresh interaction when the interaction URI changes; there is a fresh role when the role URI changes; and there is a fresh certificate reference when the combination of (type, usages, value) changes. The resulting set would be as indicated below.

Interaction: URI = retrieve

Target: <same as var org = 8036000000000002>

Service Category: <same as var cat = medications>

Role: URI = retrieve

Service Interface: http://retrieve

Service Provider: 8036000000000002

Service Endpoint: http://o2.org/med_ret

Qualified Cert Ref: type = hdi, use = pl:sign+enc, value = cert5

Qualified Cert Ref: type = hdi, use = tr:sign+enc, value = cert3

Interaction: URI = notifyAndRetrieve

Target: <same as var org = 8036000000000002>

Service Category: <same as var cat2 = discharge_summary>

Role: URI = notify

Service Interface: http://notify

Service Provider: 8036000000000003

Service Endpoint: http://org3.org/ds_not

Qualified Cert Ref: type = hdi, use = pt:sign+enc, value = cert5

3.4 Standard Update Operations

There are two standard operations to modify SIL data: `addInteraction` and `removeInteraction`. Both are coarse-grained, consistent with recommended practice for distributed service interfaces, but lacking the requisite control to fully create and update `Interaction` record components.

The semantics for publish operations are not specified, so implementers have considerable flexibility. It is recommended that standard operations disallow implicit updates for atomic components. If using the proposed data model, the easiest approach would be to check that URIs corresponding to `SvcCat`, `InteractName`, `RoleName`, and `SvcIntf` exist in the database. If any are absent, the standard error `badParam` should be returned to the client.

Disallowing implicit insertion of the various URIs implies that there must be an alternate means for low-level updates, not necessarily a Web service. Some suggestions are given in 5.1.

According to [SA2008] if the `Entity` URI does not exist a `SILError` should be returned with the enumeration set to `unknownTargetID`, which tells the client there is no HPIO associated with this SIL.

3.4.1 addInteraction

Insertion of an *Interaction* requires several checks to be made. Each of the constant URIs must be determined to be present, as well as any referenced service provider entities and certificate references. Note that an implementation could allow implicit insertion of certificate references or require them to pre-exist in the database. Since all fields have to be present in the input, implicit insertion requires additional SQL insert statements.

When the NASH is deployed, it may be a reasonable expectation that the SIL instance checks to ensure the referenced certificates have actually been issued to the entity hosting a service. Until then, there is no standard way to accomplish this, so the SIL may have little choice other than to trust the input. If the certificate(s) are invalid or not trusted, there will be problems when the client attempts to use the service. When it becomes possible for the SIL instance to verify the certificate references, standard error `badParam` should be returned to the client as a fault.

Section 3.4.1.1 contains a sequence of SQL statements to implement `addInteraction` for a sample *Interaction* structure. This is simply to give potential implementers a basic idea for use with the suggested data model. It does not check for invalid URIs, bad inserts, etc, and there is no rollback path.

Validations are usually simple - for example, an invalid identifier will result in a NULL-valued variable; a failed insert will result in a SQL error, etc. It is generally easier for such checks to be carried out from a 3GL DAO or some other façade, rather than directly from a stored procedure and/or associated error handlers. Another argument in favour of the DAO approach is that it will be much easier to deal with variations in cardinality of the sets contained by input *Interactions* (i.e. *InteractionRole* and *QualifiedCertRef*).

3.4.1.1 Example SQL for Operation

The example below accomplishes an insert for a *DeliverAndNotify Interaction*, where each role is implemented by two different service providers. One service requires two qualified certificate references and the other requires only one. This example assumes the input services do not exist. Depending on the adopted approach, this may not be a valid assumption.

The most desirable approach would be to implement composition-style semantics, whereby `addInteraction` adds a service only when it does not already exist and `removeInteraction` removes it only when it is no longer

associated with any interaction. Composition is more difficult to implement and may not be needed for early adopters. The examples in this document assume that interactions will not be composed in such a manner. However, if it does become desirable, modifications to the DAO/SQL would be quite straightforward. Comments in the example indicate how to proceed.

```
# input is of class Interaction as per [SA2008]
set @endpoint1=<input.InteractionRole[1].serviceEndpoint>;
set @endpoint2=< input.InteractionRole[2].serviceEndpoint >;
set @provider1=<input.InteractionRole[1].serviceProvider>;
set @provider2=<input.InteractionRole[2].serviceProvider>;
begin;

set @org_pk = (
    select pk from org
        where uri = <input.target>
);

set @cat_pk = (
    select pk from cat
        where uri = <input.serviceCategory>
);

set @prov1_pk = (
    select pk from org
        where uri = @provider1
);

set @prov2_pk = (
    select pk from org
        where uri = @provider2
);

set @iname_pk = (
    select pk from InteractName
        where uri = <input.interaction>
); # deliver and notify

set @role1_pk = (
    select pk from rolename
        where uri = <input.InteractionRole[1].roleName>
); # deliver

set @role2_pk = (
    select pk from rolename
        where uri = <input.InteractionRole[2].roleName>
); # notify

set @intf1_pk = (
    select pk from svcintf
        where uri = <input.InteractionRole[1].serviceInterface>
);

set @intf2_pk = (
    select pk from svcintf
        where uri = <input.InteractionRole[2].serviceInterface>
);

# set cert = <input.InteractionRole[1].QualifiedCertRef[1]>;
set @cert1_delv = (
    select pk from QualCertRef
        where value=<cert.value>
            and type=<cert.typeQualifier>
            and pkiuse=<cert.useQualifier>
);

# reset cert = <input.InteractionRole[1].QualifiedCertRef[2]>;
set @cert2_delv = (
    select pk from QualCertRef
        where value=<cert.value>
            and type=<cert.typeQualifier>
            and pkiuse=<cert.useQualifier>
);

# reset cert = <input.InteractionRole[2].QualifiedCertRef[1]>;
set @cert_notf = (
    select pk from QualCertRef
        where value=<cert.value>
            and type=<cert.typeQualifier>
            and pkiuse=<cert.useQualifier>
```

```
);

# **** ERROR IF ANY VARIABLE IS SET TO NULL ****

# *** if allowing composition of services -- ***
# *** then do select first to determine      ***
# *** service if already exists e.g.        ***
# ***
# *** set @svc2_pk = (select pk from service
# *** join svc_qcr s_c on pk = s_c.svc2_fk
# *** and s_c.qcr_fk = @cert_notf
# *** where endpoint = @endpoint2 and rolename_fk = @role2_pk
# *** and svcintf2_fk = intf2_pk and entity_fk = @prov2_pk);
# ***
# *** insert only if service(s) not present e.g. @svc2_pk = NULL

insert into service
    (endpoint, rolename_fk, svcintf_fk, entity_fk)
    values
        (@endpoint1, @role1_pk, @intf1_pk, @prov1_pk),
        (@endpoint2, @role2_pk, @intf2_pk, @prov2_pk);

# *** if allowing composition of services -- ***
# *** determine primary key of service(s)      ***
# *** from select statement                    ***
# *** else perform statements below            ***

set @svc2_pk = (select max(pk) from service);
set @svcl_pk = @svc2_pk - 1;

insert into svc_qcr
    (svc_fk, qcr_fk)
    values
        (@svcl_pk, @cert1_delv),
        (@svcl_pk, @cert2_delv),
        (@svc2_pk, @cert_notf);

insert into interact
    (entity_fk, svccat_fk, interactname_fk)
    values
        (@org_pk, @cat_pk, @iname_pk);

set @interact_pk = (select max(pk) from interact);

insert into interact_svc
    (interact_fk, svc_fk)
    values
        (@interact_pk, @svcl_pk),
        (@interact_pk, @svc2_pk);

commit;
```

3.4.2 removeInteraction

This operation is the inverse of `addInteraction`. It requires the same checks for consistency of input. The SIL Architecture [SA2008] leaves the exact semantics unspecified, so the implementer can choose the extent to which implicit deletes will be executed. The greater the extent to which such deletes are supported, the greater the amount of checking is required in the DAO. There are probably only three sensible choices.

1. Delete the *Interaction* only, i.e. no implicit deletes.
2. Delete the *Interaction* and the associated *Services*.
3. Delete the *Interaction*, associated *Services*, and associated *QualifiedCertRefs*.

It is important that `removeInteraction` semantics are consistent with those of `addInteraction`. In the example of 3.4.1.1 services and interactions are added but certificate references must pre-exist, otherwise it is considered an error. This is perhaps the easiest approach, since certificates may be reused for numerous services. If this approach is adopted for `addInteraction`, then `removeInteraction` should delete the *interaction* and associated *services* but it should not delete certificate references, even if removal of a *service* would leave them without a reference in table `Svc_Qcr`.

It is possible that some rows, particularly in table `SvcIntf`, may be left unreferenced. To prevent this, triggers could be defined or a script could be periodically run to check for unreferenced entries. However, such an action is not really necessary because the only foreign key constraints are against tables `Interact`, `Service`, and `QualCertRef`. *Entities*, *categories*, *interfaces*, *interaction names*, and *role names* should be allowed to exist in isolation, becoming associated with *services* and *interactions* as required.

3.4.2.1 Example SQL for Operation

The example below accomplishes a delete. It is the inverse operation to that defined in 3.4.1.1. This example assumes that the input services are associated with a single interaction but depending on the adopted approach, this may be an invalid assumption (see 3.4.1.1). Comments in the example indicate alternative statements if providing for service composition.

```

# input is of class Interaction as per [SA2008]
set @endpoint1=<input.InteractionRole[1].serviceEndpoint>;
set @endpoint2=< input.InteractionRole[2].serviceEndpoint >;
set @provider1=<input.InteractionRole[1].serviceProvider>;
set @provider2=<input.InteractionRole[2].serviceProvider>;
begin;
set @org_pk = (
    select pk from org
        where uri = <input.target>
);
set @cat_pk = (
    select pk from cat
        where uri = <input.serviceCategory>
);
set @prov1_pk = (
    select pk from org
        where uri = @provider1
);
set @prov2_pk = (
    select pk from org
        where uri = @provider2
);
set @iname_pk = (
    select pk from InteractName
        where uri = <input.interaction>
); # deliver and notify
set @role1_pk = (
    select pk from roles
        where uri = <input.InteractionRole[1].roleName>
); # deliver
set @role2_pk = (
    select pk from roles
        where uri = <input.InteractionRole[2].roleName>
); # notify
set @intf1_pk = (
    select pk from intf
        where uri = <input.InteractionRole[1].serviceInterface>
);
set @intf2_pk = (
    select pk from intf
        where uri = <input.InteractionRole[2].serviceInterface>
);
# set cert = <input.InteractionRole[1].QualifiedCertRef[1]>;
set @svcl_cert1 = (
    select pk from QualCertRef
        where value=<cert.value>
            and type=<cert.typeQualifier>
            and pkiuse=<cert.useQualifier>
);
# reset cert = <input.InteractionRole[1].QualifiedCertRef[2]>;
set @svcl_cert2 = (
    select pk from QualCertRef
        where value=<cert.value>
            and type=<cert.typeQualifier>
            and pkiuse=<cert.useQualifier>
);
# reset cert = <input.InteractionRole[2].QualifiedCertRef[1]>;
set @svc2_cert = (
    select pk from QualCertRef
        where value=<cert.value>
            and type=<cert.typeQualifier>

```

```

        and pkiuse=<cert.useQualifier
    );
# **** ERROR IF ANY VARIABLE IS SET TO NULL ****
set @svcl_pk = (
    select pk from service
        join svc_qcr s_c on pk = s_c.svc_fk
            and s_c.qcr_fk = @svcl_cert1
    where entity_fk = @prov1_pk
        and endpoint = @endpoint1
            and rolename_fk = @role1_pk
            and svcintf_fk = @intf1_pk
    union
    select pk from service
        join svc_qcr s_c on pk = s_c.svc_fk
            and s_c.qcr_fk = @svcl_cert2
    where entity_fk = @prov1_pk
        and endpoint = @endpoint1
            and rolename_fk = @role1_pk
            and svcintf_fk = @intf1_pk
);

set @svc2_pk = (
    select pk from service
        join svc_qcr s_c on pk = s_c.svc_fk
            and s_c.qcr_fk = @svc2_cert
    where entity_fk = @prov2_pk
        and endpoint = @endpoint2
            and rolename_fk = @role2_pk
            and svcintf_fk = @intf2_pk
);

set @interact_pk = (
    select pk from interact
        join interact_svc i_s on pk = i_s.interact_fk
            and i_s.svc_fk = @svcl_pk
    where entity_fk = @org_pk
        and svccat_fk = @cat_pk
            and interactname_fk = @iname_pk
    union
    select pk from interact
        join interact_svc i_s on pk = i_s.interact_fk
            and i_s.svc_fk = @svc2_pk
    where entity_fk = @org_pk
        and svccat_fk = @cat_pk
            and interactname_fk = @iname_pk
);

# **** ERROR IF ANY STATEMENT RETURNS #ROWS <> 1 ****
# **** if allowing service composition ****
# **** need to check for assoc with > 1 interaction e.g. ****
# ****
# *** set @svc2_cnt = (
# ***     select count(*)
# ***         from interact_svc
# ***         where svc_fk = @svc2_pk
# *** );
# ***

# disassociate cert refs from services
delete from svc_qcr
    where svc_fk = @svcl_pk;
    # logically -qcr_fk = @svcl_cert1 or qcr_fk = @svcl_cert2

delete from svc_qcr
    where svc_fk = @svc2_pk;
    # *** composition e.g. include clause: and @svc2_cnt > 1;

# disassociate services from interaction
delete from interact_svc
    where int_fk = @interact_pk;

# delete services (possibly conditionally - see notes above)

```

```
delete from service
  where pk = @svc1_pk
     or pk = @svc2_pk;
# *** composition example - replace or clause as:
# *** or (pk = @svc2_pk and @svc2_cnt = 1)

# delete interaction

delete from interact
  where pk = @interact_pk;

commit;
```

4 Trust and Security

4.1 SIL Certificates

The SIL provider must obtain its own certificate which the client will use to encrypt the request and which will also be used to sign the response (a list of interactions) (See [WSP2008].)

From the client point of view, there must be a mechanism to obtain the certificate to be used with any SIL instance. If the UHI service is not available, this may be done by out-of-band means. The SIL service provider will be responsible to define some mechanism and communicate it to potential clients. Some examples are:

- Download the certificate from a Web site.
- Attach the certificate to emails to clients.
- Incorporate SIL address and corresponding certificates in the proposed national infrastructure SIL.

4.1.1 SIL Associations

There are no official guidelines about associating SIL services with corresponding certificates or HPIOs. These issues should be addressed in the near future.

If there is one central SIL implemented, the problem of associating HPIOs will be moot since every HPIO would be associated with the same SIL. Similarly, since there would only be one certificate, for signing responses, this could be obtained quite easily, e.g. from a well-known Web site.

If there is to be a distributed set of SILs, ad-hoc mechanisms for setting and retrieving associations are less acceptable, because synchronization and consistency issues could easily occur. Before the UHI service is operational, it is recommended that NEHTA take responsibility by publishing SIL associations on an external Web site.

4.2 Lookup Operations

Any entity with a certificate signed by the trusted CA(s) is allowed to perform a lookup operation (see [SILR2008] and 2.2.1). It is a core requirement that a SIL does not restrict information to any e-health community member.

4.3 Update Operations

4.3.1 Updates by Owners

According to [SILR2008] organisations must be able to update their own records. SIL implementations need to check request signatures to determine whether an update operation is being attempted by the logical owner of the information. To accomplish this, the HPIO number present in the interaction has to match a certificate issued to the corresponding organisation by a trusted CA.

4.3.2 Updates by Non-owners

In general, an organisation should not be allowed to update information on behalf of some other organisation. However, existing relationships exist between many healthcare providers and entities which administer local service directory entities. In such a relationship the directory stores service

location information on behalf of more than one healthcare provider. It is likely that in some circumstances the directory provider will also host Web services to transfer clinical documents and other messages.

Where such a relationship exists, a HPIO entity should be able to delegate update operations on its SIL entries. In the absence of an official e-health authorisation strategy, a SIL instance must still allow such updates to occur. Currently, there is no standard strategy to accomplish this. Below are a few suggestions.

4.3.2.1 Role Based Authorisation

Define authorisation roles (not to be confused with *Interaction Roles*) corresponding to capabilities. An example set may include:

- Owner– Corresponds to the HPIO whose SIL records may be updated. Owners should have all CUD privilege for records (to be) associated with their HPIO.
- Agent– Corresponds to the HPIO(s) that an owner authorizes to perform its standard inserts and deletions.
- Agent Insert– Corresponds to the HPIO(s) that an owner authorizes to insert interactions.
- Agent Delete– Corresponds to the HPIO(s) that an owner authorizes to delete interactions.
- Agent Update– Corresponds to the HPIO(s) that an owner authorizes to update interactions. There is no update operation in the SIL specification, so any updates would constitute extra operations implemented by a SIL provider.
- Certificate Administrator – Allowed to insert and remove qualified certificate references.
- SIL Owner – Corresponds to the SIL implementer. May be responsible for adding HPIO URIs to the SIL data set.

Unless the SIL implementer required a fine degree of authorisation granularity, there is no need for all the roles listed above. Minimally, *SIL Owner* and *Owner* would be needed.

If particular directory provider HPIOs were allowed full CUD privileges, an *Agent* role would be needed. There would have to be an operation of some kind for a *SIL Owner* or *Owner* to grant the privilege.

4.3.2.2 Explicit v Implicit Roles

It is possible to define roles and associated privileges explicitly. This would require configuration files, or more likely, tables in the database containing roles, associated HPIOs, and privileges.

The *Owner* role can easily be made implicit. CUD permissions would be contingent on the X.509 certificate matching the private key used to sign the operation request.

It may not be possible for an *Agent* role to be implicit. However, if the suggested data model is used (see 3.1), it is easy to delegate CUD access with the addition of another table. The table could be named *DELEGATED_AGENTS* and would require two foreign keys on table entity (*TARGET, AGENT*). If an agent organisation attempted to update a record for a target organisation, this table would be checked. Again, a non-standard operation would be required for a target organisation to insert a record for its agent. In this way, support for roles would be implicit in the logic of the operations; there would be no need for tables of roles and privileges. Similarly, tables such as *DELEGATED_INSERT_AGENTS* or *DELEGATED_DELETE_AGENTS* could be created for finer granularity. Such a scheme could simplify the logic of CUD operations.

If transitive privileges are required, the simple schemes outlined above may not be sufficient. In that case defining explicit tables for roles, delegated access control and associated entities may be required.

Unauthorized Web service requests should always return the standard error fault - `notAuthorised`.

4.3.2.3 Certificate Based Authorisation

Role-based permissions are probably easiest to implement when the underlying data model uses relational databases. However, there are alternatives to role-based updates. One of these is to grant CUD privileges based on certificates.

If a particular entity signs a Web service request with a private key corresponding to a supported certificate (or uses a supported certificate in a SSL handshake using mutual authentication), that entity would be granted permission to update the records for an associated target entity. For this to succeed, it would be necessary to ensure a representative of an HPIO entity contacts the SIL provider with a list of certificates authorized to update SIL records on its behalf. This could be done by:

- providing a Web service operation whereby the HPIO entity can submit certificates authorizing update operations on its behalf, or
- providing a Web form allowing an HPIO entity to submit certificates that can be used for update requests on its behalf.

Internally, these solutions rely on the SIL provider maintaining a set of certificates which are allowed to sign update operations for each HPIO. If the list is empty, only certificates issued to the organisation corresponding to the HPIO itself would be allowed to execute the operation.

Unauthorized Web service requests should always return the standard error fault `notAuthorised`. This is a course-grained approach, but it could be refined by maintaining separate lists for varying levels of access control.

5 Maintenance

5.1 Extensions to the Standard Interface

Standard SIL update operations are inadequate even for storing all the required associations. For example there is no operation to associate a SIL with a HPIO or any other target identifier. This means that SIL implementers are free to choose their own methods to create the associations.

5.1.1 Associating Primary Identifiers

Depending on how many HPIOs or other identifiers are intended to be associated with a particular SIL, operations to add and remove them may not be necessary. If there are only a few associated primary identifiers, updating a configuration file may be sufficient for the purpose. If many hundreds or thousands were anticipated, a relational database would be more appropriate.

It is up to the implementer to decide the form, if any, of operations to associate/disassociate top level identifiers. An obvious choice would be to provide a Web form for a user to complete. Of course, a SIL cannot use the certificate of some entity trying to associate its own identifier unless prior arrangements have been made to trust the certificate used to sign requests.

5.1.2 Lower-level Updates

Update operations of the SIL specification are limited to adding and deleting an entire interaction. The semantics of this are unspecified but the implementers need to decide on a strategy for updating individual tables.

URIs for providers, interactions, service categories, roles, and interfaces should be able to be updated directly. The most practical strategy would be to disallow implicit insertions of URIs on insertion of an interaction. If a URI contained in an input interaction of some kind cannot be located, the implementation should return the standard error fault `badParam` (or `badlyFormedMsg` if the URI is malformed). Similarly, URIs should not be implicitly removed by a delete operation.

It should be possible to add qualified certificate references independently of a standard insert request since different services may reference the same certificates. Similarly, it should be possible to add services independently, since different interactions can refer to the same service. However, if services are not present in the database, they should be added with a new interaction.

When an interaction is added and its referenced services and/or certificates are already present in the database, the insert operation boils down to linking the pre-existing elements through updates to tables *interact*, *interact_svc* and *svc_qcr*.

5.1.3 Bulk Interfaces

If a provider directory has a large number of existing interactions and wants to transfer those to a SIL, a bulk interface of some kind may be in order. Usually, there are two methods of bulk loading a database.

1. Source a SQL file containing insert and update statements, possibly constructed by adapting a dump of the first database. This could entail high overhead if the schema of the first database is significantly different from the schema of the target database.
2. Load the database tables from comma separated value (CSV) files. It is probably necessary to write a script (e.g. sh, bash, Python, Perl, PHP) to

coalesce all the CSV files around the defined database schema and sequence calls to the database's load utilities.

If adopting a schema similar to that suggested in this document, both these approaches may be difficult due to the reliance on foreign keys. It may be advisable to create stored procedures to link individual records through the intersection tables to be repeatedly called by the scripts.

5.1.4 Non-Web service Updates

It is not necessary that non-standard updates are Web services and it is envisaged that the most maintenance will be carried out by means of a custom application or, more likely, a browser-based interface. If Web services are not being used, the Web service profile does not apply and there must be an alternate means of securing messages.

In using a Web server application, communications should be secured using HTTPS with mutual authentication. Mutual authentication allows the server to authenticate the client (user) through their X.509 certificate. The certificate itself could be used for access decisions if certificate-based authorisation was being employed (see 4.3.2.3).

If role-based authorisation was employed, as is more likely, the entity would be resolved using the certificate and the specific access checked using the appropriate *DELEGATED_** table (see 4.3.2.2). (There is no need to use the schemes described in 4.3.2.2 - permissions may be stored in specific privilege tables or configured through access control lists).

Alternatively, the entity could be authenticated at the application level over a HTTPS session, most likely through username and password. However, for a SIL associated with a large number of HPIOs there would be the additional overhead of maintaining identifying information, either in database tables or in a separate repository such as an LDAP directory. For a small SIL it would not make much difference which approach was taken. For a large SIL, the development, maintenance and storage overheads could become quite high.

5.2 Update Responsibility

If a source entity cannot locate a target's supported services through its SIL there is a possibility that one or more patients could be temporarily deprived of a healthcare service they may need. It is therefore very important that each provider's SIL remains up to date. Typically, it is the service owner that is responsible for ensuring a smooth maintenance cycle.

5.2.1 Owner and SIL Updates

An organisation is referred to as a *service owner* when it hosts some service for document exchange or such a service is hosted on its behalf. It is possible for a service to be redeployed as a fresh implementation without impacting its metadata, i.e. details stored in the SIL. Whenever any detail of a service is modified, including its service interface, supported category, role, endpoint, or certificates used in service invocation, it is the responsibility of the owner organisation to ensure the SIL is modified. Similarly, it is the responsibility of the owner to add or remove interactions when required.

If all services are outsourced, the outsourcing entity may find it more expedient to perform SIL updates. In that case, the owner must at least ensure that the outsourcer does in fact keep the SIL up to date. This implies a situation similar to that discussed in 4.3.2. Some strategy is required to deal with this situation by the SIL implementation. The service owner would need some process whereby it could delegate its maintenance responsibilities. At some point in the future, maintenance privilege may have to be removed from the outsourcer, for example because the owner enters into a different contract for services.

5.2.2 Changes Affecting HPIO

A healthcare provider organisation will be allotted a single HPIO, by which it will be identified in the E-health community. Some organisational changes will impact this identifier and its associated SIL entries, while others will have no effect. Listed below are some common circumstances and maintenance expectations.

Circumstance	Action
Healthcare provider ceases to practice/trade.	Unless another organisation takes over the old operations, associated SIL entries must be removed. Since the owner has ceased to do business, it will be incumbent on the SIL provider to delete the records. The HPIO must also be removed from the HPIO service. When this happens, reference to the organisation's SIL will also be deleted.
Provider is split into two (or more) entities.	Details could vary here. If the original HPIO continues to be associated with a provider that maintains its old operations no SIL updates may be necessary. The new organisation will set up its own SIL and add interactions as necessary. If the old services are to be divided between the new entities, the original entity must remove records that have become invalid. It is the new entity's responsibility to add its own SIL records. If existing services are to be maintained under the new entity's HPIO, there is no standard operation to accomplish the switch. The most suitable course of action would be to contact the SIL administrator (possibly the original organisation) to effect the change. In addition, a new HPIO record, including SIL details, must be made available from the HPIO service.
Provider is merged with another entity.	Assuming that both organisations have existing services, it is clearly the responsibility of the new provider entity to ensure the services of the old provider become associated with its HPIO. Because there is no standard operation (or authorisation strategy) to simply switch services to a new provider, it is probably a good idea to have the SIL administrator carry this out in the period leading up to the merge date. If records of the entity to be merged (whose HPIO is to be deleted) need to be concurrently maintained until the actual merge date, the updates should be scheduled to take place at the last moment. If required, the two entities could continue to exist at the HPIO level for some time but refer to a common service. This would require different <i>service</i> entries to refer to the same endpoint.
Service provider(s) are changed.	The service owner must remove the old records and add or arrange for the new provider to add the new ones.

Provider changes its SIL.	Provider is responsible for duplicating the entries for the new SIL and updating its HPIO record. If the old SIL instance is not being decommissioned, the provider should also be responsible for removing its records and arranging for its HPIO to be disassociated.
Core business changes.	This is unlikely, but would seem to entail old services being removed and new services being added. If the SIL instance changes the HPIO record must reflect that.
Organisation incorporates.	HPIO may change, in which case the new HPIO record must be arranged and the old one removed. Assuming the same SIL is used, the records must be switched over to the new HPIO. There is no standard operation for that, so it would be best to have the SIL administrator carry it out. If the HPIO remains unchanged, it is unlikely any change is required.
Acquisition occurs.	Similar to the situation where existing providers are merged. If the acquired organisation has no SIL and/or existing services, no action would be required in the short term.
X.509 certificate associated with a service is superseded, e.g. because the old certificate expires or is revoked.	As usual, it is the owner's responsibility to perform the update, in this case of the certificate references. If the service in question is provided by a third party, it may be better for the service provider to do the task. That would only be possible if the owner had delegated the necessary update privileges to the provider. There is no standard operation to change certificate references. An alternative interface (e.g. Web application) could be used if provided. Alternatively, out of band arrangements could be made with the SIL implementer. An almost standard way would be to remove the interaction entirely, remove the certificate reference, insert a new certificate reference (e.g. through a Web application) and finally re-insert the interaction and service(s). The number of steps this requires underlines the desirability of extending the update mechanisms.

6 Reliability

6.1 System Down-time

Like all systems, it will be necessary to bring down the SIL instance from time to time for server maintenance, software updates, etc. Despite care in administering application and database servers, etc., it is likely that system degradation or suspensions will occasionally take place, necessitating a system reset. Since there is only one SIL instance per healthcare provider, downtime implies a potential to impede or delay timely provision of healthcare services. It is therefore recommended to determine a strategy for mitigating down-time.

6.1.1 Clustered Application Server Environment

In a clustered environment there are two or more application servers running the SIL application, generally on different hosts. Typically both would connect to the same database. Each server in the cluster is delivered a request through a pass-through proxy. In normal mode, the proxy would route requests equally among the servers.

While the use of a configurable proxy is advisable, other solutions may be acceptable, e.g. using DNS round robin.

6.1.1.1 Rolling Maintenance Strategy

When the SIL application requires updating, the proxy would be temporarily configured to discontinue routing to one of the servers. After the appropriate maintenance is performed, e.g. a new version of the SIL service is deployed, the server re-enters the cluster and the proxy is set to direct fresh requests to the new service. The process is repeated for every server in the cluster. Since a SIL is not a session-based application, there should be no client-side issues in bringing down an instance. There should be no appreciable difference in response times if the absent server's load can be substantially carried by a different instance.

6.1.2 Failover Databases

A running SIL instance is little use if it cannot rely on its data stores. For SILs associated with a large number of HPIOs it may be beneficial to select a database with failover capabilities. For SILs associated with only a few HPIOs, the data store may be implemented in files, in which case only corruption of the file system (a rare occurrence) would cause a problem.

7 Network Configuration

Similar considerations apply to those documented in the Security section of [CIG2008]. See that document for more information.

8 Existing Provider Directories

As outlined in the SIL Architecture [SA2008], certain technical service providers have built directories for healthcare clients that have similar capabilities to a SIL. They list services and endpoints for healthcare providers and in addition may coordinate message exchange. Messaging often takes the form of secure email, where documents are exchanged as S/MIME attachments.

Some of these provider organisations may wish to extend their capabilities by also implementing a SIL. However, if the provider directory coordinates document exchange in one particular category, there could be issues if the healthcare client participates in exchanges for other categories. This is because a healthcare provider may only be associated with one SIL, whereas there is no such restriction for third-party directories.

If a healthcare provider is referenced by more than one directory and both directory providers wish to implement a SIL, there would have to be negotiation as to which SIL the healthcare provider would be associated with. The decision would likely be based on business considerations. Association with more than one directory can continue as before, but lack of consistency may be the outcome of several agencies maintaining related data.

Directory providers must ensure that any services they refer to are also referenced by the SIL. It may be necessary for the service owners to register their directory provider organisations as trusted entities in their SIL (see 4.3.2) if that feature is supported.

Appendix A: Informative references

- [CIG2008] NEHTA, *Connectivity: Implementation Guide v1.0*, 1 December 2008.
- [EIIWSJAXWS] NEHTA, *Example Implementation of Interoperable Web Services: JAX-WS v2.0*, 1 December 2008.
- [EIIWSWCF] NEHTA, *Example Implementation of Interoperable Web Services: WCF v2.0*, 1 December 2008.
- [EIIWSWSE] NEHTA, *Example Implementation of Interoperable Web Services: WSE v2.0*, 1 December 2008.
- [IF2007] NEHTA, *Interoperability Framework v2.0*, 17 August 2007.
- [SILR2008] NEHTA, *SIL Requirements v1.1*, 1 December 2008.