nehta

Endpoint Location Service

Solution Design

Version 1.3 — 15 November 2010

National E-Health Transition Authority

National E-Health Transition Authority Ltd
Level 25
56 Pitt Street
Sydney, NSW, 2000
Australia.
www.nehta.gov.au

Table of contents

Document information

Change history

Version	Date	Comments
1.3	2010-11-15	First release

This page is intentionally left blank

1 Introduction

1.1 Background

In a service-oriented e-health environment, tasks are performed by service invokers that invoke operations on service instances. For example, a *client program* making SOAP Web services calls on a *target service instance*. The national e-health environment uses service invocations to facilitate communications between different healthcare organisations.

In a national e-health environment *client programs* in one organisation need to know how to invoke the *target service instances* provided by other organisations. There is a scalability problem, since there are many *target service owners* and many *client programs*. There is also a maintenance problem, since the set of *target service instances* changes over time. It is not practical to inform every *client program* about every *target service instance*. The Endpoint Location Service provides a solution to this problem.

The Endpoint Location Service is a mechanism to allow *client programs* to get the technical information needed to invoke the *target service instance*. This can be done when the *target service instance* is needed, which addresses the scalability and maintenance problems.

1.2 Purpose

This is the solution design for the Endpoint Location Service (ELS).

This document contains the solution design from the business architecture, information architecture and technical architecture perspectives.

1.3 Scope

This document contains a technology independent description of the architecture. It does not provide the technology specific details for the service interface of ELS.

The technology specific details for a SOAP Web services ELS are defined in the *Endpoint Location Service: Technical Service Specification* [ELSTSS2010] and *Endpoint Location Service: WSDL and XML Schema files* [ELSWXS2010].

This solution design satisfies the requirements from [ELSR2008] and is derived from material in Standards Australia *TR 5823—2010 Endpoint Location Service* [TR5823—2010].

1.4 Intended Audience

This document is intended for:

- Architects who are designing solutions which involve ELS.
- Developers who are implementing software related to ELS.

1.5 References

[CPIS2010] NEHTA, *Concepts and Patterns for Implementing Services v2.1*, 30 June 2010.

[ELSR2008] NEHTA, *Endpoint Location Service: Requirements v1.1*, 1 December 2008.

[ELSTSS2010] NEHTA, *Endpoint Location Service: Technical Service Specification v1.3*, 15 November 2010.

[ELSWX2010] NEHTA, *Endpoint Location Service: WSDL and XML Schema files v1.3*, 15 November 2010.

[TR5820—2010] Standards Australia, *TR 5820—2010 Endpoint Location Service*, Technical Report, 5 March 2010.

1.6 Definitions

Certificate use
: Value that is associated with semantics about how a certificate is used when invoking a service interface.

Client program
: Service invoker that wishes to invoke a *target service instance* of a particular *target service owner*. It uses an *ELS service instance* to obtain an *interaction record* that it uses to invoke a *target service instance*.

ELS service instance
: Service instance that makes *interaction records* available to *client programs* and they are maintained by a *management program*.

ELS service operator
: Organisation that is responsible for the technical operation of the *ELS service instance*.

ELS service owner
: Organisation that is responsible for the *ELS service instance*.

Interaction record
: Technical information needed to invoke a *target service instance*.

Management program
: Service invoker that is used by the target service owner (or its delegate) to maintain the *ELS service instance*.

Service category
: Value that is associated with semantics about the business aspects of a service interface.

Service interface
: Value that is associated with semantics about the technical aspects of a service interface.

Target service instance
: Service instance that the *client program* wishes to invoke.

Target service operator
: Organisation that is responsible for the technical operation of the *target service instance*.

Target service owner
: Organisation that is responsible for the *target service instance* and is the organisation that the *client program* wishes to contact.

1.7 Acronyms

ELS	Endpoint Location Service

1.8 Overview

The business perspective is documented in chapter 2.

The information perspective is documented in chapter 3.

The technical perspective is documented in chapter 4.

2 Business Perspective

2.1 Community

The ELS community consists of the following software roles:

- *Client program*
- *Target service instance*
- *ELS service instance*
- *Management program*

And the following organisation roles:

- *Target service owner*
- *Target service operator*
- *ELS service owner*
- *ELS service operator*

This document focuses on defining the software roles. The organisation roles are included to allow this document to refer to processes which are out of scope for this document to define, but which are helpful for understanding how ELS works in the wider e-health environment.

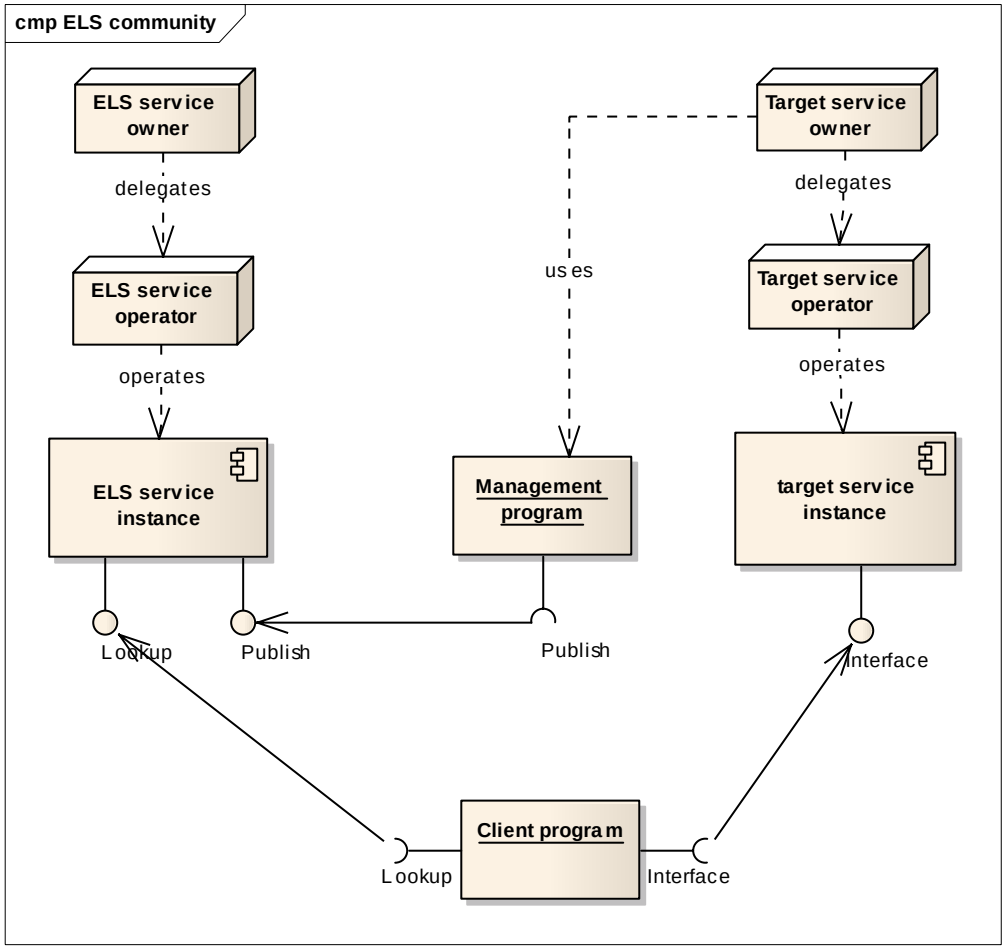

Figure 1 ELS community diagram

2.1.1 Client program

The *client program* is a service invoker that needs to invoke an operation on a *target service instance* associated with a particular *target service owner*.[1]

2.1.2 Target service instance

The *target service instance* is a service instance that provides the operation that the *client program* needs to invoke.[2]

2.1.3 ELS service instance

The *ELS service instance* is a service instance that provides the ELS lookup service and the ELS publish service.

It makes *interaction records* available to *client programs*. *Interaction records* contain the technical information that a *client program* needs to invoke an operation on the *target service instance*, and is defined in section 3.1.

2.1.4 Management program

The *management program* is a service invoker that is used to add and/or remove *interaction records* from an *ELS service instance*.

2.1.5 Target service owner

The *target service owner* is the organisation that the *target service instance* is associated with.

For example, the *target service owner* can be a healthcare provider organisation.

2.1.6 Target service operator

The *target service operator* is the organisation that operates the *target service instance*.

If the *target service owner* runs their own *target service instance*, then the *target service operator* is the same as the *target service owner*. If the *target service owner* outsources the *target service instance*, then it will be a different organisation.

For example, the *target service operator* can be an Information Technology services company contracted by the *target service owner*.

2.1.7 ELS service owner

The *ELS service owner* is the organisation that owns the *ELS service instance*.

If the *target service owner* runs their own *ELS service instance* (which is possible under some of the owner-operated distributed models described in Appendix A) the *ELS service owner* is the same as the *target service owner*.

---

[1] The term “service invoker” refers to a party that invokes a service instance by sending it a service request. The term comes from [CPIS2010].

[2] The term “service instance” refers to a specific deployment of a service implementation. The term “service implementation” refers to a product (i.e. software) that conforms to a service interface. The term “service interface” refers to the definition of the functionality of a service. These terms comes from [CPIS2010].

2.1.8 ELS service operator

The *ELS service operator* is the organisation that operates the *ELS service instance*.

If the *ELS service owner* runs their own *ELS service instance*, then the *ELS service operator* is the same as the *ELS service owner*. If the *ELS service owner* outsources the *ELS service instance*, then it will be a different organisation.

2.2 Processes

This section documents four processes:

- Target service invocation process;
- Validate interaction record process;
- Add interaction record process; and
- Remove interaction record process.

2.2.1 Target service invocation process

2.2.1.1 Introduction

In this process a *client program* invokes an operation on a *target service instance*. It uses ELS to obtain the technical information that is needed to perform the invocation.

For example, a pathology laboratory information system (the *client program*) wishes to deliver a pathology report to an identified General Practice Clinic (the *target service owner*). It wishes to invoke the pathology report receiving operation where that General Practice Clinic expects their pathology results to be delivered (the *target service instance*).

2.2.1.2 Prerequisites

The *client program* has:

- Identifier of the *target service owner*.

  As a part of the wider business process the *client program* is performing, it will have identified the organisation it needs to communicate with.

- Category of service the client needs to invoke.

  As a part of the wider business process the *client program* is performing, it will have identified the *service category* values it needs.

- Implementation of the ability to invoke certain service interfaces.

  The *client program* is capable of invoking one or more service interfaces. This will have been determined by the developer of the *client program* (or in some cases also during deployment of the *client program*) and will correspond to *service interface* values.

- Technical information about how to invoke the lookup service on an *ELS service instance* that contains *interaction records* for the *target service owner*.

  For convenience, the “technical information about how to invoke the lookup service on an *ELS service instance*” will be called an “ELS interaction record” (section 3.2). An *ELS interaction record* is information about how to invoke an *ELS service instance*, whereas a normal *interaction record* is information about how to invoke a *target service instance*.

Determining which *ELS service instance* contains *interaction records* for the *target service owner* is out of scope for this document to define, because it depends on the deployment model being used (Appendix A).

2.2.1.3 Process

1. *Client program* sends a list interactions request to the *ELS service instance*. The request contains the identity of the *target service owner*, the *service category* the *client program*s needs to invoke, and the *service interfaces* the *client program* supports.
2. The *ELS service instance* sends a response to the *client program*. The response contains a list of *interaction records* that matches the request.
3. The *client program* chooses an *interaction record* from the list.
4. The *client program* invokes the operation on the *target service instance* as described by the *interaction record*.

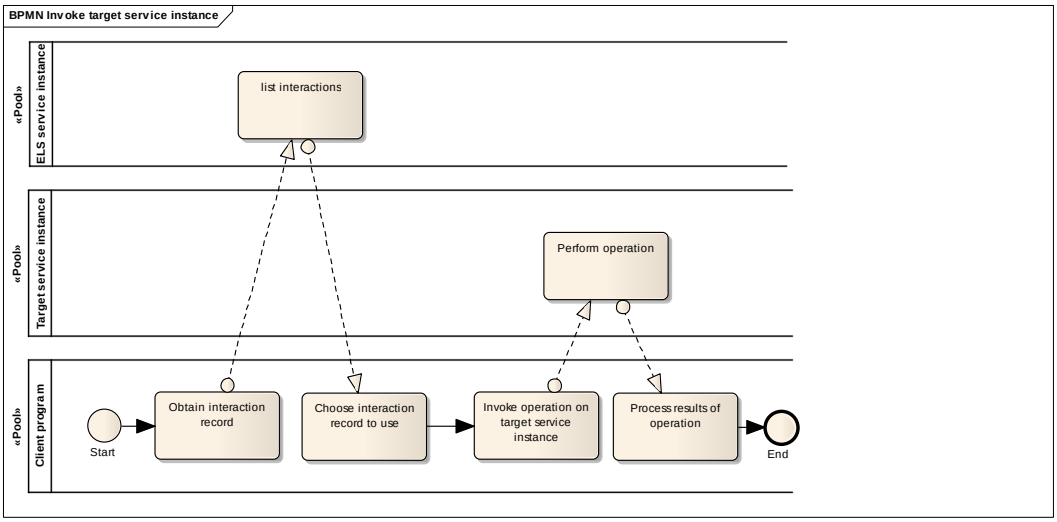

Figure 2 Target service invocation process

2.2.1.4 Alternative paths

2.2.1.4.1 Alternatives to ELS

The *client program* is responsible for obtaining the *interaction record* (or equivalent information) before being able to invoke the operation on the *target service instance*. Although the main focus of this document is the use of ELS, a *client program* is permitted to use other mechanisms instead of, or in conjunction with, ELS.

For example, the *interaction record* could be obtained from the *client program*'s cache (see section 2.3.2 on the caching of *interaction records*). Alternatively, the information might have been manually entered into the *client program*. The ELS could also be used as a final alternative mechanism, when the *interaction record* cannot be obtained by other means.

2.2.1.4.2 No suitable service interface

If zero *interaction records* are listed, the *client program* will be unable to invoke the *target service instance*.

The *client program* will have to handle it in a way that is appropriate for the wider business process it is trying to accomplish. Some business process might have specified alternative steps to follow (including failure), others might ignore it (because the invocation was optional to the business process).

2.2.1.5 Notes

If the list interactions operation returns more than one *interaction record*, the *client program* needs to choose which one(s) it invokes.

This situation could arise because the *target service owner* has multiple service instances to support different service interfaces for the same *service category*. Alternatively, the *target service owner* might be operating multiple service instances (with the same *service interface* and *service category*) to provide redundancy. In both these situations, the *client program* is always free to choose any one (and possibly only one) of the returned *interaction records*.

2.2.2 Validate interaction record process

2.2.2.1 Introduction

If the invocation of the *target service instance* fails, this could be because the *interaction record* was stale.

An *interaction record* is either valid or it is stale. A stale *interaction record* is one that has been removed by the Remove Interaction Record process (section 2.2.4). Usually it is removed because the *target service owner* does not want *client programs* to be using it anymore.

An *interaction record* could be stale because the *client program* has obtained it from its own cache.

An *interaction record* could also be stale because the value returned from a list interactions operation was stale. This is important to recognise: an *ELS service instance* provides no guarantee that the *interaction records* it returns are valid. This behaviour is permitted to allow the ELS service implementation to also use caching, and (from a practical point of view) can never be guaranteed due to race conditions (e.g. the *interaction record* was removed immediately after it was listed).

2.2.2.2 Prerequisites

The *client program* has:

- *Interaction record* to check; and
- *ELS interaction record*.

2.2.2.3 Process

1. The *client program* invokes the validate interaction operation on the *ELS service instance*.
2. The *ELS service instance* sends a validation status (true or false) to the *client program*.
3. If the response indicates that the *interaction record* was stale, the *client program* repeats the Target Service Invocation process (section 2.2.1). This time the *ELS service instance* should not include that stale *interaction record* in the list it returns.[3]

[3] There are some obscure scenarios where the same stale *interaction record* is listed again. For example, if that *interaction record* was added back and then removed again (both occurring after the validation operation, but before the list operation).

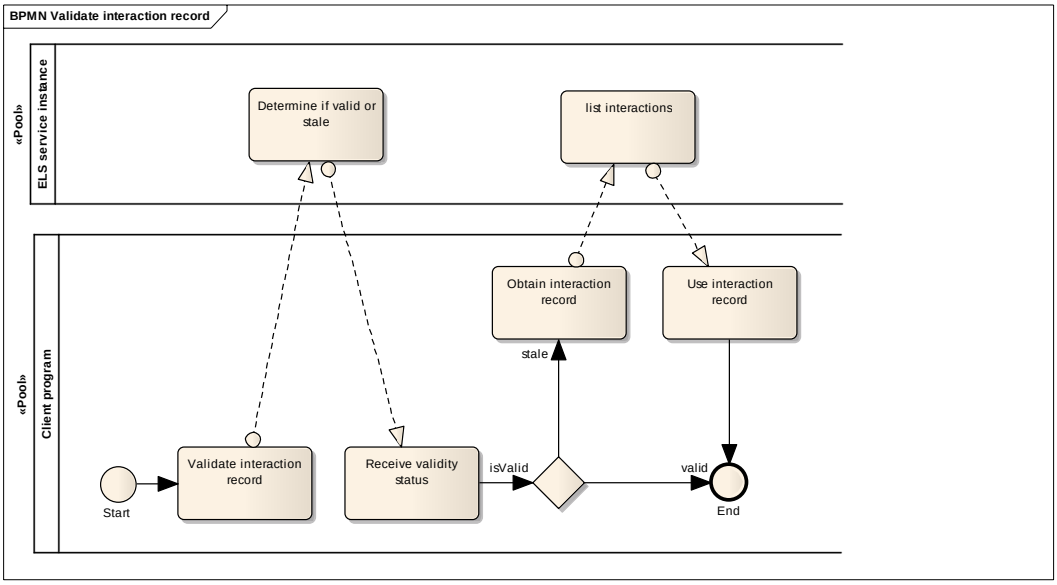

Figure 3 Validate interaction record process

2.2.2.4 Alternative paths

2.2.2.4.1 Interaction record was valid

If the response indicates that the interaction record was valid, then the client program follows the process required by the wider business process. For example, it could wait and retry the invocation or abort.

2.2.3 Add interaction record process

2.2.3.1 Introduction

In this process the *target service owner* adds an *interaction record* to the *ELS service instance* that *client program*s will use.

This process can be used when the *target service owner* deploys a new *target service instance*, or modifies an existing *target service instance*.

2.2.3.2 Prerequisites

The *target service owner* needs to know which *ELS service instance* (or possibly more than one instance) that the *interaction record* needs to be added to. This depends on the deployment model used (see Appendix A).

The *target service owner* must have established a business relationship with the *ELS service owner* so that they are registered in that *ELS service instance*.

The *ELS service owner* will have registered the *target service owner* with the *ELS service instance*.

2.2.3.3 Process

1. The *target service owner* provides the *interaction record* information to a *management program*.
2. The *management program* invokes an add interaction operation on the *ELS service instance*.

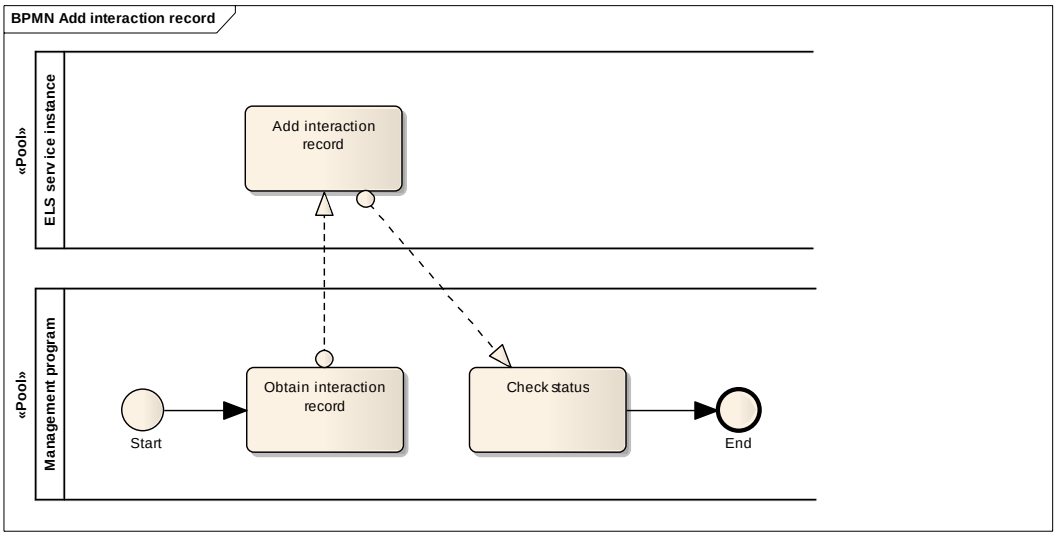

Figure 4 Add interaction record process

2.2.3.4 Alternative paths

2.2.3.4.1 Delegation

The *target service owner* can delegate this process to another party.

For example, the *target service instance* might be implemented with the ability to automatically update an *ELS service instance* when it is deployed or configured—this ability could help ensure that the ELS is always up to date. Alternatively, the *target service operator* might perform this process when they deploy or modify a *target service instance*. It is even possible that the *ELS service owner* or *ELS service operator* performs this process, upon instruction from the *target service owner*.

2.2.4 Remove interaction record process

This process is similar to the Add Interaction Record process, except that an *interaction record* is removed instead of added. See section 2.2.3 for the details.

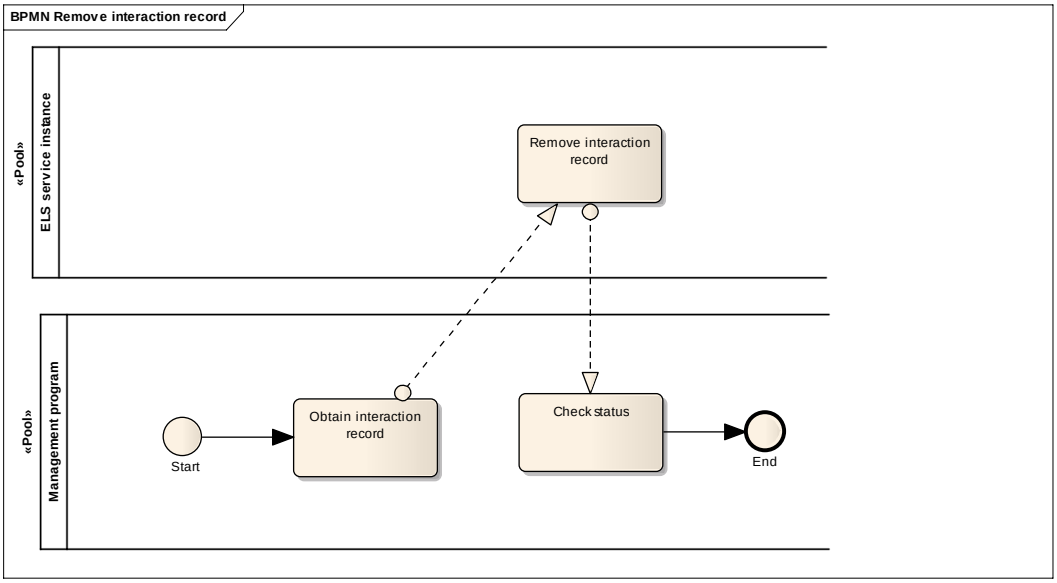

Figure 5 Remove interaction record process

2.3 Validation

2.3.1 Proxy implementations

This section introduces the possibility of proxies. Proxies are an implementation and deployment issue that is outside the scope of this specification to define, but is described as background for describing caching.

ELS is designed to allow *ELS service instance*s to be proxies, where the *ELS service instance* invokes other *ELS service instance*s to perform its function.

For example, consider an *ELS service instance* operated by a large hospital. *Client programs* from inside that hospital send all their list interactions operations to the hospital's *ELS service instance*. If the list interactions request is for a department inside that hospital, it is returned immediately. If the list interactions request is for an external organisation, then the hospital *ELS service instance* invokes the list interactions operation on an external *ELS service instance* to obtain the results which it then returns to the *client program*. This might be done to improve performance and security for its *client programs*.

The use of proxies is an implementation and deployment issue. As such, this document does not require, nor prohibit, its use.

The processes described in this document do not change in the presence or absence of proxies.

2.3.2 Caching of interaction records

This section introduces the concept of caching. It is not intended to be a complete discussion of how to implement and deploy *ELS service instances*, but as background to describing the *interaction record* validation mechanism and why it is needed.

Caching is an implementation and deployment issue, but it needs to be explicitly defined as a part of the design of ELS for it to operate properly.

The use of caching is encouraged: to improve performance, robustness and scalability. Performance is improved because an implementation does not have to invoke a list interactions operation on an *ELS service instance* for every service invocation. Robustness is improved in the unlikely event that the list interactions operation cannot be performed. Scalability is improved because the *ELS service instance* can receive fewer list interactions operations.

Client programs can cache *interaction records*.

*ELS service instance*s can also cache *interaction records* (especially if they are implemented as an ELS proxy).

All implementations need to be aware of caching, otherwise they could be incorrectly interpreting results.

2.3.2.1 Cache miss

If a *client program* does not already have an *interaction record* that it can use, it can perform a list interactions operation on an *ELS service instance* to obtain it.

2.3.2.2 Saving to the cache

If it implements caching, the *client program* should save *interaction record*s in its cache for future use.

Interaction records do not have an expiry time. This is a deliberate design decision, because it is expected that most *target service instance*s will be deployed with the intention of running them on a continuing basis. So there is

rarely any basis for determining when an *interaction record* will become stale. Any expiry time would usually be a guess and provide no value to the *client program*. Implementations are free to use a cache management algorithm that is appropriate to their environment (this includes guessing their own local expiry time if they want to use a time based algorithm).

2.3.2.3 Cache hit

If the *client program* has a suitable cached *interaction record* it should try to use it, instead of performing a list interactions operation on an *ELS service instance*.

If the invocation using a cached *interaction record* fails, the *client program* can validate whether that cached *interaction record* is stale or valid—using the Validate Interaction Record process (section 2.2.2).

The error handling process of the client program needs to take into account of a number of possible problems:

- The target service instance is (perhaps temporally) not functioning;
- The interaction record is stale; and
- In some deployment scenarios, the target service owner has changed to a different ELS service instance (Appendix A).

2.4 Not in scope

The *ELS service instance* is not designed to be a general purpose service discovery mechanism. The *ELS service instance* is designed to provide a minimal set of operations to satisfy the requirements. Unnecessary features are not included to limit the complexity and cost of implementing ELS.

The *ELS service instance* does not support service invocations where the *target service owner* is unknown.

The *ELS service instance* is not a healthcare provider directory. *Client programs* must have identified the *target service owner* (usually a healthcare provider organisation) before they can use ELS.

The *ELS service instance* is not a service discovery mechanism. *Client programs* have to have identified the *target service owner*, *service category* and *service interfaces* it needs to invoke. *Client programs* could use a separate service discovery mechanism to obtain that information before using the *ELS service instance*.

Interfaces which do not have to be standardised are not defined as a part of this solution design. For example, implementations are allowed to provide their own solutions for registering a *target service owner* with an *ELS service instance*.

3 Information Perspective

3.1 Interaction record

An *interaction record* represents the technical information that a *client program* requires to invoke an operation on the *target service instance*.

It contains the following information:

- Identity of the associated *target service owner*;
- The *service category*;
- The *service interface*;
- Address of the operation on the *target service instance*;
- Identity of the *target service operator*;
- Security credential(s) for the operation on the *target service instance*.

3.1.1 Identity of target service owner

This identity is used to identify the *target service owner*.

3.1.2 Service category and service interface

The *service category* is a value that is associated with semantics about the business aspects of a service instance.

The *service interface* is a value that is associated with semantics about the technical aspects of a service instance.

The associated semantics need to be documented, so that it is interpreted in the same way by the *target service owner* and the *client program*.

For example, there could be a *service category* value to indicate a service for receiving a pathology report and a different value for receiving a discharge summary. There could be a *service interface* value to indicate a SOAP Web services prescription receiving interface that uses TLS for security.

3.1.2.1 Assigning values

It is expected that the author of service specifications will assign both *service categories* values and *service interface* values for each service interface defined by their specification. That way, the values are made known to both the *target service instance* and the *client program* (since they both have to implement the same specification).

3.1.2.2 Treat them as opaque values

The *service category* and *service interface* values are values that identify specific semantics.

Avoid trying to embed all the semantics into the identifiers. Services have many different aspects that need to be described, making it impossible to represent all of them inside the value.

Client programs are expected to treat the *service category* and *service interface* values as opaque strings; they are not expected to parse them to extract semantics. Leave the semantics to the service specification documentation where it can be clearly defined.

3.1.2.3 Service category vs service interface

The ELS provides a two level mechanism for identifying a specific service through the combination of *service category* and *service interface*. The two

level approach was chosen as a compromise between the limitations of having just a single level and the complexity of supporting more levels.

The guidelines are: *service category* for identifying business semantics, and *service interface* for technical semantics. But it is up to the specification author to decide how they are assigned.

Some semantics are obvious. The difference between a pathology service and a discharge summary service is related to business processes, so they are assigned different *service category* values. The difference between a TLS secured service interface and a WS-Security secured interface is technical, so they are assigned different *service interfaces* values.

Some semantics are not so obvious. For example, is using different clinical terminologies in the same data structure a different business process or a technical change? Is a priority response that has different quality of service guarantees from a normal service a different business process or a technical change? Sometimes the decision will be guided by the wider business process.

The specification author, who assigns values for the *service categories* and *service interfaces*, will have to decide. Whatever is chosen will work with ELS, because the *client program* only treats them syntactically (the pair of values just have to be equal to a pair that it supports) and is not concerned with how the semantics have been divided between the two values.

3.1.3 Address

This is the address for invoking the *target service instance*.

For SOAP Web services, this will be a Uniform Resource Locator (URL).

Although ELS was developed to support *target service instances* that are SOAP Web services, other types of services can be supported by ELS. Those other types of services will be distinguished by their *service category* and *service interface* values.

3.1.4 Identity of target service operator

This identity is used to identify the *target service operator*.

It is provided so that the *client program* can identify the organisation that is responsible for the technical operation of the *target service instance*.

A possible use of this value is to allow the *client program* to use out-of-band means to telephone them if there was a technical problem involving the *target service instance*.

3.1.5 Security credentials

The ELS is designed to support *target service instances* that use public key cryptography to provide confidentiality. Therefore, to invoke an operation a *client program* might need to identify suitable public keys to encrypt data.

The *interaction record* contains zero or more references to public keys. These references to public keys refer to X.509v3 public key certificates.

Note: These certificates are necessary in the *interaction record*, because otherwise a *client program* does not have any reliable mechanism for determining the correct public keys to use. For example, a *target service owner* might have been issued multiple certificates, so the *client program* does not know which one is expected for a particular operation. Also, some *target service owners* might have delegated the operation of the *target service instance* to a *target service operator*, so those different certificates might be needed by some *target service instances*.

3.1.5.1 Certificate use

Each certificate needs to be associated with a *certificate use* value which is associated with semantics about how the certificate is used when invoking the *target service instance*.

Common certificate use values will be defined to indicate common ways of using certificates. The author of service specifications can also assign *certificate use* values, if their service interface requires certificates for purposes not covered by the common values.

3.1.5.2 Certificate examples

Here are some examples of the number and usage of certificates in an *interaction record*.

These examples are based on the current known set of services which optionally uses XML Encryption to encrypt payloads that are sent over either Transport Layer Security (TLS) or WS-Security.

These are the certificates that the *client program* requires:

- With XML Encryption an encrypting certificate is needed to encrypt the payload.
- With WS-Security an encrypting certificate is needed to encrypt the SOAP message.
- With TLS no certificate is required, because the TLS server certificate for the *target service instance* is sent to the *client program* as a part of establishing the TLS connection.

Therefore, these are some possibilities:

- A SOAP Web services operation that is only secured using TLS will have zero certificates in the *interaction record*.
- A SOAP Web services operation that is only secured using WS-Security will have one certificate in the *interaction record*. That certificate will be used for encrypting the SOAP request message using WS-Security.
- A SOAP Web services operation that receives an XML Encryption encrypted payload over TLS will have one certificate in the *interaction record*. That certificate will be used for encrypting the payload.
- A SOAP Web services operation that receives an XML Encryption encrypted payload over WS-Security will have two certificates in the *interaction record*. One will be the certificate will be used for encrypting the payload. The other will be the certificate used for encrypting the SOAP request using WS-Security.

More complicated operations might involve encrypting for different parties, and could require additional certificates to be stored in the *interaction record*. Although currently it is expected that a maximum of two certificates is required, an implementation of ELS should not have that restriction.

3.2 ELS interaction record

The term “ELS interaction record” is used in this document to mean the technical information that a *client program* needs to invoke an *ELS service instance*.

This information does not actually exist in any *interaction record*, because it is not obtained from an *ELS service instance,* but from a source determined by the deployment model (see Appendix A).

The *ELS interaction record* mainly refers to the address and any certificates needed. Depending on the deployment model, the other values in the *interaction record* are probably not applicable.

4 Technical Perspective

4.1 Overview

An implementation of an *ELS service instance* provides at least two services:

- Lookup service; and
- Publish service.

An *ELS service instance* will also have to provide a service to allow *target service owners* to be registered, but that can be agreed upon between the *ELS service operator* and the *ELS service owner* (which might be the same organisation).

This chapter is brief because the behaviours of these services and operations have already been described in Chapter 2.

4.2 Lookup service

The lookup service allows *client programs* to obtain *interaction records* about a particular *target service owner*.

The lookup service provides two operations:

- List interactions; and
- Validate interaction.

4.2.1 List interactions

Inputs:

- *Target service owner*;
- *Service category*;
- *Service interface*.

Output:

- List of matching *interaction records* (which could include stale *interaction records as well as valid ones*).

4.2.2 Validate interaction

Inputs:

- *Interaction record*.

Output:

- Boolean status indicating whether the *interaction record* was stale or valid.

4.3 Publish service

The publish service allows *management programs* to add and/or remove *interaction records* from an *ELS service instance*.

The publish service provides two operations:

- Add interaction; and
- Remove interaction.

4.3.1 Add interaction

Input:

- *Interaction record* to add.

Output:

- Status indicating whether the add interaction operation succeeded or not.

4.3.2 Remove interaction

Input:

- *Interaction record* to remove.

Output:

- Status indicating whether the remove interaction operation succeeded or not.

Appendix A : Deployment models

This solution has been designed to support a number of different e-health environment deployment architectures.

The design and operation of an *ELS service instance* does not change between the different deployments. What does change is how the *client program* obtains the *ELS interaction record*—the technical information it requires to invoke an *ELS service instance*. Also, the relationship between the *target service owner* and the *ELS service instance* will differ between the different deployment models.

There are two factors to consider in a deployment model: the ownership and distribution of the *ELS service instance*(s).

A.1 Ownership

A.1.1 Owner operated

With owner operated *ELS service instances*, the *target service owner* is also the *ELS service owner*.

A.1.2 Delegated

With delegated *ELS service instances*, the *target service owner* is different from the *ELS service owner*. The operation of the *ELS service instance* has been outsourced.

A.2 Distribution

A.2.1 Distributed

With a distributed model, there are many different instances of the *ELS service instance* in the e-health environment.

The most extreme situation would be to have a different *ELS service instance* for each *target service owner*.

If a distributed model is used, *client programs* will need to handle situations where a *target service owner* changes which *ELS service instance* is associated with them. This is especially important if they cache the *ELS interaction records*, as well as the ordinary *interaction records*.

A.2.2 Centralised

With a centralised model, there is a single *ELS service instance* in the entire e-health environment.

The centralised ELS might be implemented using different distributed techniques. For example, it could be mirrored (where there are several instances and they contain exactly the same data) or federated (where there are several instances that can use the data on the other instances if the request cannot be satisfied using local data).

The main feature of the centralised model is that *client programs* know—in advance—the *ELS interaction record* for every *target service owner*. There is only one *ELS service instance*, so every *target service owner* can be found in it.

A.2.3 Limited

With a limited distribution model, there are a small number of instances of ELS in the e-health environment. Small being defined as a number that makes it feasible to populate and maintain every *client program* with *ELS interaction records* for every one of them.

The *client program* can try each of the *ELS service instances* to find the one that has the *target service owner* in it, since there are only a small number to try. A partitioning algorithm may help reduce the number of tries required (e.g. each *ELS service instance* contains *target service owners* based on geographic location).

A.3 Possible deployment models

The actual deployment model that the e-health environment will use was not determined at the time this document was written.

Two possible deployment models were being preferred:

- A distributed model (supporting both owner operated and delegated ELS) using the NEHTA Healthcare Identifiers (HI) services to provide the *ELS interaction record* for a particular *target service owner* (i.e. HPI-O).
- A centralised model (delegated, since every *target service owner* will not be operating their own ELS) with a single *ELS service instance* operated for the entire nation.

There could also be a hybrid model, where more than one deployment model is being used.

Appendix B : Dependencies

B.1 Identifiers

The ELS depends on the *target service owners* and *target service operators* being identified.

Although there could be different types of identifiers in the healthcare environment, the *interaction record* in the *ELS service instance* must use the same identifiers that the *client program* will use to reference the same *target service owner*.

It is expected that Healthcare Provider Identifiers for Organisations (HPI-O) numbers will be used to identify the *target service owners*.

B.2 X.509v3 certificates

The *interaction records* are designed to support X.509v3 certificates.

It is expected that National Authentication Service for Health (NASH) certificates will be the certificates used in the *interaction records*.

B.3 Service categories and service interfaces

The *service categories* and *service interfaces* correspond to particular services.

It is expected that the technical service specification for each different service would define values for the *service categories* and *service interfaces*.

B.4 ELS interaction records

If a distributed distribution model is used (see Appendix A) it is expected that the NEHTA Healthcare Identifiers (HI) services will be used to provide the *ELS interaction records* of each *target service owner*.

Appendix C : Change log

Version 1.3

- First release.

  This document is a refinement of the *Endpoint Location Service: Architecture*, which was incorporated into the Standards Australia *TR 5823—2010 Endpoint Location Service*. To avoid confusion with those earlier documents, the first release of this Solution Design document was version 1.3.