

nehta

Concepts and Patterns for Implementing Services

Version 2.0 draft — 1 September 2008

Draft for comment

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	7
1.1 Background	7
1.1.1 Document history	7
1.2 Purpose	7
1.3 Scope	7
1.3.1 Content	7
1.3.2 Audience	7
1.3.3 Out of scope	8
1.3.4 Lifecycle	8
1.4 Status	8
1.5 Normative references	8
1.6 Definitions, acronyms, abbreviations	8
1.7 Conformance	9
1.8 Overview	9
2 Service concepts	10
2.1 Technical service definitions	10
2.1.1 Technical service	10
2.1.2 Service interface	11
2.1.3 Service implementation	12
2.1.4 Service instance	12
2.2 Synchronous and asynchronous services	13
2.3 Service specifications	13
2.3.1 Publishing	13
2.3.2 Conformance and certification	14
2.3.3 Interpretation	14
3 Patterns	15
3.1 Using patterns	15
3.2 Composing patterns	15
4 Service patterns	16
4.1 Invocation pattern	16
4.1.1 Request-response	16
4.2 Operation patterns	18
4.2.1 PutData	18
4.2.2 GetData	19
4.2.3 PutNotification	20
4.2.4 GetNotification	21
4.3 Interaction patterns for document delivery	22
4.3.1 Get	22
4.3.2 Put	23
4.3.3 Notify and get	24
4.3.4 Put and notify	25
5 Security patterns	27
5.1 Authentication patterns	28
5.1.1 Identified origin	28
5.1.2 Authenticated origin	28
5.2 Confidentiality patterns	30
5.2.1 Confidential to receiver	30

5.2.2	Confidential to destination.....	30
5.3	Payload security patterns.....	32
5.3.1	None.....	32
5.3.2	Signed.....	33
5.3.3	Encrypted.....	34
5.3.4	Signed Before Encrypted.....	35
6	Versioning patterns.....	36
6.1	Artefact identification.....	36
6.1.1	Sequential version identifiers.....	36
6.2	Composite artefacts.....	37
6.2.1	Composite artefact identifier.....	37

Document information

Change history

Version	Date	Comments
1.0	2006-12-21	Release for public comment
2.0 draft	2008-09-01	Draft for comment

This page intentionally left blank.

1 Introduction

1.1 Background

The National E-Health Transition Authority (NEHTA) was formed to enable interoperability across Australia's e-health environment.

The aim of NEHTA is to enable semantically rich communications between healthcare organisations. It does this by defining standards for external communications between organisations. The internal operation of an organisation is not directly in the scope of NEHTA. However, NEHTA standards will have an indirect effect on them, because the internal systems will need to support the needs of external communications.

1.1.1 Document history

This document was previously called the *Technical Architecture for Implementing Services 1.0* [TAIS2006].

Version 2.0 of this document was renamed to more clearly reflect its focus as a set of concepts and patterns.

1.2 Purpose

This document is a toolkit for authors of service specifications. It defines concepts that can be used to communicate with. It also defines patterns that they can use in their designs. This toolkit assists them in producing and communicating their designs.

1.3 Scope

1.3.1 Content

This document contains a set of technical concepts and technical patterns.

The concepts extend and refine the concepts defined in NEHTA's Interoperability Framework 2.0 [IF2007]. These concepts define a common set of terms which can be used for communicating ideas. These concepts are also used to describe the patterns in this document.

The patterns provide a catalogue of design patterns that are used in the design of solutions in NEHTA's e-health environment. These patterns are a set of mechanisms for implementing different policies. The problem domain's business requirements define the policies, and these patterns provide mechanisms to use to implement those policies.

1.3.2 Audience

The primary audience for this document are the authors of service specifications. They can use the concepts to document their service specifications. They can also choose appropriate patterns to use in their service specifications.

The secondary audience for this document are the solution architects that design systems that use the service interface specifications. They can use this document as a guide to understanding the concepts and patterns that have been used in the service interface specifications.

The tertiary audience for this document are the software developers who implement the systems specified by the solution architects. They can use this document as a guide to understanding the concepts and patterns that have

been used in the service interface specifications and in the system specification.

1.3.3 Out of scope

This document defines common design patterns for specifying services. Therefore, the following are not covered by this document:

- Process patterns, which are defined by business requirements;
- Performance patterns, which are defined by implementation policies.

1.3.4 Lifecycle

This document is intended to be a repository of reusable service patterns. It is intended to release new versions of this document as new patterns are developed and documented.

1.4 Status

This document is a draft and has been released for comment and feedback purposes.

1.5 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[RFC2110] IETF, *RFC 2119, Keywords for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997, <http://ietf.org/rfc/rfc2119.txt>

Non-normative references can be found in Appendix A: "Informative References".

1.6 Definitions, acronyms, abbreviations

Qualified identifier

A pair of values consisting of a qualifier and an identifier. The qualifier provides the context for the use of the identifier. This avoids the possibility of clashes between identifiers allocated in different systems.

Service implementation

A product (i.e. software) that conforms to a service interface.

Service instance

A specific deployment of a service implementation.

Service interface

The definition of the functionality of a service.

Service invoker

Party that invokes a service instance by sending it a service request.

Service provider

Party that makes a service instance available. It processes a service request.

1.7 Conformance

To conform to a pattern, a specification **MUST** satisfy every criteria in the profile associated with that artefact.

The keywords **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** in this document are to be interpreted as described in IETF's RFC 2119 [RFC2119].

1.8 Overview

Chapter 2 "Service concepts" introduces the concept of services.

Chapter 3 "Patterns" introduces the concept of patterns and how they can be used.

Chapter 4 "Service patterns" documents the patterns for services.

Chapter 5 "Security patterns" documents the patterns for security.

2 Service concepts

The NEHTA Interoperability Framework 2.0 defines a service as “functionality of relevance for business” [IF2007]. This chapter expands on that definition by providing a more detailed set of concepts for describing services.

2.1 Technical service definitions

This document is concerned with technical services. These services are provided by computer programs and are used by other computer programs.

Services are operated and made available by **service providers**. A service comprises of one or more *operations*. Service operations are used by **service invokers**; the process of using a service operation is called **invoking** the operation.

This document uses the terms *service invoker* and *service provider* instead of client and server. This is because client and server have traditionally been used to refer to a centralised model instead of a distributed environment. This also avoids the possible confusion when a program acts as both a service invoker and a service provider.

The external behaviour of a service is defined by its **service interface**. The service interface functions as a contract between the service invoker and the service provider: it indicates what the service invoker must do to invoke the service, and what the service provider must do when it is invoked.

The service interface is an important part of the **service specification**.

A solution can make use of zero or more technical services, as illustrated in Figure 1.

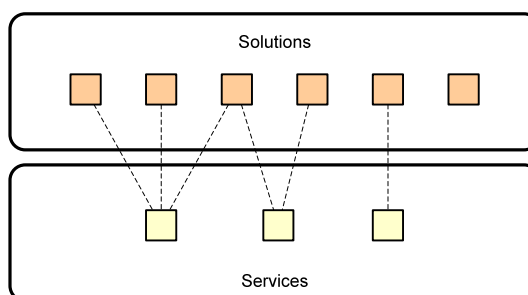


Figure 1: Solutions built using a set of services

2.1.1 Technical service

The term “service” has many different definitions in both technical and non-technical contexts. Sometimes the term is used to refer to healthcare delivery services that a healthcare organisation provides to patients. However, this document is concerned only with technical services, as defined by the NEHTA Interoperability Framework.

Technical services are loosely coupled, so service instances can individually be easily substituted or upgraded. This is possible because a service is defined by its service interface, which is separate from how it is implemented—the interface can remain the same, while the implementation behind it can be changed. They can be developed and deployed by different parties, and they can be deployed in different locations.

Some technical services are standalone, while some services depend upon other services to perform their tasks. The implementation of a service can invoke other services to carry out its functionality, but it does not have to.

Technical services are not accessed directly by people, rather they are invoked (i.e. programmatically used) by a software component or another technical service. One possible method to allow users to use a technical service is via a “portal” or Web application. However, in that situation it is the portal that programmatically uses the technical service; the user only interacts with the user interface provided by the portal.

The technical services in the e-health environment need to have these properties:

- Have semantics which are defined through standards;
- Be connected via a common network; and
- Be defined using open standards which are available to all vendors and implementers.

In this document, the term “service provider” refers to the party operating the service. The term “service invoker” refers to the party using the service. The service invoker “invokes” the service to use it.

An important feature of good services is the documentation and agreement on the service interface. The “service interface” describes how the service is to be used and what it does.

Services must be well specified and documented, because it is a part of the agreement between the service provider and the service invoker.

2.1.2 Service interface

The **service interface** defines the external view of a technical service: how external parties interact with the service, and what the service contractually promises to provide. However, it does not specify how the service itself should be constructed. The service itself can be constructed using any computer platform or technology, as long as it provides a service that conforms to the service interface.

The service interface is a specification. It is a document or set of documents that describe the external view of a service. The documents can be written in natural language for people to use, or in a formal language for computer programs to use, or a combination of both.

This Architecture has chosen to divide a service interface into three separate sets of attributes:

- **Behavioural** attributes, which define the tasks or functions provided by the service;
- **Informational** attributes, which define the data that is used and produced by the service; and
- **Non-functional** attributes, which defines constraints that affect the qualities of the service, but not the functionality of the service. For example: security, reliability, and performance.

It is recommended that services be specified by separating their interfaces into behavioural, informational, and non-functional attributes.

It is important that these different attributes are kept distinct as this makes the service interfaces more adaptable, and allows the e-health environment to be more consistent, scalable, and cost effective.

The separation of these three attributes allows the different parts of a service interface to be reused for other service interfaces. For example, “store and retrieve” type of behaviour could be used with x-ray images, to define a radiology service. A referral service could be defined that handles referral documents instead of x-ray images—they both share the same behavioural, but use different informational, attributes. Another example is when there are two referral services: one referral service could specify a high level of security

and reliability as its non-functional attributes, and a different referral service could specify a lower level of security and reliability—they both share the same behavioural and informational attributes, but have different non-functional attributes.

A technical service needs all three attributes to be specified as a part of its service interface. In the past, e-health standards have focussed more on the information (i.e. the content of the messages) than on how that information is supposed to be used. It is important to recognise that all three attributes are required for technical services to be interoperable.

It should be noted that this Architecture is at the technical level, so it only addresses the technical aspect of these attributes. For example, the behaviour is only about the service behaviour at the technical level. Non-technical aspects are not covered. For example, the business policies and business processes associated with the service are not covered. A complete service specification will include more than just the technical aspects of the service.

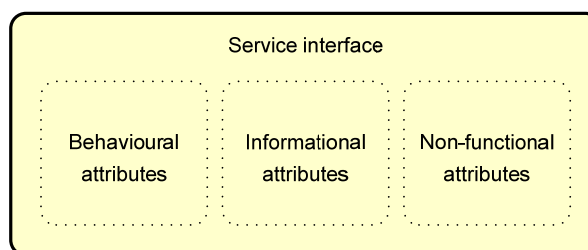


Figure 2 Attributes making up a service interface

2.1.3 Service implementation

The term **service implementation** refers to a software component that conforms to a service interface.

There can be zero or more service implementations for a particular service interface. That is, different products can be created to implement the same service interface.

In this document, the term “service implementation” refers to an artefact, such as a computer program. The term “implementation” is not used here to refer to the development, training, roll out or deployment process. However, the term “implement” is used in this document to refer to the development or creation of the service implementation artefact.

2.1.4 Service instance

The term **service instance** refers to a particular deployment of a service implementation.

These deployments can be on one computer, or on several different computers. There can be multiple service instances running at the same time. Since a service interface can have multiple service implementations, a single service interface can also have multiple service instances (and those service instances could consist of the same service implementation or different service implementations.)

To bring these three concepts together, consider an example of a bank automatic teller machine (ATM). There is a specification that describes how an ATM communicates with a banking network that is used by many different banks—that is the service specification. It describes how other systems expect to communicate with an ATM, but not how it should be constructed. A particular manufacturer creates a model of an ATM—that is a service implementation. A bank buys several ATMs from that manufacturer and locates them at its branches—each one of those ATMs is service instance. Other manufacturers can also build ATMs which conform to that specification.

That bank can choose to buy ATMs from a number of different manufacturers if it wanted to. As long as the service specification is followed, those different service instances can work together.

2.2 Synchronous and asynchronous services

The terms synchronous and asynchronous mean different things depending on the layer in which they are applied.

For example, informal definitions of three possible layers are:

- Business layer
This layer deals with business processes and events. Events are synchronous if an event occurs in direct response to another event. Events are asynchronous if there is no relationship between the events.
- Services protocol layer
This layer deals with the data that is transmitted between computer programs. This data is known as messages, documents, or packets, depending on the protocol. A synchronous response is delivered immediately upon receiving the request. An asynchronous response can occur at a later time, and can be delivered in an independent manner.
- API layer
This layer deals with how software program code is written. A synchronous function call returns the result from the function call. An asynchronous function call does not return the result, but the program can obtain the result through either calling a different function or by setting up callback handlers.

There are other layers, and each of these layers could contain multiple distinct layers. For example, a SOAP protocol could be transmitted over a HTTP layer, which is implemented over a TCP/IP protocol layer.

Different approaches can be used in different layers. For example, a programming library can provide an asynchronous API for a protocol that is synchronous. Alternatively, an API which is synchronous can be provided for an asynchronous protocol (although this is less likely).

Important points to note:

- Be aware of which level is being discussed when the terms asynchronous or synchronous is being used; and
- Do not assume that using an asynchronous or synchronous approach in one layer dictates how it must be done in another layer.

2.3 Service specifications

The NEHTA *Interoperability Framework 2.0* adopts the IEEE definition of a specification as “a document that prescribes, in a complete, precise, verifiable manner, the requirements, design, behaviour, or characteristics of a system or system component” [IF2006].

A service specification is a document that prescribes a service. In particular, it documents the behaviour of the service from an external point of view (i.e. the service interface) as well as internal behaviour.

2.3.1 Publishing

The service specifications must be published so that other parties can write software for service invokers. In this way, there are no technical restrictions on who can invoke a service; although there will be business policies to determine who can actually invoke a service. This allows service providers and

service invokers to use different products from different vendors to communicate.

2.3.2 Conformance and certification

The different products must ensure that they follow the service specifications correctly. They need to conform to the specifications. This can be enforced by suitable conformance testing and accreditation processes.

2.3.3 Interpretation

Service specifications usually contain normative and non-normative parts. The normative parts are the prescriptive part of the specification. They describe “what” the service does—not “how” it does it. Therefore, software is free to use any approach to implement the service—as long as it conforms to the service specifications. The non-normative parts contain supporting material which is not prescriptive (i.e. they are explanations or suggestions that be followed or not).

It is significant that a service specification define “what” the behaviour is, and not “how” it must be implemented. Service specifications will contain artefacts that can be used to help implement the service (e.g. machine readable files such as XML Schema and WSDL). Their primary purpose is to specify behaviour. These can be used—but they do not have to be used—when implementing the service. The implementation must still conform to them, regardless of whether they are used for implementation or not.

3 Patterns

3.1 Using patterns

The patterns in this document are described using a structure consisting of the:

- Problem, describing the problem that the pattern addresses.
- Solution, describing the solution to the problem.
- Conformance, describing the conformance criteria that must be followed when using the pattern.
- Features, describing other advantages of the solution.
- Tradeoffs, describing compromises that must be acceptable.

To choose a pattern to use, the pattern's problem needs to be the same as the problem being solved and the pattern's tradeoffs needs to be acceptable. If there are multiple patterns which are acceptable, then a pattern can be chosen based on the features which are most desirable to the problem being solved.

To be able to claim that a solution uses a particular pattern, the conformance criteria in the pattern must all be satisfied.

3.2 Composing patterns

Multiple patterns can be used in one solution.

If more than one pattern is being used, all the conformance criteria from each pattern needs to be satisfied.

4 Service patterns

This chapter describes patterns for defining services. The patterns are divided up into:

- Invocation patterns;
- Operation patterns; and
- Interaction patterns for document delivery.

4.1 Invocation pattern

This section describes the patterns for the invoking operations.

The invocation patterns describe how a single service operation is invoked. Currently, there is only one invocation pattern, request-response. This invocation pattern is used by the operation patterns in section 4.2.

4.1.1 Request-response

4.1.1.1 Problem

Invoking an operation offered by a service provider.

4.1.1.2 Solution

1. The service invoker sends a request to the service provider and waits for a response.
2. The service provider sends a corresponding response immediately back to the service invoker.

The term “immediately” in this context means quick enough so that the service invoker does not timeout waiting for the response.

4.1.1.3 Compliance

PA 4.1.1.3-1 The specification **MUST** define a request for the operation.

PA 4.1.1.3-2 The specification **MUST** define responses for the operation.

4.1.1.4 Features

This pattern allows data to be transmitted to the service provider, obtained from the service provider, or both. It also allows the service to perform an action on behalf of the service invoker, and return results of that action to the invoker.

In normal situations, the service invoker will know if the operation was successful or not.

- If the service invoker receives a successful response, it knows that the request has been received and processed successfully.
- If the service invoker receives an error response, it knows that the request has been received but not successfully processed.

In abnormal situations, the service invoker cannot be sure if the operation was successful or not.

- If the service invoker does not receive a response at all, it cannot determine whether:

- The operation was processed by the service provider (either successfully or not), but the response was not successfully delivered back to it.
- The request was received by the service provider, but not processed.
- The request did not arrive at the service provider.

The service provider has no reliable mechanism for determining if the response has successfully been received by the service invoker.

When using this pattern, additional mechanisms might be needed to deal with this situation.

4.1.1.5 Tradeoffs

The service provider must be available to process the request when the request is sent. Either the service provider must be available all the time, or some form of coordination is required so that the service invoker can send the request during the times that the service provider is available.

The service provider needs to be able to generate the response immediately upon receiving the request. If this is not possible, then the business function must be redefined into a sequence of multiple invocations. For example, one request-response invocation to start the process, and another request-response invocation to retrieve the results of the process.

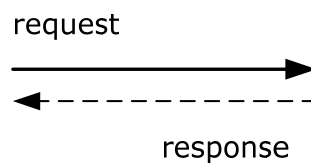


Figure 3 Request-response invocation pattern

4.2 Operation patterns

This section describes a set of patterns for the behaviour of individual operations provided by services. These patterns identify different types of data which is being conveyed between the service invoker and the service provider; and in which direction data flows. These operation patterns are used by the interaction patterns for document delivery in section 4.3.

TYPING

NEHTA develops services with strongly typed interfaces. Therefore, when these patterns are used they must be defined for a specific type of data or notification (e.g. instead of "putData", an operation would have a specific name like "putPathologyReport" and its data will be precisely defined).

4.2.1 PutData

4.2.1.1 Problem

An operation is required for the service provider to receive data from the service invoker.

4.2.1.2 Solution

Provide an operation that follows the request-response invocation pattern (section 4.1.1).

In this operation, the:

- Request contains the data to transfer;
- Response contains an acknowledgement of delivery success or failure.

4.2.1.3 Compliance

PA 4.2.1.3-1 The specification **MUST** define an operation based on the request-response invocation pattern.

PA 4.2.1.3-2 The specification **MUST** define the service request to contain the data being sent.

4.2.1.4 Features

The data is delivered when the operation is invoked.

4.2.1.5 Tradeoff

The data receiver needs to be operating as a service provider.

This pattern only deals with delivery acknowledgements. Application or process level acknowledgements need to be handled by other operations.

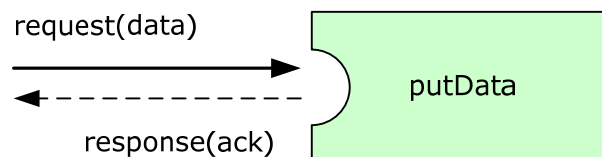


Figure 4 PutData operation pattern

4.2.2 GetData

4.2.2.1 Problem

An operation is required for the service invoker to retrieve data from the service provider.

4.2.2.2 Solution

Provide an operation that follows the request-response invocation pattern (section 4.1.1).

In this operation, the:

- Request contains information that determines what data is required;
- Response contains the data being transferred.

4.2.2.3 Compliance

PA 4.2.2.3-1 The specification **MUST** define an operation based on the request-response invocation pattern.

PA 4.2.2.3-2 The specification **MUST** define the service response to contain the data being sent.

4.2.2.4 Features

The data is retrieved when the operation is invoked.

Operations following this pattern can participate in a number of architectural patterns. For example, polling or retrying an operation if it did not succeed and possibly caching of results.

4.2.2.5 Tradeoffs

The source of the data needs to operate as a service provider.

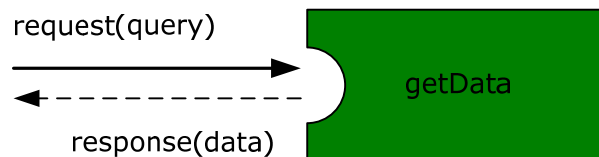


Figure 5 GetData operation pattern

4.2.3 PutNotification

4.2.3.1 Problem

An operation is required to receive notifications from the service invoker.

A notification is an indication that an event has occurred. As part of this event, additional data may also be available. The “putNotification” is similar to the “putData” pattern, but it is defined as a separate pattern because the “putData” operation pattern assumes the data in the operation is complete, whereas the notification itself indicates there may be additional data elsewhere.

4.2.3.2 Solution

Provide an operation that follows the request–response invocation pattern (section 4.1.1).

In this operation, the:

- Request contains the notification;
- Response contains an acknowledgement of delivery success or failure of the notification.

4.2.3.3 Compliance

PA 4.2.3.3-1 The specification **MUST** define an operation based on the request-response invocation pattern.

PA 4.2.3.3-2 The specification **MUST** define the service request to contain a notification about the data being sent.

4.2.3.4 Features

A notification can be used as a trigger for the recipient to perform another action. For example, the notification can cause the recipient to invoke another service operation.

Transmitting a notification may be more desirable than transmitting the actual data. The notification may be smaller in size, making it require less resources to send and process. The notification may have different (usually less) security requirements than the actual data, allowing it to be handled differently.

4.2.3.5 Tradeoffs

The data is not delivered when the putNotification operation is invoked. The receiver must process the notification before it can perform another operation to obtain the data.

If the entity is not the one that needs to process the notification, a mechanism is required to deliver the notification to that entity. This occurs if the notification is delivered to an intermediary.

For delivering data, this operation pattern must be used with another operation to actually obtain the actual data.

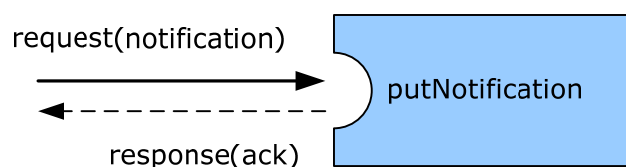


Figure 6 PutNotification operation pattern

4.2.4 GetNotification

4.2.4.1 Problem

An operation is required to provide notifications to the service invoker.

A notification is an indication that an event has occurred. As a result of this event, additional data may also be available. The “getNotification” is different from “getData”, because the “getData” operation pattern assumes the data in the operation is complete, whereas the notification itself indicates there may be additional data elsewhere.

4.2.4.2 Solution

Provide an operation that follows the request–response invocation pattern.

In this operation, the:

- Request contains information that determines the notification;
- Response contains the notification.

4.2.4.3 Compliance

PA 4.2.4.3-1 The specification **MUST** define an operation based on the request-response invocation pattern.

PA 4.2.4.3-2 The specification **MUST** define the service response to contain a notification about the data being sent.

4.2.4.4 Features

This type of operation allows the notification recipient to receive notifications without needing to be a service provider.

4.2.4.5 Tradeoffs

The data is not delivered when the getNotification operation is invoked. The receiver must process the notification before it can perform another operation to obtain the data.

If the entity is not the one that needs to process the notification, a mechanism is required to deliver the notification to that entity. This occurs if the notification is delivered to an intermediary.

For delivering data, this operation pattern must be used with another operation to actually obtain the actual data.

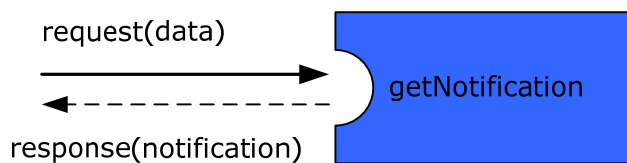


Figure 7 GetNotification operation pattern

4.3 Interaction patterns for document delivery

This section describes patterns for interactions between service operations to delivery documents. A document is data that needs to be delivered according to the business requirements.

This is not intended to be a complete list of all possible interaction patterns. Only the common interaction patterns that are being used in NEHTA's e-health environment are described.

In the figures of this section, only the service request is shown, as an arrow from the service invoker to the service provider. Every invocation also has a service response, but the service responses are not shown in the figures to keep the diagrams uncluttered.

The following patterns use the term "services challenged" to mean a party is not functioning as a service provider. This term encompasses the many different reasons for this situation: lack of permanent network connectivity, inability to manage the computing infrastructure, cannot reliably operate servers on a 24x7 basis, or not willing to be a service provider.

4.3.1 Get

4.3.1.1 Problem

A document needs to be delivered from party A to party B.

Party B is services challenged.

Party B knows when the document is available, or is able to poll for it.

4.3.1.2 Solution

Party A hosts a service with a "getData" style operation.

Party B functions as a service invoker.

Party B invokes the operation on party A, supplying a query to identify the document in the operation's request, and receives the document in the operation's response.

4.3.1.3 Compliance

PA 4.3.1.3-1 The specification **MUST** include a service interface that complies with the getData operation pattern.

4.3.1.4 Features

- No third party is introduced. Party B interacts directly with party A.
- Party B operates as a service invoker.
- The "none" payload security pattern (see section 4.4.1) is sufficient for ensuring confidentiality between party A and party B.

4.3.1.5 Tradeoffs

- Document is not delivered only when party B chooses to invoke party A.
- There is no reliable acknowledgement of delivery, because party A has no way of determining if the response message to the getData operation successfully got to party B.
- Party A must be a service provider.
- Party B must know when the document is available on A.
- Delivery of the document is delayed until party B invokes party A.

- If party B periodically polls party A, there may be times when there is no document available for party B to retrieve.
- If party B does not periodically poll party A, an external mechanism is required to inform party B of when it is necessary to invoke party A.

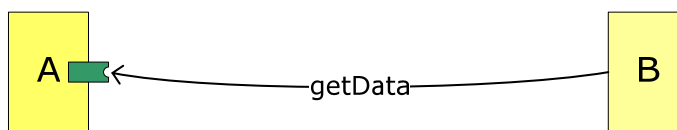


Figure 8 Direct get interaction pattern

4.3.2 Put

4.3.2.1 Problem

A document needs to be delivered from party A to party B.

4.3.2.2 Solution

Party A functions as a service invoker. Party B hosts a service with an operation that follows the “putData” operation pattern.

Party A invokes the operation on party B, supplying the document in the operation’s request. Party A receives an acknowledgement of successful delivery in the operation’s response, or it receives an error.

4.3.2.3 Compliance

PA 4.3.2.3-1 The specification **MUST** include a service interface that complies with the putData operation pattern.

4.3.2.4 Features

- Document is delivered immediately.
- Acknowledgement of delivery is received immediately.
- No third party is introduced. Party A interacts directly with party B.
- Party A operates as a service invoker.
- The “none” payload security pattern (see section 5.3.1) is sufficient for ensuring confidentiality of the document between party A and party B.

4.3.2.5 Tradeoffs

- Party B must be a service provider.

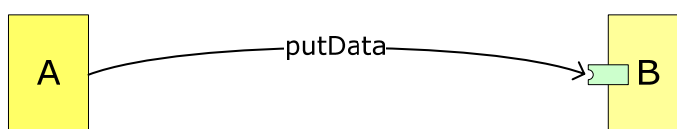


Figure 9 Direct put interaction pattern

4.3.2.6 Variation: Party B obtains the report indirectly

Introduce a third party. The document is sent to the third party using the same “putData” operation pattern. Then party B obtains the document from the third party (either by using a “getData” operation pattern or some other mechanism).

4.3.3 Notify and get

4.3.3.1 Problem

A document needs to be delivered from party A to party B.

Party B is services challenged.

4.3.3.2 Solution

Introduce a notifications service.

Party A functions as a service invoker and a service provider.

Party B functions only as a service invoker.

1. Party A puts a notification into the notification service.
2. Party B gets the notification from the notification service.
3. Party B gets the document from party A.

The notifications would normally include data useful in the processing the actual document. For example, it might indicate a unique identifier for the document, or where to get the document from (which would be important if there are multiple sources delivering documents to party B).

4.3.3.3 Compliance

PA 4.3.3.3-1 The specification **MUST** include a service interface that complies with the putNotification operation pattern.

PA 4.3.3.3-2 The specification **MUST** include a service interface that complies with the getOperation operation pattern.

PA 4.3.3.3-3 The specification **MUST** include a service interface that complies with the getData operation pattern.

4.3.3.4 Features

- Party B operates as a service invoker.
- The “none” payload security pattern (see section 5.3.1) is sufficient for ensuring confidentiality between party A and party B for the document transfer (but not for the notification).
- The notification could have different confidentiality requirements from the document.

4.3.3.5 Tradeoffs

- Document is not delivered immediately to party B. It is delivered after party B retrieves the notification and then chooses to invoke party A.
- There is no reliable acknowledgement of delivery, because party A has no way of determining if the response message to the getData operation successfully got to party B.
- Party A must be a service provider.
- A notification service is introduced.
- The notification service must be a service provider.
- Delivery of the notification is delayed until party B invokes the notification service.
- If there are multiple notification services, both parties must agree to use the same notification service.
- Party B needs to poll the notification server for new notifications.

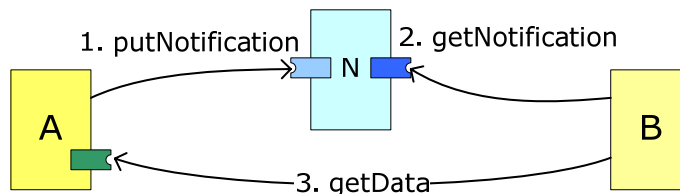


Figure 10 Notifications interaction pattern

4.3.4 Put and notify

4.3.4.1 Problem

A document needs to be delivered from party A to party B.

Party A and party B are services challenged.

4.3.4.2 Solution

Introduce a notifications service and a document store.

Party A functions as a service invoker and a service provider.

Party B functions only as a service invoker.

4. Party A puts the document into the document store.
5. Party A puts a notification into the notification service.
6. Party B gets the notification from the notification service.
7. Party B gets the document from the store.

The notifications would normally include data useful in the processing the actual document. For example, it might indicate a unique identifier for the document, or where the store is (which would be important when there are multiple stores that the documents could be put into).

4.3.4.3 Compliance

- PA 4.3.4.3-1 The specification **MUST** include a specification of a service interface that complies with the putNotification operation pattern.
- PA 4.3.4.3-2 The specification **MUST** include a specification of a service interface that complies with the getNotification operation pattern.
- PA 4.3.4.3-3 The specification **MUST** include a specification of a service interface that complies with the putData operation pattern.
- PA 4.3.4.3-4 The specification **MUST** include a specification of a service interface that complies with the getData operation pattern.

4.3.4.4 Features

- Party A operates as a service invoker.
- Party B operates as a service invoker.

This pattern can be used to support multiple stores. Party B could have a single (or a small number of) notification services it polls. Party A can put the document into any store, as long as it informs party B (using the notification) which store the document is in.

This pattern can support confidentiality in a number of ways. Firstly, the notification service does not handle the actual document. Secondly, the document store can be secured using the “signed”, “encrypted” or “signed and encrypted” payload security patterns (see sections 5.3.2, 5.3.3, 5.3.4).

4.3.4.5 Tradeoffs

- Document is not delivered immediately to party B. It is delivered after party B retrieves the notification and it retrieves the document from the store.
- There is no acknowledgement of delivery.
- A notification service is introduced.
- The notification service must be a service provider.
- A store is introduced.
- The store must be a service provider.
- If there are multiple notification services, both parties must agree to use the same notification service.
- If there are multiple stores, either:
 - Both parties must agree to use the same store; or
 - The notification must contain information that allows party B to determine which store to retrieve the document from.
- The “none” payload security pattern (see section 5.3.1) is not sufficient for ensuring confidentiality between party A and party B. Either the “encrypted” or “signed before encrypted” payload security patterns needs to be used to provide confidentiality.
- Party B needs to poll the notification server for new notifications.
- Party B needs to know which notification servers to poll. If there are multiple notification servers with notifications for party B, a list must be maintained.

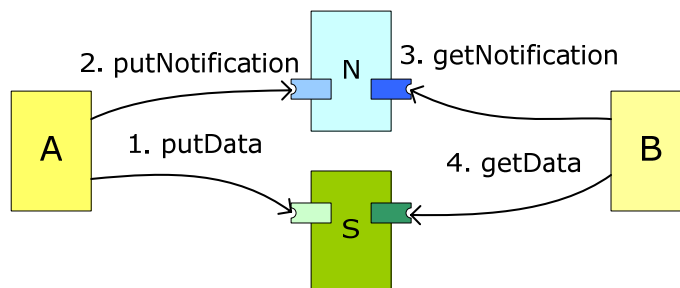


Figure 11 Notifications interaction pattern

5 Security patterns

This chapter describes patterns for providing security in the national e-health environment.

ROLES

These technical patterns are described using these four communication roles:

- **Origin**, the party responsible for creating the contents being communicated.
- **Sender**, the party that performs the sending of the communications.
- **Receiver**, the party that receives the communications.
- **Destination**, the party that acts on the contents being communicated.

For example, consider the situation of a discharge summary that is sent from a hospital to a general practice. The origin is the discharge nurse who creates the discharge summary. The sender is the hospital, or more precisely the hospital's discharge system. The receiver is the general practice's practice management system. The destination is the General Practitioner.

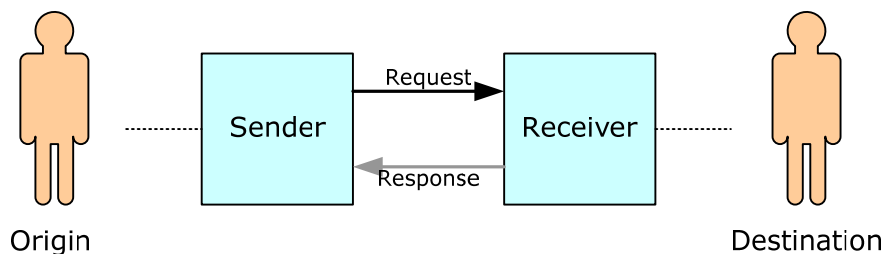


Figure 12 Roles in the security patterns

These technical patterns are described in terms of the request from the sender to the receiver. When using the request-response invocation pattern (section 4.1.1) there is also a response going in the opposite direction. The response can also be secured using these same patterns by reversing the roles.

5.1 Authentication patterns

The following patterns show ways that the origin and sender can be authenticated.

In the national e-health environment, all service invocations occur between organisations, and those organisations must always be authenticated. Therefore, there are no patterns where the sender is not involved and no patterns where the sender is anonymous or not authenticated.

Two authentication patterns are defined:

- Identified origin; and
- Authenticated origin.

5.1.1 Identified origin

5.1.1.1 Problem

Origin needs to be identified, but does not need to be authenticated.

Sender needs to be identified and authenticated.

5.1.1.2 Solution

Origin is identified in the service request.

Sender signs the service request.

5.1.1.3 Compliance

PA 5.1.1.3-1 The message **MUST** contain the identity of the origin.

PA 5.1.1.3-2 The message **MUST** be signed by the sender.

5.1.1.4 Features

This pattern is useful for operations where the business relationship exists between the organisations, and not the individuals in the organisation.

5.1.1.5 Tradeoffs

It is the responsibility of the sender to authenticate the origin.

5.1.2 Authenticated origin

5.1.2.1 Problem

Origin needs to be identified and authenticated.

Sender needs to be identified and authenticated.

5.1.2.2 Solution

Origin signs the data in the service request.

Sender signs the service request.

5.1.2.3 Compliance

PA 5.1.2.3-1 The data in the message **MUST** be signed by the origin.

PA 5.1.2.3-2 The message **MUST** be signed by the sender.

5.1.2.4 Features

The data and the origin's signature can be extracted from the request and processed independently from it.

5.1.2.5 Tradeoffs

This pattern can make use of the following payload security patterns to implement it:

- 5.3.2 Signed; or
- 5.3.4 Signed Before Encrypted

5.2 Confidentiality patterns

The following patterns identify ways that data can be kept confidential, so that only the service provider and/or destination can access it.

Two confidentiality patterns are defined:

- Confidential to receiver; and
- Confidential to destination.

5.2.1 Confidential to receiver

5.2.1.1 Problem

Confidentiality to the receiver is required.

5.2.1.2 Solution

Service request is encrypted for the receiver.

5.2.1.3 Compliance

PA 5.2.1.3-1 The message **MUST** be encrypted for the receiver.

PA 5.2.1.3-2 The message **MUST** contain the identity of the destination.

5.2.1.4 Features

The receiver is responsible for maintaining confidentiality once it has received the data.

This pattern is useful if flexibility is required for seeing the data. For example, the receiver could allow another authorised provider at the organisation to view the data in special circumstances (e.g. a locum GP is replacing a regular GP in a practice).

5.2.1.5 Tradeoffs

Confidentiality to the destination can be provided. This confidentiality will have to depend on trusting the receiver to maintain that confidentiality, instead of enforcing it through cryptographic mechanisms.

5.2.2 Confidential to destination

5.2.2.1 Problem

Confidentiality to the destination is required.

5.2.2.2 Solution

Service request is encrypted for the receiver.

The data in the service request is encrypted for the destination.

5.2.2.3 Compliance

PA 5.2.2.3-1 The data in the message **MUST** be encrypted for the destination.

5.2.2.4 Features

The receiver is prevented from accessing the data.

5.2.2.5 Tradeoffs

The receiver can never access the data. Even in an emergency situation (e.g. when the destination's private key is not available) the data cannot be accessed—the destination, and only the destination, can access the data.

This pattern can make use of the following payload security patterns to implement it:

- 5.3.3 Encrypted; or
- 5.3.4 Signed Before Encrypted

5.3 Payload security patterns

This section defines patterns for creating a secured piece of data. The secured piece of data is called a payload, because its primary use is in the payload of a service request or the response. The term “data” in this section will refer to any type of data, including notifications.

These payload security patterns can be applied to each of the operations in the interaction patterns for document delivery (described in section 4.3).

Note: It is expected that all operations will also be secured by operation level security mechanisms. Sometime this is referred to as transport level security, but that term is ambiguous because different protocols can be referred to as “transport level” depending on the context.

Four payload security patterns are defined:

- None;
- Signed;
- Encrypted; and
- Signed before Encrypted.

5.3.1 None

5.3.1.1 Problem

No additional security is required over that required to secure the link between the two parties.

5.3.1.2 Solution

Use the data and/or notification directly.

5.3.1.3 Compliance

None.

5.3.1.4 Features

The structure of the data can be exposed at the service interface. The interface can be strongly typed.

5.3.1.5 Tradeoffs

Security is only provided by any operation level security mechanisms.

5.3.2 Signed

5.3.2.1 Problem

The origin of the data needs to be authenticated.

The recipient needs proof that the data has not been tampered with.

5.3.2.2 Solution

Send the data with a digital signature over the data.

5.3.2.3 Compliance

PA 5.3.2.3-1 The specification **MUST** identify the data that must be digitally signed.

PA 5.3.2.3-2 The signing public certificate **SHOULD** be sent with the data. Alternatively, an identifier for the public certificate is sent, but then the recipient will need to obtain the public certificate by other means.

5.3.2.4 Features

The structure of the data can be exposed at the service interface. The interface can be strongly typed.

The origin of the data does not have to be the service invoker or service provider. For example, it can be a person (using their personal keys) instead of the system performing the operations.

5.3.2.5 Tradeoffs

The service interface needs to be modified to include the signature and public certificate, as well as the actual data.

An extra level of processing will be required to validate the signature. Depending on the business requirements, signature validation may be optional in some situations. For example, the signature might only be checked in exceptional circumstances.

The signature only authenticates the origin of the data: not the service invoker. It must not be assumed that the service invoker is the same entity as the origin.

5.3.3 Encrypted

5.3.3.1 Problem

The data needs to be kept confidential from all parties, other than the destination.

This problem can arise when a payload is sent to the service provider via a third party that is not trusted to access the confidential data. For example, the “**Error! Reference source not found.**” pattern (see section **Error! Reference source not found.**) when confidential data needs to be delivered without the store having access to it.

5.3.3.2 Solution

Encrypt the payload for the destination.

Note that this is an extra encryption of the payload, in addition to the encryption that is performed for the link between the service requestor and service provider.

5.3.3.3 Compliance

PA 5.3.3.3-1 The specification **MUST** identify the data that is to be encrypted.

PA 5.3.3.3-2 An identifier for the public certificate used for the encryption **SHOULD** be sent with the encrypted data. The receiver can then use the identifier to locate the corresponding private key to use to decrypt the data.

5.3.3.4 Features

Confidentially is achieved.

The party that the payload is encrypted for does not necessarily have to be the service provider.

5.3.3.5 Tradeoffs

The origin of the data cannot be authenticated. This pattern only does encryption without any signing (for both signing and encryption, see section 5.3.4).

An extra level of processing is required to handle the payload security. The service interface will extract out the payload, which is an encrypted block of data. This payload is then decrypted, before it can be deserialised and/or validated.

5.3.4 Signed Before Encrypted

5.3.4.1 Problem

The identity of the origin of the data needs to be asserted.

The recipient needs proof that the data has not been tampered with.

The data needs to be kept confidential from all parties, other than the destination.

5.3.4.2 Solution

Sign the payload and then encrypt it.

5.3.4.3 Compliance

PA 5.3.4.3-1 The specification **MUST** identify the data that must be signed and encrypted.

PA 5.3.4.3-2 The signing public certificate **SHOULD** be sent with the data. Alternatively, an identifier for the public certificate is sent, but then the recipient will need to obtain the public certificate by other means.

PA 5.3.4.3-3 An identifier for the public certificate used for the encryption **SHOULD** be sent with the encrypted data. The receiver can then use the identifier to locate the corresponding private key to use to decrypt the data.

5.3.4.4 Features

Authentication, integrity, and confidentiality is achieved.

5.3.4.5 Tradeoffs

An extra level of processing is required to handle the payload security. The service interface will extract out the payload, which is an encrypted block of data. This payload is then decrypted, before its signature can be validated and the data used.

This pattern is only valuable if the data is signed and/or encrypted for a third party. The signing and encryption being provided by the service operation already provides authentication and confidentiality between the two communicating parties.

6 Versioning patterns

6.1 Artefact identification

6.1.1 Sequential version identifiers

6.1.1.1 Problem

Different versions of one artefact are being released.

For any two versions, it is necessary to determine which one is newer.

6.1.1.2 Solution

Assign every version an identifier which has a defined collation order. This ordering allows any two identifiers to be compared to determine which one is newer.

The identifier needs to be made up of two parts:

- The artefact identifier part that remains the same for all versions of the same artefact.
- The version number part that changes between each different version. Although it is called a “number” it does not have to be a single mathematical number—any value that has a defined ordering can be used. For example, a popular convention is to use number dot number dot number (e.g. 1.0.0, 1.2.3, 3.1.4).

6.1.1.3 Compliance

PA 6.1.1.3-1 Each version **MUST** be assigned a unique version identifier.

PA 6.1.1.3-2 The version identifier **MUST** be made up of an artefact identifier and a version number.

PA 6.1.1.3-3 The artefact identifier **MUST** uniquely identify the artefact and remain the same for every version of that artefact.

PA 6.1.1.3-4 The version number **MUST** be different for every version of the same artefact.

PA 6.1.1.3-5 The version number **MUST** be a value that has a defined collation order. That is, there is a manner in which values can be compared and ordered.

6.1.1.4 Features

- Does not require the nature or degree of changes between different versions to be identified.

6.1.1.5 Tradeoffs

- This versioning pattern does not provide any information about the differences between the different versions.

6.2 Composite artefacts

6.2.1 Composite artefact identifier

6.2.1.1 Problem

A mechanism is needed to uniquely identify and version an artefact which consists of one or more other artefacts.

The artefact being identified will be referred to as the “composite artefact,” and the artefacts in it will be referred to as the “component artefacts.”

The component artefacts each have their own independent versioning. Versions of the component artefact are being updated and released at different times.

6.2.1.2 Solution

Assign a new unique identifier to the composite artefact. That artefact is versioned independently from its parts.

An external mapping needs to be used to determine which component artefact is in a particular composite artefact.

6.2.1.3 Conformance

PA 6.2.1.3-1 A composite artefact **MUST** be made up of particular versions of each component artefact.

PA 6.2.1.3-2 A new identifier **MUST** be assigned to the composite artefact.

PA 6.2.1.3-3 The identifier for the composite artefact **MUST** be versioned independently from the component artefacts.

PA 6.2.1.3-4 The version of the composite artefact **MUST** change if the composite artefact is changed to contain different component artefacts or uses a different version of a component artefact.

6.2.1.4 Features

The component artefacts can have their own lifecycle. They can be versioned independently from the versioning of the composite artefact.

6.2.1.5 Tradeoffs

There is no direct association between the versioning of the composite artefact and its component artefacts.

Appendix A: Informative References

- [IF2007] NEHTA, *Interoperability Framework 2.0*, 2007-08-17.
- [TAIS2006] NEHTA, *Technical Architecture for Implementing Services: Concepts and Patterns 1.0*, 21 December 2006.