

nehata

Connectivity Coordinator

High Level Design

Version 1.0 — 1 December 2008

Release

National E-Health Transition Authority Ltd

Level 25

56 Pitt Street

Sydney, NSW, 2000

Australia.

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

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

Document Control

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

Copyright © 2008, NEHTA.

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

Table of contents

| | |
|---|------------|
| Table of contents | iii |
| Document information | iv |
| Change history | iv |
| 1 Introduction | 1 |
| 1.1 Purpose | 1 |
| 1.2 Scope | 1 |
| 1.3 Normative references | 1 |
| 1.4 Definitions, acronyms, abbreviations | 1 |
| 1.4.1 Terminology | 2 |
| 1.5 Document Overview | 2 |
| 2 Environment Overview | 3 |
| 2.1 Infrastructure Services | 3 |
| 2.1.1 UHI | 3 |
| 2.1.2 NASH | 3 |
| 2.1.3 SIL | 4 |
| 2.2 Interaction Types | 4 |
| 3 Computational View | 5 |
| 3.1 Primary Requirements | 5 |
| 3.2 Notes on Diagrams | 5 |
| 3.3 High level overview | 6 |
| 3.3.1 Outgoing Requests | 6 |
| 3.3.2 Incoming Requests | 7 |
| 3.4 Objects Referenced | 8 |
| 3.5 Send Processing | 9 |
| 3.5.1 Gather Address Information | 11 |
| 3.5.2 Retrieve Recipient Messaging Patterns | 12 |
| 3.5.3 Process Recipient Messaging Pattern | 14 |
| 3.5.4 Common Send Pattern Processes | 17 |
| 3.6 Receive Processing | 19 |
| 3.6.1 Process Request | 21 |
| 3.6.2 Invoke Recipient Code | 21 |
| 3.7 Acknowledge Processing | 22 |
| 3.8 Acknowledgement Processing | 22 |
| 4 Engineering View | 23 |
| 4.1 Design goals | 23 |
| 4.2 Core components | 23 |
| 4.2.1 Application Facing Adapter | 23 |
| 4.2.2 Connectivity Framework | 23 |
| 4.2.3 Services (Domain and Infrastructure) | 24 |
| 4.3 Connectivity Framework in more detail | 24 |
| 4.3.1 Application Programming Interface (API) | 25 |
| 4.3.2 Workflow Control | 25 |
| 4.3.3 Workflow Steps | 25 |
| 4.4 Common Workflow Steps | 26 |
| 4.4.1 Logging | 26 |
| 4.4.2 Tracking | 26 |
| 4.4.3 Configuration | 26 |
| Appendix A: Informative references | 28 |

Document information

Change history

| Version | Date | Comments |
|---------|------------|----------|
| 1.0 | 2008-12-01 | Release |

1 Introduction

1.1 Purpose

The purpose of this document is to provide a non-normative high level software design for one possible approach of an implementation focused on being conformant with the NEHTA Connectivity related specifications. One of the primary goals of this document is to draw out and highlight some of the subtleties and design constraints of implementing these specifications in a real world operational environment. This approach will open the path to greater adoption and understanding of the specifications and ultimately lead towards the interoperability goals they are trying to achieve.

1.2 Scope

This document can be viewed as a complete high level software design for an implementation aiming to be compliant with the NEHTA Connectivity related specifications but **MUST NOT** be considered the only way to implement the specifications or as the suggested way to implement these specifications.

The high level software design described here will be focussed on a generic solution that can either encompass all package requirements with some package specific components or be specialised to cater for package specific requirements only.

This document only describes design related to interactions for business functions. It does not include design for components or behaviour required for administration or management aspects of the e-health environment (e.g. publishing endpoint details to a SIL).

The intended audience of this document include Architects and Software Developers. This document should be used to form a working understanding of the processes required to support interactions within the e-health environment, aspects of caching used to improve performance and key error handling scenarios required to be handled by an implementation.

1.3 Normative references

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

[RFC2119] IETF, RFC 2119: *Keywords for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997, <http://ieft.org/rfc/rfc2119.txt>

1.4 Definitions, acronyms, abbreviations

| | |
|-------|---|
| NEHTA | National E-Health Transition Authority |
| UHI | Unique Health Identifier |
| HPI-I | Healthcare Provider Identifier – Individual |
| HPI-O | Healthcare Provider Identifier – Organisation |
| IHI | Individual Healthcare Identifier |
| SIL | Service Instance Locator |
| NASH | National Authentication Service for Health |

1.4.1 Terminology

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].

1.5 Document Overview

The core of this document has been broken into three major sections, Environment Overview, Computational View and Engineering View.

The Environment Overview section describes the environment within which implementations are expected to operate.

The Computational View section describes the processes and activities that are required to support the operation of business interactions within the environment described by the previous section.

Engineering View section describes the key components and collaborations that are needed to support the processes described in the Computational View section.

2 Environment Overview

2.1 Infrastructure Services

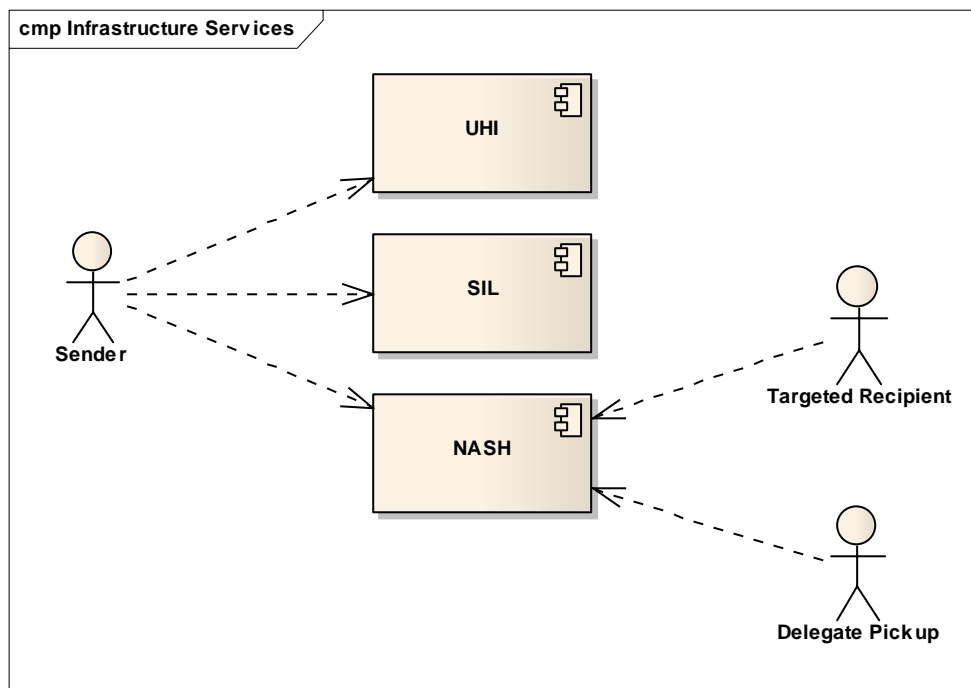


Figure 1: Infrastructure Services

In order for communication to occur within an environment that contains many parties that may have limited knowledge of one another a number of infrastructure related services are required to provide either authoritative information or bootstrap a process of information discovery to support interaction capabilities. NEHTA has identified three primary infrastructure services in this space, Unique Health Identifier (UHI), Nation Authentication Service for Health (NASH) and Service Instance Locator (SIL).

Details of how these components are involved in e-health interactions will be described in more detail in the Computational View section. The Computational View section hinges on many of the concepts identified in Connectivity Architecture [CONA2008].

2.1.1 UHI

The UHI is the authoritative source of e-health related identifiers. It stores information about identifiers and the entity that these identifiers are relate to and allows information to be retrieved about either aspect by trusted parties. It is also likely that the UHI will provide a discovery capability allowing the lookup of the SIL that is representative of the entity that will be the receiver in an e-health interaction.

2.1.2 NASH

The NASH provides authoritative cryptographic type information about entities involved in e-health interactions and is used to provide details to verify the identity of an entity involved in an interaction or to provide details required to obfuscate information that will be sent to such an entity. This functionality will be implemented by PKI technology and often delivered as X.509 certificates.

2.1.3 SIL

This SIL provides an information discovery capability to find the appropriate location and communication method to deliver information to a receiving party involved in an e-health interaction. For a more in-depth discussion on the concepts of SIL please refer to [SILA2008].

2.2 Interaction Types

The interaction type used in an e-health interaction is driven by the capabilities of the parties involved in the interaction. Often this is driven primarily by the capabilities of the receiver however there can be other influencing factors. These capabilities, or lack thereof, can also affect the roles required to achieve the desired result of the interaction. For example, if the target recipient is not capable of receiving a message via Web services directly a third party could be included to receive the message on behalf of the recipient. The recipient can then retrieve the message at some later time from the third party.

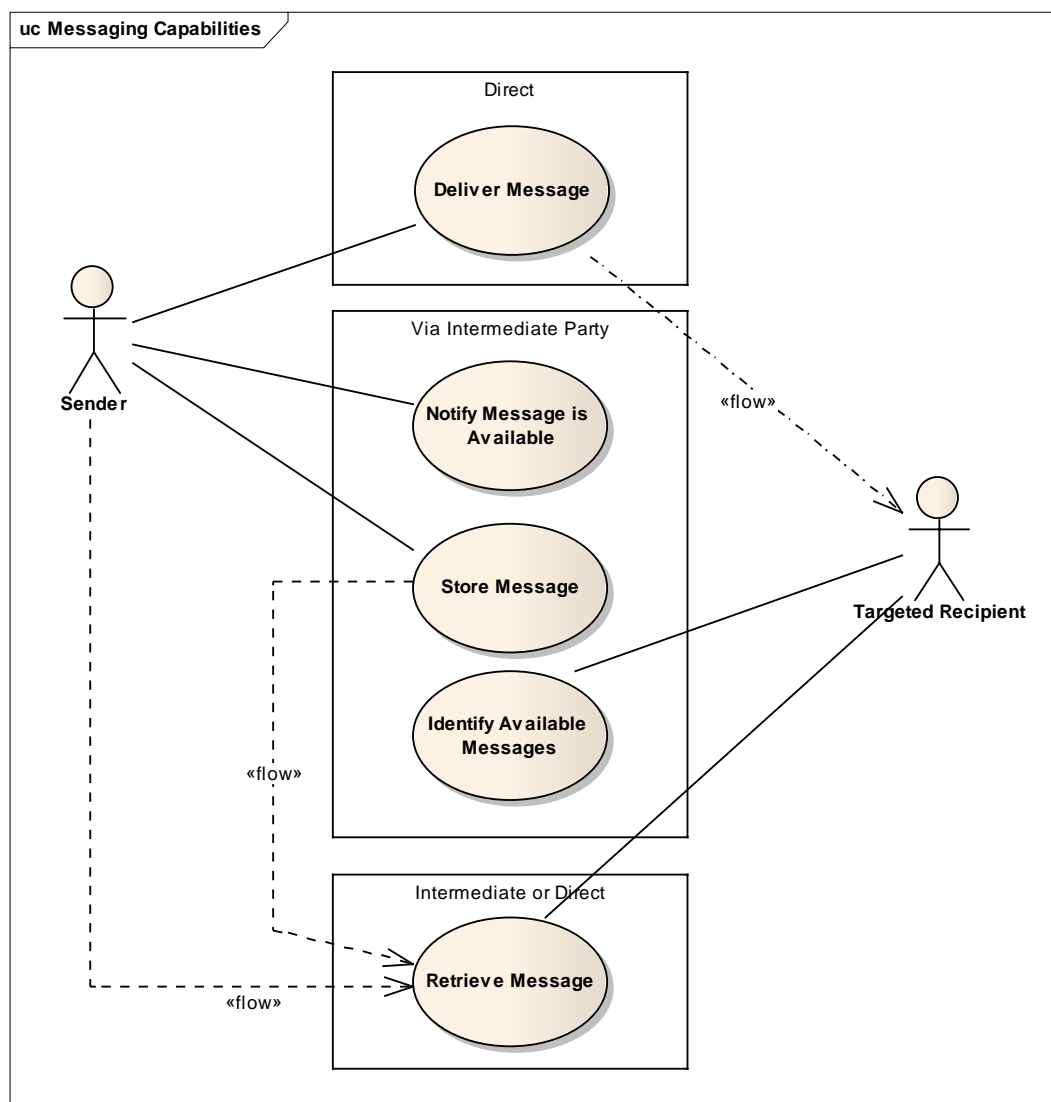


Figure 2: Messaging Capabilities

In order to support interactions between parties with these varying constraints on their messaging capabilities a number of messaging patterns have been identified to deal with the various communication scenarios. The diagram above provides a summary of the use cases that these patterns must support. For a more in-depth discussion on these communications patterns please refer to [CPIS2008].

3 Computational View

This section describes computational processes that will provide the capabilities necessary to participate in e-health related interactions. The processes described here are only one solution to the problem domain and it should not be inferred that these are the preferred solution.

It is suggested that prior to reading this section you make yourself familiar with [CONA2008], [SILA2008] and [CPIS2008].

3.1 Primary Requirements

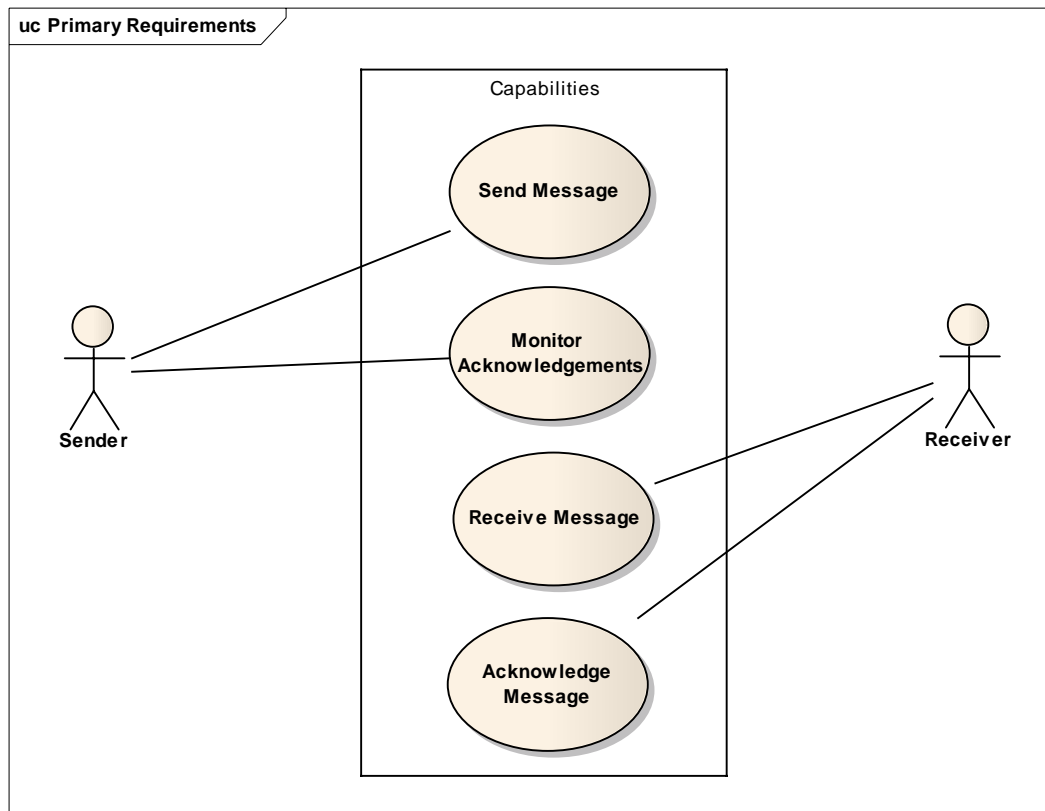


Figure 3: Primary Requirements

The above diagram gives a high level view of how the roles of sender and receiver need to interact with the system.

The Sender needs to be able to:

- Send messages; and
- Monitor acknowledgement messages received in response to their sent messages

The Receiver needs to be able to:

- Receive messages; and
- Send acknowledgements (positive or negative) to the messages received

3.2 Notes on Diagrams

The activity diagrams that follow show both the control flow and the data object flow in unison. Object flow is distinguished by the flow line traversing between two or more square boxes contained on the outer edge of activities (activity parameters) or actions (object nodes). Where there is only one

possible channel of object and control flow only the object flow is shown. From a processing perspective an activity (or action) can only be considered active if the control has been passed to it by the control flow and all input object flows to the activity have delivered their associated objects. Control flows may also be assigned guard constraints (labels within []) that must be satisfied before the control path can be taken. Guard constraints are only used where two control flows exit and action to show under what conditions the various paths are taken.

Although not explicitly shown on all diagrams this design assumes that input and output information for activities are received and passed back via a context object. Providing the inputs and outputs via a single context object rather than strongly typed parameters to a method allows a generic framework to be created in which components can easily be replaced, swapped or extended as they can all comply to a single, simple interface.

The state diagrams identify states that may be encountered while interacting with remote services. The 'Error' final state identifies that a rule based error (validation or constraint violation) condition has been encountered and the operation must not be retried without first taking corrective action. The 'Failure' final state identifies that an environment type error has been encountered and it is possible that if the environmental issue is rectified a retry of the operation may succeed.

Each text description of an activity starts with the declaration of the expected inputs and outputs. These named values **SHOULD BE** extracted and placed into the context object by the associated activity during the course of its processing.

3.3 High level overview

3.3.1 Outgoing Requests

The following diagram gives a high level pictorial representation of the processes that take place within a system when it issues an outgoing service request to a domain service.

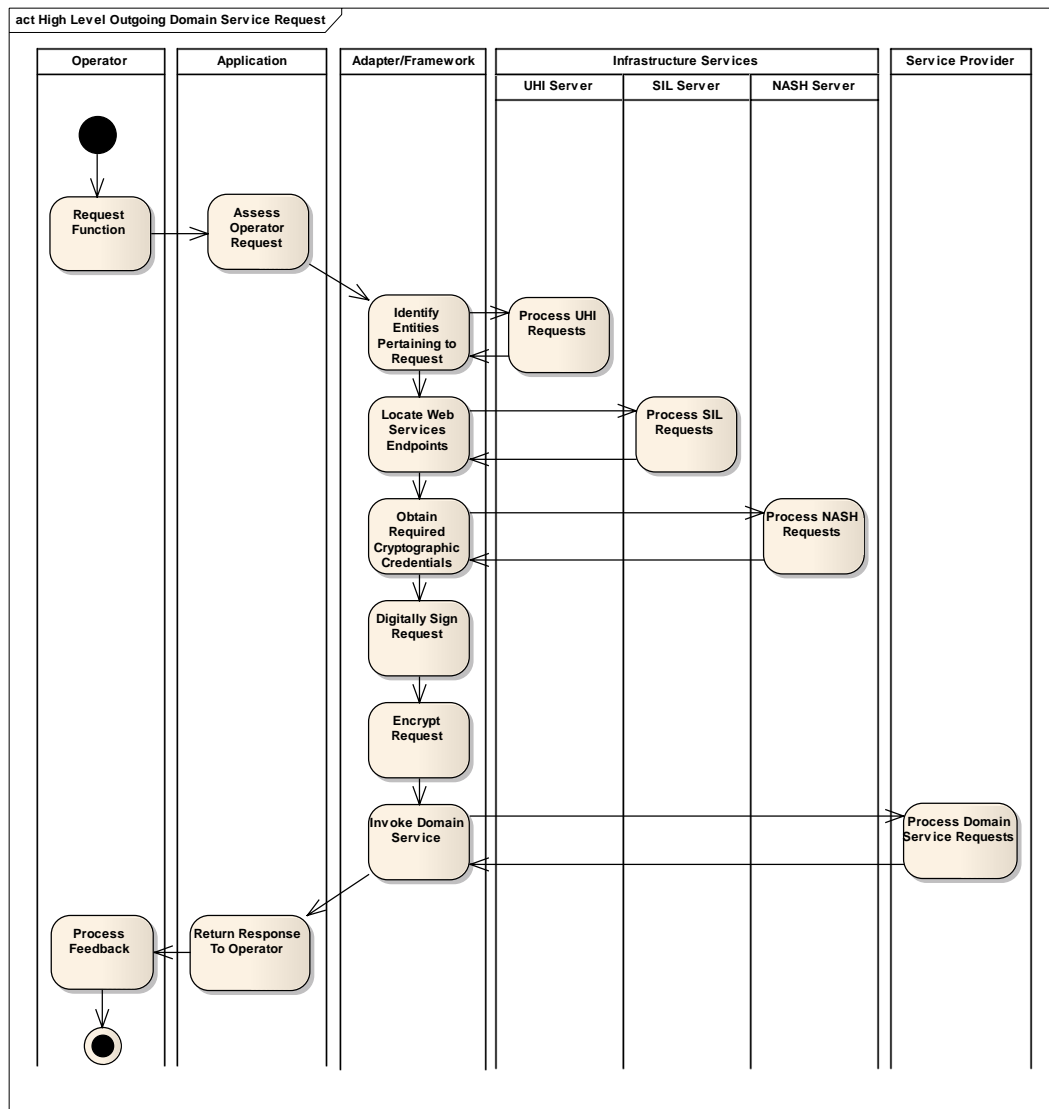


Figure 4: High Level Outgoing Domain Service Request

1. The system operator performs an operation on the Application.
2. An assessment is made by the Application on the operation which determines that an outgoing request needs to be made to an external service via the application’s associated connectivity adapter.
3. The Adapter interacts with the HPI-O Server to obtain the endpoint of the SIL associated with the HPI-O identified as the target of the request.
4. The Adapter interacts with SIL to locate the domain service endpoints for communication to the identified organisation.
5. The Adapter interacts with the NASH to obtain/validate the currency of the PKI certificate for the identified organisation.
6. The Adapter digitally signs the request.
7. If required by the interaction the Adapter encrypts the request
8. The Adapter returns a response consistent with the operation performed to the Application which is presented to the Operator in the appropriate manner.

3.3.2 Incoming Requests

The following gives a high level pictorial representation of the processes that take place within a system when it acts as Domain Server and receives incoming service request.

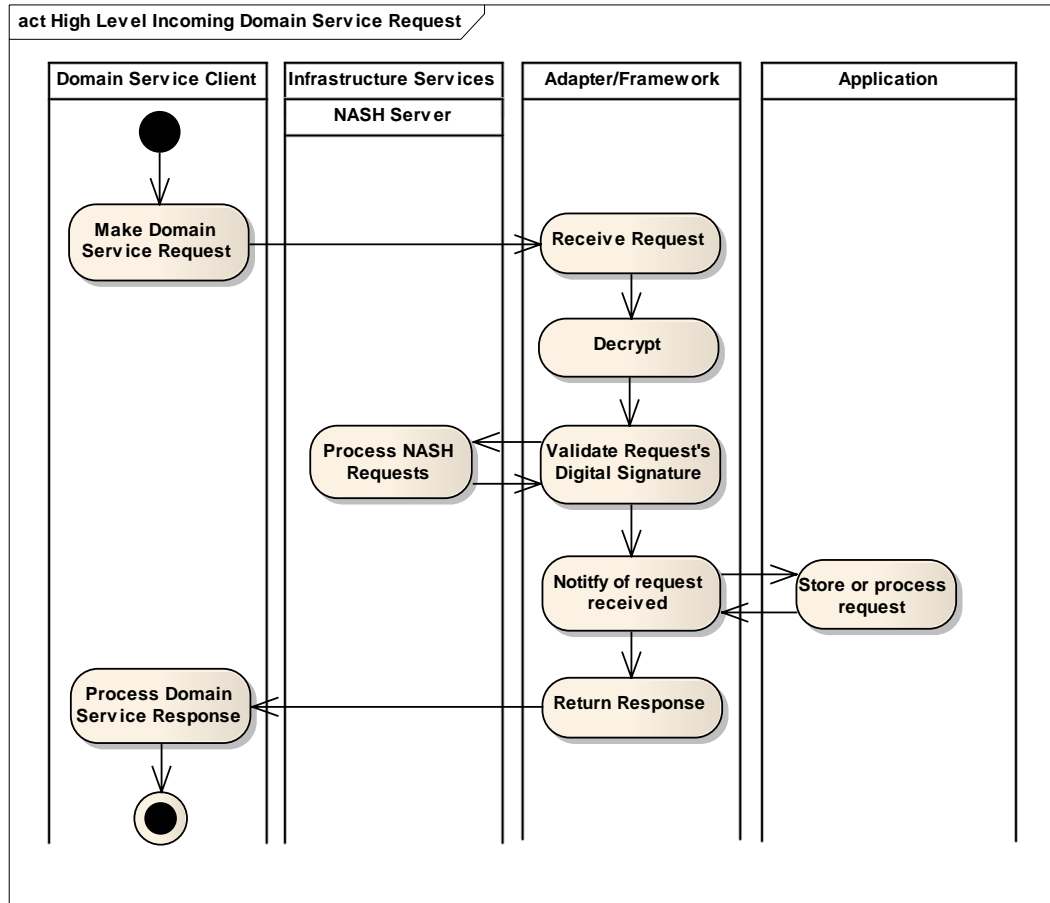


Figure 5: High Level Incoming Domain Service Request

1. The Domain Service Client makes a service request on the Domain Service Provider (Adapter).
2. The Adapter decrypt the incoming request
3. Validates the currency of the PKI certificate for the Domain Service Client and ensures the signature associated with the message is valid and is the signature of a trusted party (this may involve interaction with NASH).
4. Dispatches the request to the appropriate processing code.
5. Returns a response indicative of the success of the operation to the Domain Service Client.

3.4 Objects Referenced

Objects that are referred to in later sections and their associated figures are briefly described here.

Address An Address object provides convenient access to details of the parties involved in the interaction and identifies the interaction itself. The attributes for address will include:

- receiver[]
- sender
- destination[]
- origin

Cert A Cert object refers to either a X.509 certificate or a

| | |
|---------------------|---|
| PickupCert | reference to such a certificate. |
| Context | A Context object is used to provide and receive details from a processing activity. It provides associative array type capabilities to allow a key to be associated with a specified value. |
| CryptoDetails | A CryptoDetails object provides details of what parties (via certification or certificate references) were involved in signing and/or encrypting the received message. |
| EnvelopeDetails | An EnvelopeDetails object details the parties involved in securing the transport aspects of the interaction. |
| KeyRef | A KeyRef object identifies the key that is associated with a given Certificate. |
| NotifyPayload | A NotifyPayload object is the object that is sent as a notification for the availability of an item for a target. |
| Pattern Patterns | A Pattern object (Patterns is list of type Pattern) represents the data structure that is returned by the SIL that identifies the interaction patterns the target can support. Refer to [SILA2008]. |
| Payload | A Payload object represents the information that is being conveyed in the business interaction. Usually a document or report. |
| PrivateKey | A PrivateKey object represents the private key component of an asymmetric key pair. |
| Role | A Role object identifies an entity within a pattern that is performing a specific role. For example this could be: receiver of notifications or intermediate party or possible end target. |
| SecurePayload | A SecurePayload represents a payload that has been encrypted as per [XSP2008]. |
| SignedPayload | A SignedPayload represents a payload that has been signed as per [XSP2008]. |
| SIL | A SIL object represents an endpoint reference to Web service providing SIL capabilities. |
| Status | A Status object identifies whether the operation succeeded or failed. |
| Target | A Target object provides the end point details for Web service identified as fulfilling a role within an interaction pattern. |

3.5 Send Processing

The processing described here assumes that the application has already obtained and validated Identifiers as they were selected by the operator and prior to the dispatch of the request to the Connectivity Framework.

The following diagram shows the high level processing activities involved in sending a message to a recipient. The first activity is used to extract the addressing type details for the input message. These details are required to dispatch the request in later activities. The second activity is required to lookup the SIL associated with the target organisation to discover which patterns are supported. The third and final step processes the Patterns identified in the previous activity to find an appropriate method to communicate with the target organisation.

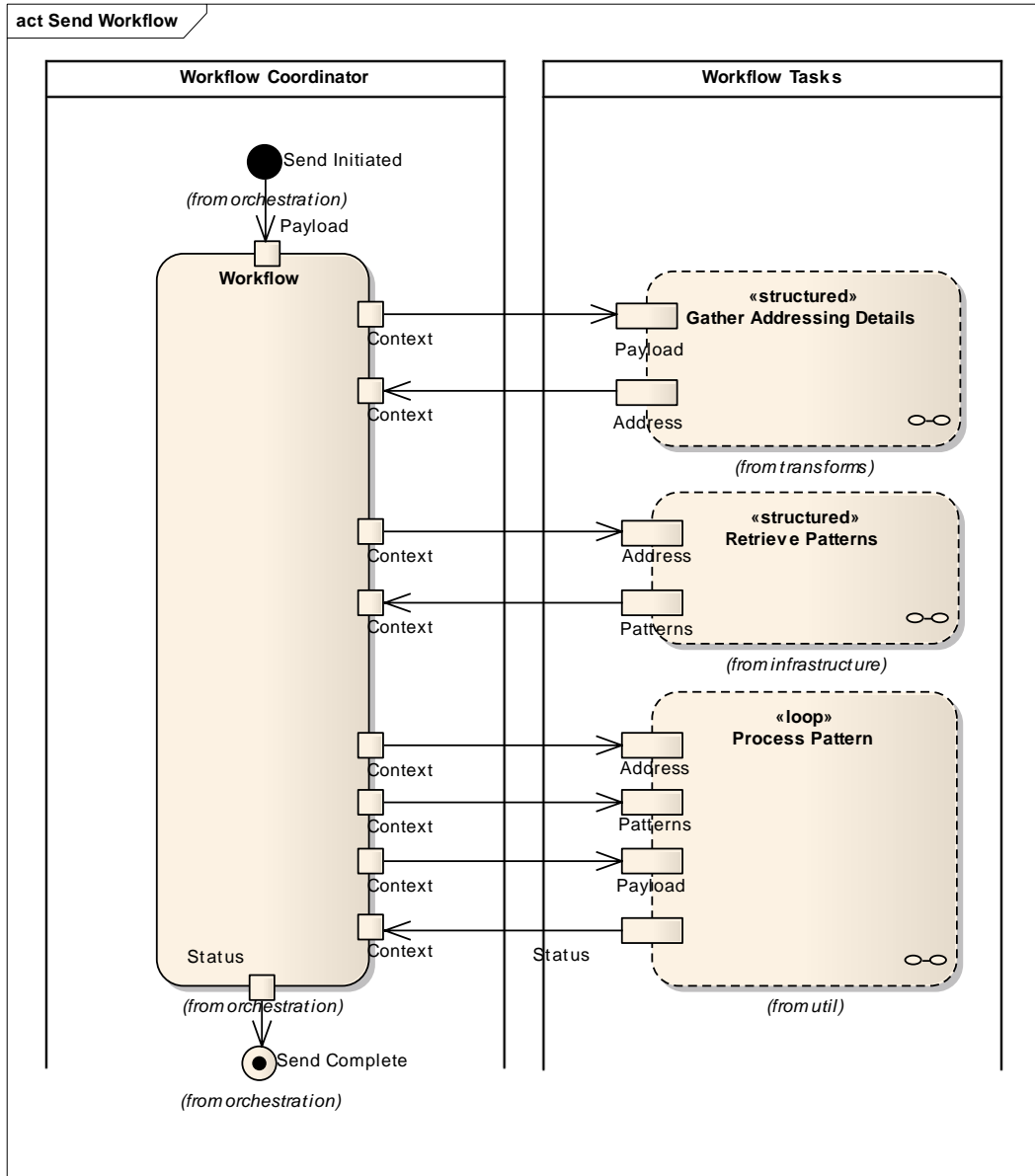


Figure 6: Send Workflow

The following sections provide a more detailed breakdown of the activities shown in the above diagram.

3.5.1 Gather Address Information

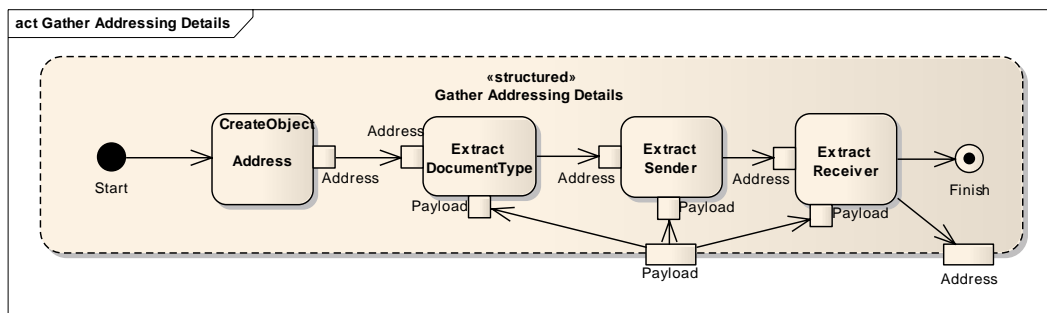


Figure 7: Gather Addressing Details

Inputs: Payload

Outputs: Address

This activity assumes that interaction addressing type information will always be present somewhere within the business orientated payload.

The purpose of the Gather Addressing Details activity is to construct and populate an Address object by extracting addressing related and document type details from the incoming payload. This activity **SHOULD BE** implemented as a generic component where possible but there may be instances where it may need to be a specific implementation aligned to package or similar. The generic capability could be achieved by using XPath for XML type payloads or offsets or field matching methods for binary or structured text to extract the relevant details.

If identifier mapping is required (i.e. local to global or jurisdiction) this could be achieved by appending an additional action at the end of the flow in this activity to provide the mapping capabilities. An alternate solution, and one that provides a separation of concerns (extract details, transform), is to place the mapping activity directly off of the main work flow between the 'Gather Addressing Details' and the 'Retrieve Patterns' activities.

3.5.2 Retrieve Recipient Messaging Patterns

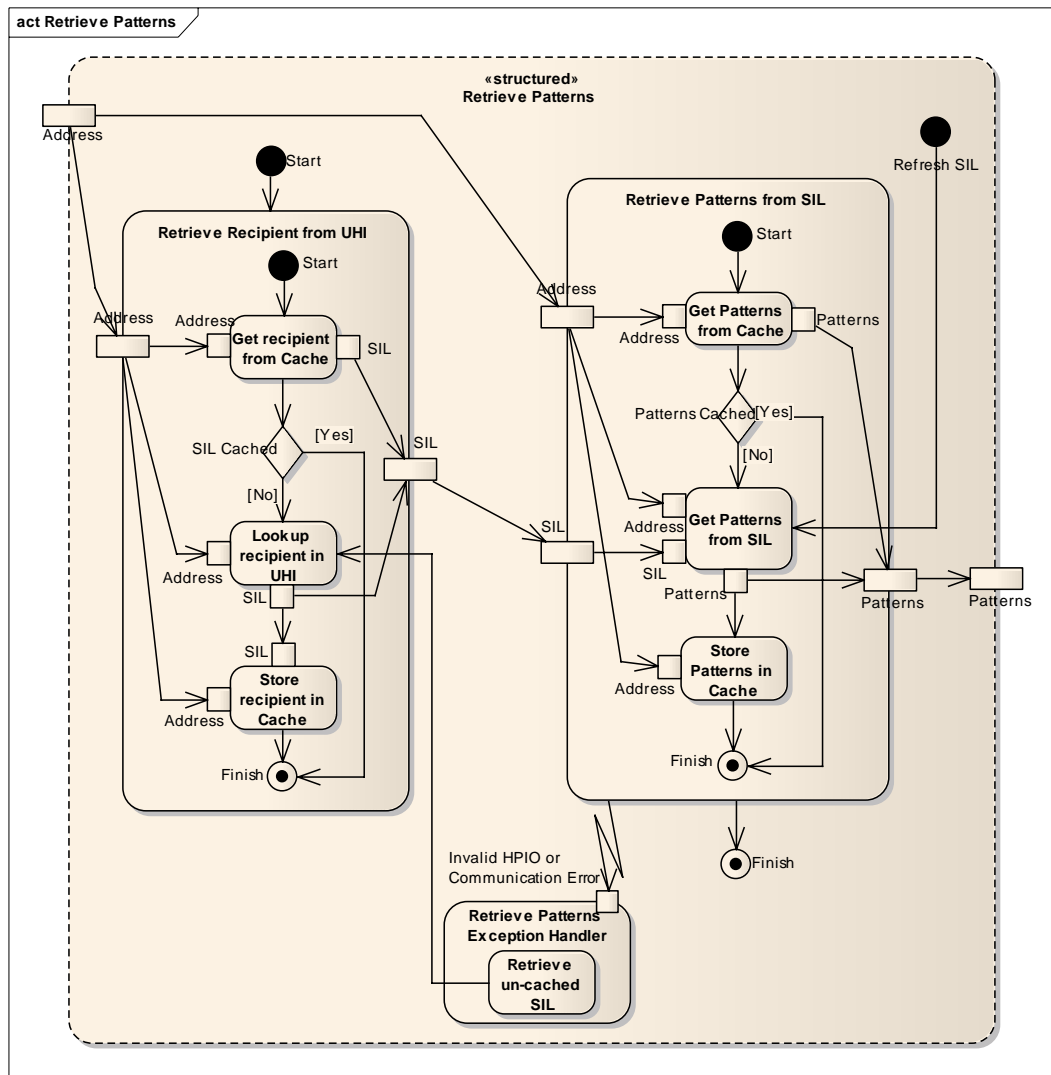


Figure 8: Retrieve Patterns

Inputs: Address

Outputs: Patterns

Retrieving the message patterns that the receiver supports involves two key steps. The first step is to retrieve the recipient’s associated SIL endpoint reference. The second is to use the reference gained in the first step to contact the recipient’s associated SIL to retrieve the patterns that the recipient, or their delegated service provider, supports. These sub activities are described in more detail in the following sections.

3.5.2.1 Retrieve Recipient from UHI

Inputs: Address

Outputs: SIL

As it is expected that the mapping between a HPI-O and their nominated SIL will remain predominately static these mappings **SHOULD BE** cached to reduce the overheads of retrieving these details over the network for subsequent requests to the same recipient. This cache would be referred to in the first instance and if an entry cannot be found the details would be requested from the UHI with the results being stored in the cache to minimize the overhead for subsequent references to the same HPI-O.

Where the request must be issued to the UHI consider the following state diagram for the types of issues that may occur while interacting with the UHI.

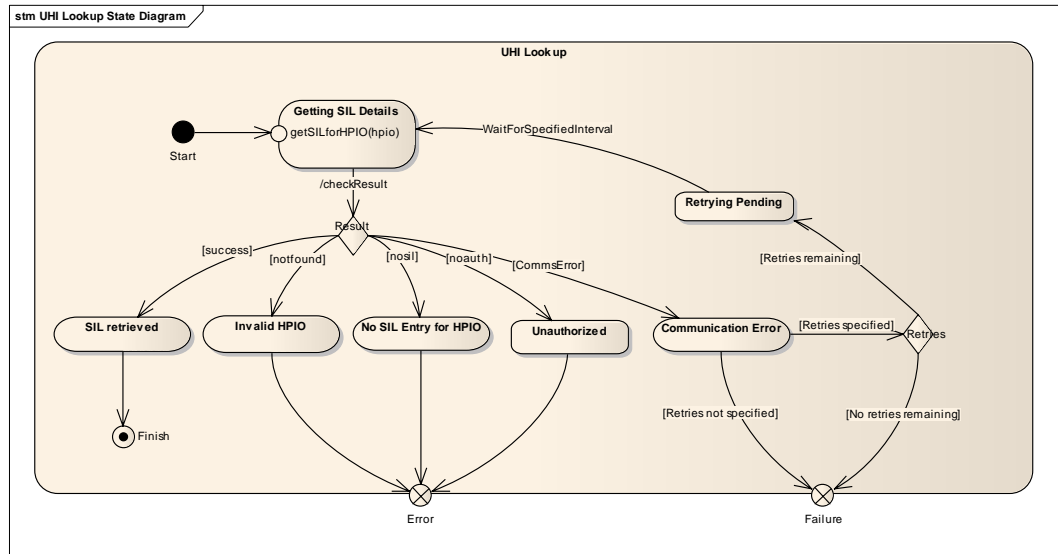


Figure 9: UHI Lookup States

3.5.2.2 Retrieve Patterns from SIL

Inputs: Address, SIL

Outputs: Patterns

As it is expected that the mapping between a HPI-O’s Business Service and the supported patterns for this Business Service will remain predominately static these mappings **SHOULD BE** cached to reduce the overheads of retrieving these details over the network for subsequent requests directed at the same Business Service for the same recipient. This cache would be referred to in the first instance and if an entry cannot be found the details would be requested from the associated SIL with the results being stored in the cache to minimize the overhead for subsequent references to the same HPI-O and Business Service.

The SIL specification states that an ‘Invalid HPIO’ or Communication type error should cause the reference held for the SIL to be refreshed. This is shown in Figure 8 by the exception handler towards the bottom of the diagram. The exception handler passes control back to the remote UHI lookup action to allow the refresh of the SIL endpoint and then continuing to refresh the Pattern entries.

Where the request must be issued to the recipients nominated SIL consider the following state diagram for the types of issues that may occur while interacting with the SIL.

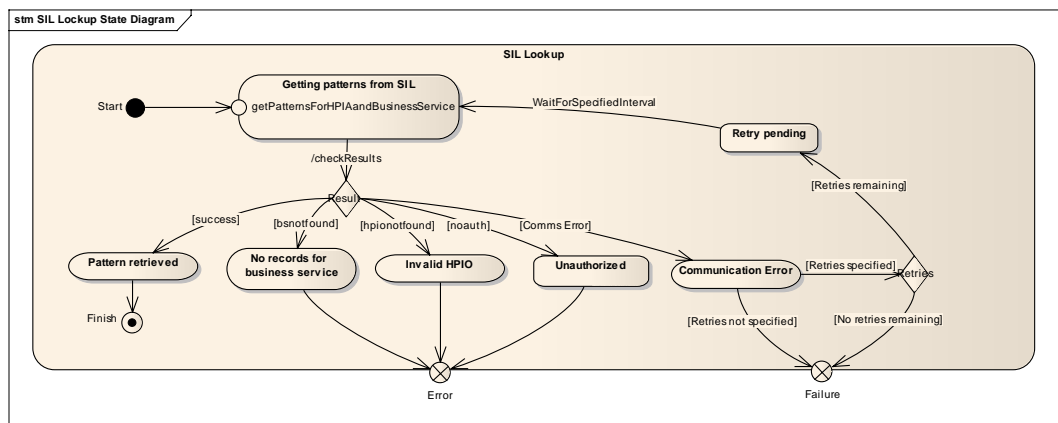


Figure 10: SIL Lookup States

3.5.3 Process Recipient Messaging Pattern

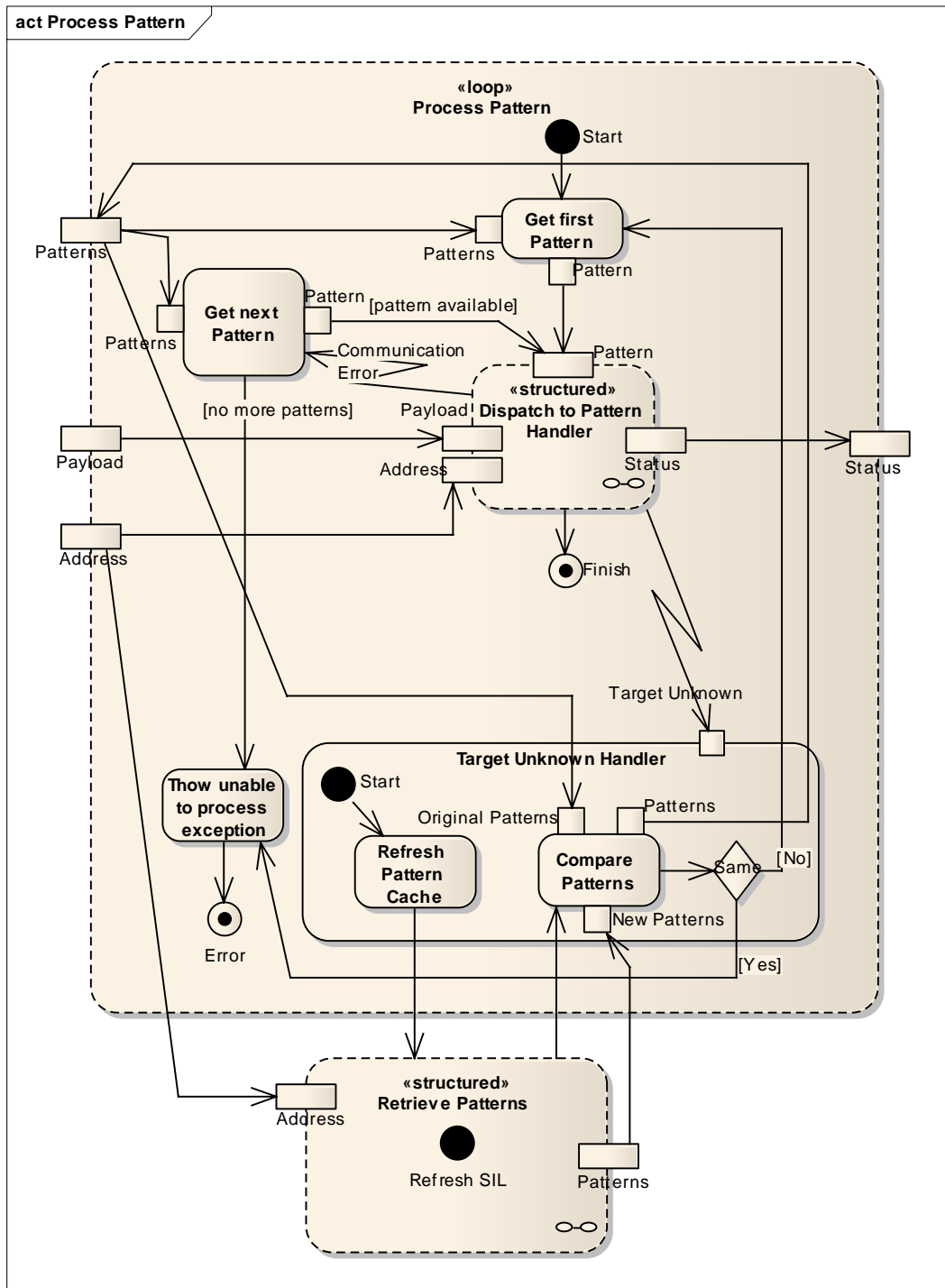


Figure 11: Process Pattern

Inputs: Patterns, Address, Payload

Outputs: Status

The primary goal of the Process Pattern activity is to provide a generic error handling capabilities which will try all identified patterns until a successful pattern is found to deliver the message to the recipient (or their delegate) or all patterns identified have been exhausted through this process (which results in an error condition).

The shortest successful execution path involves retrieving the first pattern from the list of patterns supplied, dispatching to the associated pattern handler and returning the status from the handler. The handler may also

choose to implement prioritisation criteria over the returned patterns to sort the returned Patterns in order to align with local preferences.

If a communication type error occurs the next pattern from the list of patterns is selected and the process is tried again until there are no more patterns in the list. Once the list of patterns is exhausted the handler **MUST** escalated the condition to be analysed and potentially rectified.

When the list is exhausted an exception is generated identifying the message is unable to be processed at this time.

If a 'target unknown' type exception is encountered during the processing an exception handler must capture this exception and attempt to refresh the SIL Patterns cache. The handler must check the results of the refresh to ensure that the patterns have changed else the details are still incorrect and the problem **MUST BE** escalated to be rectified.

3.5.3.1 Dispatch to Pattern Handler

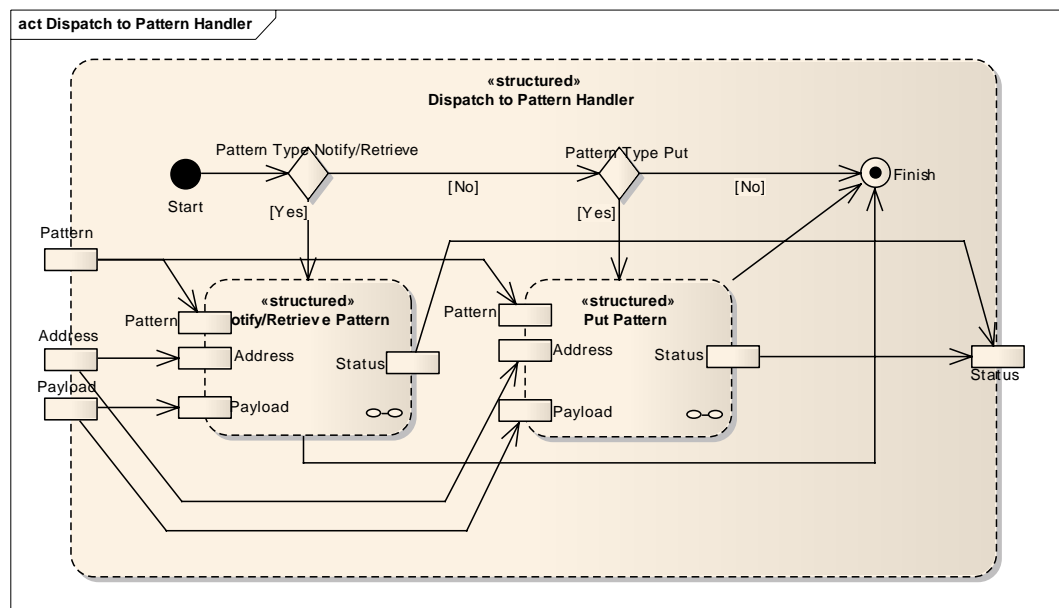


Figure 12: Dispatch to Pattern Handler

Inputs: Patterns, Address, Payload

Outputs: Status

The Dispatch to Pattern Handler consists primarily of a case statement to identify the appropriate Pattern Handler to execute for the pattern that has been given as an input parameter. Both Dispatch to Pattern Handler and the Pattern Handlers themselves comply with the same interface. This allows the addition of new handlers without affecting the structure or behaviour of the encapsulating process, Process Pattern.

3.5.3.1.1 Notify and Retrieve Pattern

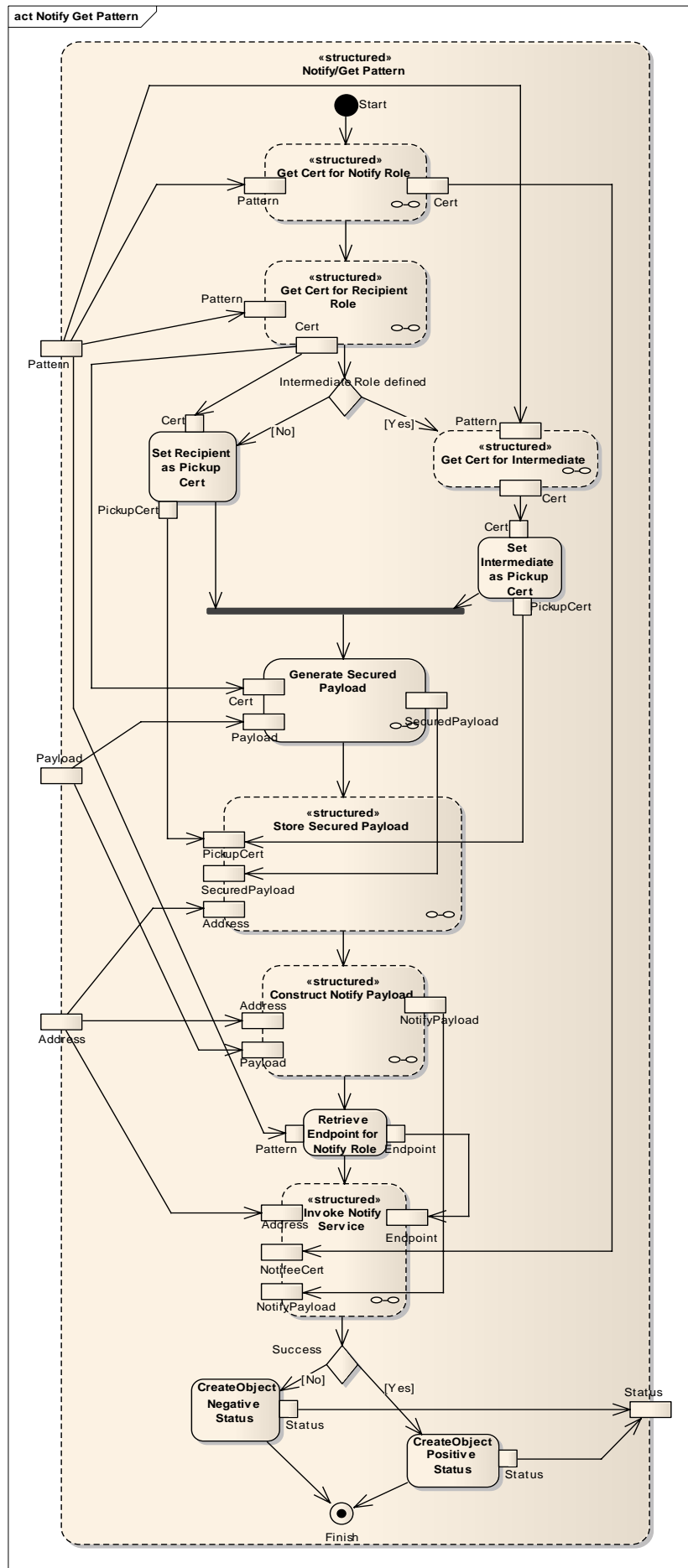


Figure 13: Notify and Retrieve Pattern

Inputs: Patterns, Address, Payload

Outputs: Status

The Notify and Retrieve Pattern activity implements the steps required to store the message for future retrieval by the target recipient and notify a designated party of its availability. See [CPIS2008] for further details.

The initial steps are focused on retrieval of certificates for the signing and encrypting steps that follow later in the activity. There is one conditional step involved in the retrieval of the certificates and this relates to whether an intermediate party will retrieve the message on behalf of the recipient or the recipient will retrieve the message directly. The certificate of the specified retriever must be stored with the message to allow the authorisation of message retrieval at some later time.

After the required certificates have been retrieved the Secure Payload can be constructed and then stored awaiting retrieval by the designated party at some later time.

The next steps are to construct the notification payload, retrieve the endpoint associated with the notification target and issue the notification to the endpoint. Some of the key states for the issuing of the notification to the remote service are shown in Figure 14 below.

The final step is to issue a response to initiating process related to the success of the overall operation. This is done by constructing an status object identifying either success or failure and other ancillary information.

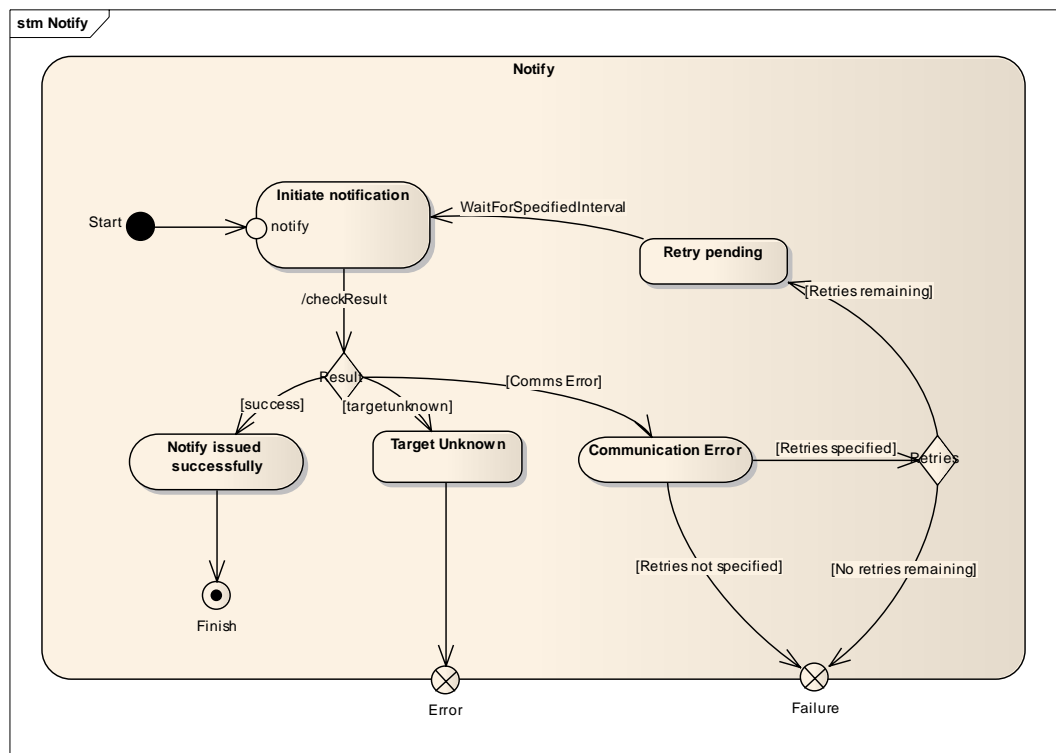


Figure 14: Notify States

3.5.4 Common Send Pattern Processes

There are number of processes that are common across all patterns used to send messages. These are described in the following sections.

3.5.4.1 Get Certificate for Pattern Role

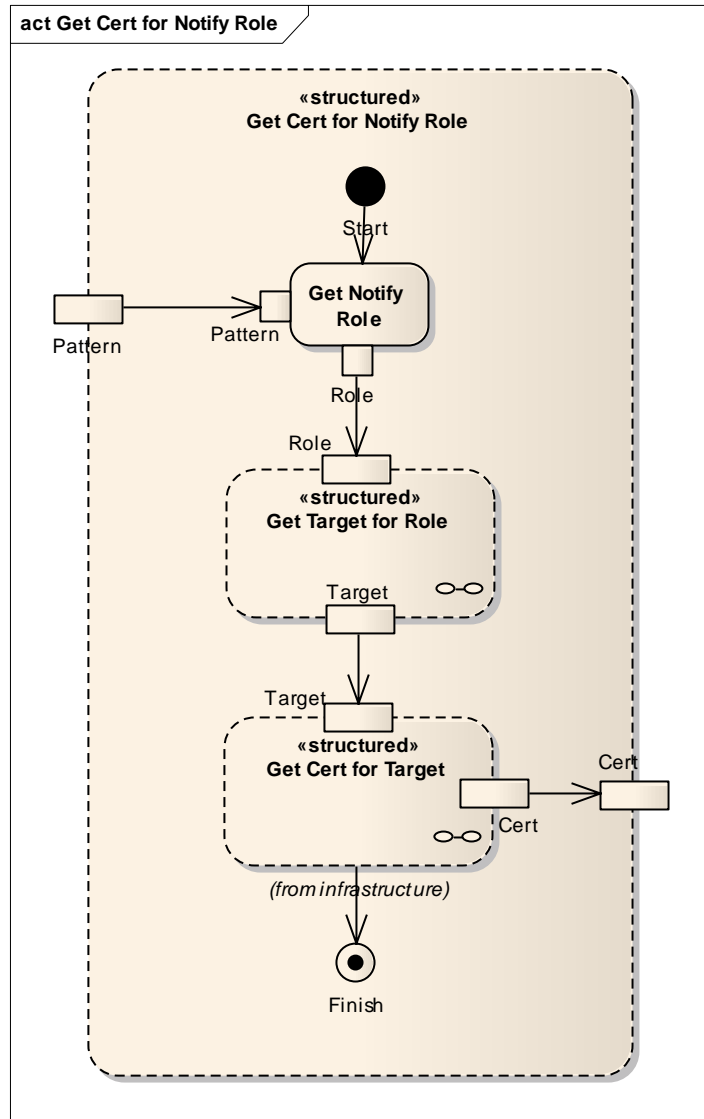


Figure 15: Certificate Retrieval

The diagram above shows the key steps needed to retrieve a certificate for a target entity (i.e. HPI-O). The example given above is for the Notify role in the Notify and Retrieve pattern but is equally applicable to other roles. To provide this capability for other roles the first action would be changed to extract the required role details from the Pattern given. The Get Cert for Target activity is shown in more detail in the diagram below. The Get Cert for Target activity is primarily a pattern for caching or storing certificates locally after the initial retrieval.

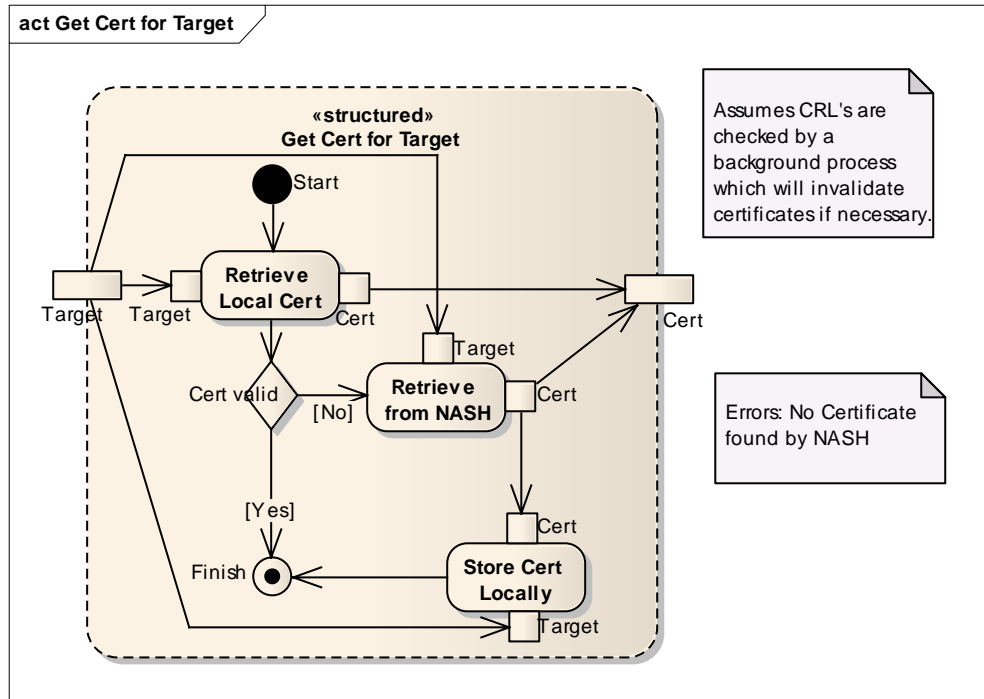


Figure 16: Certificate Caching

3.6 Receive Processing

The follow diagram shows the high level processing activities. The first activity is used to decrypt and verify the secure payload. This activity returns the original payload and details of the party that signed and encrypted the secure payload. The second activity passes the payload along with the details of the party who signed and encrypted the secure payload and the party that provided the transport layer security to the code nominated by the recipient for application specific processing. The transport layer security details must be provided by the proxy component receiving the request when initiating the Workflow Coordinator. The status returned from the application specific processing is then used to send either a successful or unsuccessful response.

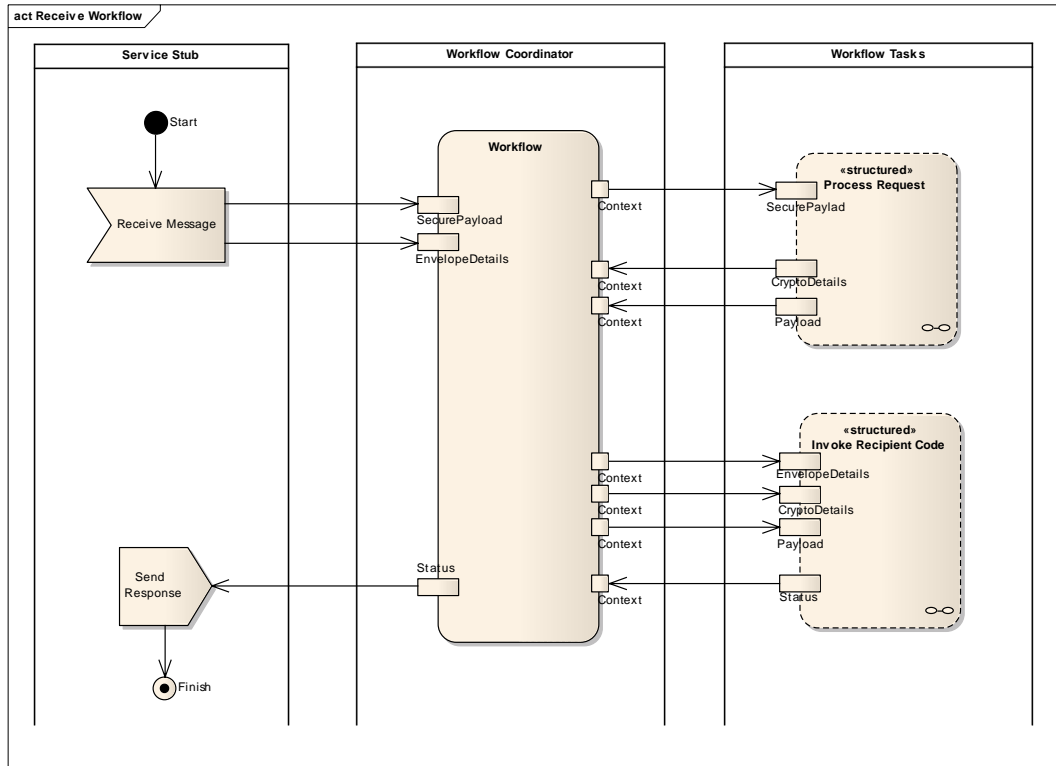


Figure 17: Receive Workflow

3.6.1 Process Request

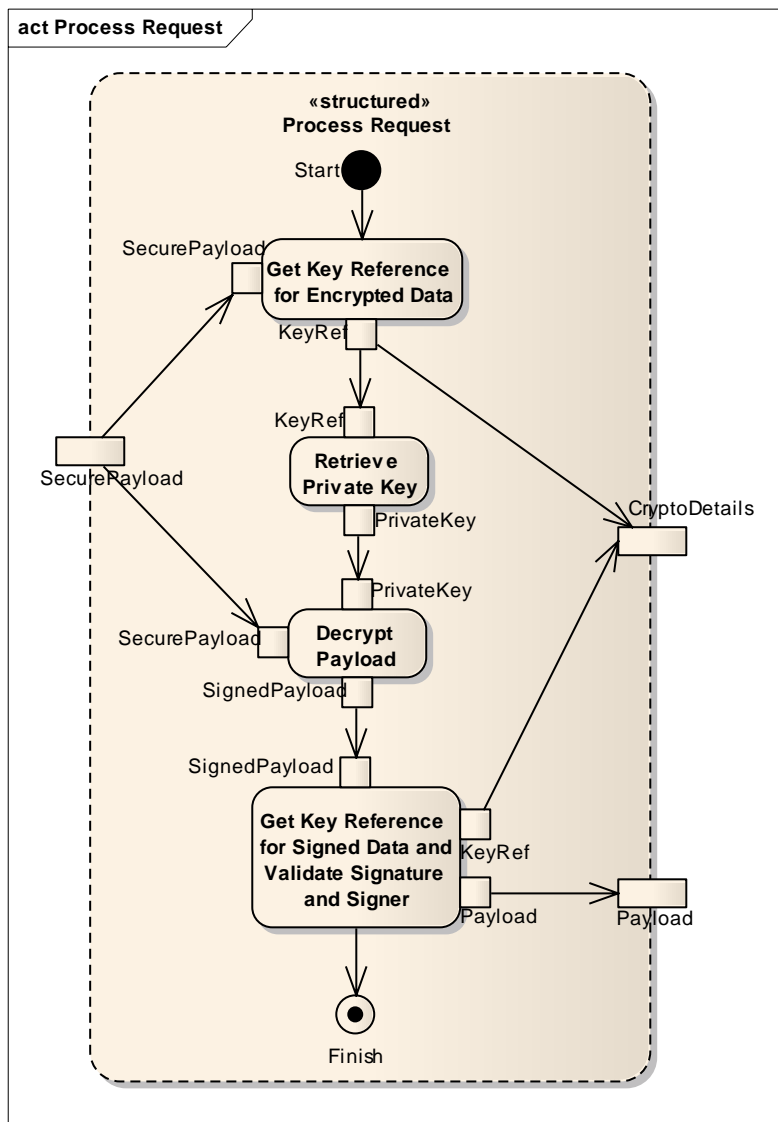


Figure 18: Process Request

Inputs: SecurePayload

Outputs: CryptoDetails, Payload

The Process Request activity is primarily focussed on deciphering the secured payload and returning the original payload and the details of who encrypted and signed the secure payload.

3.6.2 Invoke Recipient Code

Inputs: EnvelopeDetails, CryptoDetails, Payload

Outputs: Status

The Invoke Recipient Code activity provides the deciphered payload and details of how the message was secured prior to processing beginning. These details include information on who signed and encrypted the SOAP message components (EnvelopeDetails) and who signed and encrypted the secure payload (CryptoDetails). These details allow the recipients code base to determine the validity or trustworthiness of the message being processed. The result from the executed recipient code must be a response that represents the status of the execution (i.e. success, failure). This will be packaged up in the response back to the issuing client.

3.7 Acknowledge Processing

There are no distinguishing concepts in regards to acknowledgement processing. Acknowledge processing is performed using the same process as Send processing.

3.8 Acknowledgement Processing

There are no distinguishing concepts in regards to acknowledgement processing. Acknowledgement processing is performed using the same process as Receive processing.

4 Engineering View

Pre-requisite reading for this section include

- Web Service Profile
- Connectivity Architecture document

4.1 Design goals

Some of the key design patterns and goals considered in this design include:

- Dependency injection
- Configuration driven components
- Encapsulation
- Leveraging all of the above to gain agility and flexibility

4.2 Core components

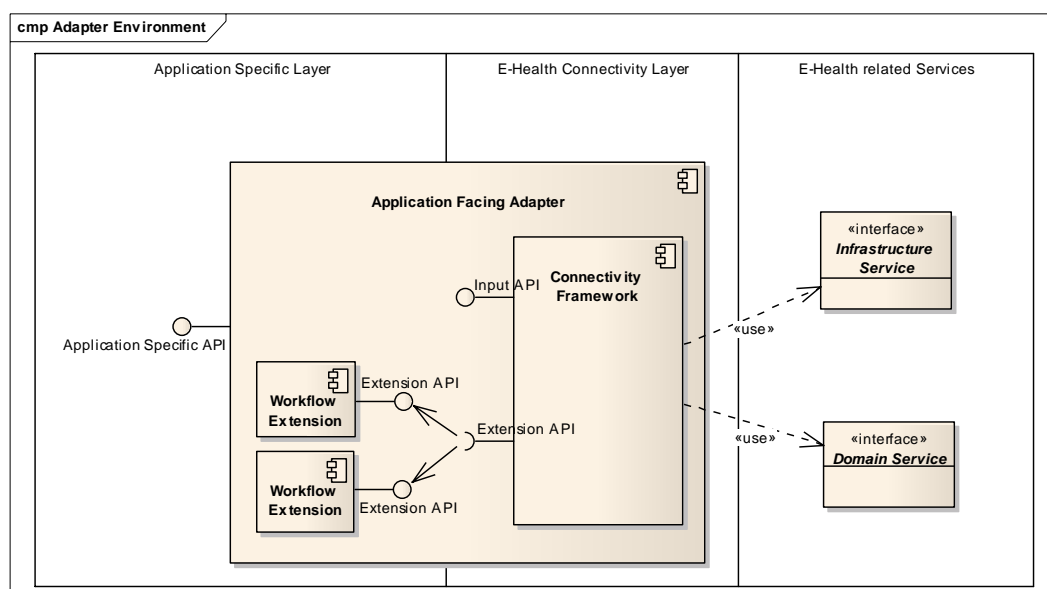


Figure 19: Adapter Environment

4.2.1 Application Facing Adapter

The Application Facing Adapter can be either an active component being driven by events (i.e. files being deposited in a directory or messages arriving on a queue) or passive component that is directly embedded in a larger application. It is responsible for bridging the gap between the applications preferred interaction methods and that of the e-health connectivity environment. It may also provide additional components to extend the basic workflows which have been described in the Computational View section. This could include extensions to provide domain specific transformations.

4.2.2 Connectivity Framework

The Connectivity Framework is a passive component that will generally be embedded in either an active application adapter component or directly in the application itself. The Connectivity Framework is responsible for providing the basic connectivity capabilities to ensure connectivity can be achieved with minimal custom components. The Connectivity Framework can, and is likely

to, be extended via components complying with an extension API and changes to workflow steps.

4.2.3 Services (Domain and Infrastructure)

Services consists of both infrastructure and domain related services. The infrastructure services were introduced in the Infrastructure Services section of the Environment Overview. The domain services are the services that are specific to the various domain related packages. These services are invoked according to the calling pattern by the Connectivity Framework.

4.3 Connectivity Framework in more detail

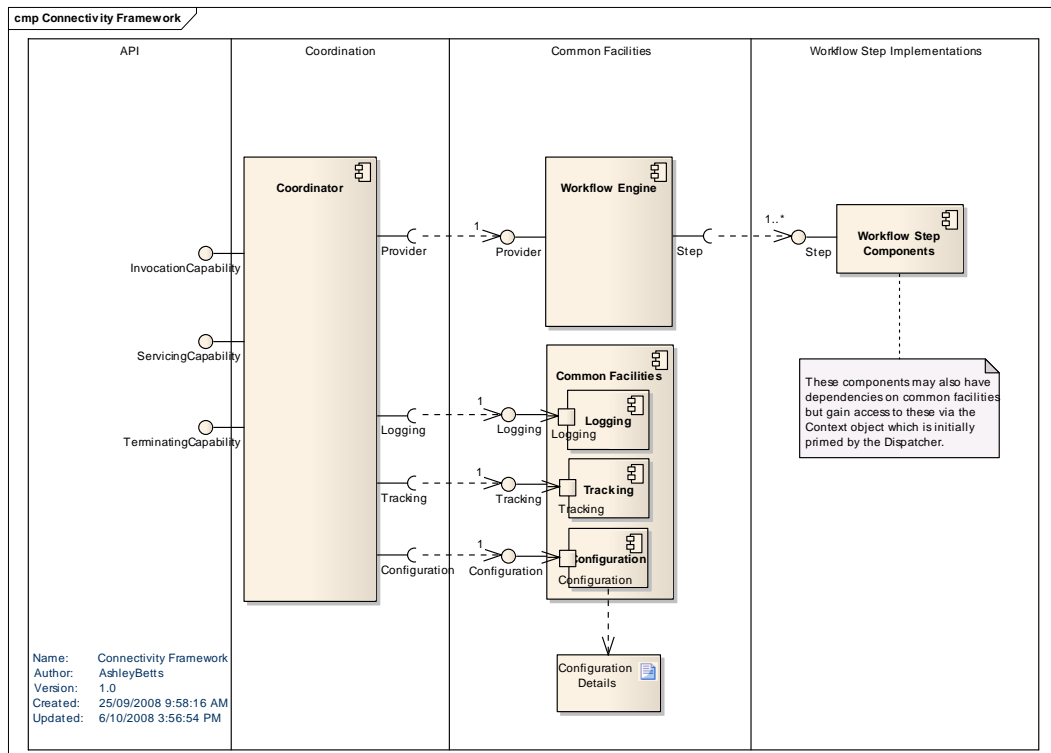


Figure 20: Connectivity Framework

The diagram above gives a high level overview of the more significant components required by the Connectivity Framework and also the interfaces across which these components communicate with one another. The scale of the component in the diagram is by no means an indication of the complexity or size of the implementation. For example, it is expected the Coordinator component would primarily be responsible for initiating some dependency injection of common components into a context for easy access by Step components and then delegating to the Workflow Provider to perform the actual work.

The Workflow Step components are identified and coordinated by the Workflow Engine. Some steps would be supplied by common components and specialised components can be supplied via the extension API. Steps and the components implementing them would be specified to the Workflow Engine via the common Configuration component.

4.3.1 Application Programming Interface (API)

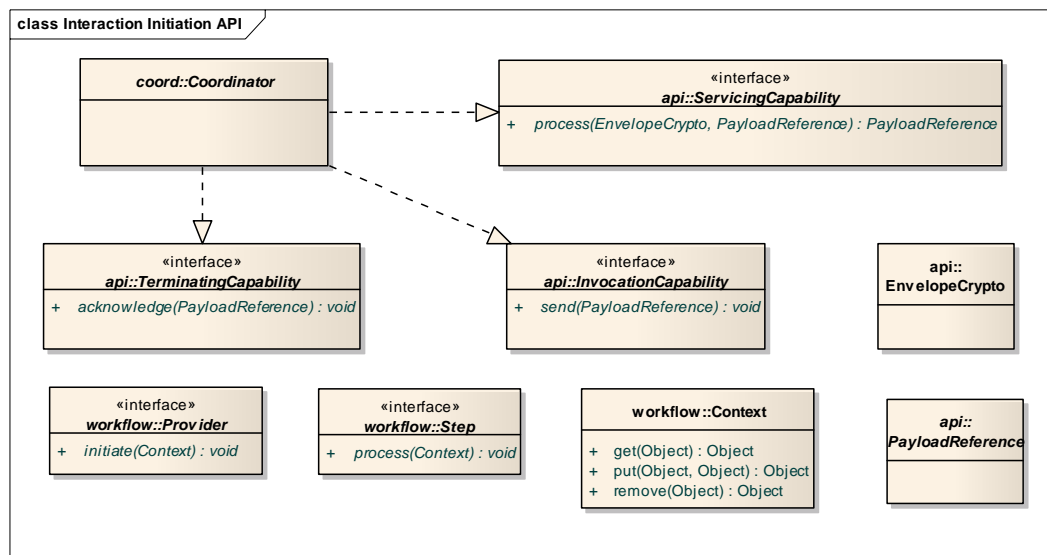


Figure 21: API Structure

The API provides a well defined abstraction layer over the core functionality of the Connectivity Framework and its extension points. In the diagram above, the interfaces in the `api` package for the core API and interfaces in the `workflow` package form the extension API. These interfaces are not to be considered complete in this high level design but rather a foundation on which a more thorough specification can be based in a low level detailed design.

A service receiving a request would dispatch using the `process` method of the `ServicingCapability` interface exposed by the `Coordinator` component. This provides domain specific Web service server side stubs a common entry point into the framework allowing processing steps required by the [WSP2008], [XSP2008] and [CONA2008] to be performed prior to dispatching the resulting payload to the code implementing the domain specific functionality.

4.3.2 Workflow Control

This high level design doesn't specify exactly what implementation must be used to fulfil the Workflow Control component of the framework or describe its design in significant detail. There are numerous tools that can be used to provide this functionality, Apache Chains and State Chart XML to name a few. A design requirement however of this component is that it must allow components to be easily plugged into the workflow and preferably be configurable by configuration files or other flexible means.

4.3.3 Workflow Steps

This design assumes that input and output information for Workflow Steps are received and passed back via a context object. This was briefly discussed earlier in the Notes on Diagrams section. This provides an environment allowing separation of concerns to exist between the Step components. The Steps are decoupled from one another only knowing the expected inputs and outputs for their operation without any dependencies on up or down stream components. This will allow a flexible environment where components can easily be replaced or slotted in to enhance an existing process.

The context object **SHOULD BE** implemented as an associative array allowing an object of interest to be associated with a specified key. All runtime state for Steps **MUST BE** store only in the context object. This will allow Steps to be re-entrant and support multiple concurrency. Configuration state can be

specified at initialisation time to configure the same component to have various behaviours depending on parameter values but once set for an instance these values **SHOULD NOT** be changed during normal execution.

The following diagram provides a high level example of the Coordinator component priming the context object with objects from the common facilities for reference and later use by downstream Steps.

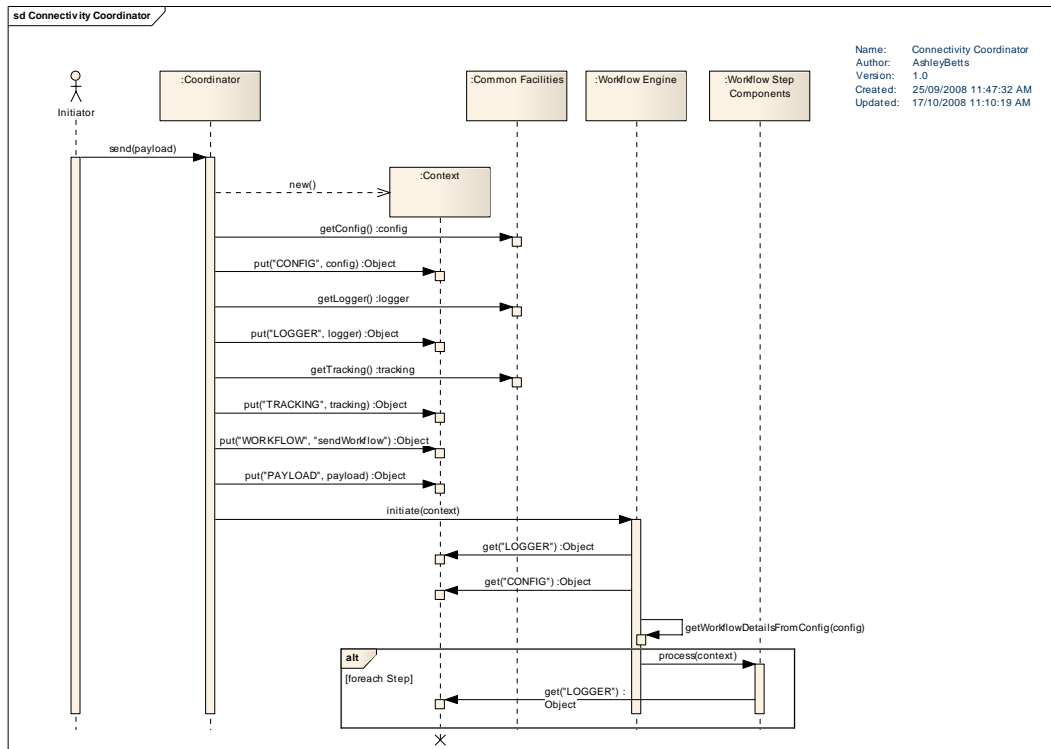


Figure 22: Priming the Workflow Context

4.4 Common Workflow Steps

4.4.1 Logging

The logging capabilities should be provided via an interface which is made available for use by each of the steps as a value in the context object. The disassociation with the implementation allows new methods of logging to be provided without any changes being required to the code that depends purely on the interface.

4.4.2 Tracking

Tracking is used by components that interact with services to identify who has, or has had, control of the message processing. This facility would be used primarily for problem management and possibly business activity monitoring capabilities. This could be implemented as an extension to the logging capability. When control is passed to and received from an external service, tracking information must be logged. It is also suggested that tracking information is logged when control passes between major components, is awaiting action or is stored locally.

4.4.3 Configuration

The Configuration component provides configuration information to the common and extension components of the framework. Providing configuration via a single component and interface ensure that all configuration information

is presented in a consistent manner to other components. This also provides the ability to have a single source of all configuration information possibly leading to better configuration management practices.

One aspect of the design that should rely on the configuration component is a mapping of service provides a mapping between a service identifier (likely an URI) and a class representing a client stub for an external service. The class would comply with an interface containing a method accepting a context object. This client stub would be responsible for extracting the required details from the context object and calling the specific Web service and adding the results from this call back into the context object for later steps to refer to. This method will allow the framework to implemented in a more generic fashion with only the client stubs being aware of the specifics of definition of remote Web services.

Appendix A: Informative references

- [CONA2008] NEHTA, *Connectivity: Architecture v1.0*, 1 December 2008.
- [CPIS2008] NEHTA, *Concepts and Patterns for Implementing Services v2.0*, 1 December 2008.
- [SILA2008] NEHTA, *Service Instance Locator: Architecture v1.1*, 1 December 2008.
- [UHICO2007] NEHTA, *Unique Healthcare Identification – Concept of Operations v2.0*, 3 September 2007.
- [WSP2008] NEHTA, *Web Services Profile v3.0*, 1 December 2008.
- [XSP2008] NEHTA, *XML Secure Payload Profile v1.0*, 1 December 2008.