Enterprise Architecture Policy

Integration Patterns Procedure

1. Purpose
This procedure guides decision making for designing an interface in an integration environment that allows information to be shared quickly and accurately throughout the organisation and across organisational boundaries, in a secure fashion satisfying privacy and consent concerns in the delivery of healthcare for the Department of Health.

2. Scope
This procedure applies to all employees, contractors and consultants within the Department of Health divisions, agencies and commercialised business units.

The scope of this document is limited to Functional Integration and Data Integration. This document does not include Presentation Integration.

3. Supporting documents
Authorising Policy and Standard/s:
- Enterprise Architecture Policy
- Enterprise Architecture Framework Implementation Standard
- Enterprise Architecture Foundations Implementation Standard
- Enterprise Business Architecture Implementation Standard
- Enterprise Information Architecture Implementation Standard
- Enterprise Applications Architecture Implementation Standard
- Enterprise Technology Architecture Implementation Standard

Procedures, Guidelines and Protocols:
- ICT Cabling Protocol (under development)
- Identity Management and Demographics Protocol (under development)
4. Related documents

- Queensland Government Enterprise Architecture Framework 2.0
- Department of Health Information Security Policy

5. Approach

Queensland Health requires an integration environment that allows information to be shared quickly and accurately throughout the organisation and across organisational boundaries, in a secure fashion that satisfies privacy and consent concerns.

In particular, the integration environment is intended to:

- Enable systems to access information that is integrated, accurate and relevant
- Enable information to be consolidated and shared from multiple source systems
- Enable the sharing of information and collaboration amongst a network of providers across organisational boundaries
- Support participation in national health initiatives
- Support a decentralised organisational delivery model
- Support reuse of integration services
- Support Enterprise Application Integration (EAI) for message based interfaces (such as HL7), and integrating legacy applications
- Support enterprise services and standards based web service integration
- Support Service Oriented Architecture design, build and delivery concepts.

To achieve these outcomes, a number of different integration patterns are described to provide direction on how integration technologies should be configured. This paper should be read in conjunction with the Department of Health Systems Integration Protocol.

6. Functional Integration Patterns

Each pattern has been allocated a status to indicate usage within the enterprise. These notations are shown at the top right corner of each pattern diagram.
The three main groups of patterns described in this section are:

- Broker
- Managed Process
- Point to Point (Direct connection).

These main groups are further broken down into a number of separate patterns as shown in Figure 2 below. The Message Exchange patterns are the building blocks that are used in various combinations to construct the functional integration patterns. The Message Exchange Patterns are not specific to the enterprise, and are summarised in Appendix A.
All of the patterns use the integration platform except the Point to Point patterns.

6.1 **Point to Point (Direct Connection)**

This pattern allows two applications to communicate directly with each other. In general this pattern is discouraged, as it tends to result in many point-to-point ‘spaghetti’ connections, bypassing the integration platform for controlling and mediating the messages. As such, it cannot be governed for conformance to data integration and exchange standards and supported mechanisms. Centralised data validation and mediation controls are bypassed and other systems interested in the data may be excluded from the information. Using this pattern can result in data inconsistency, as transforms are done differently, based on different understanding of the business domain by different teams at different times.

This pattern decreases the standardisation of the system and sustainability of the system but increases the potential maintenance and fragility of the system. This pattern features prominently in the current enterprise ICT environment.

The Point to Point or Direct Connection pattern has two variations depending on whether a response is required, and can be either two way (request / response) or one-way (unsolicited). Both variations may be used either with synchronous or asynchronous communication protocols.

---

**Figure 3 – Point to Point – One Way**

---

**Figure 4 – Point to Point – Two Way**

Point to Point web service integration is a special case of this pattern where web services technologies are used without an Enterprise Service Bus to mediate. This is discouraged as it results in many point-to-point connections in the same way as traditional point to point connections.
6.2 Broker

Endorsed Australian and international standards shall be adopted wherever appropriate.

This group of patterns separates the message transformation and routing rules from the source and target applications. It allows a single interaction from the source application to be routed to one or more target applications. One of the key benefits this pattern offers is eliminating point-to-point connections, enabling more flexible solutions that are easier to maintain. This pattern is fundamental to traditional Enterprise Application Integration (EAI) and is also applicable to Service Oriented Architecture (SOA) (although use of this pattern does not guarantee SOA). Figure 5 shows the generic Broker pattern which is broken down into more specific patterns in the subsequent sub-sections.

![Broker Diagram]

There are several patterns within this group:

- Mediated Request / Response – send a message to one or more target systems and receive a response. This exchange can be both synchronous and asynchronous provided that the exchange is mediated by the integration platform.
- Router - several variations which send message to one or more targets (based on message content or context).
- Publish/Subscribe - send message to list of subscribers based on topics of interest selected by subscriber.
- Scatter-Gather - send a message to multiple targets and aggregate the responses back into a single message to return to the recipient.

The integration platform should not contain any business logic, apart from what is required to validate messages and perform basic routing.

When to use:

- to eliminate point to point solutions
- to integrate heterogeneous applications
- to perform integrations involving a mix of protocols and message communication patterns
- to apply updates to multiple target applications
- to aggregate information from multiple applications
- to route to one of several possible target systems
- to notify interested subscribers of events.
- to broker SOA services
- to centrally manage quality of service, logging, auditing and security
The Broker pattern is generally stateless, or maintains state only for correlation of messages over near real-time short-running integration processes. Where state needs to be maintained over a long-running process then the Process Manager pattern should be used.

### 6.3 Mediated Request / Response

The Mediated Request / Response pattern is initiated by the data recipient sending a query to the data producer. This exchange can be both synchronous and asynchronous provided that the exchange is mediated by the integration platform. The event that triggers the data recipient to make the request can be the admission of a new patient, the login of a care provider or numerous other activities.

This pattern is preferred for service oriented architecture where data is not cached or duplicated and systems interoperate to exchange information. This pattern is not widely used in the current enterprise integration environment, but will become increasingly important as common business services are deployed.

**Figure 6 – Mediated Request / Response Pattern**

Figure 6 shows four variations of the Mediated Request / Response Pattern:

1. Full synchronous from source to target
2. Synchronous to integration platform and asynchronous to target
3. Asynchronous from integration platform and synchronous to target
4. Full asynchronous from source to target.

The variations involving asynchronous processing show acknowledgements being sent in response to messages. These are required to support guaranteed delivery which is commonly used with this pattern.
Department of Health Integration Patterns Procedure

System acknowledgements are acknowledgements sent from the integration platform. Receipt acknowledgements are those sent from the receiving application.

6.4 Content-based Router

**Pattern Status: Support**

The pattern is used when the integration platform makes a decision on where to route the message based on the contents of the message (e.g. the target may be one of several possibilities depending on the value of an attribute or attributes in the message). This pattern does not use the publish and subscribe capability to determine the data recipients. This pattern is acceptable where the recipients do not change frequently. Although the recipients are not pre-defined from the perspective of the sender, changes are required in the integration platform every time a new recipient has to be added (to map content fields to new endpoints in the endpoint database).

In accordance with the integration guidance document statement ‘separate integration logic from business logic’, care should be taken to ensure that business logic is not duplicated in the integration platform – the routing should be driven by simple integration logic. Where more complex business logic is required to drive the routing, the Process Manager pattern should be used (with rules stored in a rules engine).

Each receiving node may have their own filters to ignore unwanted messages and process only the messages that it is interested in. This pattern features prominently in the current enterprise ICT environment. Although supported, other patterns such as the Publish / Subscribe pattern are preferred. Where the set of recipients is subject to change and/or cannot be unambiguously determined from the content, the Publish / Subscribe pattern should be used.

![Content-based Router Filtered Diagram](image)

**Figure 7 – Content-based Router - Filtered**

Unfiltered routing is functionally equivalent to email spam. This is because this pattern results in large amounts of data being sent to recipients who neither need nor want the data. This pattern puts unnecessary strain on the network, the integration platform and on the recipient systems. Unfiltered routing is discouraged.
6.5 Publish / Subscribe

The Publish / Subscribe pattern allows topics of various descriptions and granularities to be created that align to events that occur within a system (e.g. topics could include 'new and updated pathology records', 'new and updated records for a specific patient' etc.). Systems wanting to be informed of events subscribe to the corresponding topic(s) through the integration platform.

Systems that execute one of the events of interest publish a notification of the event to the integration platform by sending a message. The integration platform maintains lists of published topics (subjects) and subscribers, and notifies subscribers as events of interest occur.
There are two types of subscriptions, durable and non-durable. The difference is whether the subscriber has changed as the result of an event instead of a notification that the event has occurred. The integration platform then determines who to send the data to, based on the subscriptions. With this pattern, the data recipient receives the data without solicitation. However, the data producer relinquishes control over which systems are allowed to have the data with this pattern.

There are three key pre-conditions for implementation of the publish/subscribe pattern:

- Topics should be created within the integration platform for events to align to.
- All interested systems should subscribe to topics of interest to receive notification.
- Subscribers should be capable of receiving one of the pre-determined message/notification formats supported by the integration platform.

There are two types of subscriptions, durable and non-durable. The difference is whether the subscriber wants to receive messages that are published when the subscribing application is not running. When a durable subscriber is offline, the integration platform will store messages published to the topic, and deliver them when the subscriber reconnects. With non-durable subscribers, the subscriber does not receive any messages published to the topic while the subscriber was disconnected. Durable subscriber is the default configuration for Department of Health.

The Publish/Subscribe pattern may be used in conjunction with the Content-based Router pattern to enable Content-based Publish / Subscribe. With this pattern combination, the integration platform matches the event message against a set of subscribers, and also checks if the subscriber matches a
particular message field or fields and a set of values. If a match exists between the message content and a subscriber, the message is then forwarded to each matching subscriber. The Content-based Publish / Subscribe approach is more flexible, enabling routing to be refined based on individual fields in the message.

**HL7 Publish / Subscribe**

For clinical message-based integration, this pattern may be implemented using the HL7 support for Publish / Subscribe. For each message stream, the publisher (or integration platform) can define a publish and subscribe ‘Conformance Statement’ which sets up the rules for filtering and query matching. To establish a subscription, an authorised subscriber sends a query message which specifies the pattern to select messages of interest. The integration platform has a list of all subscription requests, and for each message it checks the query filters associated with the subscriptions. If the message matches a query filter, it is sent to the recipient.

### 6.6 Scatter-Gather

This is a broker pattern that sends a message to multiple targets and aggregates the responses back into a single message to return to the recipient. This pattern can be used to communicate with multiple target systems in update mode to apply updates across multiple target systems, or in read only mode to aggregate responses from multiple systems.

The integration platform uses rules-based mediation to coordinate message interaction with the various target applications, and perform the appropriate message transformations. Messages to each target are usually request /response (the pattern can be thought of as a composite of several instances of the Mediated Request/Response pattern.

The messages may be sent to all targets at the same time (no sequence or synchronisation between individual requests), or there may be some sequence or synchronisation required between the various updates. Messages may be synchronous or more commonly asynchronous.

Where updates to multiple systems are involved, compensating transactions may be required to reverse partially completed failed processes in the event of failure.

Superficially, this pattern is similar to the Process Manager Orchestration pattern, but the Scatter-Gather
Broker pattern is more concerned with technical integration (EAI integration) as opposed to orchestration of business services (SOA services).

This pattern is used to expose services through an Enterprise Service Bus (ESB). The consuming application accesses the service via the ESB which deliver the information to the requestor without the calling application needing to know what happens ‘under the covers’. For example, the service could present information from multiple sources such as Client Directory and ePADT, without the calling application knowing anything about these applications.

This pattern can also be used in legacy enablement to expose services from legacy functionality.

6.7 Process Manager

The Process Manager pattern is used when multiple integration operations are aggregated into a multi-step process. This pattern refers specifically to integration-centric process management where the process manager is being used to facilitate a near-real-time integration process, with no human interaction between the steps.

The process may be stateless or may maintain state to correlate messages. The process may be synchronous and complete in a single unit of work, or asynchronous with separate units of work for each step. Where multiple units of work are involved, compensating transactions may be required to reverse partially completed failed processes.

The Process Manager pattern has all of the capabilities of the Broker pattern, including message transformation, message routing, protocol conversion, message broadcasting. In addition, it enables process logic to be managed by process rules that define execution rules for each target application. It also maintains state enabling results of intermediate steps to be stored.

This pattern does not cover long-running Workflow processes involving human interaction steps.

Where there is no complex business logic involved, and there is not a requirement to maintain state (other than to correlate messages), the Broker pattern should be used in preference to the Process Manager pattern. Due to the overlap in capability of the Process Manager pattern and the Broker pattern, there are some situations where either pattern could be used.

6.8 Orchestration – Serial Process

In the context of this document, this pattern refers specifically to integration-centric orchestration where the process manager is being used to facilitate a near-real-time integration process with no human interaction between the steps. This pattern does not cover long-running Workflow processes involving human interaction steps.

The pattern is usually used within a Service Oriented Architecture context to orchestrate standards-based services. The pattern could also be used in to orchestrate low level technical (non-SOA) ‘services’ into a more coarse-grained SOA service.

This pattern enables an application to invoke a service that performs a process involving the sequential or parallel execution of a number of other services hosted by several target applications. This pattern can increase flexibility by externalising the process flow logic from individual applications.
This pattern will become more prevalent as SOA matures within the enterprise and the pool of services to orchestrate or compose increases.

Where update services are involved, this pattern may require the use of retry logic and compensating transaction logic to reverse parts of partially executed processes in the event of being unable to complete the process.

This pattern can be a combination of other patterns. e.g. some steps of the multi-step integration process may use the Broker pattern to take advantage of its routing and transformation capabilities.

This pattern should be used:
- when there is a complex multi-step integration where results from one step influence subsequent steps
- where there is a need to maintain state across the multiple steps of an integration process
- rules-based routing where rules are specified in BPEL or equivalent technology.

Superficially, this pattern is similar to the Broker pattern, but the broker is more concerned with technical integration (EAI integration) as opposed to orchestration of services.

### 6.9 Orchestration – Parallel Process

This pattern extends the Orchestration - Serial Process pattern by allowing process steps to be executed concurrently. Parallel processing enables a process to be orchestrated through multiple processing steps when the required steps may not be known at design-time and are not be sequential, and also enables processing time to be reduced by implementing concurrent processing streams.
Parallel processes are more complex to design and manage than serial processes with the processing complexity and number of error scenarios that need to be catered for rising steeply as the number of parallel steps increases.

This pattern is generally discouraged - the preferred approach is to use a serial process, using parallel processing only where there is a compelling benefit.

The pattern is suited for use in a Service Oriented Architecture context for automating straight-through processing (i.e. workflow). It is less commonly used as an integration pattern.

6.10 Process Choreography

Choreography is a set of peer to peer interactions usually involving B2B interaction, defining the steps and conditions under which cooperating independent participants exchange messages in order to perform a task. Unlike orchestration, there is no central controlling service or 'conductor'. The integration platform coordinates interaction with its immediate peers, but plays no part in coordinating interaction between other systems involved in the process.
Choreography is collaborative in nature with each participant involved describing its part in the interaction, and interacting in a peer to peer fashion. Choreography allows a service consumer to control the execution of service interactions based on the intermediate results of service execution. Choreography typically involves interactions across organisational boundaries where control of the service interactions is not owned by a single organisation.

Choreography is usually implemented by breaking it down into separate orchestrations (one for each participant involved).

This pattern is generally used in a workflow context rather than an integration context.

### 6.11 Complex Event

The integration platform collects multiple events and processes them using filtering, aggregation and correlation. When a combination of events is detected that satisfies prescribed rules, a message or notification is sent to interested parties. The recipients of the composite event may be determined by topic subscription (Publish / Subscribe) or by some other pattern such as Content-based Routing, Orchestration etc.

The multiple event processing capability differentiates this pattern from the basic Publish / Subscribe pattern which responds to single (simple) events by sending out messages or notifications.
The various components in the complex event pattern are loosely coupled or decoupled with the publish component having no knowledge of the consumer of the event information. The integration platform provides the common messaging to achieve loose coupling and asynchronous messaging.

Complex-Event Processing (CEP) systems are used to identify patterns in events such as when anomalies or opportunities are detected, expiry dates are imminent, or preset thresholds are exceeded. Alerts or reminders are then sent to interested recipients, often using Business Activity Monitoring (BAM) dashboards to represent the data graphically. BAM enables business decisions to be made based on information extracted from business activities in near real time, as opposed to making decisions based on last night’s (or last month’s) data in a batch data warehouse.

7. Composite Patterns - Functional & Data Integration

These patterns are specialisations of the Broker group of patterns where either the source or target is a database or file (these patterns can be thought of as hybrid patterns used to integrate two applications where one application is using ‘Functional Integration’ and the other is only capable of ‘Data Integration’).
7.1 Database in - Message out

Pattern Status: Discourage

This pattern is used to upload information from a database to the integration platform. The integration platform uses a database adapter (e.g. Java Database Connectivity JDBC) to execute an SQL query to fetch database information (this may be achieved by calling a database stored procedure). The request may be triggered by periodic polling or by an input message from another system.

The integration platform reads the database result set and extracts, transforms and formats it into messages. The result set from the database may be transformed into a stream of separate messages, or embedded in a single message. The messages are sent to a target system using one or more of the Message Exchange Patterns described in Appendix A.

This pattern is discouraged as it bypasses the business logic of the owning application by interfacing at the database layer as opposed to the application layer.

7.2 File in – Message out

Pattern Status: Discourage

This pattern is similar to the Database Upload to integration platform pattern above. The pattern is used to upload information from a batch file to the integration platform. The integration platform uses a file adapter to transfer a file from a directory to the integration platform. The request may be triggered by periodic polling for the existence of a file in a pre-defined location, or may be triggered by an input message from another system.

The integration platform reads the file, and extracts, transforms and formats it into messages. Once the entire file has been read and persisted by the integration platform, the integration platform usually initiates a physical or logical deletion of the source file to indicate that it has been processed.

The information from the file may be transformed into a stream of separate messages, or embedded in a single message. The messages are sent to a target system using one or more of the Message Exchange Patterns described in Appendix A.
Where large files are involved (over 10 Mb), a specialist ETL (extract, transform, load) tool should be used to implement this pattern.

File integration is a fairly inefficient method of integration and adds latency to the process. There are situations where this pattern may be appropriate, but in general this pattern should only be considered as a last resort after other patterns have been exhausted.

This pattern should be used when the input can only be obtained from a file, but the target system can support functional integration.

![Diagram: File in – Message out]

**Figure 18 – File in – Message out**

### 7.3 Message in – Database out

Pattern Status: Discourage

This pattern is used to apply messages from a sending application directly to a database. The main issue with this pattern is that it bypasses the business logic of the owning application by interfacing at the database layer as opposed to the application layer. Essentially it is using the integration platform to circumvent the principle that an application should not update a database owned by another application. This pattern is therefore generally discouraged, and should be used as a last resort.

The sending system sends messages individually to the integration platform (which may accumulate the messages in a queue). The individual messages are applied to a database using a special database adapter (e.g. JDBC adapter). The trigger for database insert or update could be the receipt of a message, or the update could be initiated according to a predefined schedule.

The database interaction is normally executed as a single unit of work (XA compliant transaction) and the message is deleted from the integration platform queue on success of the database operation; on failure the transaction will be rolled back and retried (if guaranteed delivery is specified).
7.4 Message in – File out

This pattern is used when an application capable of sending messages needs to send data to an application which expects to receive a batch file as input. The integration platform accumulates the messages from the sending system in a queue. The individual messages are persisted in the integration platform and written to a network file using a special file adapter. The trigger for file transfer could be the receipt of a message, or could be initiated according to a predefined schedule.

When the file has been successfully transferred, the transactions are physically or logically deleted from the queue to mark them as delivered.

Where large files are involved (over 10Mb), a specialist ETL tool should be used to implement this pattern.

It is difficult to manage the guaranteed delivery and processing of file integration and consequently this pattern should only be considered as a last resort after all other patterns have been exhausted.
8. Data Integration

The patterns described in this section are:

- Shared Database
- File Transfer
- Maintain Data Copies
- Federation
- Operational Data Store.

These patterns are all supported in specific situations where data movement or data synchronisation is required; as outlined in the pattern descriptions below. However these patterns should not be used as an alternative to functional integration where applications need to be integrated at the business logic layer.

8.1 Shared Database

This pattern allows multiple applications to access a single data store directly. In general, applications should not make direct connections to a database owned by another application, as this can bypass security controls, bypass business rules, and create performance problems. The Shared Database pattern requires all applications to use a common schema.

The most common use of this pattern is within transactional systems where related applications or sub-systems may share a common database (e.g. a Radiology Information System and a Picture Archive and Communication System system may share the same database). It may also be used in to enable multiple Business Intelligence applications to access a common source of data (e.g. a Data Mart).
8.2 File Transfer

Pattern Status: Contain

File Transfer has been used traditionally to transfer batches of data between applications; often using FTP (e.g. a file might contain all the updates that have occurred to a collection of records over the last 24 hours). File Transfer is an effective pattern for sending data that is not time-critical (i.e. doesn’t need to be known to the data recipient in near ‘real-time’) and/or for sending a large volume of data.

The trigger for the file transfer could be the receipt of a message by the integration platform, or it could be initiated according to a predefined schedule, or via a file listener.

Disadvantages of this pattern are that data is not delivered in real-time, transformation rules are often embedded in the source application, resulting in tight coupling, although using ETL tools to extract information can alleviate this. Unless a file transfer management system is implemented, file management problems are common (e.g. missing or late files going undetected until too late, or duplicate files processed).

The HSIA Secure Information Transfer Service uses Secure FTP (SFTP) for external file transfers. Although not generally recommended, this pattern may be used when:

- Transferring very large files (e.g. gigabytes, or hundreds of megabytes).
- Introducing the preferred functional integration patterns adds unacceptable degree of complexity and cost.
- Transferring batch information to or from external organisations that only support this pattern.
- A low cost solution is required for periodic transfer of information that is not time critical.
8.3 Maintain Data Copies

With this pattern multiple copies of a database are maintained with different applications accessing each copy. The databases may be images of each other, or there may be some transformation in the copy process. To keep the databases synchronised there is a periodic refresh, either near real-time or in batch. Synchronisation may be one-way (population) or two-way (replication).

At any point in time, there is a possibility of the databases being out of sync due to the delays in propagating or replicating information from one database to the other.

8.4 Population

This pattern is used when information from one data store is used to populate another. Data is propagated without requiring changes to applications. The pattern may be initiated by a trigger, or be scheduled periodically.

Near real-time propagation where updates are published when they occur (Event-based population) is described in the functional integration Publish / Subscribe pattern. Generally near real-time propagation is preferred, but whether to use batch population or near real-time population depends on the latency tolerance of the applications in question. Batch population usually involves fairly large volumes of information, and is an effective approach when time delays are not a concern.

This pattern may be used for:

- Population of Data Warehouse and Data Marts. Data flows from operational to analytical stores will be ETL based and will occur as scheduled batch extracts (as opposed to ad-hoc). In most cases, the data flows will make intermediate stops at staging databases so that the appropriate data cleansing and filtering controls can be applied prior to loading the data into the analytical systems.
- Propagation of large volumes of data from one system to another.
- Periodic refresh of local cache copies of reference data or directory data from an authoritative source of truth.

8.5 Replication

The Replication pattern provides two-way synchronisation between databases in a distributed database...
system. The main difference between this pattern and the population pattern is that population is one-way and replication is two-way. Database changes made at one location are stored and forwarded to remote locations and applied to the remote copies of the database. Database replication technology is typically specific to the DBMS being used and promotes vendor lock-in.

In an application integration context, this pattern is generally not used, although it has valid uses outside the application integration space, particularly in maintaining database copies for load balancing, disaster recovery and similar solutions.

Two-way synchronisation is sometimes used to synchronise directory information when an authoritative source has delegated data maintenance to another information system. This type of two-way synchronisation may be achieved by specialised directory integration tools, or the integration may be simply a pair of one-way propagation patterns rather than true replication.

Two-way synchronisation is sometimes used to synchronise directory information when an authoritative source has delegated data maintenance to another information system. This type of two-way synchronisation may be achieved by specialised directory integration tools, or the integration may be simply a pair of one-way propagation patterns rather than true replication.

![Replication Diagram](image)

**Figure 25 – Replication**

### 8.6 Federation

The Federation pattern is a specialisation of the Broker pattern that enables multiple data stores to be accessed as though they were a single data store. A central metadata store is used to determine where the required data is stored and in what format it is stored. When a query is received, it is decomposed into a number of individual requests to the respective data stores. From the requesting application’s point of view, it is requesting data from a single source. Within the integration layer, adapters are used to access each target repository. The integration layer may store intermediate results and have logic to compensate for any missing data. Once all sources have been accessed, the data is aggregated and normalised and a single result is returned to the calling application.

Federation can support both structured and unstructured data, and can provide read only or update capability. This pattern should generally be used in read only mode, or for updates to a single data store. Update is complex when more than one data source is involved, and may require two-phase commit or compensating transactions to reverse partially completed transactions in the event of failure.

This pattern can be implemented using EAI technology (e.g. JCAPS), or can use a specialised data federation technology. An example application of this pattern is a record locator service to access clinical information from multiple sources.
8.7 Operational Data Store

An operational data store is a database used to store transaction level data to support operational query and reporting needs. It may be used as a source of data to feed data warehouses or data marts. This pattern uses the integration platform to enable the aggregation of data from a number of different source systems into the operational data store (ODS).

Operational Data Stores may integrate data or aggregate data. Integrating data from multiple data sources transforms the data structure and/or content to conform to a common data model and standardised terminology. Aggregating data from multiple data sources does not transform the data structures or content to conform to a common data model and standardised terminology.

Consumers applications can access the data in the ODS in near real time (e.g. using web services) to support operational processes. The consuming applications only access the operational data store and need have no knowledge of the source application where data originated. Operational Data Stores should not be used directly to support Business Intelligence or other secondary analytical purposes.

An example of this pattern is the Clinical Data Repository (CDR) which is an operational data store that aggregates operational data from multiple sources including EDIS, ORMIS, AUSLAB and QRIS. Different consumer applications such as Electronic Discharge Summary (EDS) and The Viewer can access this data using web service interfaces.
9. Review

This Procedure is due for review on: 05 June 2016

Date of Last Review: N/A

Supersedes: New

10. Business Area Contact

Director, Strategy and Architecture Office, Planning, Engagement and Performance Directorate, Health Services Information Agency (HSIA)

11. Definitions of terms used in this procedure

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition / Explanation / Details</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction</td>
<td>In the context of this paper, abstraction refers to an integration approach where existing system interface is made available externally to the application via an abstraction layer using the integration platform.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Application Integration</td>
<td>Integration refers to the set of technologies and services that enable the exchange of information between systems and applications across the enterprise.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Business Logic</td>
<td>The logic within an application that deals with the implementation of business policies, rules, practices and procedures for relevant business functions or processes.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Business Process Execution Language (BPEL)</td>
<td>BPEL is a standard executable language for specifying actions within business processes. Processes in BPEL export and import information by using web service interfaces.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Data Integration</td>
<td>Used when applications need to share data from files or databases, as opposed to integration via the business logic layer. This style of integration is widely applied in the Business Intelligence domain for population of Data Warehouses and Data Marts. There are also still many applications where File Transfer is</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Term</td>
<td>Definition / Explanation / Details</td>
<td>Source</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Term</td>
<td>Definition / Explanation / Details</td>
<td>Source</td>
</tr>
<tr>
<td>Data Integration</td>
<td>used to move large volumes around. Data Integration is widely supported by vendors, non-invasive to the application, and can handle large volumes of data.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>Encapsulation is an integration approach that enables existing system functionality to be exposed in a way where that functionality is ONLY able to be consumed as an integration service through the integration platform – applications cannot bypass the integration layer to access the functionality directly.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Enterprise</td>
<td>The Department of Health and the Hospital and Health Services (HHSs) make up the public healthcare system known as Queensland Health. The use of the word enterprise within this document indicates the public healthcare system as it is influenced by enterprise architecture.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Enterprise Application Integration (EAI)</td>
<td>EAI is the sharing of information amongst applications or data sources connected to a central integration framework that facilitates exchange of data and messages. EAI technologies generally support a range of legacy integration capabilities.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Enterprise Service Bus (ESB)</td>
<td>ESB is an application infrastructure component that enables integration using common business services that are made available across the organisation to consuming applications or business processes.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Enterprise Services</td>
<td>Enterprise Services provide core business functionality that is common to multiple applications (e.g. Clinical Data Service, Provider Directory Service etc.)</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>External Gateway</td>
<td>A software component that is used to control exchange of information (messages, documents, services) across the organisation boundary. It is used to apply appropriate security controls and transform between external and internal formats. It can be thought of as an extension to the ESB for external integration.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>External Rules Engine</td>
<td>A software engine used to record, track, manage and revise system business processes, without having to modify the software application itself. The Rules Engine ‘externalises’ the rules for quick and easy modification</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Extract, transform and load (ETL)</td>
<td>ETL is a process in database usage and especially in data warehousing that involves: • Extracting data from data sources • Transforming it to fit operational needs Loading it into the end target</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Functional Integration</td>
<td>The integration of the functional flow of processing between applications. Used to integrate applications via the business logic layer by allowing the business functionality in one application to be accessed by other applications. Functional Integration can be done through a number of integration technologies such as Enterprise Service Bus (ESB) and Enterprise Application Integration (EAI).</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Integration Platform</td>
<td>The central integration capability currently the JCAPS / e*Gate platform managed by the HSIA Systems Integration work unit. In future this will be the new ESB/EAI platform.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Point to Point</td>
<td>Point to Point Integration is where one application shares information with another directly, through having knowledge of the location of the other system and the system specific aspects of its</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Term</td>
<td>Definition / Explanation / Details</td>
<td>Source</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Presentation Integration</td>
<td>Includes older technologies such as screen scraping, web page links and newer technologies such as Portal integration and Presentation Integration Servers.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>Service Oriented Architecture</td>
<td>Service Oriented Architecture (SOA) is a design approach that enables application functionality to be delivered as single software services and reused within those applications that would otherwise redundantly provision that functionality. The functionality is typically deployed as a collection of interoperable services that can be used within multiple separate systems from one or more business domains. Services are abstracted away from the implementation of any consuming application using standards-based interfaces. SOA is independent of specific technologies.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
<tr>
<td>W3C</td>
<td>The World Wide Web Consortium (W3C) is an international community that develops web standards.</td>
<td>Department of Health Strategy and Architecture Office</td>
</tr>
</tbody>
</table>

12. Approval and Implementation

Policy Custodian:
Senior Director Strategy Governance and Architecture, Planning, Engagement and Performance Directorate, HSIA

Responsible Executive Team Member:
Chief Information Officer, HSIA

Approving Officer:
Ray Brown, Chief Information Officer, HSIA

Approval date: 05 June 2014
Effective from: 05 June 2014

Version Control

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Prepared by</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>09/05/2014</td>
<td>ICT Policy</td>
<td>Finalised for approval</td>
</tr>
</tbody>
</table>
Appendix A  Message Exchange Patterns

These patterns should only be used as building blocks within the functional integration patterns. They should not be used individually as integration solutions, as on their own they are simply point to point patterns.

One-Way

| Pattern Status: | Support |

One-Way is a form of message communication where a request is sent from a sender to a receiver. When a sender has made the request it continues processing without waiting for a response.

This pattern is used where information needs to be passed, but a reply or acknowledgment from the receiver is not required. Where it is important to record that a message was sent, a persistent message should be used (e.g. for logging or error reporting). A non-persistent message may be used where tracking messages is unimportant (but tracking usually is important).

This pattern corresponds to the following W3C patterns:

- In Only
- In Only Robust
- Out Only
- Out Only Robust.

The out patterns are simply the reverse of the in patterns. The Robust patterns are identical to the basic patterns except that the Robust variation can trigger a fault.

![Figure 28 – One Way](image-url)
One-Way with Acknowledgment

One-Way with Acknowledgment is a form of message communication where the sender sends a message to the receiver and waits for a reply message that acknowledges receipt of the request. There is no information returned to the sender other than the acknowledgment that message was received. The request message is normally written to persistent storage.

This pattern is used when:

- A receipt of message is required
- Guaranteed delivery of message is required without waiting for the receiving system to process the message
- Requesting system starts a transaction but has no need to wait for the result before continuing.

One-Way with Acknowledgment is the same as One Way, with the addition of an acknowledgement message from receiver to sender. This pattern corresponds to the following W3C patterns:

- In Only
- In Only Robust
- Out Only
- Out Only Robust.

The out patterns are simply the reverse of the in patterns. The Robust patterns are identical to the basic patterns except that the Robust variation can trigger a fault.

![One Way With Acknowledgement Diagram](image)

*Figure 29 – One-Way with Acknowledgement*
Request / Response

Request/Response is a form of synchronous communication where a sender sends a message to the receiver and waits for a response before continuing. This pattern is used where the sender needs some information from the receiver to enable further processing. This is a common pattern for real-time enquiry and update transactions.

Use this pattern where an immediate response is required or immediate confirmation of success or failure is required. This pattern is not suitable where delays in receiving a response have a detrimental impact on the performance of the client application performing the request.

This pattern corresponds to the following W3C patterns:
- In-out
- Out-in
- In-optional-out
- Out-optional In.

The out patterns are simply the reverse of the in patterns. The ‘optional’ variations are the same as the basic In-out, Out-in except that the subsequent message is optional.

Request / Response - No Blocking

Request / Response - No Blocking is a form of asynchronous communication where a sender sends a message to the receiver and does not wait for a reply before continuing. This pattern is essentially two ‘One-way with Acknowledgment’ message pairs. To ensure guaranteed delivery, persistent messages are used.

This pattern is used when:
- Sending a message to a receiver using the Mediated Request / Response pattern, where a delayed response is expected (the delay may be fractions of a second to hours or even days, depending on the situation)
- Sending messages to multiple targets using the ‘Broker’ pattern
- Initiating steps in a process using the ‘Managed Process’ pattern.
Figure 31 – Request / Response no Blocking

Node A sends a message to Node B which responds with an acknowledgement. Node A then processes the message and sends a response. If a fault occurs, it replaces the output message.