

Eskom iCoE

End Systems Integration Design Requirements

Author:	Zingisani M. Mpande
Creation Date:	December 04, 2018
Last Updated:	August 29, 2022
Document Ref:	N/A
Version:	Draft 0.1

Approvals:

1

1 Document Control

1.1 Change Record

Date	Author	Version	Change Reference

1.2 Reviewers

Internal

Name	Position

External

Name	Position

1.3 References

ID	Description	Reference
1		

1.4 Open issues

ID	Issue	Resolution	Responsibility	Target Date	Impact Date

Table of contents

1	Document Control.....	2
1.1	Change Record.....	2
1.2	Reviewers.....	2
1.3	References	2
1.4	Open issues.....	2
2	Background	7
3	Introduction.....	8
3.1	Purpose	8
3.2	Audience.....	8
4	Service-Oriented Architecture.....	9
4.1	Integration Service	10
4.2	Principles	10
4.3	Guidelines.....	10
4.4	Service Versioning.....	11
4.4.1	Major Release	11
4.4.2	Minor Release	11
4.4.3	Point Release	12
4.4.4	Principles	12
5	Security	14
5.1	Security Principles	14
5.2	Security Requirements.....	15
5.2.1	Transport-Level Security	15
5.2.2	Authentication.....	15
5.2.3	Authorization.....	16
5.2.4	Confidentiality	16
5.2.5	Integrity.....	16
5.2.6	Message Level Security.....	16
5.2.7	Public Key Infrastructure.....	17
5.2.8	Auditing	17

5.2.9	Security Policies	17
5.3	Web Services Security	17
5.4	Database Security	18
5.5	Managed File Transfer Security	18
6	Eskom Implementation Standards	19
6.1	Security Implementation	19
7	Design Patterns	22
7.1	Synchronous Request-Reply Pattern	22
7.2	Asynchronous Fire-and-Forget Pattern	24
7.3	Guaranteed Delivery Pattern.....	25
7.4	File Transfer Pattern	27
7.5	Message transformation	Error! Bookmark not defined.
7.5.1	Message size.....	Error! Bookmark not defined.
7.5.2	Rule of thumb	Error! Bookmark not defined.
7.6	Protocol Standards	Error! Bookmark not defined.
7.6.1	Supported Protocols	Error! Bookmark not defined.
7.6.2	Web Service Providers	Error! Bookmark not defined.
8	Canonical Message Design	28
8.1	Canonical Messages = EBM	28
8.2	Canonical Object Definitions = EBO	28
8.2.1	EBM Architecture.....	29
8.2.2	EBM Headers	30
8.3	Message Exchange Patterns (MEP)	31
8.3.1	Synchronous request/response message exchange pattern.....	31
8.3.2	One way message exchange pattern.....	32
8.3.3	Asynchronous Request/Response message exchange pattern	33
8.3.4	Asynchronous pub/sub message exchange pattern.....	35
9	Error handling.....	36
9.1	EIP Faults Classification	36
9.2	Business Faults versus Technical Faults	Error! Bookmark not defined.

9.3 Technical Fault Handling 37

9.4 Business Fault Handling 37

Table of figures

Figure 1: Service Consumers.....	9
Figure 2: Service Versions	12
Figure 3: EIP Security Overview	14
Figure 4: Synchronous Request-Reply Pattern Technology	22
Figure 5: Fire-and-Forget Pattern Using Queuing Technology	24
Figure 6: Ensuring Guaranteed Delivery Using Milestones	25
Figure 7: Synchronous request/reply message exchange pattern	31
Figure 8: One way message exchange pattern	32
Figure 9: Asynchronous Request/Response message exchange pattern.....	33
Figure 10: JMS based Asynchronous Request/Response.....	34
Figure 11: SOAP based Asynchronous Request/Response.....	34
Figure 12: Asynchronous Pub/Sub message exchange pattern	35

2 Background

Eskom enterprise applications integrate with various vendor proprietary applications through the use of the Eskom Enterprise Integration Platform (EIP) that is managed and implemented by the Eskom Integration Centre of Excellence (iCoE). iCoE has established Services-Oriented Integration (SOI) Reference Architecture which forms part of Eskom's Service Oriented Architecture (SOA) framework. This framework defines integration standards that are to be adhered to by both enterprise and external applications wishing to integrate to any enterprise applications through the use of the EIP.

3 Introduction

This document defines integration design requirement for any application that wishes to integrate with the Eskom enterprise applications through the EIP. SOI is a variant or flavour of SOA focusing on Enterprise Integration. Wherever SOA is mentioned in the document it will be focusing on the Integration context specifically.

3.1 Purpose

The objective of this document is to define the design and implementation standards for integrating to the Eskom EIP.

3.2 Audience

This document is intended for application architects, software engineers and developers. This audience must understand the relationship between integration points between Eskom EIP and application integration layers and will use this document to guide their integration architecture and design decisions.

4 Enterprise Integration Platform (EIP) Architecture

The intent of the Eskom EIP is to provide common reusable services that can be leveraged by a variety of consumers. These services will typically originate from functionality and data that already exists in the enterprise. Services are created by “service-enabling” legacy assets. As new projects are implemented, standalone services may also emerge as autonomous entities that do not have dependencies on legacy systems.

Eskom EIP uses the SOA architecture to create standardized, reusable, integration enabling services in an incremental manner.

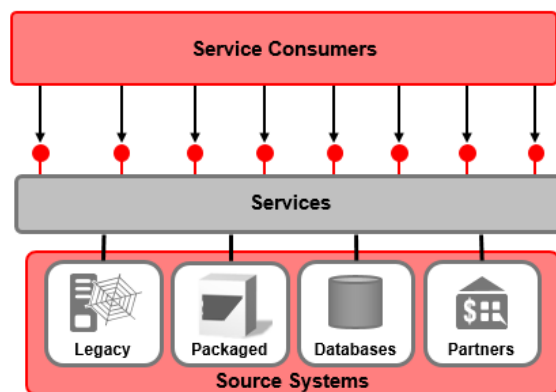


Figure 1: Service Consumers

As shown in the diagram above, services provide an intermediary between consumers and source systems (Location transparency, Platform independence and Implementation encapsulation). Services provide business-enabling capabilities (Enterprise data, High level of abstraction). Source systems provide process, functionality and data.

The information below is applicable to both EIP and End System (Application) Integration layers.

4.1 Integration Service

A key element is a common vocabulary and thorough understanding of what is meant by a “Service”. Integration Services, for the purpose of this document and that of the Eskom reference architecture document, refer to reusable code that is deployed in an IT environment. It does not include activities performed by humans, or products for sale to customers. A logical definition of a service consists of three components: A Contract, its Interface and the Implementation.

Eskom EIP exposes a set of Integration Services through which applications can integrate. In order for an application to communicate with other applications within the Enterprise it must expose a set of service providers and/or must have the ability to consume services from the EIP in a manner that is consistent with Eskom EIP standards.

4.2 Principles

- All services must have contracts.
- All contracts will be based on the approved and agreed templates.
- Any deviation from the approved service templates needs to be approved.
- Usage agreements are required for consumers

4.3 Guidelines

- The quality of services must be realistic and should adhere to the definition of “what is real-time” for Eskom.
- A SOAP interface defined by WSDL should suffice for most invocation patterns. For services exposed to Web applications, mobile applications etc. where small messages and high performance are requirements REST/JSON services may be required.
- Where performance is critical, J2EE open standard compliance should be applied.

4.4 Service Versioning

Services, like any other code assets, are bound to change and evolve over time. Changes can be accommodated in a variety of ways; however, following the loose coupling principles of SOA, the impact to consumers should be minimized. For this reason changes should be handled as versions, ideally supporting the ability to have multiple versions running concurrently.

There will be times when this is not feasible, such as when an underlying database change is made. But care should be taken to minimize these instances. Failure to do so requires consumers to deploy updates in lock-step with service updates, which can quickly become unwieldy as usage increases.

The following versioning objectives should be met:

4.4.1 Major Release

These types of changes are not backwards compatible and will therefore require Service consumers to be modified before being able to utilize the new version of the Service.

If Services tend to go through a large number of major revisions, it is an indication that there are areas for improvement, particularly around the area of Service identification and discovery.

The deployment of a major release of a Service typically requires the previous version to be co-deployed. The previous version is considered to be deprecated in order to allow Service consumers to be upgraded to the newest version of the Service.

A major change to a Service can:

- Allow deprecated operations to be dropped,
- Allow a change in an operation signature in a manner that could break consumer compatibility,
- Replace the input and output message type with a different input and output message type,
- Be used if the Service has gone through a major rewrite or has been updated considerably.

4.4.2 Minor Release

These types of changes are backwards compatible, and do not require dependent Service consumers to be updated in order to utilize the new Service version. The deployment of a minor release to a Service results in the immediate retirement of the previous version of the Service.

A minor change to a Service version can:

- Allow operations, data types, optional fields to be added (but not deleted),
- Must not break any consumer compatibility.

4.4.3 Point Release

A point release of a Service represents the least significant types of changes for a service. Changes such as defect fixes, where no new functionality is added to the service, are classified as point releases. Like the minor release, the point release is backwards compatible and upon deployment the previous version of the service is immediately retired.

A point change to a Service version:

- Can be used when fixing specific problem (e.g. bug, performance),
- Does not allow new features to be added.



Figure 2: Service Versions

4.4.4 Principles

- The architecture should support several versions of a service in production concurrently
- Governance should limit the number of concurrent versions to 2.
- “New” consumers should only discover the latest version
- Consumers should not be forced to upgrade immediately
- Consumers must migrate eventually and a timeframe of 3 months given for retiring a deprecated version.
- Services will be retired gracefully once all consumers have migrated to the latest version.
- Services should strive for backward compatibility
- Point releases should never break backward compatibility

- Service providers must be able to release new versions into production without waiting for consumers to certify on them.
- Consumers should be able to test and certify on new service versions before switching to the new version in production.
- Service providers should base their testing on the contract (black box testing by contract). Consumers may certify for their use based on their own functional test criteria.
- Service consumers should not require code modifications in order to access the latest compatible Service versions (point releases).
- Consumers should be able to determine which version they access.
- Bug fix point release should not affect the interface or contract.
- Minor release (e.g., adding a new method), or major release will require a change to all three components.
- Contract change might affect the interface and/or implementation.

5

5 Security

Eskom has taken the architectural decision to secure all communication (inbound and outbound) between the EIP and integrating partner applications. At a minimum transport level security (TLS) is required and user authentication against the corporate authentication store or embedded LDAP (for system users).

This comprises the bare minimum SOA Security requirements acceptable to Eskom. The Eskom SOI reference architecture addresses both these minimum requirements and the evolving future state security requirements of Eskom.

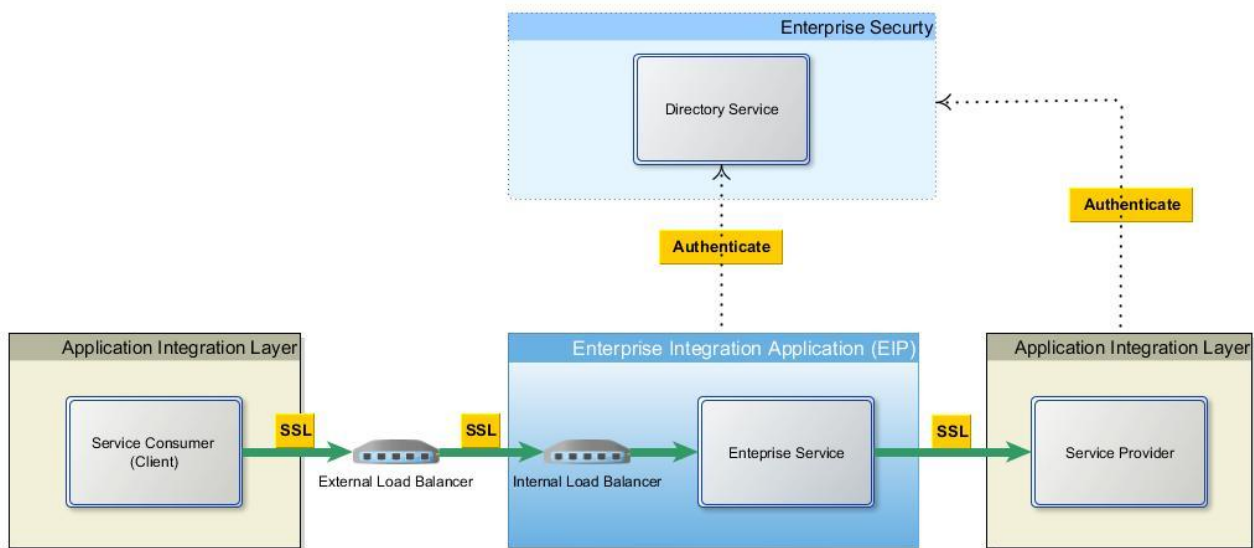


Figure 3: EIP Security Overview

Security within the RA is not meant to be an exhaustive list of all security considerations, but rather a more detailed view of specific security concerns related to the SOI. Specific type of security to be implemented is guided by the Eskom Security.

5.1 Security Principles

There are numerous Web Services security standards available, the most relevant being WS-Security. WS-Security builds on other standards (XML-Encryption...) to provide authentication, authorization, integrity and confidentiality.

Therefore the following principles will apply:

- Use appropriate WS-* standards where possible
- Clear separation between business logic security logic
- Use WS-Policies to configure security policies declaratively
- Manage/change security policies independently from the secured service (policies can be added/changed at design, deployment and run time).
- Where no applicable policies exist (for example when non WS interfaces apply) the separation of concerns between business logic and security logic will apply.

5.2 Security Requirements

The following sub-sections highlight the security requirements for integrating with Eskom EIP. Not all are required for all use cases.

5.2.1 Transport-Level Security

Secure Socket Layer (SSL), otherwise known as Transport Layer Security (TLS), the Internet Engineering Task Force (IETF) officially standardized version of SSL, is the most widely used transport-level data-communication protocol providing:

- Authentication (the communication is established between two trusted parties).
- Confidentiality (the data exchanged is encrypted).
- Message integrity (the data is checked for possible corruption).
- Secure key exchange between client and server.

SSL provides a secure communication channel, however, when the data is not “in transit,” the data is not protected, which makes the environment vulnerable to attacks in multi-step transactions (SSL provides point-to-point security, as opposed to end-to-end security).

5.2.2 Authentication

Authentication is the process of verifying that a user or resource consumer is who he/she claims to be. This is generally accomplished by providing an identifier along with a password, token, or signature that is unique to the user and trusted to be secret.

Authenticating users is a common capability of any business solution. The way in which a user is authenticated varies based on the implementation of security architecture and the degree of stringency one places on proving identity.

5.2.3 Authorization

Authorization is the process of granting or denying requests by a party (e.g. client or user) to perform actions on a resource. It is generally performed by comparing rights granted to the user with actions the user is attempting to perform. Access decisions can be divided into three topics including coarse-grained access control, fine-grained entitlements, and data redaction.

5.2.4 Confidentiality

Confidentiality in this context refers to the ability to protect data from being seen by entities it is not intended for, both while stored and in transit. Sensitive data housed within a completely secure, locked down, computing environment may be considered safe enough without the need for extra security measures. However, most data eventually must travel beyond the boundaries of locked down environments and must be protected through some means. Encryption technologies can be leveraged to solve these issues.

5.2.5 Integrity

Integrity, as it is addressed in this document, pertains to the ability to ensure:

- That requests were issued by the sender, not an imposter
- That messages from the sender have not been altered en route
- These concerns can be addressed through the use of digital signatures and Public Key Infrastructure (PKI).

5.2.6 Message Level Security

Due to the distributed nature of services in Eskom environment, the concept of service chaining (services invoking other services), and the introduction of mediation or proxy points between providers and consumers necessitates the need for message level security.

Message level security protects messages from end to end, from the origination point of a service request, through intermediaries, through service providers, and services they might call. It protects messages while in transit and at rest (in queues and databases). This makes it an ideal solution for message protection, going a step beyond transport level, or point to point security.

5.2.7 Public Key Infrastructure

In many cases, web services security relies on Public Key Infrastructure (PKI) environments. A PKI uses cryptographic keys (mathematical functions used to encrypt or decrypt data). Keys can be private or public. In an asymmetric cipher model, the receiving party's public key is used to encrypt plaintext, and the receiving party's matching private key is used to decrypt the ciphertext. Also, a private key is used for encryption with digital signatures and the public key is used for verifying digital signatures. Public-key certificates (or certificates, for short) are used to guarantee the integrity of public keys.

5.2.8 Auditing

End Systems are required to keep an audit trail of the messages sent and received from EIP. These audit trails must be readily accessible and usable. Audit trail data may include request message, response message, date & time message sent/received, message ID, correlation ID, etc. This information is necessary for message tracking, trouble shooting and dispute resolutions.

5.2.9 Security Policies

In the context of this document, a security policy is a definition of what it means for a computing environment (system, application, service, etc.) to be secure. The general definition of policy must be recorded in a way that security architects can read and understand so they can apply it to IT assets. Quite often it involves the configuration of many endpoints, gateways, firewalls, applications, and infrastructure in order to achieve a state of security described by the policy. As such, security policy specification and enforcement mostly involves manual efforts to articulate policy and enact methods for enforcement.

The advent of Web Services, SOAP, and WS-Security specifications brought forth an aspect of security policy that has achieved a level of standardization and automation. It addresses the declaration of message-based security that is applied to Web Services. Web Service security policies identify the security requirements of a service and provide a mechanism to enforce security policies so that requests must either meet the policy or be denied.

5.3 Web Services Security

Web Service Security may be applied to both SOAP Web Services as well as REST services. Security and management policies can be applied to protect multiple types of web services: Java Platform, Enterprise Edition (Java EE) Java API for XML Web Services (JAX-WS), etc.

5.4 Database Security

The EIP may be required to connect to databases using JDBC JCA adapters. Whatever security measures required by the database will be complied with by EIP.

5.5 Managed File Transfer Security

Managed File Transfer (MFT) has its own security that leverages and extends the security mechanisms of file transfer mechanisms such as FTP(S) and SFTP to provide Role based secure access to files. Encryption/decryption of files uses PGP.

6 Eskom Implementation Standards

The implementation standards elaborated on below are generally applicable to all services built for the Eskom EIP.

6.1 Security Implementation

Eskom requires all inbound and outbound communication to the EIP to be secured with SSL encryption and authenticated access control.

The following policies are the bare minimum security requirements:

Component	Security Requirement	Security Implementation
Web Service Providers	https encryption with http basic authentication or web service certificate authentication	OWSM policy (oracle/wss_http_token_service_policy) Alternatively <ul style="list-style-type: none">• Support for http basic authentication• Support for certificate authentication
External file transfer	Secure, encrypted and authenticated file transfer.	MFT security
External JDBC	JDBC authentication and encryption	Data source security
External JCA (JMS, MQ etc.).	Authentication and encryption	JCA security

6.2 Message transformation

6.2.1 Message size

The message size and transformation performance the platform is capable of handling depends on processor and RAM allocation. There is therefore no clear criteria valid for all environments.

6.2.2 Rule of thumb

Appropriate size: 100s of KBs

Upper Limit: 10 MB

Design Criteria: For message sizes greater than 5 MB evaluate alternatives.

6.3 Protocol Standards

Eskom has standardised on WSDL based SOAP as the primary protocol for the integration architecture with protocol translation to or from SOAP taking place at the periphery using transport/adaptor services.

6.3.1 Supported Protocols

The following protocols and transport mechanisms are supported.

1. HTTP(S) – SOAP (default), XML/JSON Rest Services, XML, custom message formats
2. JDBC connections
3. JMS – Text Messages (SOAP, Object, XML, Custom formats), Map messages, Bytes messages, Object messages, Stream messages.
4. File based transports (SFTP, FTPs)
5. Java APIs

6.3.2 Web Service Providers

EIP exposes two types of web services:

6.3.2.1 Enterprise Service

This is a preferred and standard highly reusable web service provider generically built to accept canonical message format from the integrating application client. Eskom uses industry standards CIM messages including the below:

Model
Enterprise Resource Planning (Human Resource)
Environmental (CIM for Weather)
Geographic Information
IEC61850
IEC61968
IEC61970
IEC62325

6.3.2.2 Mediator Service:

Web Service Provider custom built to accept proprietary message format from the integrating application client. This type of service is responsible for mediating the EIP on behalf of the Requester application. It does this by transforming messages between the Requester specific format and the canonical message format of the EIP. In order to do this the Mediator service may be required to execute the following tasks:

- Validate the request message
- Enrich the request message
- Transform to canonical EBM
- Operate – execute the EBS operation

7 Design Patterns

The following patterns are derived from the Application Integration Architecture (AIA). Below are the preferred EIP design patterns. End Systems (Source & Target) must design and implement their service providers and consumers in accordance with the recommendations defined.

- Synchronous Request-Reply Pattern
- Asynchronous Fire-and-Forget Pattern
- Guaranteed Delivery Pattern

7.1 Synchronous Request-Reply Pattern

Consumers send requests synchronously to service providers and get immediate responses to each of their requests. The consumers need to wait until the responses are received before proceeding to their next tasks.

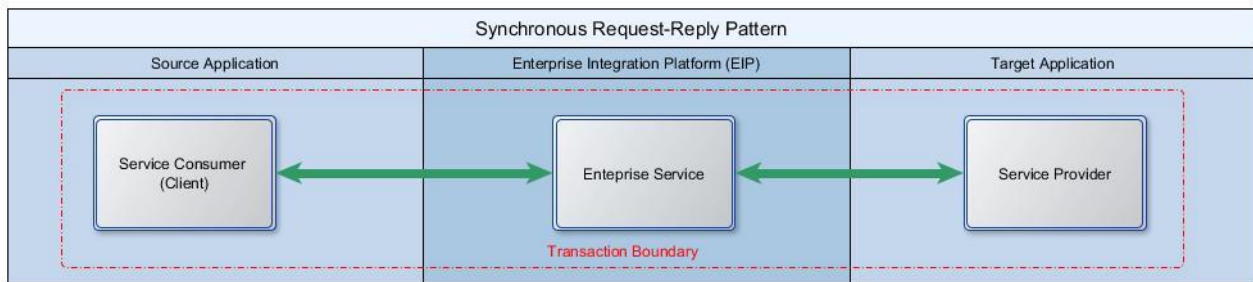


Figure 4: Synchronous Request-Reply Pattern Technology

Implementation should strive to ensure that no persistence of state information (dehydration) or audit details is done by any of the intermediary services or by the ultimate service provider. These techniques help keep the latency as low as possible.

Impact & Consideration

- Service Consumer must be able to interpret, handle and manage response messages from EIP.
- Service Provider must reply with a full service response message to EIP
- All the resources are locked in until the response from service provider goes back to the originating system or user (single Transaction Boundary).
- Either a transaction timeout or an increased latency may result if any of the services or the participating application takes more time for processing. Read and connection timeout settings are to be configured at all times.

- Service providers must always be available and ready to fulfil the service requests.
- Service providers doing real-time retrieval and collation of data from multiple back-end systems in order to generate a response message could put an enormous toll on the overall resources and increase the latency. The synchronous request-response design pattern should not be used to implement tasks that involve real-time complex processing.

7.2 Asynchronous Fire-and-Forget Pattern

The requester application should be able to continue its processing after submitting a request regardless of whether the service providers needed to perform the tasks are immediately available or not. Besides that, the user initiating the request does not have a functional need to wait until the request is fully processed.

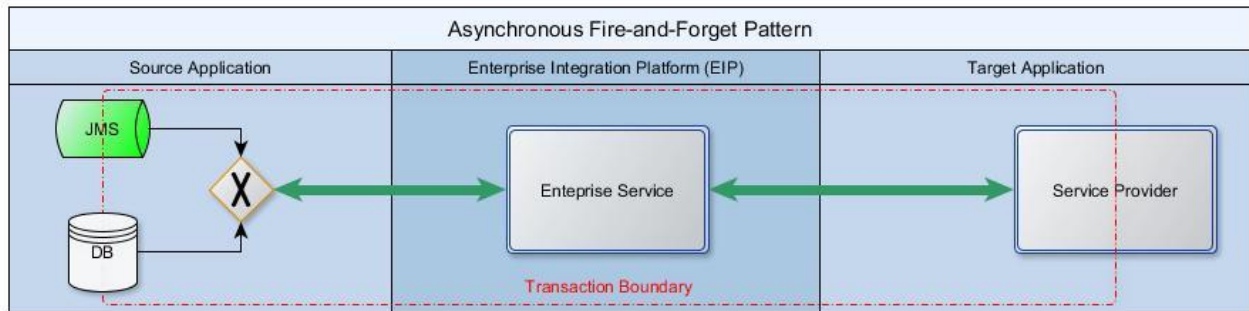


Figure 5: Fire-and-Forget Pattern Using Queuing Technology

Eskom recommends the fire-and-forget pattern using queuing or database technology to decouple the participating application from the integration layer. The queue/database acts as a staging area allowing the requester applications to place the request messages. The request messages are subsequently forwarded to service providers on behalf of requester as and when the service providers are ready to process them. It is highly recommended that the persistence of the request message into the queue/database is within the same transaction initiated by the requester application to perform its work.

Impact & Consideration

- The default implementation does not have inherent support for notifying the requester application of success or the failure of messages.
- Refer to the Error Handling section on the recommendations on how to handle responses
- If a database is used, the database table must be built with sufficient data to allow for the tracking of the message and reporting or errors. The technical columns required may include Date the record is inserted; message ID, flag indicating processing status of the record, etc.

7.3 Guaranteed Delivery Pattern

The integrating applications use a built-in local data store to persist messages. When the Requester sends a message, the send operation does not complete successfully until the message is safely stored in the sender's data store. Subsequently, the message is not deleted from one data store until it is successfully forwarded to and stored in the next data store. In this way, once the sender successfully sends the message, it is always stored on disk on at least one computer until it is successfully delivered to and acknowledged by the receiver.

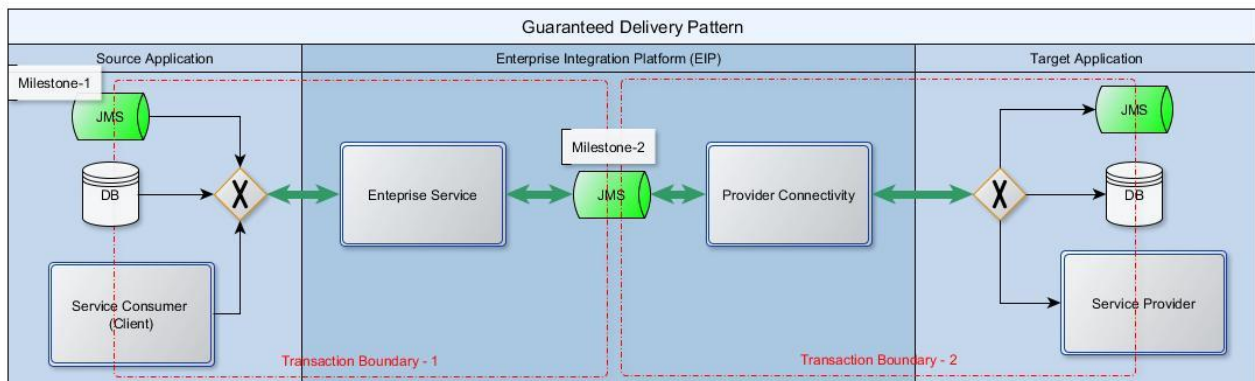


Figure 6: Ensuring Guaranteed Delivery Using Milestones

A “**milestone**” is a message checkpoint where the message is preserved after completing a certain amount of business functionality at this logical point or is submitted to the participating application for further processing of that message in an end-to-end composite business process. A milestone helps to commit the business transactions done to that logical point and also releases the resources used by services and the participating application.

Milestones are to be introduced at appropriate logical points in an end-to-end integration. EIP uses JMS Destinations (Queues or Topics) to implement milestones. However, the end system may use a persistent store of choice to implement milestone.

AIA recommends that the source application push the request message to Milestone -1 (JMS Queue) in a transactional mode to ensure the guaranteed delivery of the message. Once the message is stored in Milestone-1, the source application resources can be released because AIA ensures the guaranteed delivery of that message to the target application.

The request messages are subsequently forwarded to service providers or Milestone-2 on behalf of requester as and when the service providers are ready to process them.

The messages would either be transferred to Milestone-2 or rolled back to Milestone-1 for corrective action or resubmission.

Once the messages are transferred to Milestone-2, all the Milestone-1 implementation service instances are released to accommodate new requests.

The provider implementation service picks up the message and ensures it is delivered to the target application.

The target application must have capability to ensure guaranteed delivery. The implementation service should be designed for manual compensations or semi-automatic or automatic compensations based on the target system's capability to acknowledge the business message delivered from EIP

7.4 File Transfer Pattern

Eskom EIP can receive files from external partners through FTP(S) and SFTP file transfer mechanisms, transform the data to target application formats and distribute the files to the target applications.

8 Canonical Message Design

The uses of canonical message formats are a key ingredient of a loosely coupled Integration Architecture. Within the context of SOI a service layer exposing business functions in a platform neutral (canonical) language is required to provide a flexible architecture able to adapt to changes in the underlying EIS systems.

AIA canonical messages are called Enterprise Business Messages (EBM) which is application independent message formats. The EBM comprises a Header and a data element known as an EBO. The EBO is the canonical format defining the data load and structure of the business entity.

Eskom utilises and extends the Common Information Mode (CIM), a standard developed by the electric power industry and officially adopted and maintained by the International Electrotechnical Commission (IEC). The CIM is currently maintained as a UML model from which design artefacts such as XML schemas (XSDS) may be derived. The Eskom specific extensions to the CIM are known as the CIM@E. The sections below seek to align and combine the AIA and CIM@E practices.

8.1 Canonical Messages = EBM

At the most basic levels, EBMs are the messages exchanged between two applications. The EBM represents the specific content of an EBO needed for performing a specific activity. For example, an invoice might be used in three contexts: add, cancel, and update. The context for processing the invoice might require most elements present in the EBO. However, the context for cancelling the invoice might require only the specific invoice instance to be cancelled.

8.2 Canonical Object Definitions = EBO

An EBO is a standard business data object composed of reusable data components. The library of all EBOs yields a domain model or an EBO data model. The EBO represents a layer of abstraction on top of the logical data model. Developers, business users, and system integrators use EBOs. In the integrations developed using AIA, the EBO data model serves as a common data abstraction across systems. It supports the loose coupling of systems in AIA and eliminates the need for one-to-one mappings of the disparate data schemas between each set of systems. Each application data schema is mapped to the EBO data model only once.

EBOs have the following characteristics:

- They contain components that satisfy the requirements of business objects from the source and target application data models.
- They are not data repositories.
- Instead, they provide the structure for exchanging data. XML provides the vocabulary for expressing business data. The XML schema is an XSD file that contains the application-independent data structure to describe the common object.
- They are represented in an XML schema (XSD) file format.

An EBO represents a business concept, such as a customer, a sales order, a payment, and so forth. Each EBO has a primary business component that identifies the object, and optionally multiple supplementary components required to complete the definition of the EBO. Sales Order Header and Purchase Order Header are examples of primary business components; Sales Order Line and Sales Order Schedule are examples of supplementary business components.

Eskom uses the CIM@E model to derive XML schema representations of these canonical business concepts.

8.2.1 EBM Architecture

Every EBM has the same message architecture. An EBM encompasses details about the action to be performed using the data, one or more instances (EBOs) of the same type, and the EBM header. Each service request and response is represented in an EBM using a distinct combination of an action and an instance. For example, a single Query Customer Party EBM business document sends the request to a billing system for retrieving account details for one or several customer accounts.

The EBM cannot process details about multiple types of action. For example, you cannot have *Query* and *Update* actions in the same message.

When using EBMs, consider the following points:

- If an application were allowed to invoke the Enterprise Business Services Layer (EBSs), the application itself would have to generate the EBM to pass the EBM as a payload to the EBS.
- The action in the EBM identifies the action that the sender or the requester application wants the receiver or provider application to perform on the EBM.
- The action also stores additional information that must be carried out on the EBO instance. For example, the Create action may carry information about whether it wants the target application to send a confirmation message. The Query action may carry information about

the document header section of the original EBM that resulted in the performance of this action.

- The business object portion of the data area element contains the business object data element definitions that can or must be sent in the message.
- This is the content that is carried from one point to another. The element reflects the action-specific view of the EBO.
- An EBM can be defined to carry multiple instances. Only the actions that support bulk processing use EBMs that support multiple instances.
- The information present in an EBM header is common to all EBMs.
- The information present in the data area and the action are very specific to a particular EBM.
- The message architecture is detached from the underlying transport protocol.

Any transport protocol such as HTTP, HTTPS, SMTP, SOAP, and JMS should be able to carry these documents.

Eskom message designs have typically combined the EBM Action and the EBM header concepts into a single structure known as the StandardCombinedHeader.

8.2.2 EBM Headers

The EBM header is an integral part of every EBM. The EBM header is like a wrapper or envelope around transactional data messages. It comprises functional data representations. For example, Involved Parties might include Sender, Provider, intermediary services, Security, and Transaction Rules (Transaction State and Exceptions).

The EBM header provides the ability to:

- Carry information that associates the message with the originator.
- Uniquely identify the message for auditing, logging, security, and error handling.
- Associate the message with the specific instance of the sender system that resulted in the origination of the document.
- Store environment-specific or system-specific information.

Infrastructure-related service requirements such as auditing, logging, error handling, and security necessitate additional attributes in the EBM message header section.

8.3 Message Exchange Patterns (MEP)

The following message exchange patterns are supported by the architecture:

- Synchronous request/response
- One way
- Asynchronous request/response
- Asynchronous pub/sub

8.3.1 Synchronous request/response message exchange pattern

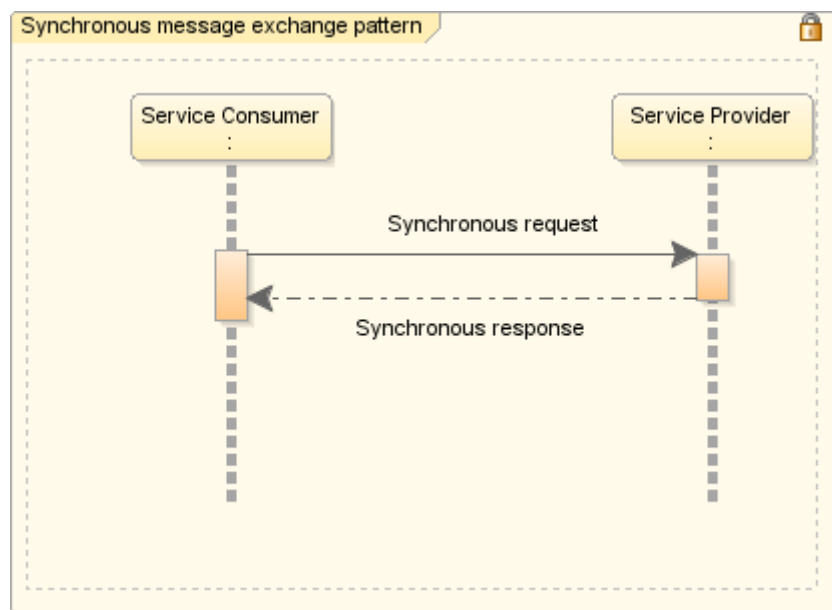


Figure 7: Synchronous request/reply message exchange pattern

The service consumer initiates the interaction by submitting the request message. The communication channel with the service provider is kept open until the service provider returns the response message. The service consumer thread is blocked until the response is received.

This message exchange pattern is used when the consumer requires a response before it can proceed and the expected latency is low.

This message exchange pattern is supported by the http(s) and Java RMI transports.

8.3.2 One way message exchange pattern

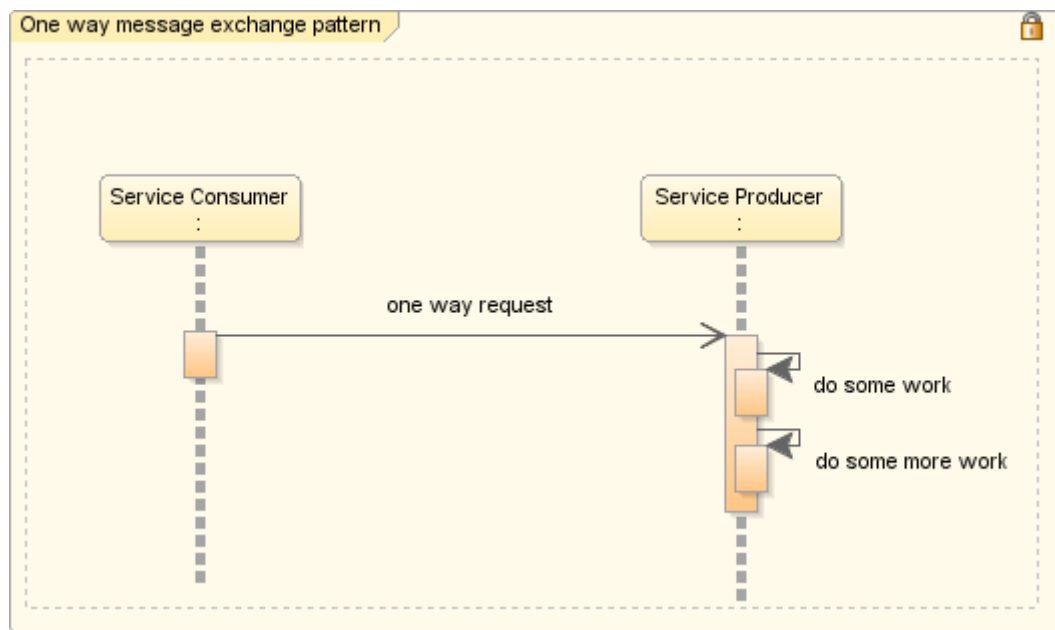


Figure 8: One way message exchange pattern

The service consumer sends a request message to the service producer and immediately continues with its next task without waiting for a response.

This message exchange pattern is used when the service consumer wants to initiate an operation on the service producer and does not require anything in response.

This message exchange pattern is supported by the http(s), Java RMI and JMS transports. For SOAP web services the WSDL defines an operation with a request message, no response message and no fault messages.

8.3.3 Asynchronous Request/Response message exchange pattern

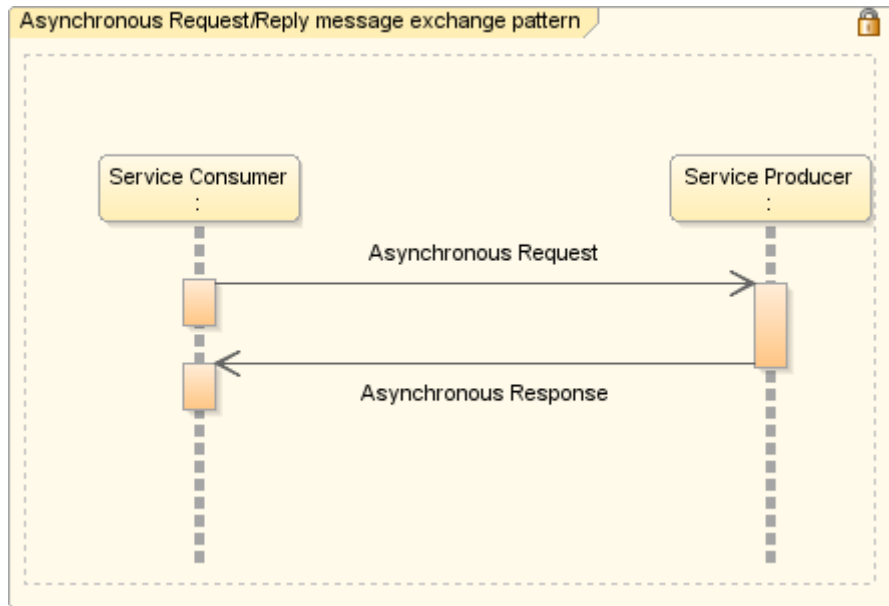


Figure 9: Asynchronous Request/Response message exchange pattern

The service consumer initiates the interaction by submitting the request message. The communication channel with the service provider is closed and the consumer may proceed with other work without blocking. When the service producer has completed its processing and is able to provide a response it makes an asynchronous call back to the consumer in order to deliver the response message.

This message exchange pattern is used when the consumer requires a response to the request but the expected latency is too high to wait for the response synchronously.

This message exchange pattern is supported by the SOAP/http(s) and JMS transports.

There are two mechanisms dependent on the transport used. For the JMS transport two queues are used, one for the request and one for the response. The consumer publishes to the request queue and listens on the response queue for the response. The request and response are correlated by a JMS Message ID header variable on the request message and a JMS Correlation ID header variable on the response message.

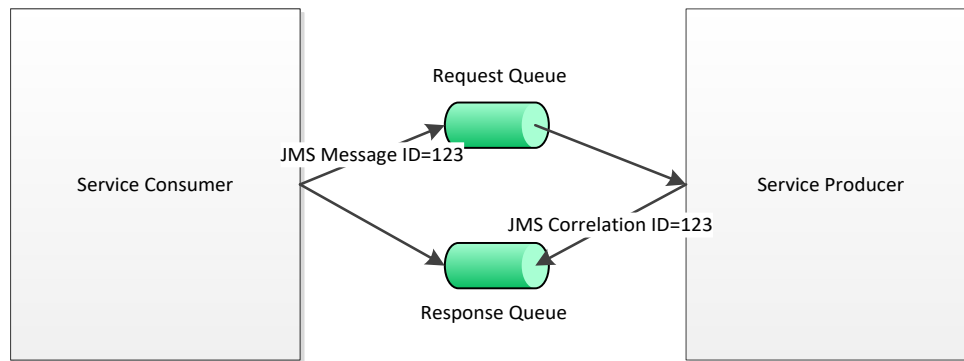


Figure 10: JMS based Asynchronous Request/Response

For SOAP web services the WSDL defines two port types, one for the requests and one for the responses. Each port type defines one way operations for the request and responses. The response port type is exposed by the consumer (as a web service) to receive the response messages. The request message SOAP header contains a WS-Addressing section specifying the ReplyTo address and the MessageID. The ReplyTo address is the service URL of the service consumer and the MessageID identifies the request. This enables the consumer to receive the responses pertinent to its requests and to match request and response by MessageID.

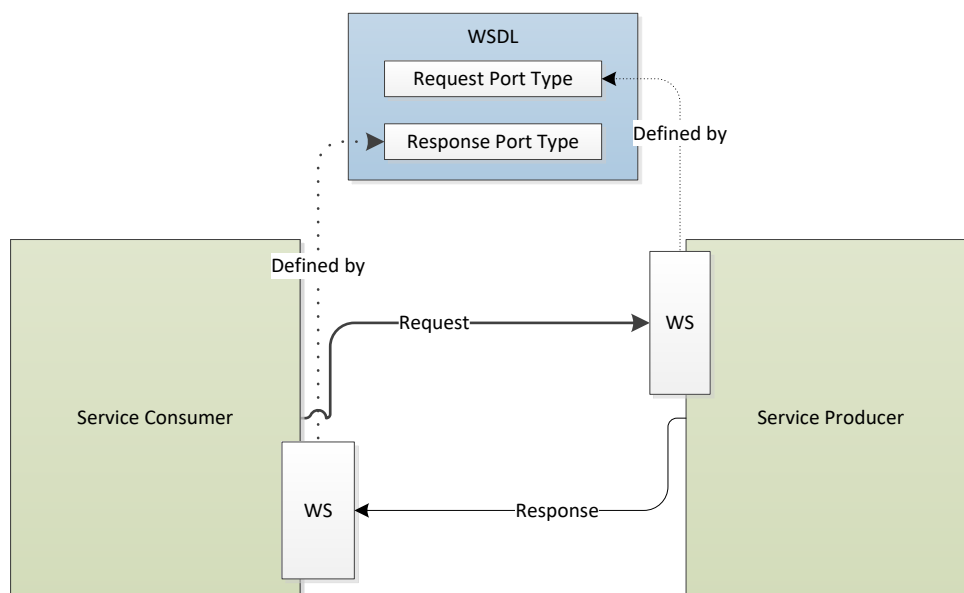


Figure 11: SOAP based Asynchronous Request/Response

8.3.4 Asynchronous pub/sub message exchange pattern

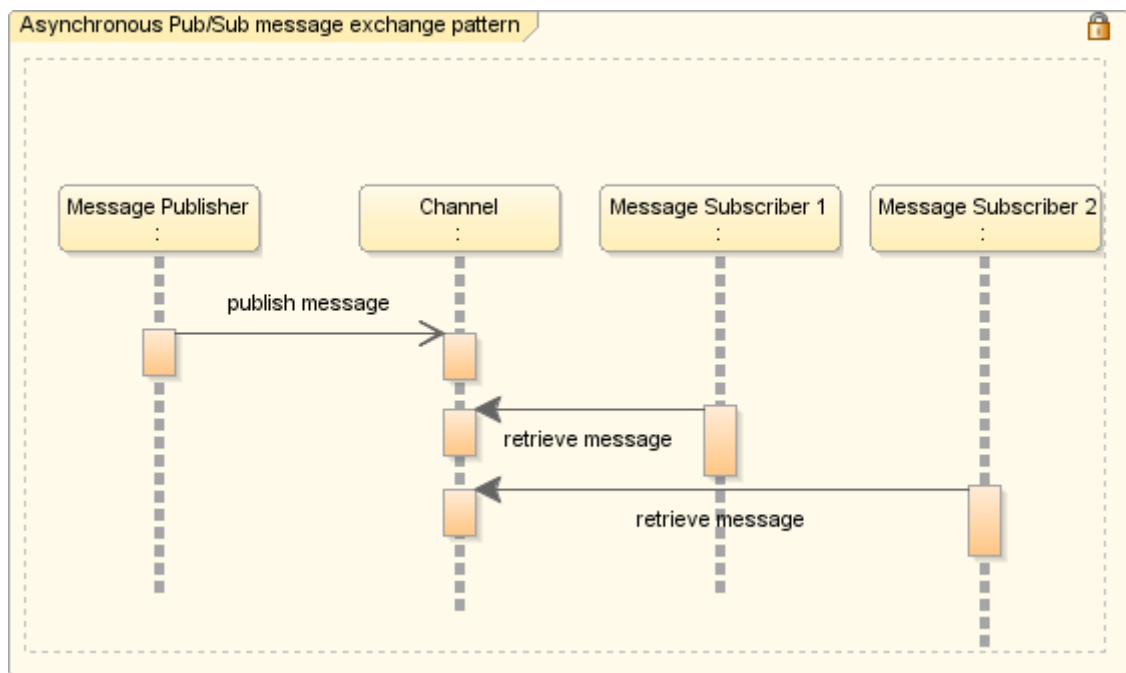


Figure 12: Asynchronous Pub/Sub message exchange pattern

The message publisher initiates the interaction by publishing the message to a delivery channel. The delivery channel holds the message until each registered message subscriber retrieves a copy of the message and then discards the message.

This message exchange pattern is supported by JMS Topic transports.

9 Error handling

Faults in IT-systems can be categorized into the following types:

- **Technical errors** - Faults caused by errors in the underlying infrastructure or middleware components on which applications run. Examples are network errors, server failures, corrupt disks, full tablespaces, and so on.
- **Software errors** - Faults caused by programming errors in custom-made applications, faults in 3rd party software libraries that are used, software faults and bugs in packaged applications, etc. Think of division by zero, infinite loops, memory leaks, null pointer exceptions, and so on.
- **Faulty operation by users** - Faults caused by human errors when using IT-systems. Examples of such faults are entering a wrong credit card number, accidentally switching the “to” and “from” date when booking flight tickets, ordering the wrong eBook in a web shop, and so on.
- **Exceptional business behaviour** - Faults caused by a failure to meet a certain business rule. For example, a customer with a bad credit rating, a new customer that wants to purchase something but is unknown in the CRM system, an invoice with an incorrect invoice amount.

9.1 EIP Faults Classification

EIP categorises errors into two types, those that can be automatically re-tried and those that are not re-triable:

Fault Type	Fault Grouping	Fault Handling
Business Error	<ul style="list-style-type: none">• Software errors• Faulty operation by users• Exceptional business behaviour	Errors that cannot be automatically retried/resubmitted errors. Requires user intervention and manual resubmission.
Technical Error (Transient Error)	<ul style="list-style-type: none">• Technical errors	Errors that the system can recover from without the need to change message content or structure.

9.2 Technical Fault Handling

If a message cannot be delivered to the target due to transient/technical errors encountered while processing the request message,

- the message is rolled back to the JMS/persistent store or
- a service response/Fault message is returned to the client

The message will eventually be delivered after a number of retries; but if the number of retry/rollback cycles required to redeliver the message becomes large, system resources can be negatively affected. To avoid this situation, a ***Delay and Redirect*** redelivery option is to be enabled.

After configurable number of failed redelivery attempts, the message is to be redirected to a recovery state (a dead-letter queue).

Recommended values

Attribute	Value	Description
Number of Retries	3	The message must be re-submitted 3 times before it is put in a recovery state. Careful consideration should be taken for messages that are to be processed in an orderly/sequential fashion to preserve the order of the messages.
Delay Frequency	15 minutes	The number of retries is counted from the original rollback and the delay time between the retries is 15 minutes.

9.3 Business Fault Handling

Business errors are classified as errors that the system cannot automatically recover from. End System must expose to EIP service points that will accept business error messages. These messages will then be made available to the relevant support people where the fault can be manually corrected and later resubmitted.

The service points may accept data such as, date & time the error was encountered, message that caused the error, type of error raised, Short description of the error, detailed description of the error, message ID, correlation ID, etc.