

BS EN 61968-100:2013



BSI Standards Publication

Application integration at electric utilities — System interfaces for distribution management

Part 100: Implementation profiles

bsi.

...making excellence a habit.
AVA Documents™
Document Sharing Hub

National foreword

This British Standard is the UK implementation of EN 61968-100:2013. It is identical to IEC 61968-100:2013.

The UK participation in its preparation was entrusted to Technical Committee PEL/57, Power systems management and associated information exchange.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2013.
Published by BSI Standards Limited 2013

ISBN 978 0 580 77557 4
ICS 33.200

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2013.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

EUROPEAN STANDARD
NORME EUROPÉENNE
EUROPÄISCHE NORM

EN 61968-100

November 2013

ICS 33.200

English version

**Application integration at electric utilities -
System interfaces for distribution management -
Part 100: Implementation profiles
(IEC 61968-100:2013)**

Intégration d'applications pour les services
électriques - Interfaces système pour la
gestion de distribution -
Partie 100: Profils de mise en œuvre
(CEI 61968-100:2013)

Integration von Anwendungen in Anlagen
der Elektrizitätsversorgung -
Systemschnittstellen für Netzführung -
Teil 100: Implementations-Profile
(IEC 61968-100:2013)

This European Standard was approved by CENELEC on 2013-08-30. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Avenue Marnix 17, B - 1000 Brussels

© 2013 CENELEC - All rights of exploitation in any form and by any means reserved worldwide for CENELEC members.

Ref. No. EN 61968-100:2013 E

 **Documents**
Document Sharing Hub

Foreword

The text of document 57/1358/FDIS, future edition 1 of IEC 61968-100, prepared by IEC/TC 57, "Power systems management and associated information exchange" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 61968-100:2013.

The following dates are fixed:

- latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement (dop) 2014-05-30
- latest date by which the national standards conflicting with the document have to be withdrawn (dow) 2016-08-30

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 61968-100:2013 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC 61968-9	NOTE	Harmonised as EN 61968-9.
IEC 61968-13	NOTE	Harmonised as EN 61968-13.
IEC 61970-452	NOTE	Harmonised as EN 61970-452.
IEC 61970-453	NOTE	Harmonised as EN 61970-453.
IEC 62361-100	NOTE	Harmonised as EN 62361-100.

Annex ZA (normative)

Normative references to international publications with their corresponding European publications

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

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

<u>Publication</u>	<u>Year</u>	<u>Title</u>	<u>EN/HD</u>	<u>Year</u>
IEC 60050-300	-	International Electrotechnical Vocabulary - Electrical and electronic measurements and measuring instruments - Part 311: General terms relating to measurements - Part 312: General terms relating to electrical measurements - Part 313: Types of electrical measuring instruments - Part 314: Specific terms according to the type of instrument	-	-
IEC 61968-1	-	Application integration at electric utilities - System interfaces for distribution management - Part 1: Interface architecture and general requirements	EN 61968-1	-
IEC/TS 61968-2	-	Application integration at electric utilities - System interfaces for distribution management - Part 2: Glossary	-	-
IEC 61968-11	-	Application integration at electric utilities - System interfaces for distribution management - Part 11: Common information model (CIM) extensions for distribution	EN 61968-11	-
IEC 61970-301	-	Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base	EN 61970-301 ¹⁾	-
IEC 61970-552	-	Energy Management System Application Program Interface (EMS-API) - Part 552: CIMXML Model Exchange Format	EN 61970-552 ¹⁾	-
ISO 8601	-	Data elements and interchange formats - Information interchange - Representation of dates and times	-	-

¹⁾ At draft stage.

CONTENTS

INTRODUCTION.....	8
1 Scope.....	9
2 Normative References	10
3 Terms, definitions and abbreviations	10
3.1 Terms and definitions	10
3.2 Abbreviations	10
3.3 Terminology for common integration technologies	11
3.3.1 General	11
3.3.2 Enterprise Service Bus (ESB).....	12
3.3.3 Java Messaging Service (JMS).....	12
3.3.4 Service-Oriented Architecture (SOA)	12
3.3.5 Event-Driven Architecture (EDA)	12
3.3.6 Simple Object Access Protocol (SOAP)	12
3.3.7 Web Services (WS)	13
3.3.8 Web Services Definition Language (WSDL)	13
3.3.9 XML Schema (XSD).....	13
3.3.10 Representational State Transfer (REST).....	14
3.3.11 Queue	14
3.3.12 Topic	14
3.3.13 Message Destination	14
3.3.14 Request.....	14
3.3.15 Response	14
3.3.16 Query	15
3.3.17 Transaction	15
3.3.18 Event.....	15
4 Use Cases.....	15
4.1 General	15
4.2 Simple request/reply.....	16
4.3 Request/reply using an ESB.....	16
4.4 Events.....	17
4.5 Transactions	18
4.6 Callback	19
4.7 Adapters.....	20
4.8 Complex messaging	21
4.9 Orchestration	22
4.10 Application-level use cases	22
5 Integration Patterns	23
5.1 General	23
5.2 Client and server perspectives	23
5.2.1 General	23
5.2.2 Basic web service pattern	24
5.2.3 Basic JMS request/reply pattern	24
5.2.4 Event listeners.....	26
5.2.5 Asynchronous request/reply pattern.....	27
5.3 Bus perspective.....	27

5.3.1	General	27
5.3.2	ESB messaging pattern using JMS	28
5.3.3	ESB messaging patterns using web service request	29
5.3.4	ESB request handling to web service	29
5.3.5	ESB request handling via adapter	30
5.3.6	Custom integration patterns	31
6	Message organization	32
6.1	General	32
6.2	IEC 61968 messages	32
6.2.1	General	32
6.2.2	Verbs	33
6.2.3	Nouns	34
6.2.4	Payloads	35
6.3	Common message envelope	36
6.3.1	General	36
6.3.2	Message header structure	37
6.3.3	Request message structures	40
6.3.4	Response Message Structures	43
6.3.5	Event message structures	48
6.3.6	Fault message structures	49
6.4	Payload structures	50
6.5	Strongly-typed payloads	53
6.6	SOAP message envelope	54
6.7	Request processing	55
6.8	Event processing	56
6.9	Message correlation	57
6.10	Complex transaction processing using OperationSet	57
6.10.1	General	57
6.10.2	OperationSet Element	59
6.10.3	Patterns	61
6.10.4	OperationSet example	63
6.11	Representation of time	65
6.12	Other conventions and best practices	65
6.13	Technical interoperability	65
6.14	Service level agreements	66
6.15	Auditing, monitoring and management	66
7	Payload specifications	66
8	Interface specifications	70
8.1	General	70
8.2	Application-level specifications	70
8.3	Web service interfaces	72
8.3.1	General	72
8.3.2	WSDL Structure	72
8.3.3	Document style SOAP binding	73
8.3.4	Strongly-typed web services	74
8.4	JMS	76
8.4.1	General	76
8.4.2	Topic and queue naming	77
8.4.3	JMS message fields	78

9	Security	78
10	Version control	79
	Annex A (normative) XML schema for common message envelope	81
	Annex B (normative) Verbs	91
	Annex C (normative) Procedure for strongly typed WSDL generation	93
	Annex D (normative) Generic WSDL	106
	Annex E (informative) AMQP	108
	Annex F (informative) Payload Compression Example	109
	Annex G (informative) XMPP	111
	Bibliography	112
	Figure 1 – Overview of Scope	9
	Figure 2 – Simple Request/Reply	16
	Figure 3 – Request/reply using intermediaries	17
	Figure 4 – Events	18
	Figure 5 – Point-to-Point (One Way) Pattern	19
	Figure 6 – Transaction Example	19
	Figure 7 – Callbacks	20
	Figure 8 – Use of Adapters	21
	Figure 9 – Complex messaging	22
	Figure 10 – Application-level use case example	23
	Figure 11 – Basic request/reply using web services	24
	Figure 12 – Basic request/reply using JMS	25
	Figure 13 – Event listeners using JMS	26
	Figure 14 – Asynchronous request/reply pattern	27
	Figure 15 – ESB content-based routing	28
	Figure 16 – ESB with smart proxy and content-based routing	29
	Figure 17 – ESB with proxies, routers and adapters	30
	Figure 18 – ESB Integration to non-compliant resources	31
	Figure 19 – Messaging between clients, servers and an ESB	33
	Figure 20 – Example payload schema	35
	Figure 21 – Common message envelope	37
	Figure 22 – Common message header structure	39
	Figure 23 – Request message structure	41
	Figure 24 – XML for example RequestMessage	42
	Figure 25 – Example 'Get<Noun>' profile	43
	Figure 26 – ResponseMessage structure	44
	Figure 27 – Reply message states	45
	Figure 28 – Error structure	46
	Figure 29 – XML for example ResponseMessage	47
	Figure 30 – XML example of payload compression	47
	Figure 31 – XML example for error ResponseMessage	48
	Figure 32 – EventMessage structure	48

Figure 33 – XML example for EventMessage	49
Figure 34 – Fault message structure	50
Figure 35 – Message payload container – Generic.....	51
Figure 36 – Message payload container – Type specific example	54
Figure 37 – SOAP bindings	54
Figure 38 – SOAP envelope example for strong typing	55
Figure 39 – Message OperationSet Element	58
Figure 40 – OperationSet details.....	60
Figure 41 – Transactional Request/Response (non-OperationSet)	61
Figure 42 – Published events (non-OperationSet)	62
Figure 43 – Transactional Request/Response (OperationSet)	62
Figure 44 – Published event (OperationSet).....	63
Figure 45 – Information Models, Profiles and Messages	67
Figure 46 – Contextual Profile Design in CIMTool	67
Figure 47 – Example message payload schema.....	68
Figure 48 – Example payload XML schema.....	69
Figure 49 – Example message XML	70
Figure 50 – Example complex business process	72
Figure 51 – WSDL structure	73
Figure 52 – Web service usage example	76
Figure 53 – Example Organization of Topics and Queues	77
Figure C.1 – Process for WSDL Generation	93
Figure C.2 –Example sequence diagram	94
Figure C.3 – WSDL folder structure	94
Figure C.4 – WSDL type definitions	95
Figure D.1 – Generic WSDL structure	106
Table 1 – Verbs and their Usage.....	34
Table 2 – Payload usages.....	53
Table B.1 – Normative definitions of verbs	91

INTRODUCTION

This part of IEC 61968 defines a set of implementation profiles for IEC 61968 using technologies commonly used for enterprise integration. More specifically, this document describes how message payloads defined by parts 3-9 of IEC 61968 are conveyed using web services and the Java Messaging System. Guidance is also provided with respect to the use of Enterprise service Bus (ESB) technologies. The goal is to provide details that would be sufficient to enable implementations of IEC 61968 to be interoperable. In addition, this document is intended to describe integration patterns and methodologies that can be leveraged using current and future integration technologies.

The IEC 61968 series of standards is intended to facilitate *inter-application integration* as opposed to *intra-application integration*. Intra-application integration is aimed at programs in the same application system, usually communicating with each other using middleware that is embedded in their underlying runtime environment, and tends to be optimised for close, real-time, synchronous connections and interactive request/reply or conversation communication models. IEC 61968, by contrast, is intended to support the inter-application integration of a utility enterprise that needs to connect disparate applications that are already built or new (legacy or purchased applications), each supported by dissimilar runtime environments. Therefore, these interface standards are relevant to loosely coupled applications with more heterogeneity in languages, operating systems, protocols and management tools. This series of standards, which are intended to be implemented with middleware services that exchange messages among applications, will complement, not replace utility data warehouses, database gateways, and operational stores.

This standard is based upon the EPRI Technical Report 1018795 and other contributed works.

The IEC 61968 series, taken as a whole, defines interfaces for the major elements of an interface architecture for distribution systems within a utility enterprise. Part 1: Interface Architecture and General Recommendations, identifies and establishes requirements for standard interfaces based on an Interface Reference Model (IRM). Parts 3 through 9 of IEC 61968 define interfaces relevant to each of the major business functions described by the Interface Reference Model.

As described in IEC 61968, there are a variety of distributed application components used by the utility to manage electrical distribution networks. These capabilities include monitoring and control of equipment for power delivery, management processes to ensure system reliability, voltage management, demand-side management, outage management, work management, automated mapping, meter reading, meter control and facilities management. This set of standards is limited to the definition of interfaces and is implementation independent. It provides for interoperability among different computer systems, platforms, and programming languages. Methods and technologies used to implement functionality conforming to these interfaces are considered outside of the scope of these standards; only the interface itself is specified in these standards.

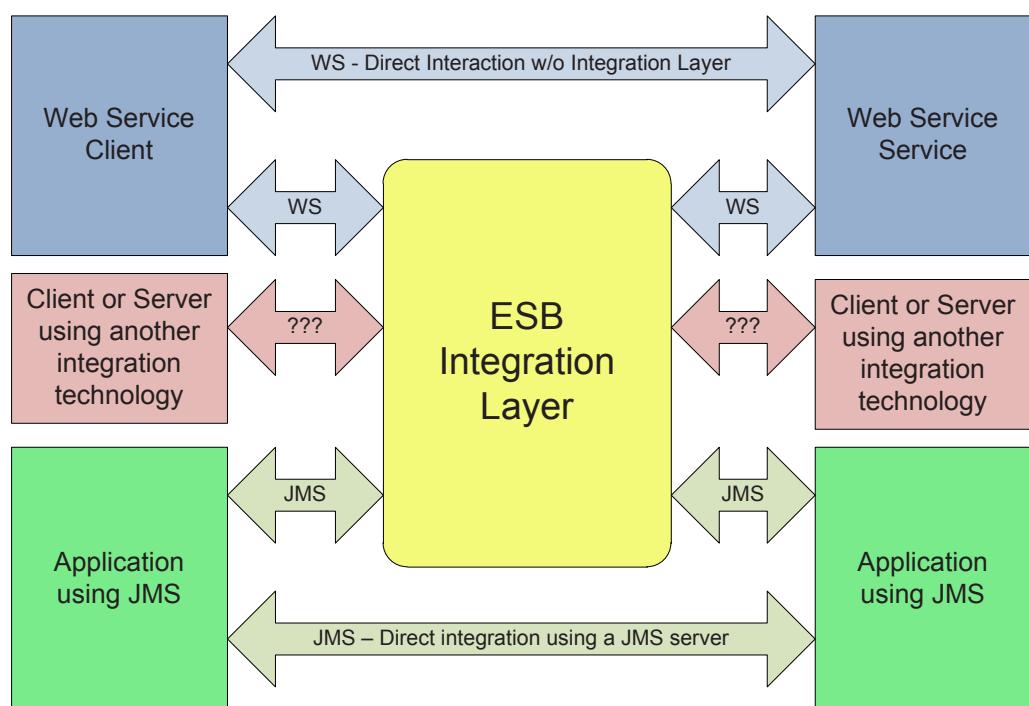
APPLICATION INTEGRATION AT ELECTRIC UTILITIES – SYSTEM INTERFACES FOR DISTRIBUTION MANAGEMENT –

Part 100: Implementation profiles

1 Scope

This part of IEC 61968 specifies an implementation profile for the application of the other parts of IEC 61968 using common integration technologies, including JMS and web services. This International Standard also provides guidance with respect to the use of Enterprise Service Bus (ESB) technologies. This provides a means to derive interoperable implementations of IEC 61968-3 to IEC 61968-9. At the same time, this International Standard can be leveraged beyond information exchanges defined by IEC 61968, such as for the integration of market systems or general enterprise integration.

Figure 1 attempts to provide an overview of scope, where IEC 61968 compliant messages are conveyed using web services or JMS. Through the use of an ESB integration layer, the initiator of an information exchange could use web services, where the receiver could use JMS, and vice versa. The integration layer also provides support for one to many information exchanges using publish/subscribe integration patterns and key functionality such as delivery guarantees.



IEC 1769/13

Figure 1 – Overview of Scope

The scope of this document specifically includes the following:

- integration patterns that support IEC 61968 information exchanges
- design of interfaces for use of strongly typed web services
- design of interfaces for use of generically typed web services
- design of interfaces using JMS

- definition of standard design artefacts and related templates
- recognition that technologies other than JMS and web services may be used for integration leveraging this standard (with some specific examples and associated recommendations described in appendices)

This profile can also be applied to integration problems outside the scope of IEC 61968.

It is important to note that other implementation profiles can potentially be defined for IEC 61968, and that this is not intended to be the only possible implementation profile. In addition, this profile can be adapted to meet specific needs of specific integration projects.

It is also not within the scope of this document to prescribe those implementation details as required for security.

2 Normative References

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

IEC 60050-300, *International Electrotechnical Vocabulary – Electrical and electronic measurements and measuring instruments – Part 311: General terms relating to measurements – Part 312: General terms relating to electrical measurements – Part 313: Types of electrical measuring instruments – Part 314: Specific terms according to the type of instrument*

IEC 61968-1, *Application integration at electric utilities – System interfaces for distribution management – Part 1: Interface architecture and general recommendations*

IEC/TS 61968-2, *Application integration at electric utilities – System interfaces for distribution management – Part 2: Glossary*

IEC 61968-11, *Application integration at electric utilities – System interfaces for distribution management – Part 11: Common information model (CIM) extensions for distribution*

IEC 61970-301, *Energy management system application program interface (EMS-API) – Part 301: Common information model (CIM) base*

IEC 61970-552, *Energy management system application program interface (EMS-API) – Part 552: CIM XML Model Exchange Format*

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this specification, the terms and definitions given in IEC 60050-300, IEC/TS 61968-2, IEC 62051, IEC 62055-31 apply.

3.2 Abbreviations

The following terms and abbreviations are used within this document:

API	Application Programming Interface
AMQP	Advanced Message Queue Protocol
CIM	Common Information Model
CME	Common Message Envelope
CRUD	Create, Read, Update, Delete
EDA	Event Driven Architecture
ESB	Enterprise Service Bus
IEC	International Electrotechnical Commission
IETF RFC	Internet Engineering Task Force Request For Comments
ISO	International Standards organization
JEE	Java Enterprise Edition
JMS	Java Message Service
JSR	Java Specification Request
mRID	CIM master resource identifier
OASIS	Organization for the Advancement of Structured Information Standards
RDF	Resource Description Framework
REST	REpresentational State Transfer
RFC	Request for Comments
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secured Socket Layer
TLS	Transport Layer Security
UML	Unified Modelling Language
URL	Uniform Resource Locators
UUID	Universal Unique Identifier
W3C	World-Wide Web Consortium
WS	Web Services
WS-*	Web Services standards
WS-I	Web Services Interoperability
WSDL	Web Services Definition Language
XML	eXtensible Markup Language
XSD	XML Schema
XSL	XML Stylesheet Language

3.3 Terminology for common integration technologies

3.3.1 General

Where there is a difference between the definitions in this standard and those contained in other referenced IEC standards, then those defined in IEC/TS 61968-2 shall take precedence over the others listed, and those defined in this document shall take precedence over those defined in IEC/TS 61968-2.

3.3.2 Enterprise Service Bus (ESB)

An Enterprise Service Bus (ESB) refers to a software architecture construct that is used as an integration layer. This construct is typically implemented by technologies found in a category of middleware infrastructure products, usually based on recognized standards, which provide foundational services for more complex architectures via an event-driven and standards-based messaging engine (the bus).

An ESB generally provides an abstraction layer on top of an implementation of an enterprise messaging system, which allows integration architects to exploit the value of messaging without writing code. Contrary to the more classical enterprise application integration (EAI) approach of a monolithic stack in a hub and spoke architecture, the foundation of an enterprise service bus is built of base functions broken up into their constituent parts, with distributed deployment where needed, working in harmony as necessary.

An ESB does not implement a service-oriented architecture (SOA) but provides the features with which one may be implemented.

3.3.3 Java Messaging Service (JMS)

The Java Message Service (JMS) API is a Java Message Oriented Middleware API for sending messages between two or more clients. JMS supports request/reply, publish/subscribe and point to point messaging patterns. JMS is a part of the Java Platform, Enterprise Edition, and is defined by a specification developed under the Java Community Process as JSR 914. It is important to note that some ESB product vendors provide language bindings for JMS using C, C++ and/or C#, making the term JMS a misnomer. Where the wire protocol is different between different JMS implementations it is often trivial to bridge between different JMS implementations.

3.3.4 Service-Oriented Architecture (SOA)

Service Oriented Architecture (SOA) is a computer systems architectural style for creating and using business processes, packaged as services, throughout their lifecycle. SOA also defines and provisions the IT infrastructure to allow different applications to exchange data and participate in business processes. These functions are loosely coupled with the operating systems and programming languages underlying the applications. SOA separates functions into distinct units (services), which can be distributed over a network and can be combined and reused to create business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services. SOA concepts are often seen as built upon and evolving from older concepts of distributed computing and modular programming.

3.3.5 Event-Driven Architecture (EDA)

Event-Driven Architecture (EDA) is a software architecture pattern that promotes the production, detection and consumption of events. An event is any change of state of potential interest. Within an EDA, events are transmitted between loosely coupled software components and services, typically using publish/subscribe messaging patterns. EDA is complementary to SOA, to the extent that SOA 2.0 is also known as 'event-driven' SOA. EDA is fundamental to a variety of business intelligence patterns, including complex event processing patterns.

3.3.6 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is a standard that defines the formatting of XML messages. SOAP serves as a foundation layer of the web services protocol stack. SOAP is also commonly used within JMS. Common transports for SOAP include HTTP, HTTPS and proprietary JMS transports. SOAP is also now sometimes referred to as 'Service-Oriented Architecture Protocol'. SOAP is a W3C Recommendation.

Two versions that are in common use include SOAP 1.1 and SOAP 1.2. New integrations or interfaces should use SOAP 1.2 when applicable.

3.3.7 Web Services (WS)

A Web Service is defined by the W3C as ‘a software system designed to support interoperable Machine to Machine interaction over a network.’ Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard. Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP *endpoint*, but it is a prerequisite for automated client-side code generation in many Java and .Net development tools.

Where web services have two primary styles, document-centric and RPC-centric, the focus of this specification is document-centric. This is supportive of SOA and the transport of IEC 61968 payloads. For increased interoperability, the document wrapped form is required.

It is important to note that web services do not readily support publish/subscribe messaging unless mechanisms such as WS-Eventing are used. However, this specification also describes an approach for the ESB to route messages asynchronously to configured subscribers.

This standard will discuss two approaches for use of web services, where WSDL operations are either generic or strongly typed. Generic web services have WSDLs and related operations that are defined in a payload type independent manner. Strongly typed web services are defined where WSDLs and operations are defined individually for specific payload types.

This also provides for automated frameworks for server-side validation of message content based on message schemas contained in the WSDL document.

3.3.8 Web Services Definition Language (WSDL)

Web Services Definition Language (WSDL) is an XML-based language that is used to describe web services. WSDL is often used in combination with SOAP and XML schema to provide web services over the internet (or an intranet). WSDL is a W3C Recommendation.

Two versions that are in common use include WSDL 1.1 and WSDL 2.0. WSDL 2.0 better supports interoperability between Java and .Net implementations.

The WSDL templates provided by this document are based upon WSDL 1.1, and consequentially WSDL 1.1 is a requirement. In order to better support interoperability between diverse implementations, all WSDL documents are WS-I Basic Profile 1.1 compliant.

3.3.9 XML Schema (XSD)

XML Schema, published as a W3C recommendation in May 2001, is one of several XML schema languages. It was the first separate schema language for XML to achieve Recommendation status by the W3C.

Like all XML schema languages, XML Schema can be used to express a schema: a set of rules to which an XML document shall conform in order to be considered 'valid' according to that schema. However, unlike most other schema languages, XML Schema was also designed with the intent that determination of a document's validity would produce a collection of

information adhering to specific data types. An XML Schema instance is an XML Schema Definition (XSD) and typically has the filename extension ".xsd". The language itself is sometimes informally referenced as XSD.

It is important to note that 61968 payload are defined using XML schemas. Those XML schemas provide normative specifications for application payloads that are conveyed using this standard.

3.3.10 Representational State Transfer (REST)

Representational State Transfer (REST) is an alternative to the use of SOAP-based web services. REST itself is not currently a standard, but instead an architectural style.

REST leverages HTTP, where each URL is a representation of some object. Interfaces can be defined in terms of XML payloads for requests and responses. A WSDL is not required for the use of REST. Additionally, the XML documents used for requests and responses do not need to be defined using an XSD, although it is common for the XML to be compliant to an XSD.

Within REST, an object can be retrieved (read) using an HTTP GET. Similarly, an HTTP POST is used to create an object, an HTTP PUT is used to modify an object and an HTTP DELETE is used to delete an object.

REST is discussed here for completeness purposes only as it may be encountered by an integration project. However, there are currently no specific recommendations for mappings by this standard. REST may be supported in the future.

3.3.11 Queue

A queue is a construct supported by many messaging products to provide reliable messaging with delivery guarantees. A standard API that includes queue-based messaging models is provided by JMS. AMQP also defines an open protocol for queue based messaging.

3.3.12 Topic

A topic is a construct supported by many messaging products to enable publish/subscribe messaging patterns where there may be potentially many consumers of a message that has been sent to a named topic by a publisher. Topics are commonly used as a destination for event messages. Topics are directly supported by JMS.

3.3.13 Message Destination

A message destination is the target address for a message, whether it be a request, response or an event message. When using JMS, the destination may be a topic or queue. When using HTTP mechanisms such as web services, the destination is specified as a URL.

3.3.14 Request

A request is a message sent from a client (or source) to a server (or target) where a response is expected. The request may be either a query (where data is returned from the target) or a transaction (where data is modified in the target). A request will use verbs such as 'get', 'create', 'change', 'delete', 'cancel', 'close' or 'execute'.

3.3.15 Response

A response is a message sent as a consequence of a request, typically from the target of the request to the source of the request. Response messages are synonymous with 'reply' messages. A response message will use a 'reply' verb.

3.3.16 Query

A query is a type of request where the target is expected to return information to the source of the request. The request message for a query will use the 'get' verb. This will typically be implemented using a request/reply pattern.

3.3.17 Transaction

A transaction is a type of request where the target typically will modify information that it manages. A transaction request will use verbs such as 'create', 'change', 'delete', 'cancel', 'close' or 'execute'. This may be implemented using a variety of patterns. If a pattern other than request/reply is used, there should be a delivery guarantee provided by the transport.

3.3.18 Event

The term 'Event' is significantly overloaded, and can have different meanings in different contexts. The most general definition of an event is 'something that happens at a given place and time', or 'a change of state of potential interest'. As a consequence of an event, 'something' may generate an 'event message' to report the fact that a certain type of event occurred at a given time. Event messages are therefore asynchronous in nature and are typically published to potentially interested subscribers using topic-based messaging.

Event messages are one type of asynchronous message. It may be common for an application to generate an event when a condition of interest is detected, or a transaction has been processed. There are several integration patterns that are related to the support of events.

Events are typically published asynchronously as event messages by a system or application in order to report conditions of interest. In some cases a system or application may internally identify a condition of interest, but in other cases the condition may be detected by an external source such as a device. Event messages will use past tense verb such as 'created', 'changed', 'deleted', 'closed', 'canceled' or 'executed'.

4 Use Cases

4.1 General

The purpose of Clause 4 is to describe several use cases related to the interactions between components within a set of systems cooperating to support a set of business processes. It is important to note that the use cases presented are from the perspective of the integration of systems, as opposed to end use application-level use cases. The actors for the use cases described in this clause include the following:

- Client
- Server
- ESB
- Adapter
- Subscriber (an Event Listener)

Three key terms related to messaging are request, reply and event. Within the terminology of IEC 61968, these are reflected in terms of the verbs used to define specific information flows.

Central to the use cases is the assumption that a variety of integration technologies may be used, where the focus of this standard is JMS and web services.

4.2 Simple request/reply

The first use case is a simple request/reply between a client and server. This is synonymous with request/response. The initiator of the request is the client, where the requested is processed by the server. The first view of this is the simple view without the use of an ESB. This use case involves one of two cases:

- A client making a query request to a server, where the server will return a set of objects to the client based upon some filter criteria
- A client making a transaction request to a server, where a set of objects will be created or modified in some way

Both cases are illustrated in Figure 2.

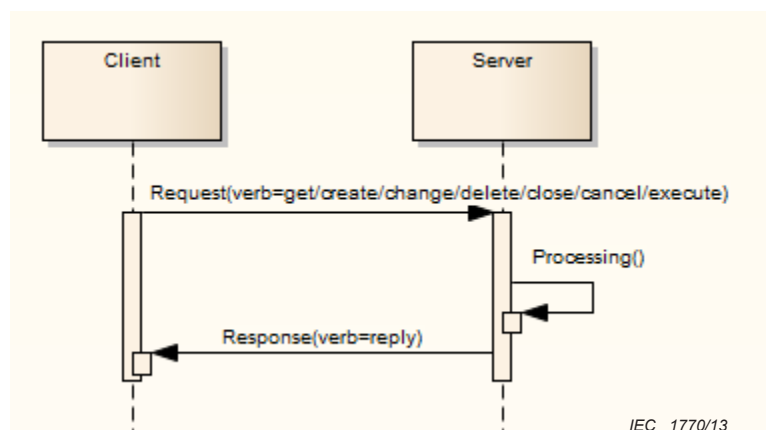


Figure 2 – Simple Request/Reply

In terms of IEC 61968, requests use verbs such as "get", "create", "change", "delete", "close", "cancel" or "execute".

4.3 Request/reply using an ESB

The simple request/reply use case can also be extended to leverage an ESB. Within the ESB many actions can be taken by intermediaries as needed to facilitate integration and the decoupling of components, such as transformations and routing.

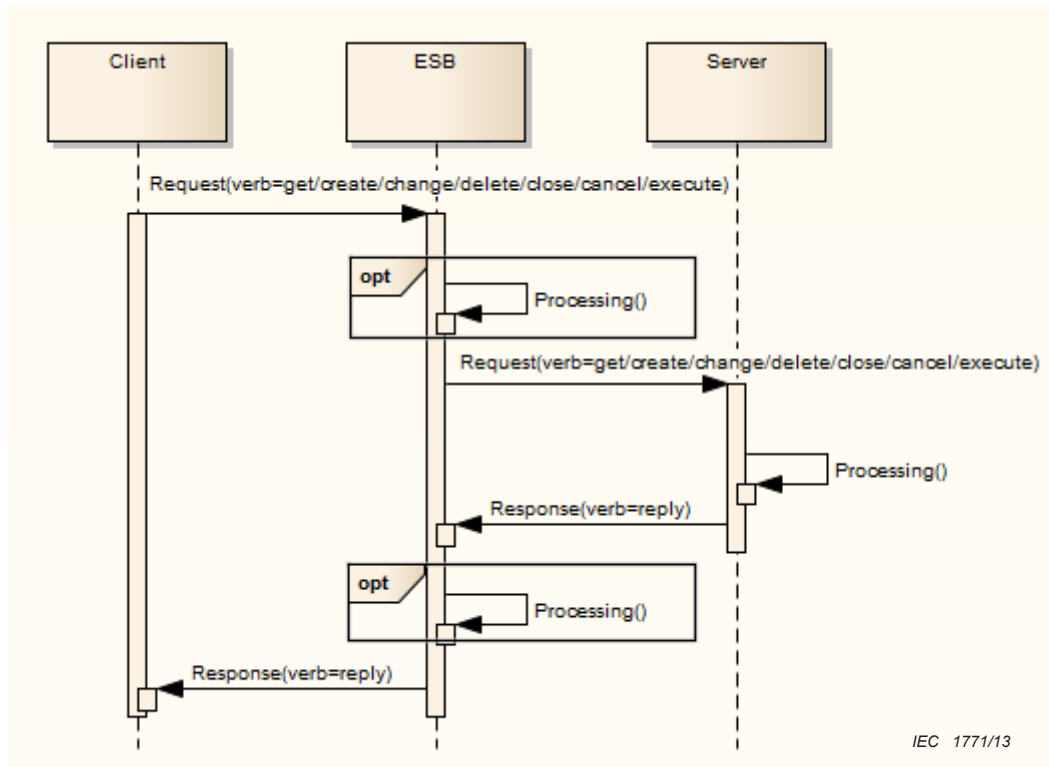


Figure 3 – Request/reply using intermediaries

A key aspect of this use case is the decoupling of the client from the server, so that the client does not have to know the exact location of the server or conform to the exact interface used by the server. The routing and mapping can take place in the integration layer.

It is also recommended that ESB intermediaries be stateless. The use of stateless intermediaries simplifies the implementation of load balancing and high availability.

4.4 Events

There is often the need for client processes to be informed of an event of potential interest. Many client processes can listen (subscribe) for events. One example of this is EndDeviceEvents messages that may be published by a metering system. Other examples include events that report the execution of a control or transaction, such as the creation or update of a work order.

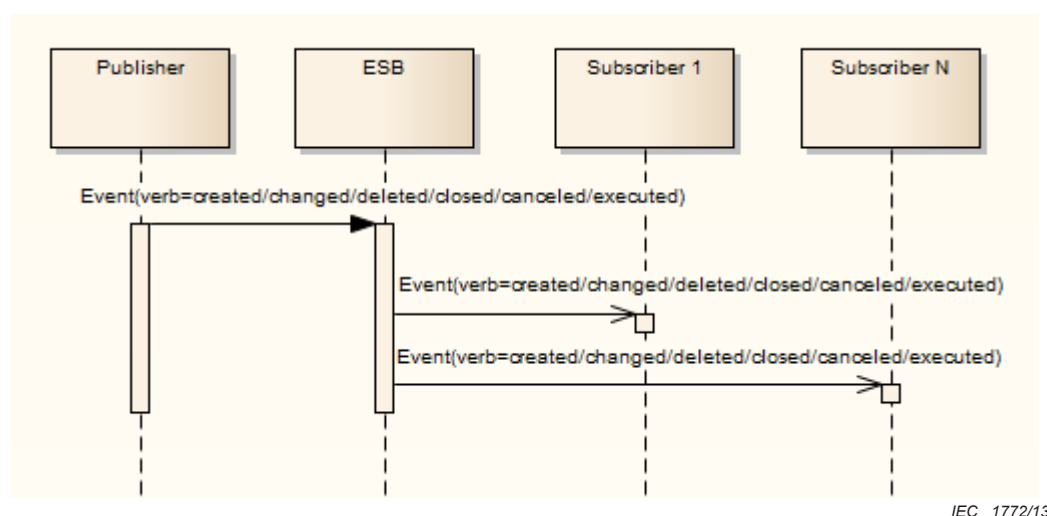


Figure 4 – Events

In terms of IEC 61968, events use the 'past tense' verbs "created", "changed", "deleted", "canceled", "closed" or "executed".

This is an example of a one way pattern, where the sender does not expect a response at the application level, although the transport used may allow for lower level acknowledgements in order to facilitate delivery guarantees when needed.

Listeners that use web services will need to have an exposed interface at a URL that is known by an intermediary so that the events can be appropriately distributed. Rules must also be defined for retry processing where guaranteed delivery is required. The implications of message ordering also need to be considered. Message ordering is a significant topic in itself that may require attention in a future edition.

4.5 Transactions

The use case for a transaction is typically a combination of a request/reply exchange between a client and a server, with a consequential publication of events. An important aspect is that the clients and services may use different transport mechanisms, where a client may use web services but a server may use JMS. An intermediary within the ESB can provide the necessary routing functions.

Transactions are commonly implemented using a request/reply pattern, but they can also be implemented using a point-to-point or one way pattern. The one way pattern can use either present tense verbs for transactions or past-tense verbs for events and the needs of a specific implementation may require.

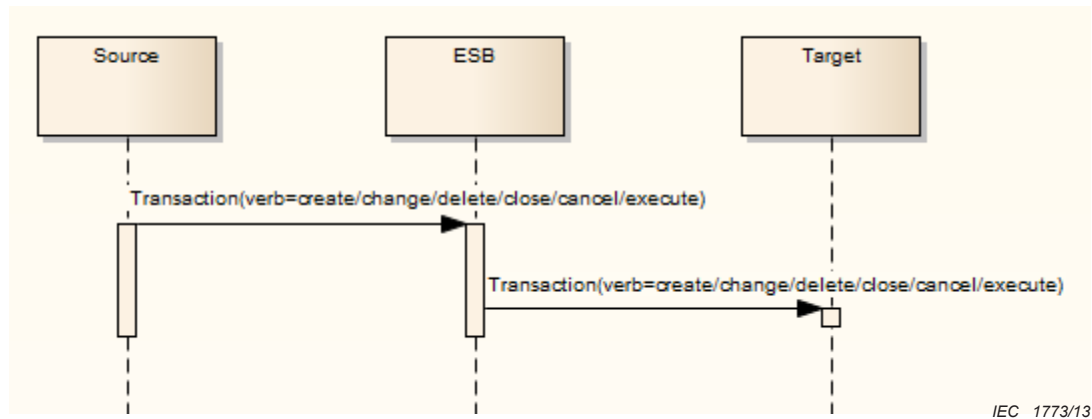


Figure 5 – Point-to-Point (One Way) Pattern

The one way pattern is used in cases where a system of record wants to forward a transaction to a peer system that also shall apply the transaction. If errors are detected by the target, they shall be logged for resolution within the target system. The following figure provides an example where a transaction will result in events that get propagated to interest subscribers.

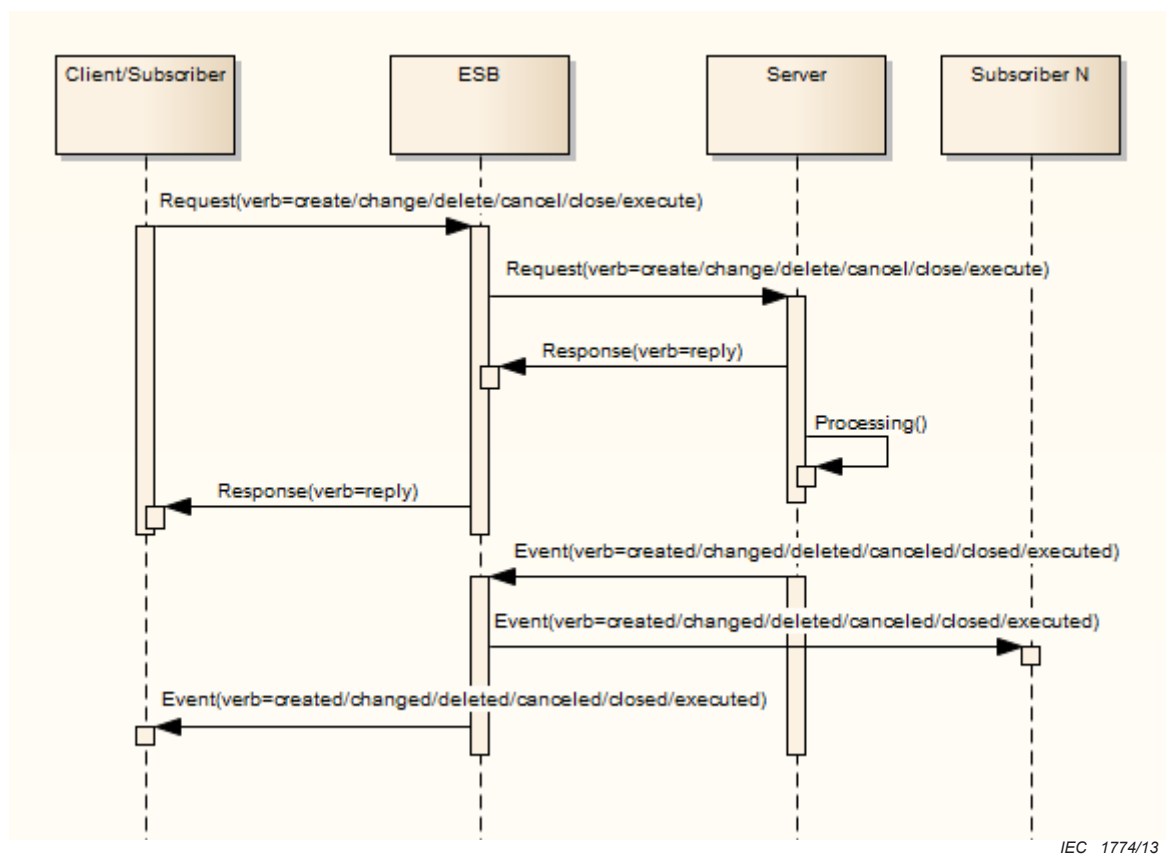


Figure 6 – Transaction Example

In terms of IEC 61968, requests related to transactions use the verbs create, change, delete, cancel, close, and execute. Verbs related to events use past tense. The use of the execute verb implies a complex transaction and the use of the Payload.OperationSet element, as described in 6.9.

4.6 Callback

A callback is an asynchronous process for message exchange. It is made of two request/response (initial and final) synchronous calls. The two are correlated in a way that

each party can unambiguously identify which callback goes with which initial request. In this case, the client sends an initial request to server. Once the server receives the message, it returns a response message back. At this point the initial message transaction is completed and client application is freed to perform other processing. Once the server has completed processing, it then invokes the final request/response sequence with a request message. The whole call-back process is completed after the client (of the initial request) replies to the final request. This is illustrated in the following sequence diagram.

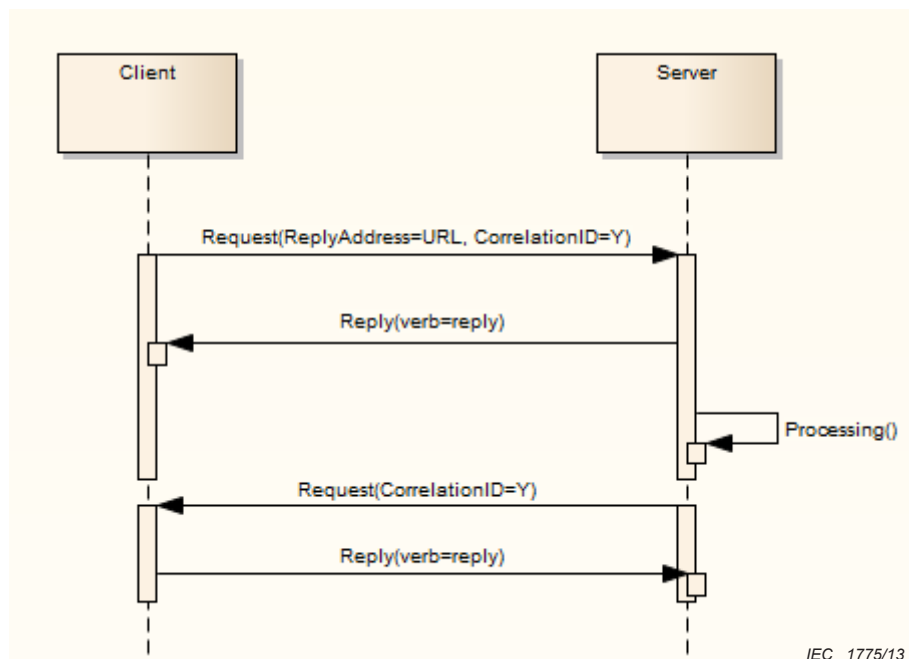


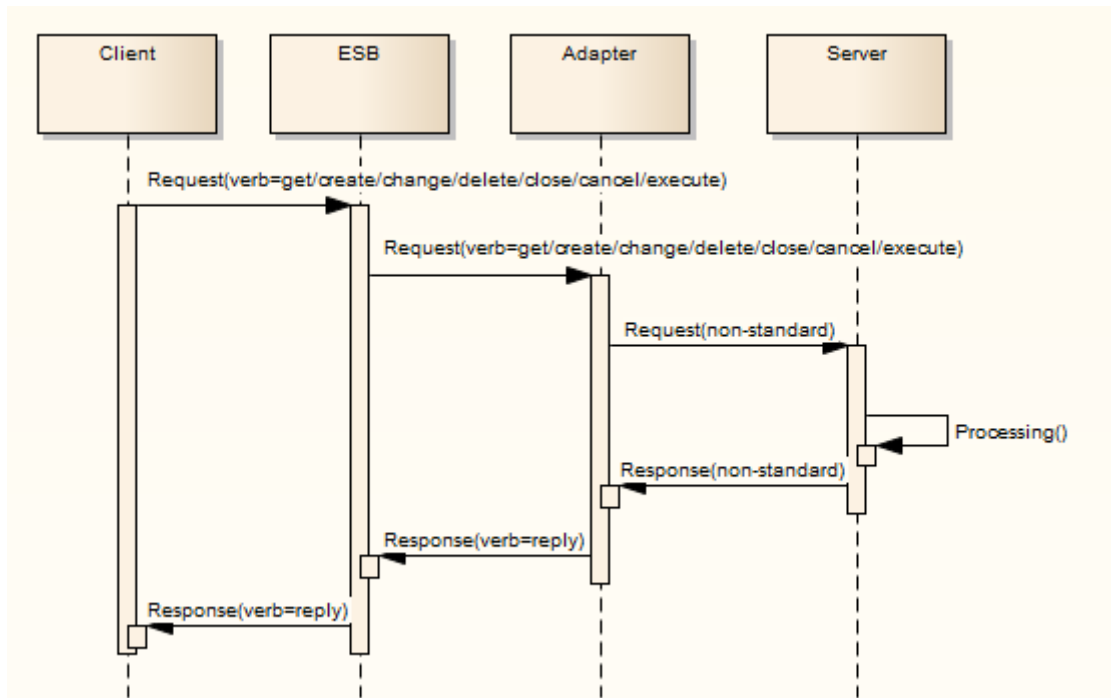
Figure 7 – Callbacks

In the call back process, the client has to inform the server of the location for the final response, often a URL called a callback address. This piece of information is included in the initial request message. 6.3.2 discusses specific message elements that are used to control this type of dialog.

Callbacks are typically implemented through asynchronous replies when using JMS.

4.7 Adapters

There are cases where an application (client or server) cannot directly connect to an ESB or another application. In these cases, an adapter can be used to handle the 'impedance mismatch' between the application and the ESB. In some cases the application may simply be a database or file directory. In the following diagram an adapter is used to connect a server to the ESB.



IEC 1776/13

Figure 8 – Use of Adapters

The adapter may perform a variety of functions, and may take actions on behalf of the application such as generation of events that are consequential to the success of a transaction. The most common activity is to convert data between an application model and an enterprise canonical model.

4.8 Complex messaging

There are some use cases that may require more complex messaging patterns. For example, there may be cases where a transaction may result in potentially many consequential events that can be sent to the client in the form of asynchronous events.

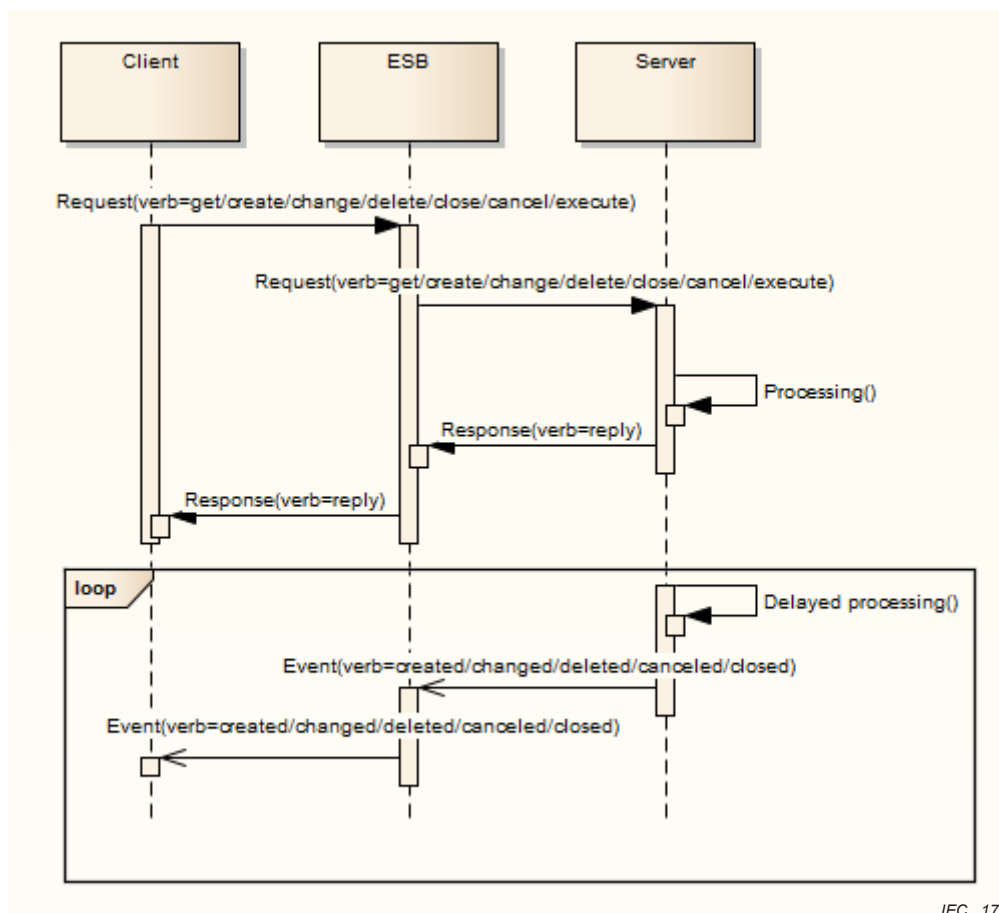


Figure 9 – Complex messaging

The above use case is seen in applications such as metering, where it may take significant time to obtain results as in the case of a disconnect or load control, where an EndDeviceControls message might be issued, but the results may be reported after some delay using an EndDeviceEvents message.

It is important to note that this is a variation of the callback pattern. The initial request could either be a query, where results are returned asynchronously, or a transaction where one or more events could be generated as a consequence.

4.9 Orchestration

Use cases related to the orchestration or choreography of a business process as well as short or long running distributed transactions are outside the scope of this specification. The integration approaches described by this document can be leveraged by more complex integrations that address those needs.

4.10 Application-level use cases

The use cases described in 4.1 to 4.9 can be applied to end use application-level use cases. In the following example application-level use case, messages are defined using IEC 61968 verbs and nouns, in the form '<verb>(<Noun>)'. Specific (or representative) types of systems are also identified as opposed to more generic types of actors.

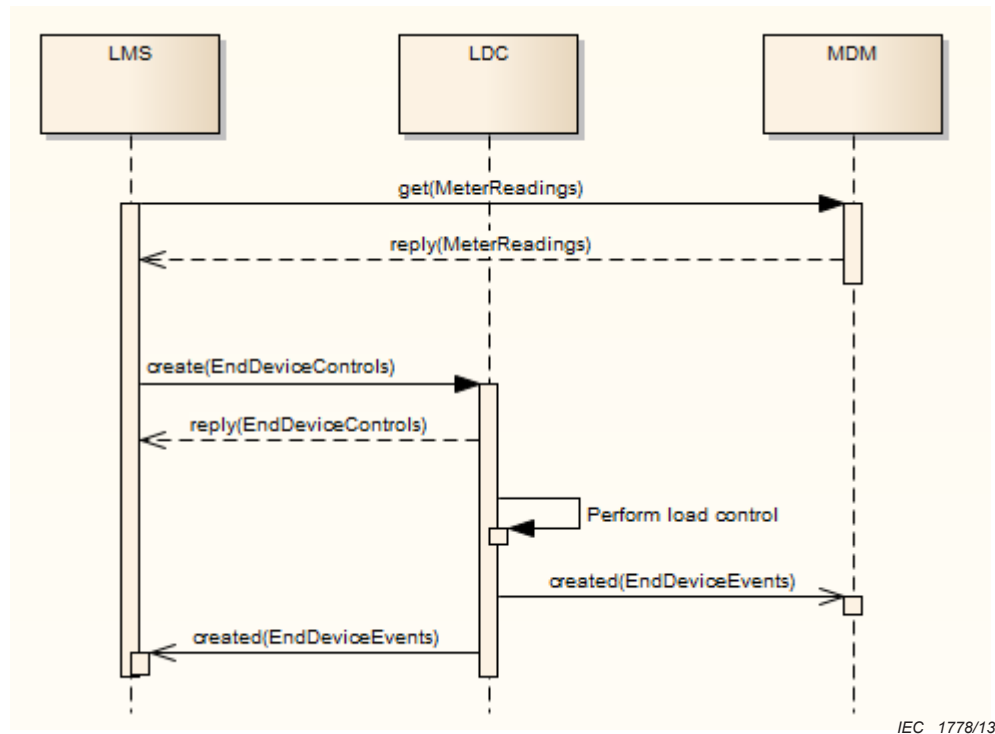


Figure 10 – Application-level use case example

5 Integration Patterns

5.1 General

This document recognizes a set of basic service integration patterns that support the previously described use cases. It also allows for more complex integration patterns through the use of intermediaries within an enterprise server bus (ESB), or where an application serves as an intermediary.

5.2 Client and server perspectives

5.2.1 General

From the perspective clients and servers there are several basic integration patterns described within this document. These patterns include, but are certainly not limited to:

- Synchronous request/reply
 - Web Service implementations use an operation with input and output messages
 - JMS implementations exchange messages on queues
- Asynchronous request/reply
 - Web Service implementations use separate operations for request and callback reply messages
 - JMS implementations exchange messages on queues
- Publish/subscribe, where potentially many targets are listening for messages
 - Web Service implementations will involve a client listening on specified URLs for events that are sent to multiple targets by an ESB intermediary
 - JMS implementations will involve targets listening for messages on a topic

From the perspective of either a client alone or a server alone, whether or not the client is communicating with the server directly, or through intermediaries is irrelevant so long as they

have chosen to use the same transport mechanism. The value of an ESB is that an integration architecture can be provided where a client has a choice of using web services, topics or queues, and each target service can also choose to use web services, topics or queues independently from the choices of any client. This can be used to isolate the technology choices of applications in the enterprise such that any bridging technology used has no direct impact on individual applications which would limit application migration options.

5.2.2 Basic web service pattern

The following diagram illustrates a basic request/reply pattern using web services.

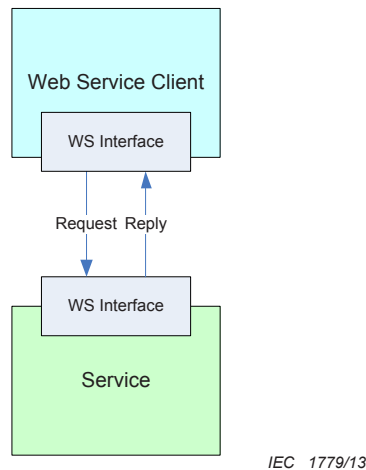


Figure 11 – Basic request/reply using web services

In this pattern the client issues a request to a web service interface exposed by some service. This interface is defined using a WSDL. However, this interaction pattern may also be realized using REST. The client should expect one of several outcomes:

- The request is successfully processed, where a reply message is returned in a timely manner (Result=OK)
- The request is accepted, but results in a reply message that returns an application level error code (Result=FAILED)
- The request is accepted, but results in a reply message with a partial set of results (Result=PARTIAL)
- The request results in a fault being returned to the client
- After sending the request, no reply or fault is returned in a timely manner

It is also important to note that there may be varying levels of security that may be required for the implementation. The extreme case is where the client is using a public network to communicate with the service, where authentication, authorization, encryption and signing may be important.

In the case where a reply is not required, this is called a 'one way' pattern. However care should be taken with respect to any required delivery guarantees.

5.2.3 Basic JMS request/reply pattern

Figure 12 describes a basic request/reply pattern where the client sends a JMS message to a topic (or queue). In this pattern the reply message is optional, where there may be some cases where a reply message is never sent.

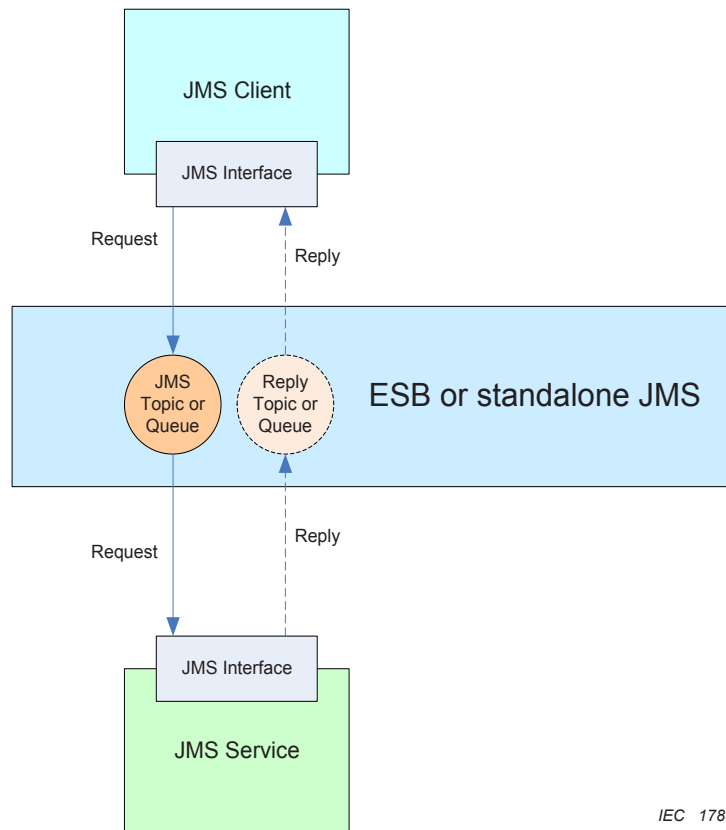


Figure 12 – Basic request/reply using JMS

A service listening on a message destination (topic or queue) consumes the request message and issues a reply to the client. The message destination is managed by the JMS implementation (typically part of an ESB). The client may consume the reply message synchronously or asynchronously, with the decision being left solely to the discretion of the client implementation.

Where this document refers to a JMS ‘message destination’, implementations may use either topics or queues for request/reply messaging. When using topics, a service would typically use a durable subscription in cases where request messages must not be lost.

The client initiating the request can expect one of the following results:

- The request message is successfully sent to a topic (or queue), where a reply message (correlated to the request using an ID) is returned in a timely manner (Result=OK)
- The request message is sent, but results in a reply message that returns an application level error code (Result=FAILED)
- The request is accepted, but results in a reply message with a partial set of results (Result=PARTIAL)
- The attempt to send a request message to the topic (or queue) fails
- After sending the request message, no reply message is ever received (which may or may not be normal behaviour depending upon the service)

Reply messages are sent using topics or queues as appropriate, where the specific topics may be statically or dynamically defined. For simplicity subsequent diagrams will not explicitly identify reply topics or queues. The delivery guarantees offered by JMS implementations provide the option for clients to making transaction requests to not require replies, where this is sometimes referred to as a one-way pattern.

JMS is typically used within a secure, private enterprise network. However there may be isolated cases where this is not the case and security is a significant concern. JMS can also be readily configured to use SSL/TLS and/or use client authentication.

This pattern actually does not require a full ESB implementation, where only a JMS implementation is actually needed. It is also important to note that JMS implementations from different vendors are typically not interoperable, and a 'bridge' may be required in cases where clients cannot use a common JMS implementation.

5.2.4 Event listeners

Another integration pattern is that of a process that listens for events that may be published. There are many cases where a service may publish event messages that are of potential interest to many other processes.

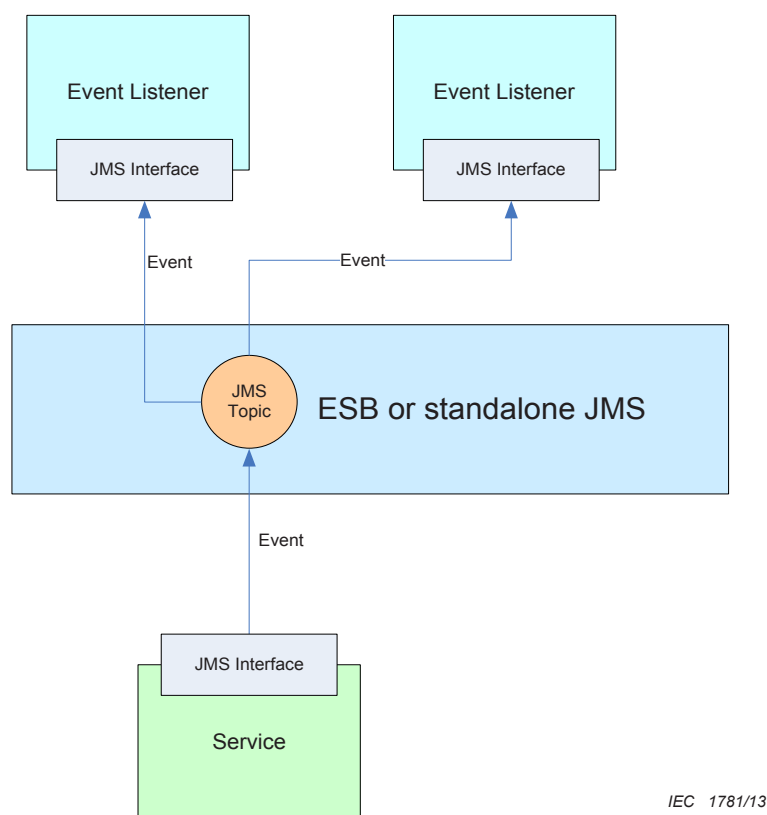


Figure 13 – Event listeners using JMS

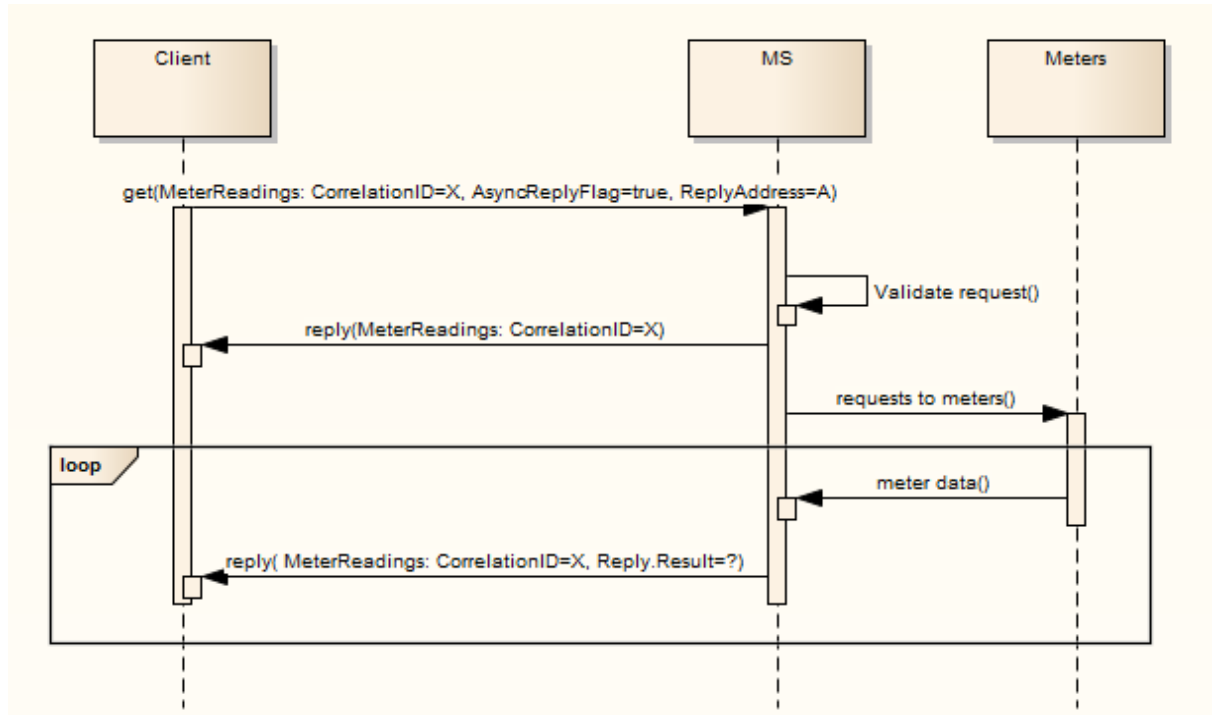
Events should typically be published on topics (as opposed to queues), as there will typically be many consumers of events. The listener is responsible to subscribe to one or more JMS topics of potential interest. Some listeners may choose to use a durable subscription in cases where events shall not be lost. Event messages are sent and consumer asynchronously. There is no acknowledgement message of any kind returned to the service.

There may be issues related to the number of topics (or queues) and the fan out (i.e. number of event listeners) that can be supported using a given JMS implementation.

It is also possible to extend the architecture described by this document to leverage the use of WS-Eventing for the publication and subscription of events by web service clients. There is currently an open source of WS-Eventing through the Apache project. However, this document will not focus on specific aspects of the implementation and use of WS-Eventing. This document also provides another example of routing events using web services.

5.2.5 Asynchronous request/reply pattern

The asynchronous request/reply allows one or more replies to be returned to a requesting client asynchronously. This is a complex integration pattern that is slightly more complex to implement using web services than for JMS.



IEC 1782/13

Figure 14 – Asynchronous request/reply pattern

In this pattern, a client makes a request to a service. Where the service may not be able to process and/or respond with the desired information immediately, the service responds immediately with a trivial acknowledgement of the request. After processing is able to occur and/or the desired information is obtained, the service can provide one or more asynchronous replies to the client using a specified URL, topic or queue. The key elements (as described in Clause 6) used to control this exchange include:

- Header.`CorrelationID` is used to logically link all messages together. Where a `CorrelationID` is provided by the client on the initial request, the server shall include it on all related response and event messages.
- Header.`AsyncReplyFlag` is set to 'true' on the initial request.
- Header.`ReplyAddress` is set on the initial request to identify the destination where replies should be set.
- `Reply.Result` is used on responses, where 'PARTIAL' indicates that more responses may be expected, or 'OK' indicates that processing is complete and no more responses should be expected. A value of 'FAILED' indicates an error condition.

One use of this pattern is to obtain meter readings, where a metering system head end shall request the desired information from one or more meters.

5.3 Bus perspective

5.3.1 General

Clients and servers can either communicate directly, or through intermediaries. The enterprise service bus (ESB) is used for the management of intermediaries. The use of an ESB provides for many variations in communication patterns. However, the client still sees the ESB as a

no different than a server. Similarly, the server sees the ESB as being no different than any other client.

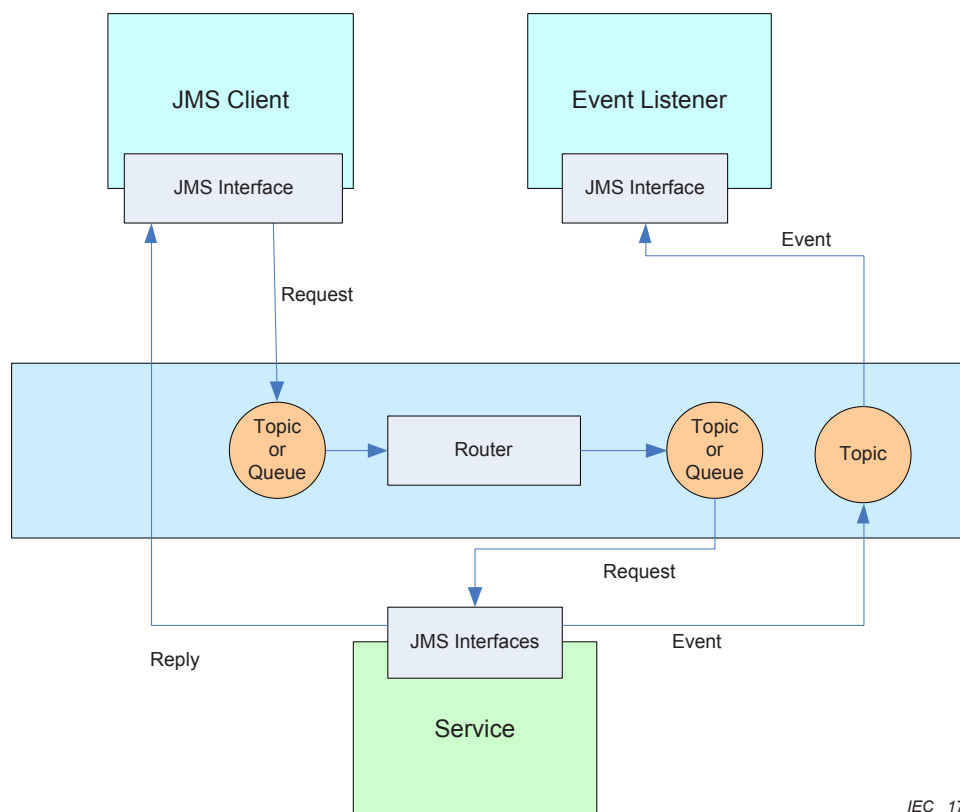
This is important in that:

- No knowledge of the ESB is imposed upon any client or server implementation of an IEC 61968 interface
- There is no requirement for an ESB product placed upon any IEC 61968 interface, except that a JMS server is needed where JMS messages are to be used
- A project implementation using IEC 61968 can use an ESB and freely implement integration patterns that are appropriate for the project and the associated integrations

Given that an ESB is not required for the implementation of an IEC 61968 interface, the remainder of 5.3 describes recommendations only, and is provided as informative material.

5.3.2 ESB messaging pattern using JMS

The basic ESB messaging pattern using JMS introduces the option for routing of requests. This serves to further decouple the client and server, where the bus (through use of routing logic, often referred to as a 'Content-Based Router' integration pattern [EIP]) can make decisions related to the handling of the request. This is described in the following diagram.



IEC 1783/13

Figure 15 – ESB content-based routing

When a client issues a request to a topic (or queue), a router on the bus can decide to forward the message to another topic (or queue). The decisions by the router may take into account any of the following:

- Contents of a message header
- Contents of a message payload (although this should be avoided)
- The status of a destination service instance

- It is important to note that one advantage of the loosely coupled approach described by this document is that routing components are not tied to messages of specific payload types. The router can be configured using XPath expressions to identify message content to determine actual routing.

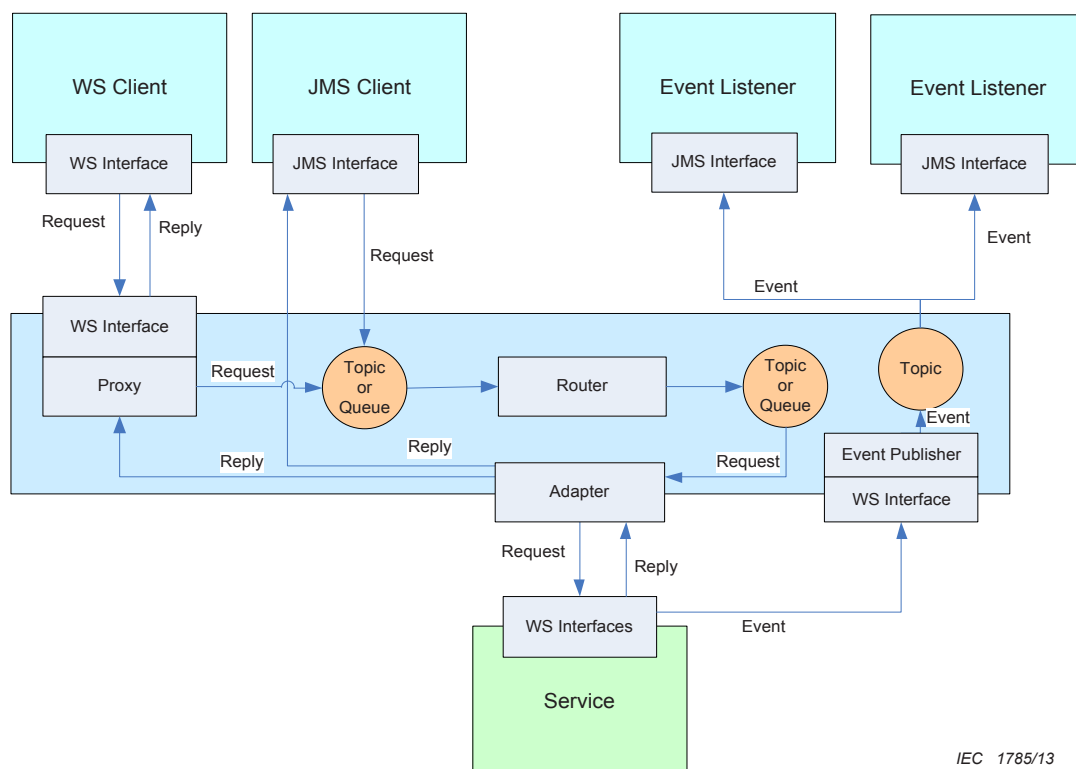
The following diagram extends the previously described pattern to permit a request to be initiated by a web service client as well as a JMS client.



Figure 16 – ESB with smart proxy and content-based routing

A proxy (sometimes referred to as a 'Smart Proxy' integration pattern [EIP]) component is implemented on the ESB to expose a web service (as defined by a WSDL). The Smart Proxy can make decisions with respect to dispatching of requests and correlation of responses. The message conveyed through the WSDL is simply converted to a JMS message and is then routed as appropriate.

The following diagram extends the previous pattern to allow for a service to expose its interface as a web service with an appropriately defined WSDL.



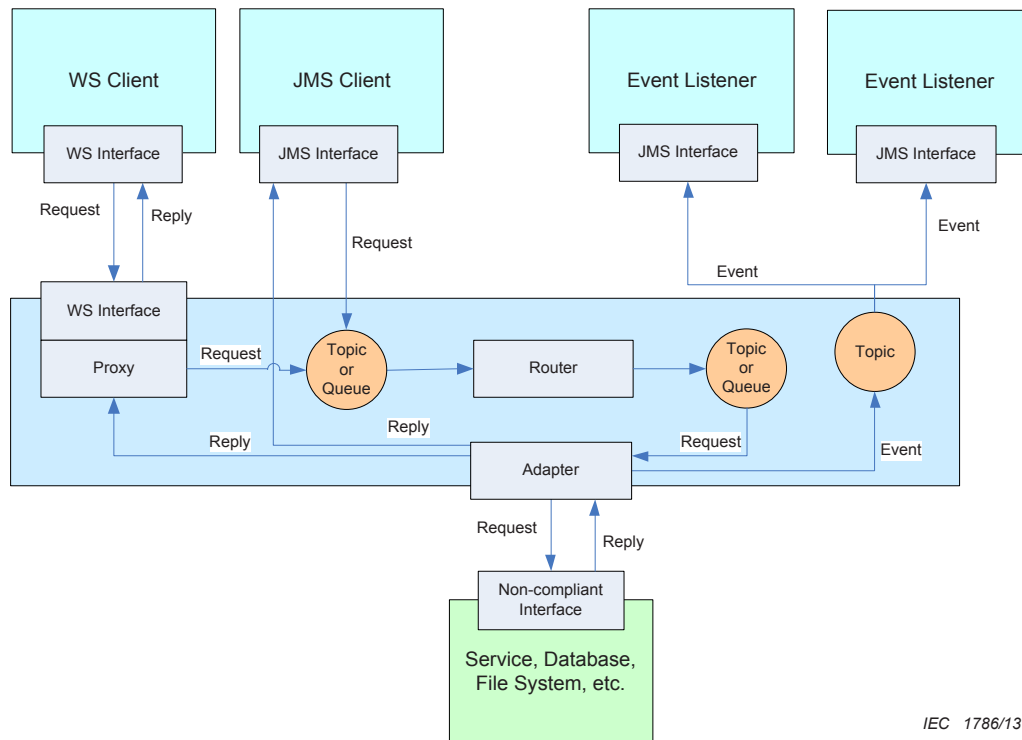
IEC 1785/13

Figure 17 – ESB with proxies, routers and adapters

Within this pattern, an adapter is implemented within the ESB to convert the internal JMS message to an appropriate web service request.

5.3.5 ESB request handling via adapter

The following diagram is a variation on the previous integration pattern, where the server uses an interface that would otherwise not be compliant with the interface profile described by this document. This shows that an IEC 61968 compliant interface can be used to integrate with a server, database, file system or other data source or sink that is otherwise not compliant with IEC 61968 through the use of an adapter within the ESB. Adapters may also be independent of an ESB.



IEC 1786/13

Figure 18 – ESB Integration to non-compliant resources

The integration between the adapter and non-compliant interface can use a variety of integration mechanisms depending upon the capabilities of the specific ESB product. These mechanisms can include, but are not limited to:

- JMS
- Web services
- HTTP
- Java Database Connectivity (JDBC)
- File Transfer Protocol (FTP)
- File read and/or writes
- Proprietary database access

Where the specification of JMS and JDBC imply the use of a Java Enterprise Edition (JEE) framework, it does not impose an actual requirement. Most databases that support JDBC can also be accessed using Open Database Connectivity (ODBC), either directly or through bridge products. Many ESB products that support JMS also have APIs that can be used to send and receive JMS messages using languages other than Java (e.g. C, C++). However, it is important to recognize that the use of the JEE framework provides for a high degree of platform independence.

5.3.6 Custom integration patterns

Typically an integration project will involve the implementation of a variety of custom integration patterns. Subclauses 5.3.1 to 5.3.5 alluded to the potential existence and use of some of these patterns implemented as intermediary processes within the ESB. These would potentially include, but not be limited to patterns such as [EIP]:

- Content-Based Router, where messages are routed based upon message content typically referenced using XPath expressions

- Smart Proxy, where messages may be re-dispatched to a specific destination service, where replies are accepted from the service and passed back to the client
- Claim Check, where a copy of an often very large file is maintained as a document for use by other processes, where the current status of the document is tracked, but the document is typically transported by means other than messaging
- Transformation, where transformations usually defined by XSL are used to reformat message contents
- Bridge, where a message published on a topic or queue may be forwarded to or received from another messaging infrastructure (this pattern can sometimes be implemented using third party products or simply through configuration)

However, it is important to note that this standard does not mandate the implementation or use of any specific custom integration pattern. It is also important to note two primary philosophies for the implementation of integration patterns:

- 1) Patterns are implemented as a single process definition that may be instantiated one or more times to support potentially many information flows, where there are no type constraints
- 2) Patterns are templates, where the template is used to implement a process to support a specific information flow, resulting in a 'type-specific' implementation

There are significant trade-offs with the above two philosophies. The focus of this specification is to recommend the first option through the use of a common message envelope that readily supports leveraging common implementations of specific integration patterns as opposed to type-specific instances of a given integration pattern.

6 Message organization

6.1 General

Each service interface is constructed to accept a message that has a verb and a noun. The noun identifies the type of the payload that may be provided on the request, response or event message. This allows the interfaces to be loosely coupled.

The service interfaces are defined using one or both of the following:

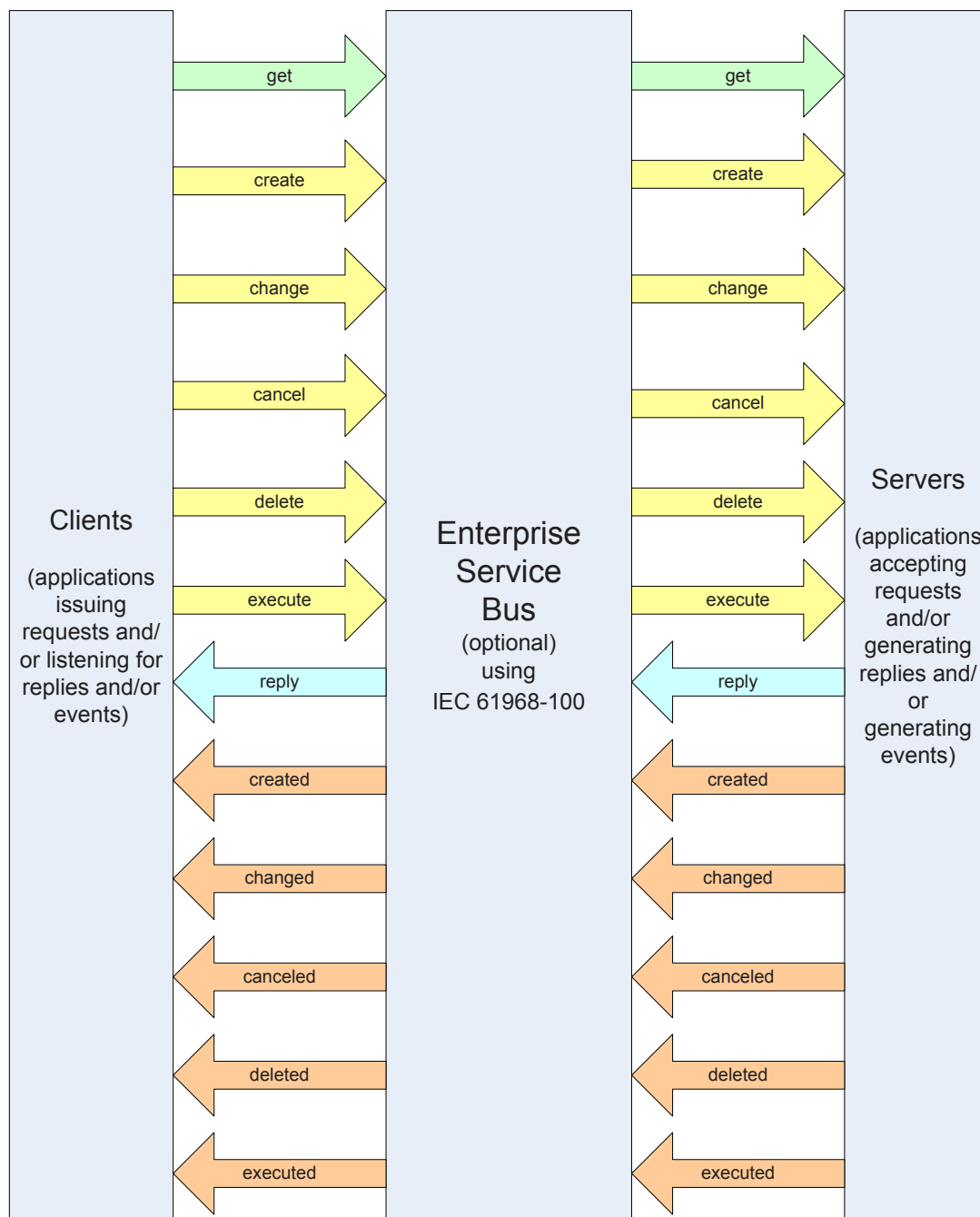
- Web Services Definition Language (WSDL), where request, response and fault messages are defined for one or more operations
- JMS message definition

In all cases, XML Schemas (XSDs) are used to define the structure of message envelopes. In most cases, XSDs are used to define the structure of message payloads. The content of message payloads is described in Clause 7.

6.2 IEC 61968 messages

6.2.1 General

IEC 61968-1 prescribes information exchanges in terms of a verb, noun and payload. Figure 19 shows the directional flow of messages between clients, servers and the ESB based upon the verb.



IEC 1787/13

Figure 19 – Messaging between clients, servers and an ESB

6.2.2 Verbs

IEC 61968-1 identifies a set of verbs, where annex B of this standard defines a normative list. This sub-clause is to provide more specificity on the usage of each verb and identify the deprecation some verbs as well as synonyms. In Table 1 verbs used for requests are associated with the verb that should be used on a response message and as such would be used for publication of an event, where often events are a consequence of the successful completion of a transaction initiated by a request.