IEC 61968-100

Edition 1.0 2013-07

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Application integration at electric utilities – System interfaces for distribution management –
Part 100: Implementation profiles

Intégration d'applications pour les services électriques – Interfaces système pour la gestion de distribution –
Partie 100: Profils de mise en oeuvre

IEC 61968-100:2013

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61968-100

Edition 1.0 2013-07

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

Application integration at electric utilities – System interfaces for distribution management –
Part 100: Implementation profiles

Intégration d'applications pour les services électriques – Interfaces système pour la gestion de distribution –
Partie 100: Profils de mise en oeuvre

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XE

ICS 33.200

ISBN 978-2-8322-1007-9

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

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

Part 100: Implementation profiles

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61968-100 has been prepared by IEC technical committee 57: Power systems management and associated information exchange.

The text of this standard is based on the following documents:

FDIS	Report on voting
57/1358/FDIS	57/1382/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts in the IEC 61968 series, published under the general title *Application integration at electric utilities – System interfaces for distribution management*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

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.

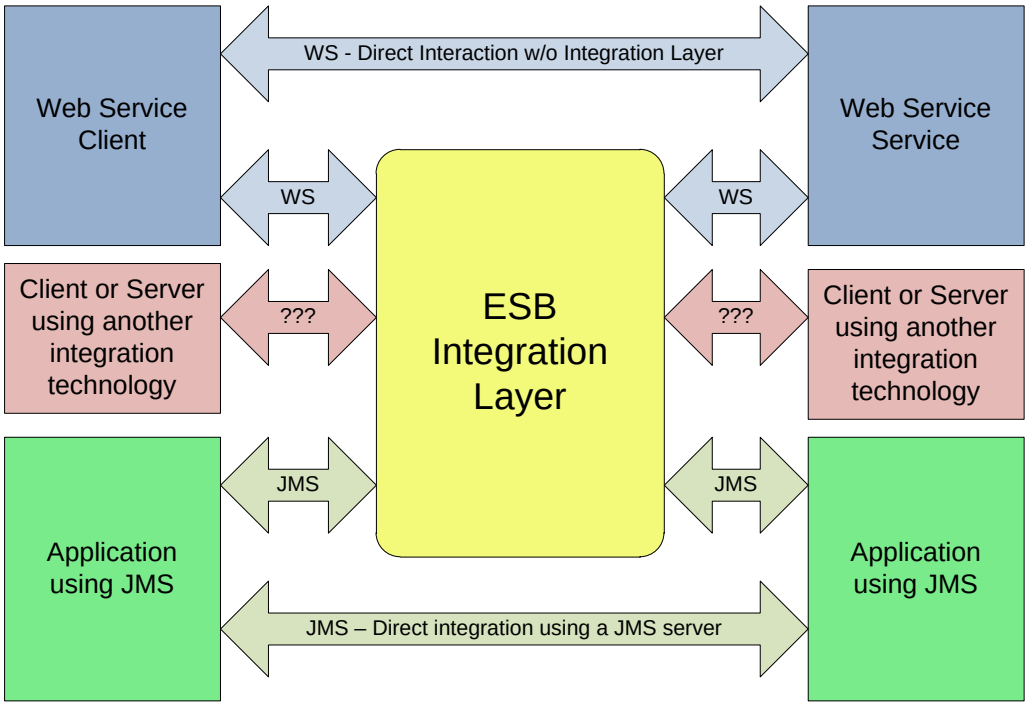


Figure 1 – Overview of Scope

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 dverse 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 synonomous 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) 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

b) 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.

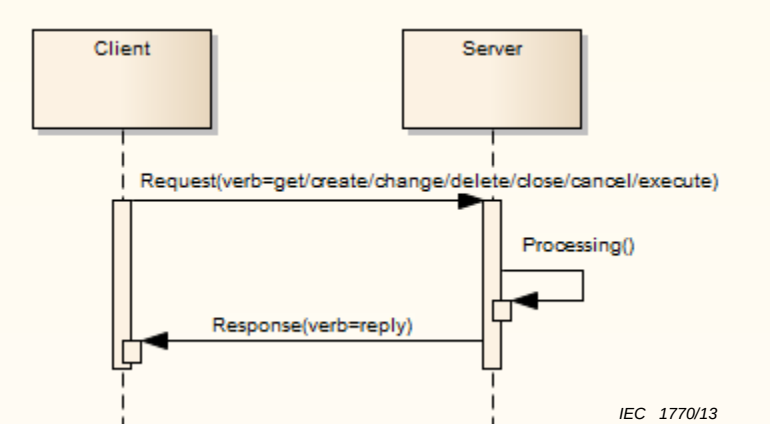


Figure 2 – Simple Request/Reply

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.

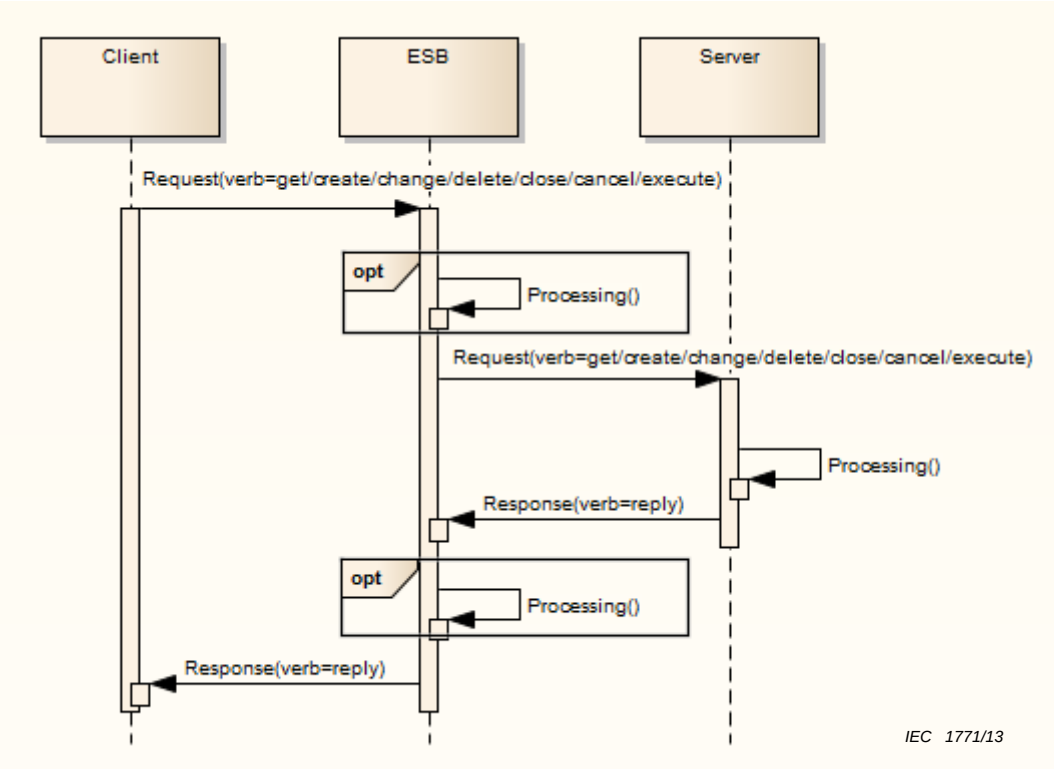


Figure 3 – Request/reply using intermediaries

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.

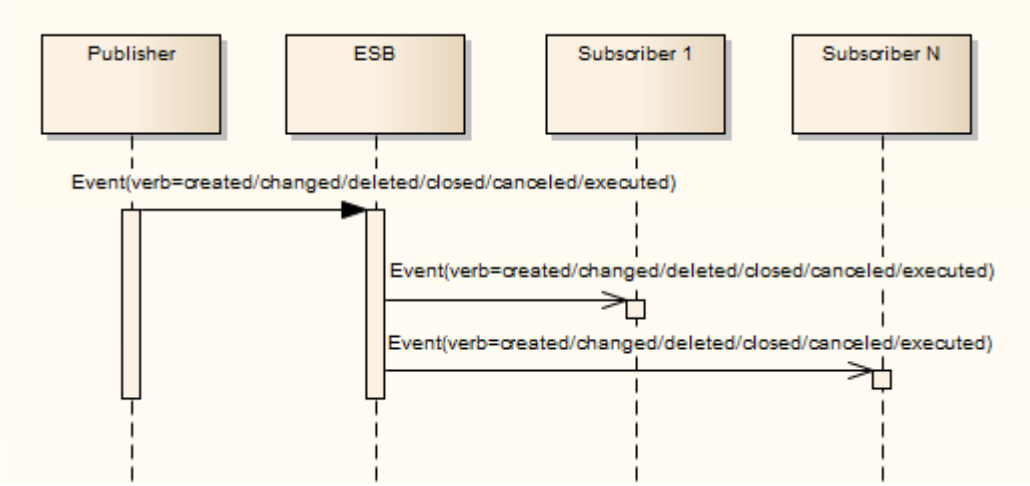


Figure 4 – Events

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.

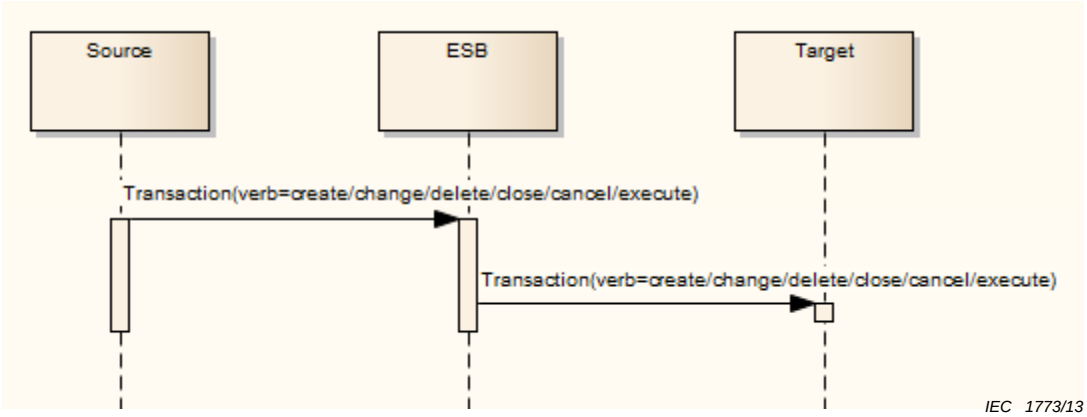


Figure 5 – Point-to-Point (One Way) Pattern

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.

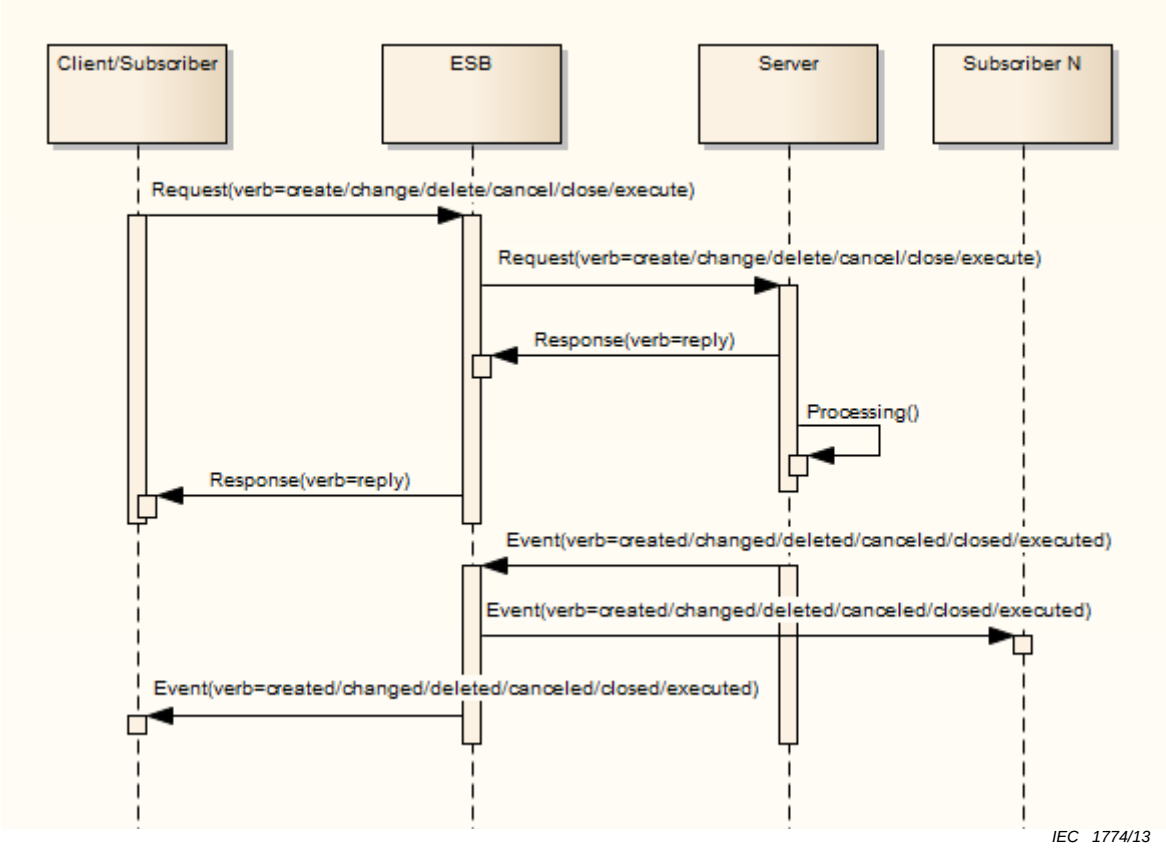


Figure 6 – Transaction Example

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.

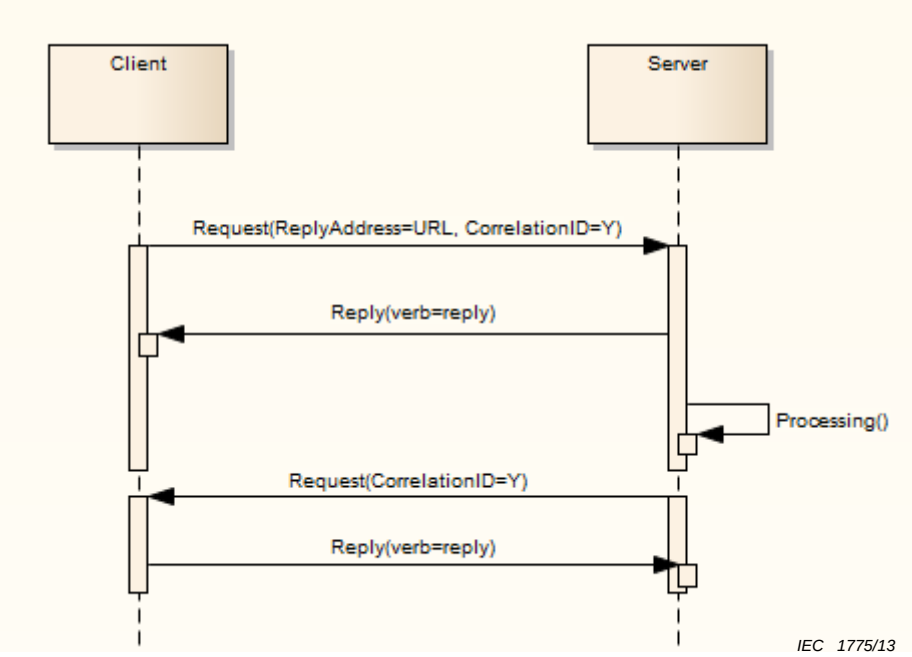


Figure 7 – Callbacks

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.

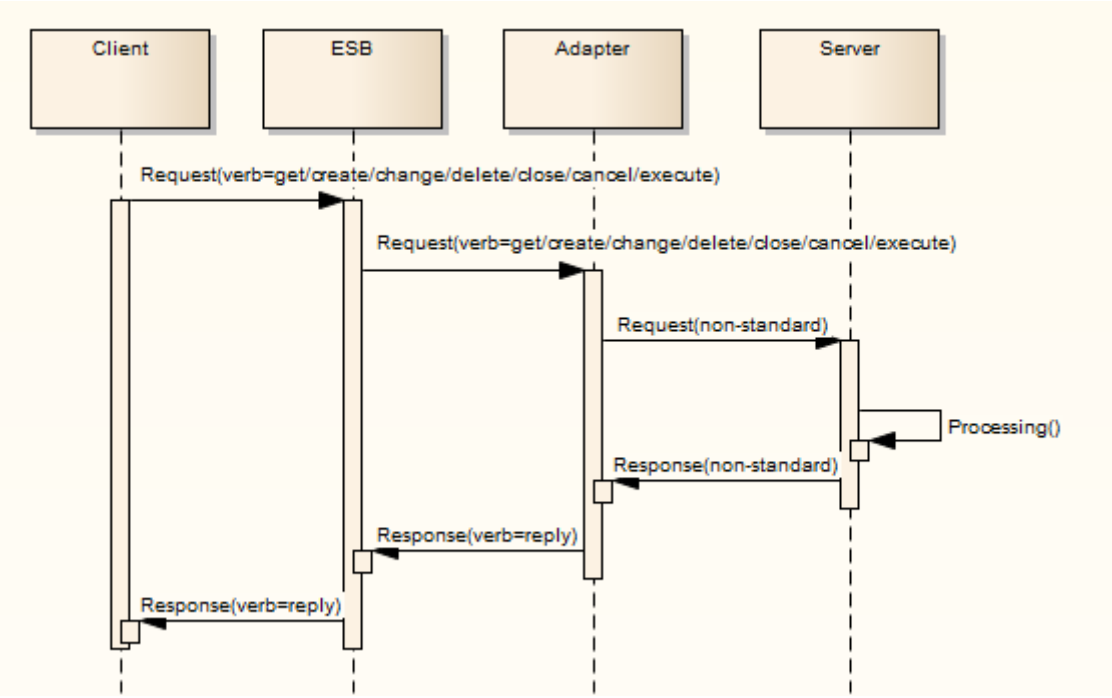


Figure 8 – Use of Adapters

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.

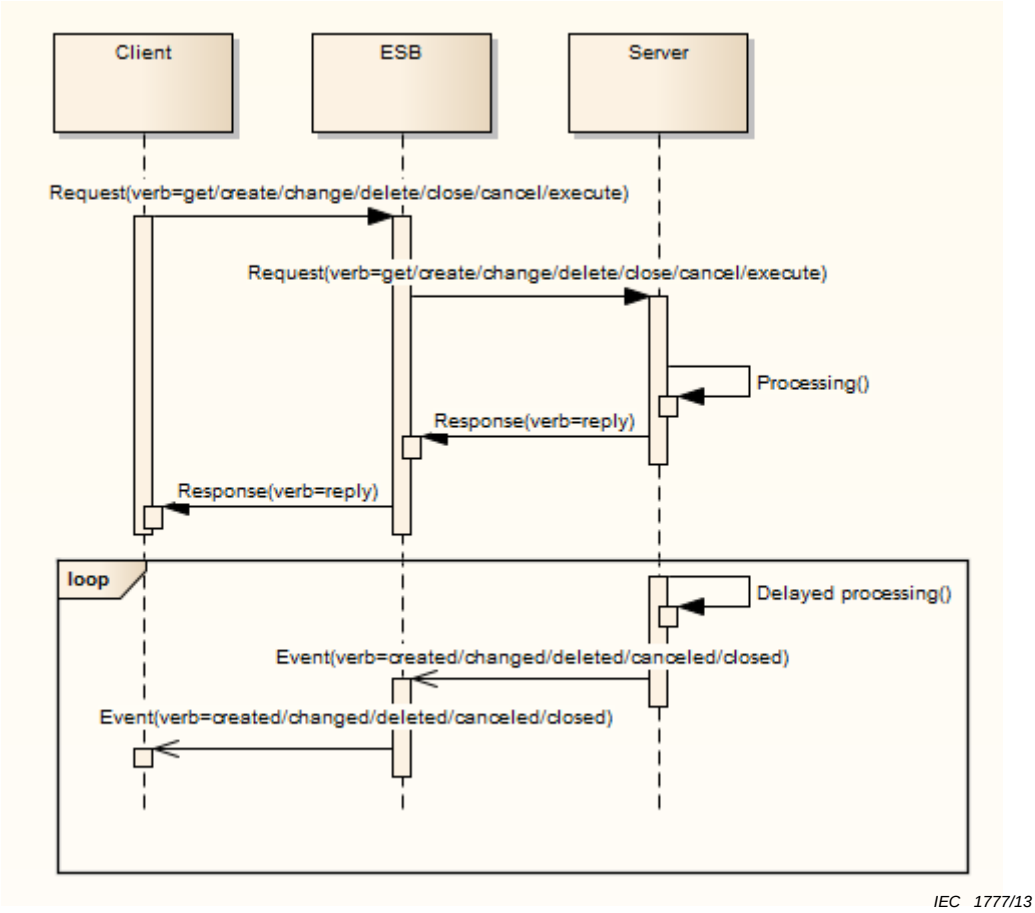


Figure 9 – Complex messaging

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.

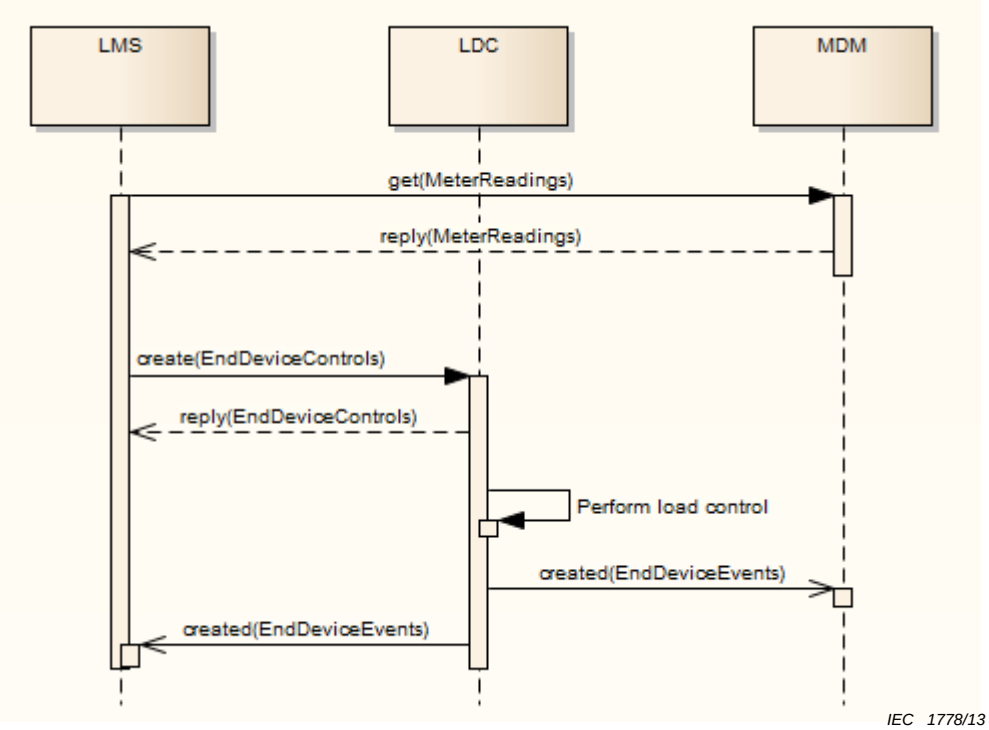


Figure 10 – Application-level use case example

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 an 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.

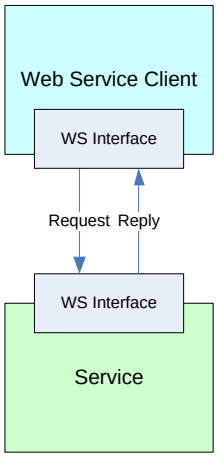


Figure 11 – Basic request/reply using web services

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.

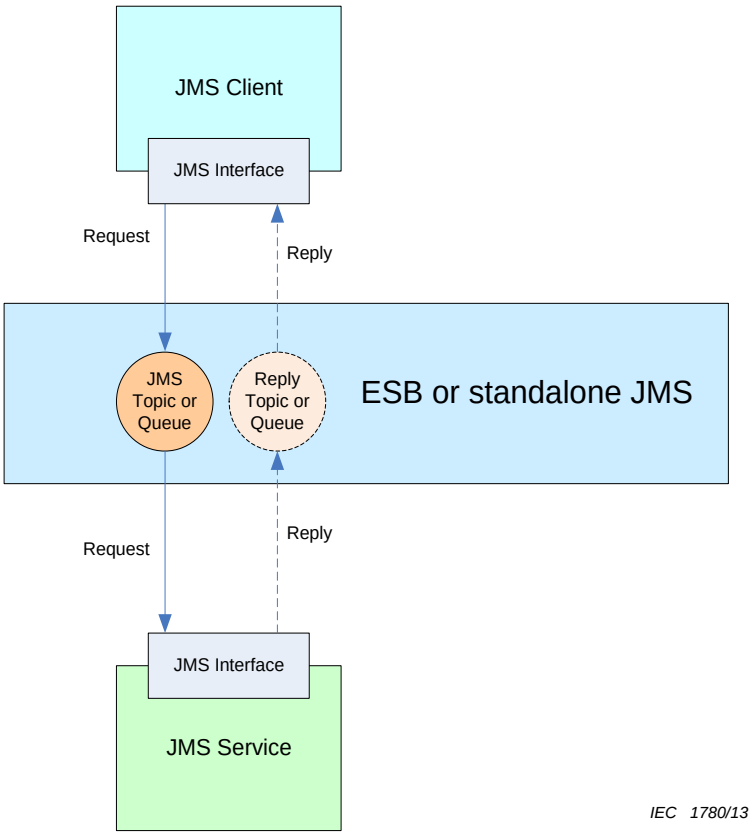


Figure 12 – Basic request/reply using JMS

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.

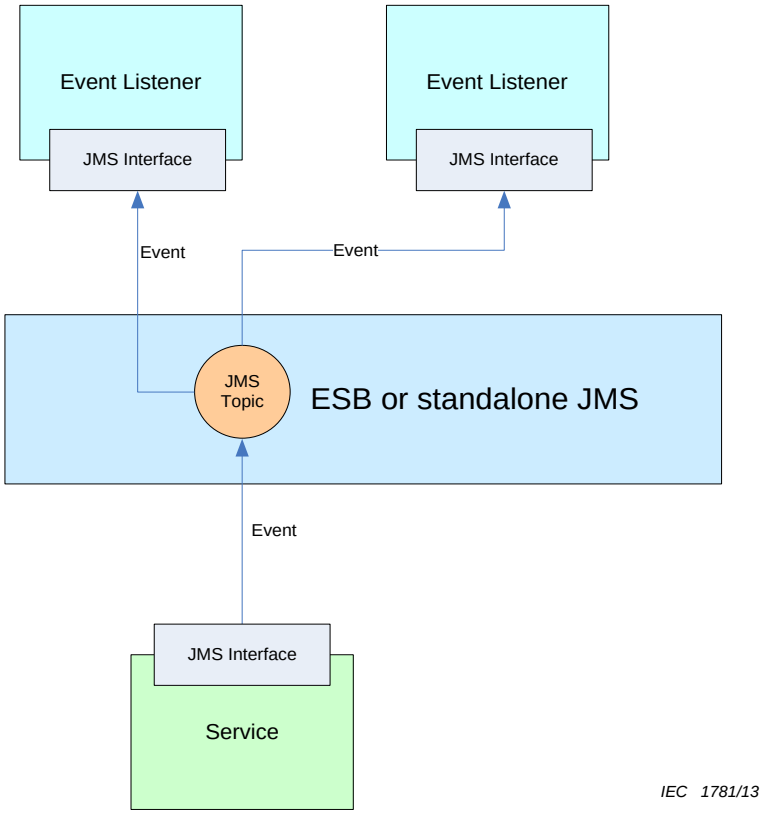


Figure 13 – Event listeners using JMS

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.

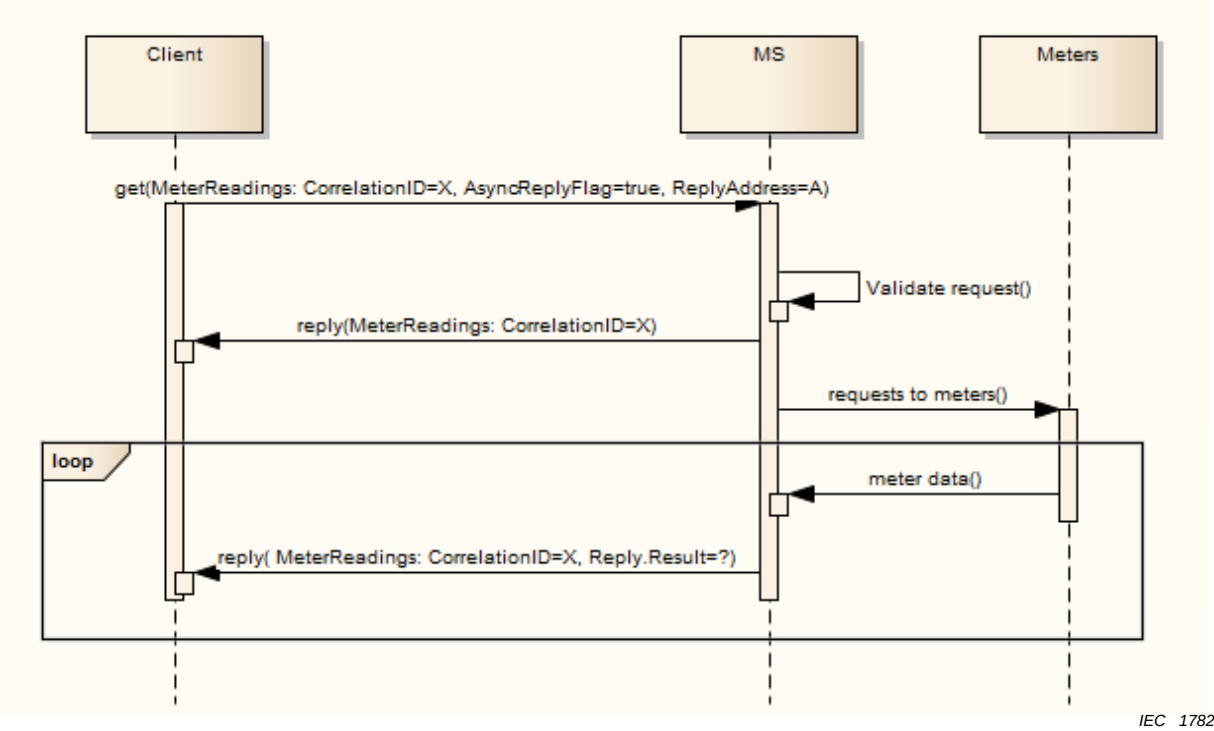


Figure 14 – Asynchronous request/reply pattern

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 being

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.

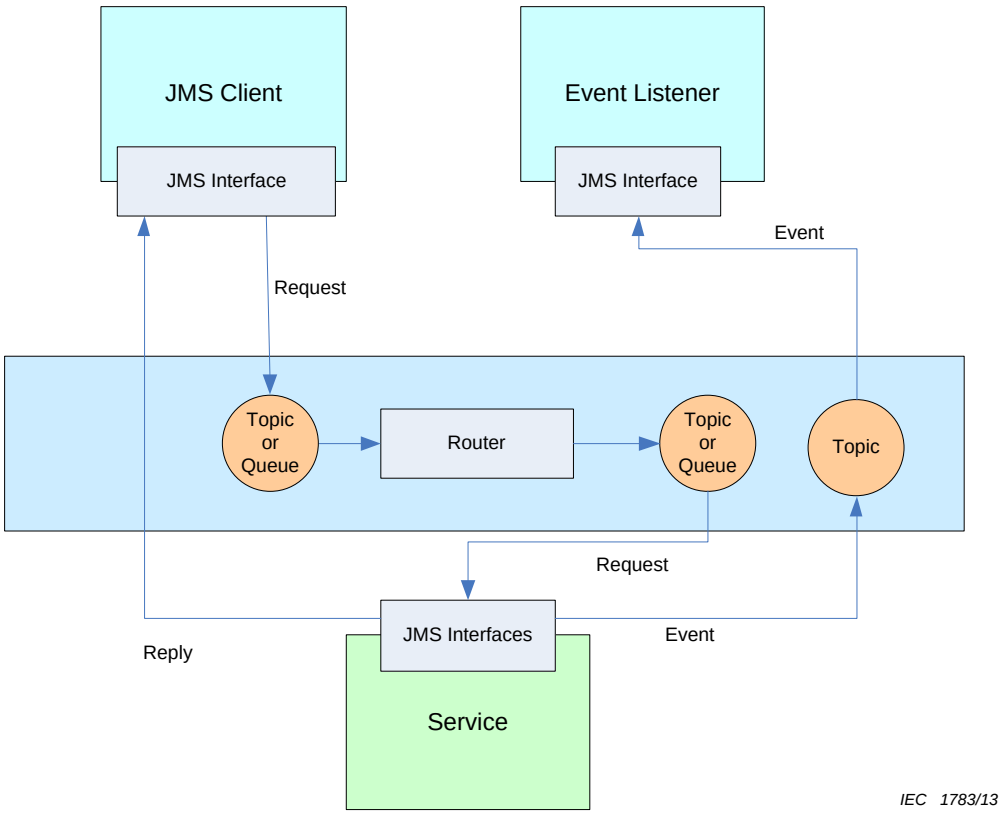


Figure 15 – ESB content-based routing

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

- The need to balance load

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.

5.3.3 ESB messaging patterns using web service request

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.

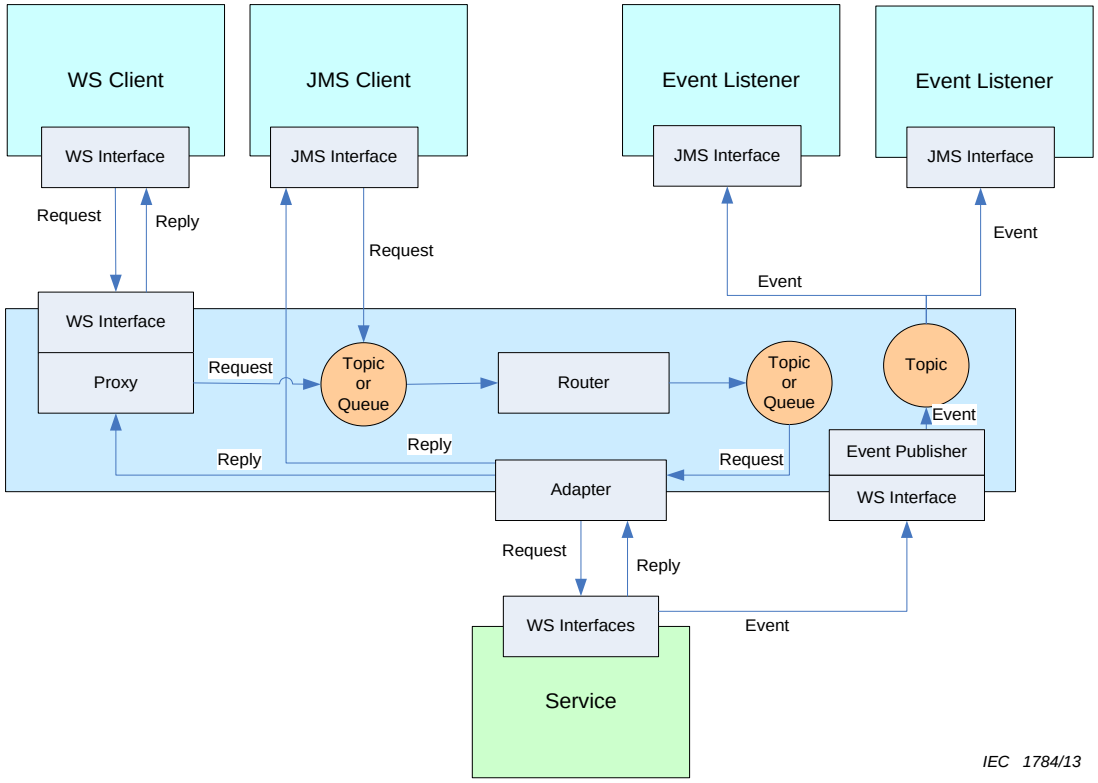


Figure 16 – ESB with smart proxy and content-based routing

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.

5.3.4 ESB request handling to web service

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.

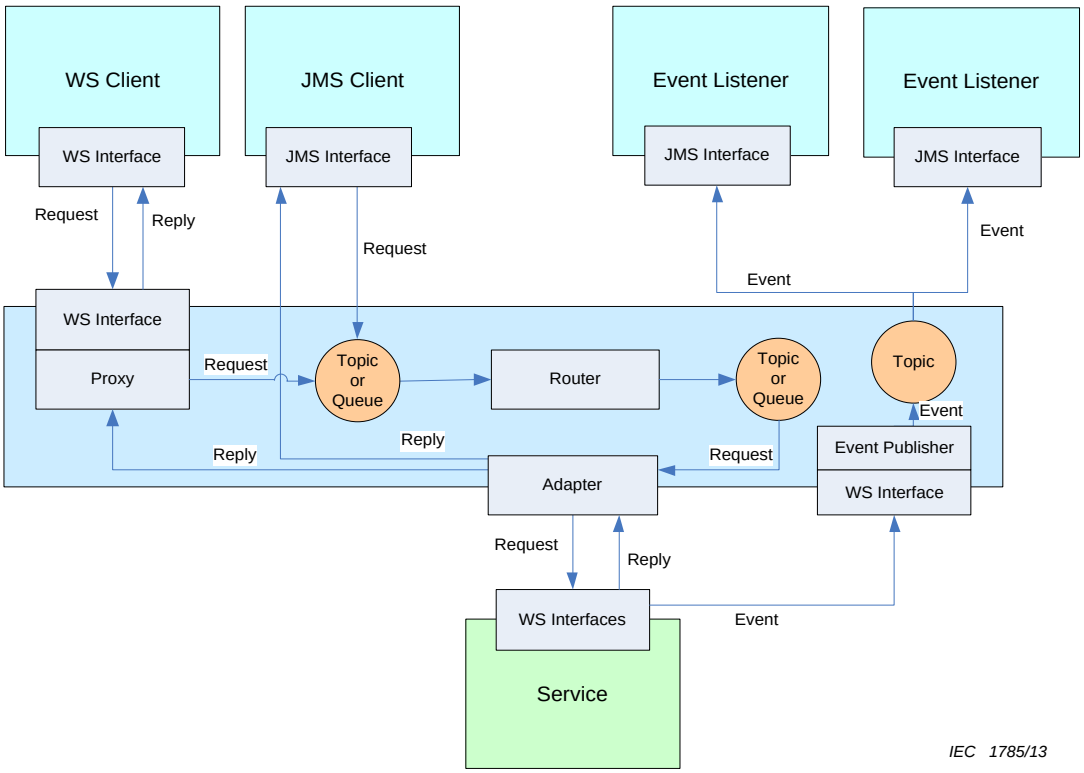


Figure 17 – ESB with proxies, routers and adapters

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.

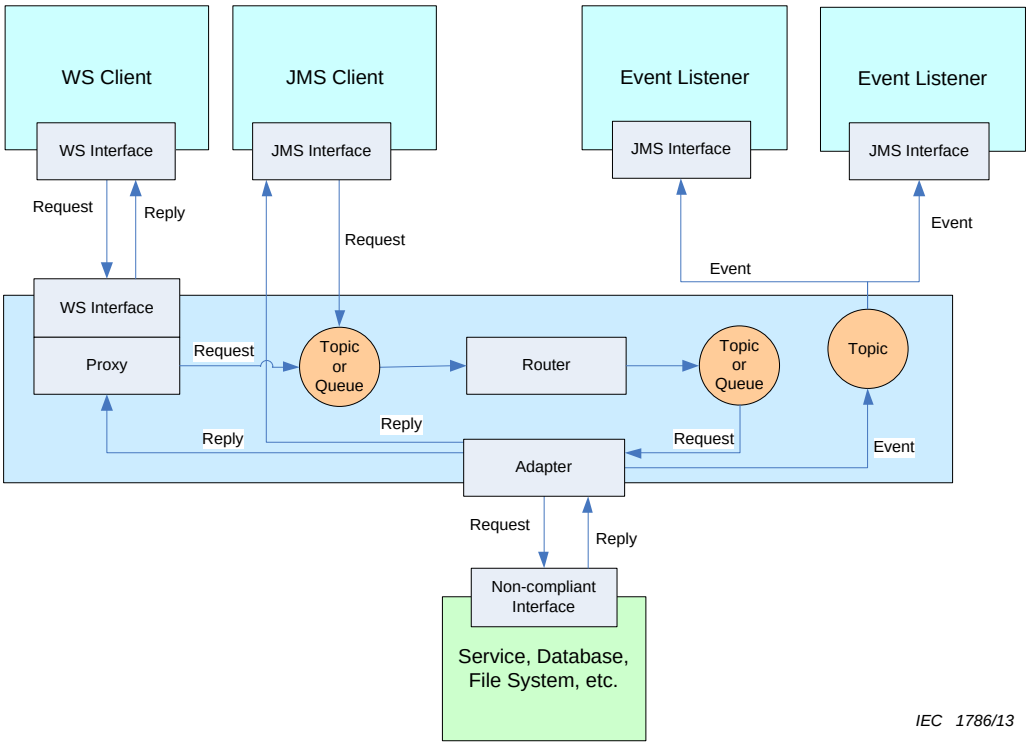


Figure 18 – ESB Integration to non-compliant resources

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 JBDC 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.

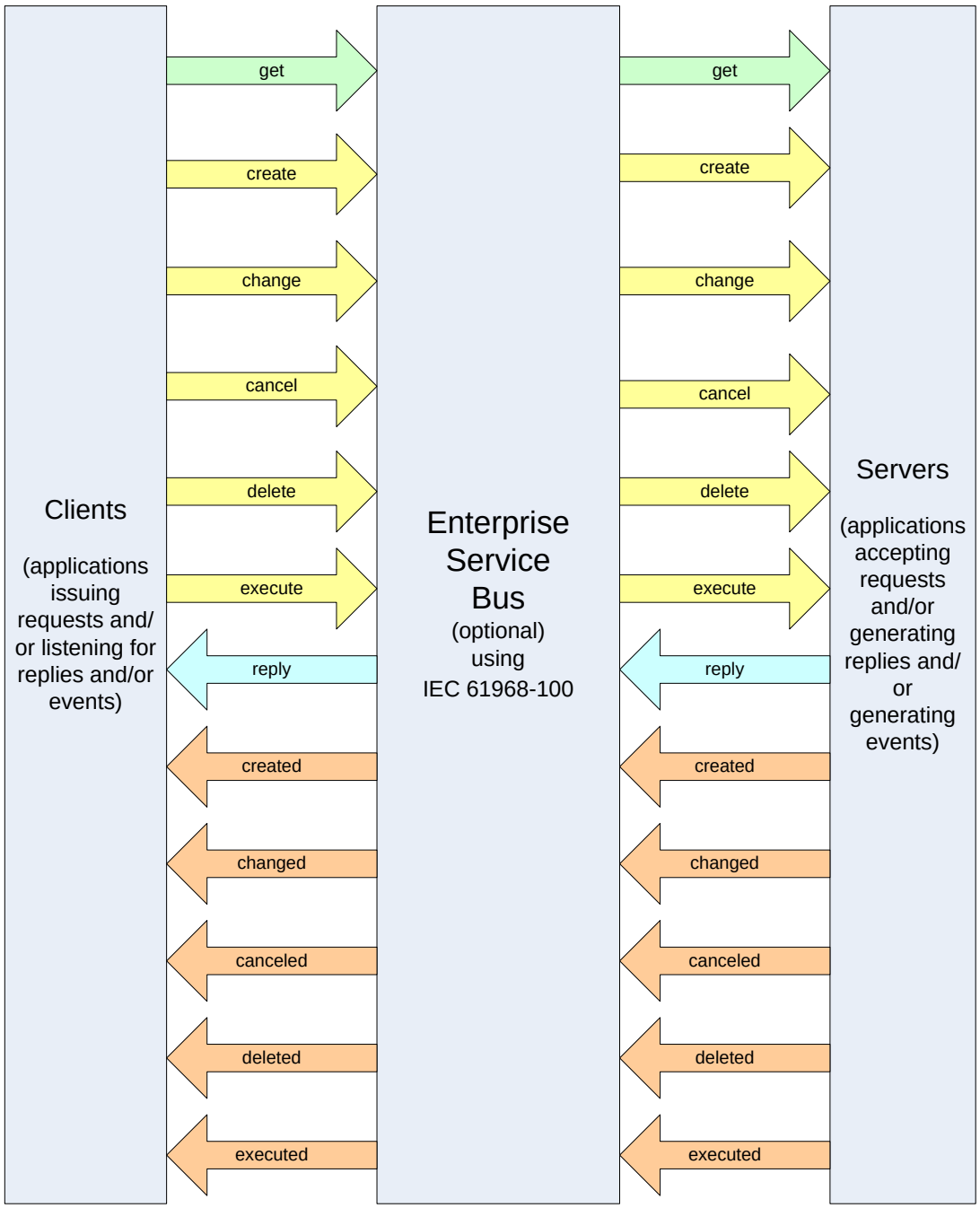


Figure 19 – Messaging between clients, servers and an ESB

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.

Table 1 – Verbs and their Usage

Request Verb	Reply Verb	Event Verb	Usage
get	reply	(none)	query
create	reply	created	transaction
change	reply	changed	transaction
cancel	reply	canceled	transaction
close	reply	closed	transaction
delete	reply	deleted	transaction
execute	reply	executed	transaction

The usage of request verbs are as follows.

- ‘get’ is used to query for objects of the type specified by the message noun
- ‘create’ is used to create objects of the type specified by the noun
- ‘delete’ is used to delete objects, although sometimes an object is never truly deleted in the target system in order to maintain a history
- ‘close’ and ‘cancel’ imply actions related to business processes, such as the closure of a work order or the cancellation of a control request
- ‘change’ is used to modify objects, but it is important to note that there can be ambiguities that need to be addressed through business rules, especially in the case of complex data sets (e.g. complex data sets typically have N:1 relationships and it is important to be clear when relationships are additive or are to be replaced by an update).
- ‘execute’ is used when a complex transaction is being conveyed using an OperationSet, which potentially contains more than one verb.

The response to each of the above requests uses the ‘reply’ verb. Event verbs are often the consequence of a request, where a ‘create’ may result in the generation of a ‘created’ event. The verbs used for events use the ‘past tense’ form of the associated request verb. There is no requirement that event be initiated through a request, as it may be appropriate for events to be generated independently of any specific request.

Validation and business rules may need to be defined for application of verbs in specific cases. This is in part true in that many rules are beyond the descriptive capabilities of UML and XML Schema.

It is also important to note that the enumerations for verbs in the standard Message XML Schema use the **lower case** form. The uppercase form is otherwise convenient for documentation purposes.

IEC 61968-1 previously identified verbs ‘update’, ‘updated’, ‘show’, ‘subscribe’, ‘unsubscribe’ and ‘publish’, all of which have been deprecated. The reason is that ‘show’ is a synonym for ‘reply’, and the verbs ‘subscribe’, ‘unsubscribe’ and ‘publish’ are functions that are performed within the transport layer (e.g. using JMS).

6.2.3 Nouns

Nouns are used to identify the type of the information being exchanged. These are also commonly called profiles. Each noun typically has a corresponding XML Schema definition defined using a namespace unique to each noun. Nouns are typically identified by use cases. Within a message, the noun is used to identify the type of the payload or the type of object to be acted or has been acted upon. Some common example nouns taken from IEC 61968-9 are:

- EndDeviceControls
- EndDeviceEvents

- MeterReadings

Nouns can be defined as needed to distinguish the contents of different information flows. They need not be defined as classes in a UML model, but instead the contents and structure of the noun are defined using classes, attributes and relationships from a UML model.

6.2.4 Payloads

Each noun identifies a payload structure that is typically conveyed using an XML document that conforms to an XML Schema. The structure of the payload is typically defined as a contextual profile from a UML model. This is the approach taken to define message structures by IEC 61968-9.

Figure 20 is an example payload structure that results from the contextual profile definition:

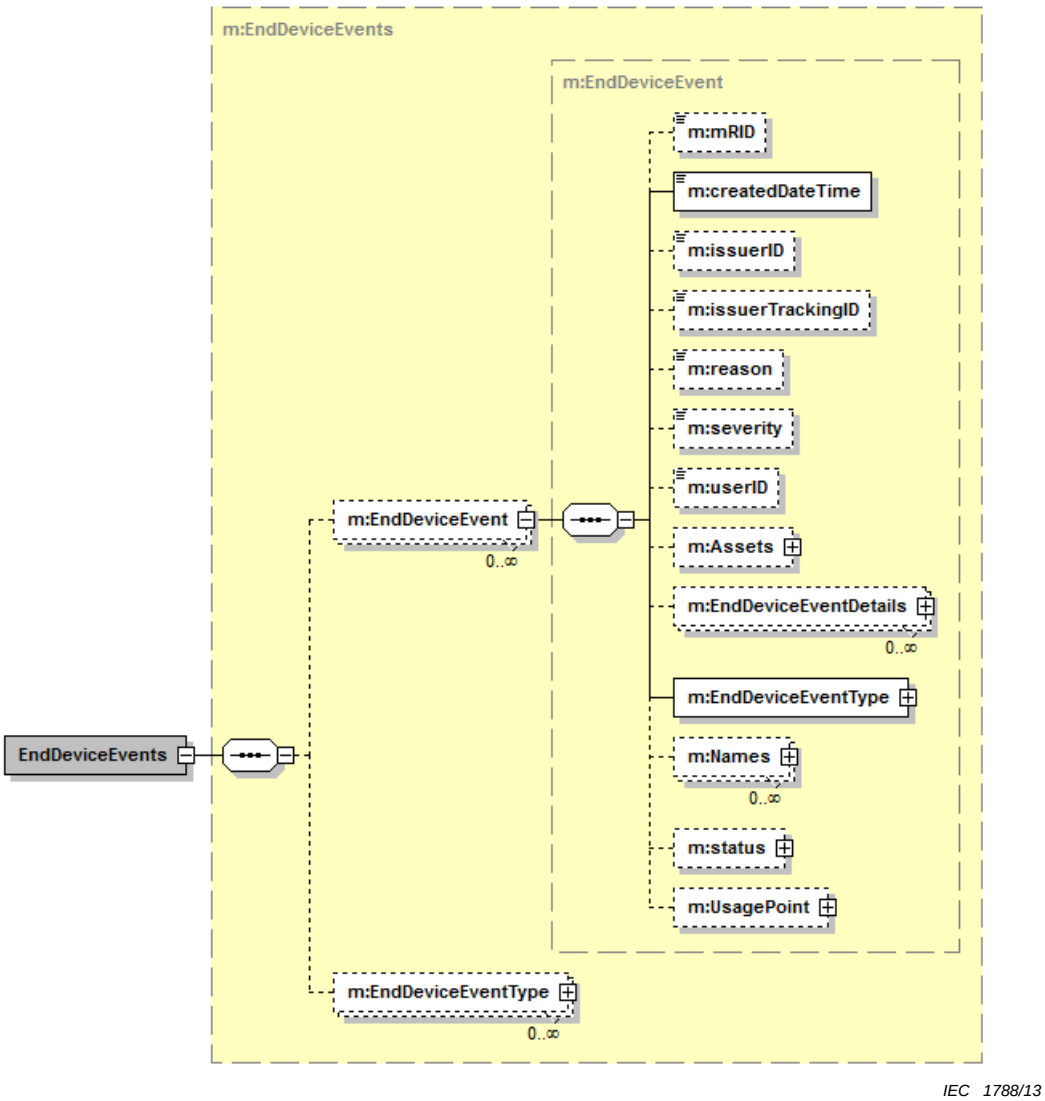


Figure 20 – Example payload schema

Depending upon the situation, a payload may or may not be required in a message. A message payload is required for the following cases:

- When issuing a ‘create’ request
- When issuing an ‘change’ request
- When issuing an ‘execute’ request

- In a reply message for a successful ‘get’ request
- For any event message (‘created’, ‘changed’, ‘deleted’, ‘closed’, ‘canceled’ or ‘executed’)

In cases where an event is a consequence of a transactional request and the noun of the request is the same as the noun of the event, the payload of the request is copied as the payload of the event.

A message payload should not be used in the following cases:

- In a reply message for an unsuccessful ‘get’ request in which no results are returned.
- In a reply to a ‘create’, ‘change’, ‘delete’, ‘close’, ‘cancel’ or ‘execute’ request
- In a ‘delete’, ‘close’ or ‘cancel’ request, since the ID of the object(s) is specified using the Request.ID elements
- In a ‘get’ request, as the parameters used to filter the request are supplied in the message Request element, optionally using a ‘Get’ profile in the Request.any element.

6.3 Common message envelope

6.3.1 General

Unless otherwise specified, all messages use a common message envelope (CME), where a predefined stereotype is used for requests and another stereotype is used for responses. There are also stereotypes for events and faults. This structure is based upon the IEC 61968-1 recommendations. Messages are constructed with several sections, including:

- Header: Required for all messages (except for fault response messages), using a common structure for all service interfaces.
- Request: optional, defining commonly used parameters needed to qualify ‘get’ query requests, or identify specific objects for ‘delete’, ‘cancel’ or ‘close’ requests. There is a provision to allow for inclusion of a complex structure using the Payload.any element. As an example, in the case of a request to get MeterReadings, a ‘GetMeterReadings’ profile can be defined to pass request qualifiers. In cases such as this, the profile should be named using the convention ‘Get<Noun>’. Not used for event or response messages.
- Reply: Required only for response messages to indicate success, failure and error details. Not used for request or event messages.
- Payload: Used to convey message information as a consequence of the ‘Verb’ and ‘Noun’ combination in the message Header. Required for ‘create’, ‘change’ and ‘execute’ requests. It is also required for event messages. Optional in other cases as described later in this document and specifically within annex B. The payload structure provides options for payload compression.

Figure 21 provides a generalize view of the high-level message structure:

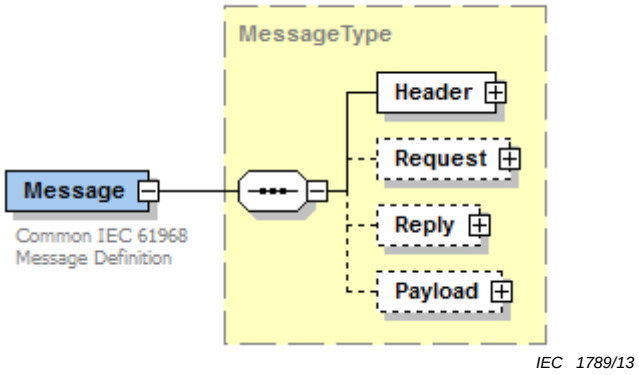


Figure 21 – Common message envelope

From this common message envelope there are four stereotypes which identify the specific subset of elements that are used for requests, responses, events and faults:

- RequestMessage
- ResponseMessage
- EventMessage
- FaultMessage

Where these stereotypes are useful when defining interfaces to clearly differentiate requests from responses from events from faults, it is a common practice to internally use the more generic Message structure within software such as common services and intermediaries.

6.3.2 Message header structure

Common to request, response and event messages is a header structure. The header currently has two required fields that must be populated, these include:

- Verb, to identify a specific action to be taken. There are an enumerated set of valid verbs, where commonly used values include ‘get’, ‘create’, ‘change’, ‘cancel’, ‘close’, ‘execute’ and ‘reply’. Within event notification messages ‘past tense’ verbs are used, which can include ‘created’, ‘changed’, ‘canceled’, ‘closed’ and ‘executed’. Implementations should treat deprecated verbs ‘update’ and ‘updated’ as synonyms to ‘change’ and ‘changed’.
- Noun: to identify the subject of the action and/or the type of the payload, such as MeterReadings, Notification, etc.

Field that can be optionally supplied include the following:

- Revision: To indicate the revision of the message definition. This should be ‘1’ by default.
- ReplayDetection: This is a complex element with a timestamp and a nonce used to guard against replay attacks. The timestamp is generated by the source system to indicate when the message was created. The nonce is a sequence number or randomly generated string (e.g. UUID) that would not be repeated by the source system for at least a day. This serves to improve encryption.
- Context: A string that can be used to identify the context of the message. This can help provide an application level guard against incorrect message consumption in configurations where there may be multiple system environments running over the same messaging infrastructure. Some example values are PRODUCTION, TESTING, STUDY and TRAINING.
- Timestamp: An ISO 8601 compliant string that identifies the time the message was sent. This is analogous to the JMSTimestamp provided by JMS. Either Zulu (‘Z’) time or time with a time zone offset may be used.

- Source: identifying the source of the message, which should be the name of the system or organization.
- AsyncReplyFlag: A Boolean (‘true’ or ‘false’) that indicates whether a reply message will be sent asynchronously. Replies are assumed to be sent synchronously by default.
- ReplyAddress: The address to which replies should be sent. This is typically used for asynchronous replies. This should take the form of a URL, topic name or queue name.This is analogous to the JMSReplyTo field provided by JMS. This is ignored when using one-way integration patterns (e.g. AckRequired=false). If the reply address is a topic, the topic name should be prefixed by ‘topic:’. If the reply address is a queue, the queue name should be prefixed by ‘queue:’. If the reply address is a web service, the reply address should be a URL beginning with ‘http://’ or ‘https://’.
- AckRequired: This is a Boolean (‘true’ or ‘false’) that indicates whether or not an acknowledgement is required. If false, this would indicate that a one-way integration pattern is being used for communicating transactional messages.
- User: A complex structure that identifies the user and associated organization. Should be supplied as it may be required for some interfaces, depending upon underlying implementations. This allows a UsersID string and optional Organization string as sub-elements.
- MessageID: A string that uniquely identifies a message. Use of a UUID or sequence number is recommended. This is analogous to the JMSMessageID provided by JMS. A process should not issue two messages using the same MessageID value.
- CorrelationID: This is used to ‘link’ messages together. This can be supplied on a request, so that the client can correlate a corresponding reply message. The server will place the incoming CorrelationID value as the CorrelationID on the outgoing reply. If not supplied on the request, the CorrelationID of the reply should be set to the value of the MessageID that was used on the request, if present. This is analogous to the used of the JMSCorrelationID provided by JMS. Given that the CorrelationID is used to ‘link’ messages together, it may be reused on more than one message. Use of a UUID or sequence number is recommended.
- Comment: Any descriptive text, but shall never be used for any processing logic.
- Property: A complex type that allows custom name/value pairs to be conveyed. The source and targets would need to agree upon usage. These are analogous to a Property as defined by JMS.
- any: Can be used for custom extensions.

Figure 22 describes the header structure used for request, response and event messages.

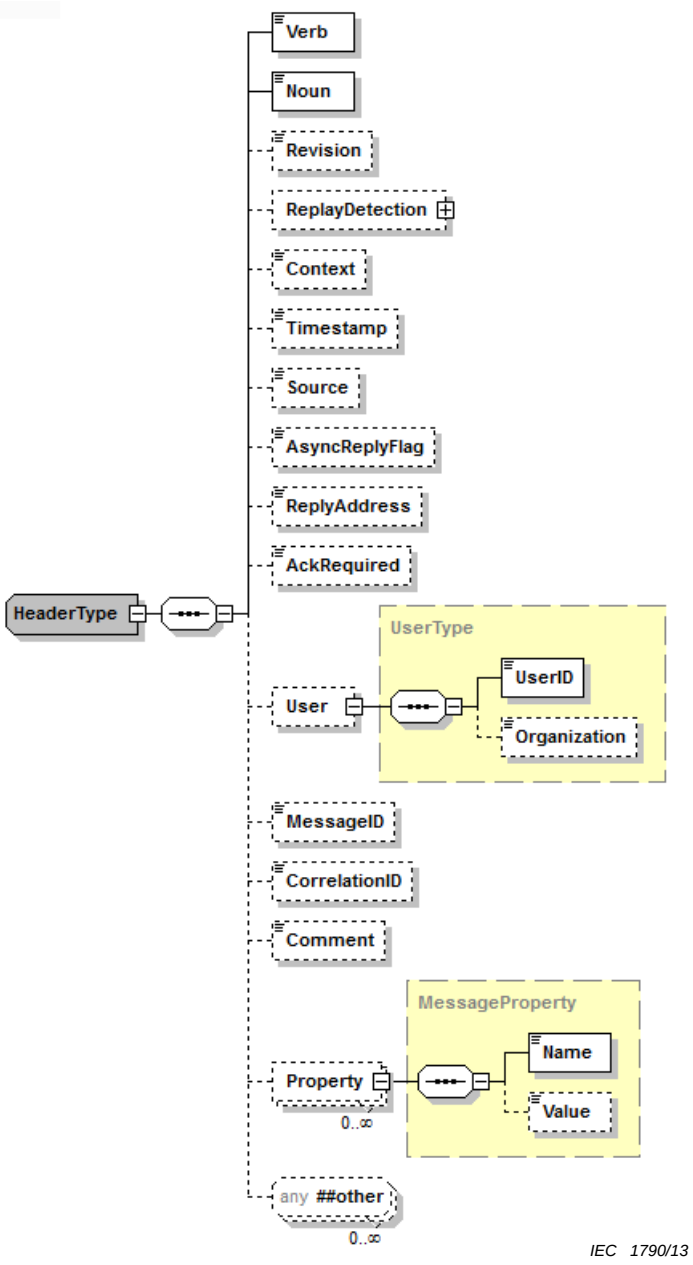


Figure 22 – Common message header structure

Where there are only two required elements, verb and noun, there are many optional elements that may be populated. In Figure 22, the optional items are represented using dashed borders.

The following is an XML example for a message that populates all header fields.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestMessage xsi:schemaLocation="http://iec.ch/TC57/2011/schema/message
Message.xsd" xmlns="http://iec.ch/TC57/2011/schema/message"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <Header>
              <Verb>get</Verb>
              <Noun>LoadForecast</Noun>
              <Revision>1</Revision>
              <ReplayDetection>
                     <Nonce>dcd98b7102dd2f0e8b11d0f600bfb0c093</Nonce>
                     <Created>2012-12-16T09:30:47.0Z</Created>
              </ReplayDetection>
              <Context>PRODUCTION</Context>
```

```
                <Timestamp>2001-12-16T09:30:47.0Z</Timestamp>
                <Source>EMS</Source>
                <AsyncReplyFlag>false</AsyncReplyFlag>
                <ReplyAddress>queue:EMS.ReplyQueue</ReplyAddress>
                <AckRequired>true</AckRequired>
                <User>
                        <UserID>Bob</UserID>
                        <Organization>Scheduling</Organization>
                </User>
                <MessageID>3432626</MessageID>
                <CorrelationID>3432626</CorrelationID>
                <Comment>Example message</Comment>
                <Property>
                        <Name>timeout</Name>
                        <Value>10</Value>
                </Property>
        </Header>
        <Request>
                <StartTime>2012-12-17T00:00:00.0Z</StartTime>
                <EndTime>2012-12-17T24:00:00.0Z</EndTime>
        </Request>
</RequestMessage>
```

In cases where the message is conveyed using a transport such as SOAP or JMS, there is some redundancy between the optional fields in the message envelope and the transport-level header. In these cases, both fields can simply be set to the same value. In cases where they are different, they shall be used as appropriate for the transport-level and application-level message envelope.

6.3.3 Request message structures

Figure 23 describes the structure of a request message that would be used in conjunction with a message or WSDL operation.

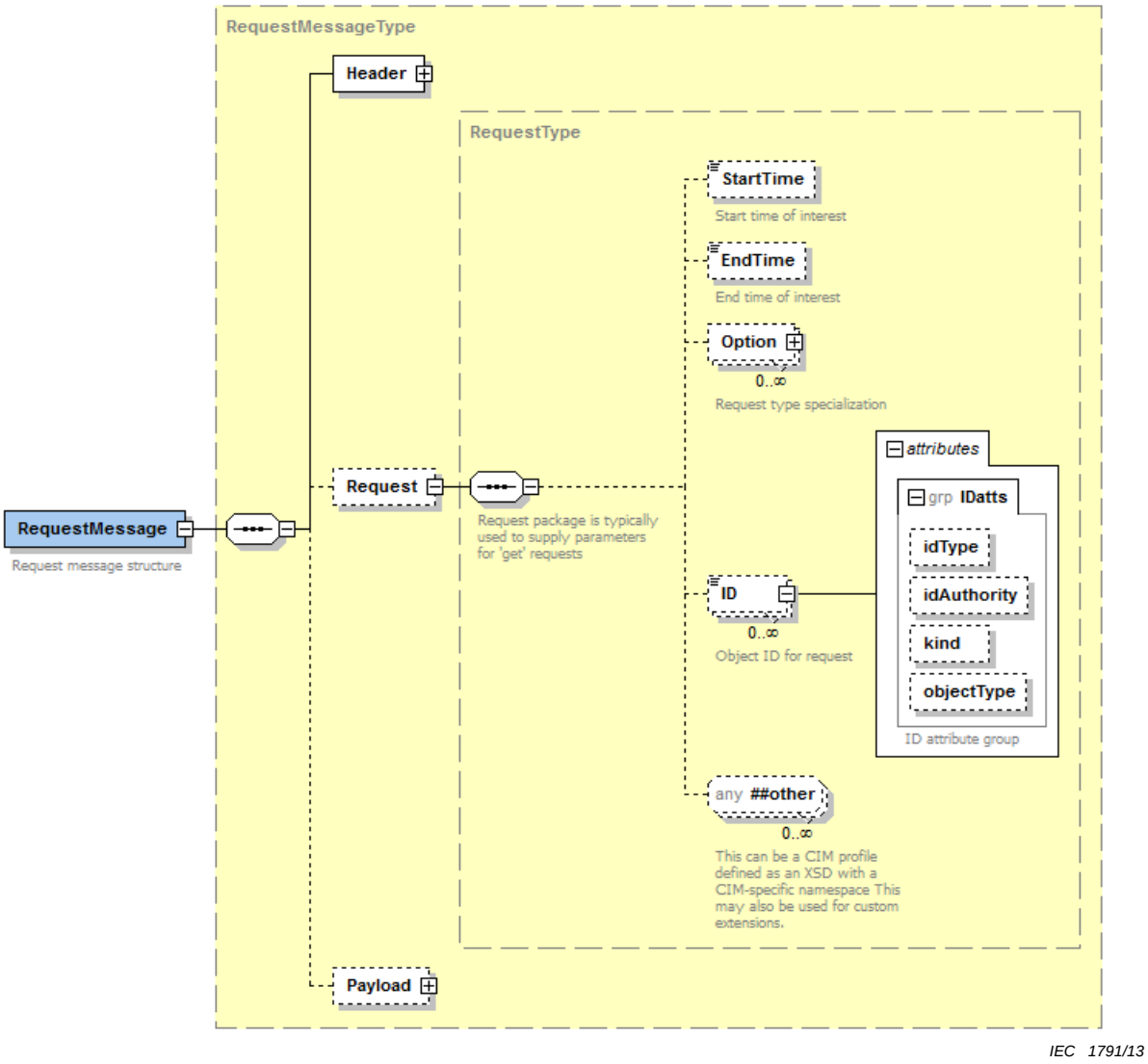


Figure 23 – Request message structure

The RequestMessage can also optionally contain an element with parameters relevant to the request, called Request. One key use of the RequestType is to avoid the placement of application specific request parameters in the header or within payload definitions.

There are no required elements in the Request element. The usage of elements within the Request element is described as follows:

- StartTime: Used when a query needs to specify a start time as a filter, but no such parameter is provided in a ‘Get’ profile. If both exist, this will be ignored.
- EndTime: Used when a query needs to specify an end time as a filter, but no such parameter is provided in a ‘Get’ profile. If both exist, this will be ignored.
- Option: Used when name/value pairs are useful in filtering a query or to convey general or custom request options. Examples of usage are the specification of a transaction timeout value or specifying a response mode such as ‘Aggregated’ or ‘Streaming’. At the current time there are no normative enumerations for these values.
- ID: Used when the ID of one or more objects are needed to filter a query request. Can also be used to identify specific objects in the case of ‘delete’, ‘cancel’ or ‘close’ transactions. Each ID can specify attributes, first to identify the kind of ID, which can be name, uuid, transaction or other. The default of uuid is used for mRID values. If a name, the idType and idAuthority can be specified.

- any: Used to supply a ‘Get<Noun>’ profile (e.g. GetMeterReadings) element to be conveyed with parameters that can qualify a request. Can also be used for other non-standard extensions. In cases where a ‘Get’ profile is used, the elements defined within the ‘Get’ profile take precedence over the StartTime, EndTime and ID elements. This recognizes the asymmetry between the information needed to qualify a request from the information that is returned on a reply.

Situations that may use the Option Name/Value pair can be described as a part of other standards. In some cases it may be decided that these should require changes to other standard request elements (i.e. the Get<Noun> elements described in Figure 25) in order to facilitate such a request.

Figure 24 gives an example of a RequestMessage where the Request.ID elements are used to identify objects of interest:

```
<ns0:RequestMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>get</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:Revision>1</ns0:Revision>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Request>
   <ns0:ID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</ns0:ID>
   <ns0:ID>e6d957ba-792a-4fcf-9f33-fd176a66dee8</ns0:ID>
   <ns0:ID>567fdc86-0ccd-4a96-a318-bdc1a3015643</ns0:ID>
</ns0:Request>
</ns0:RequestMessage>
```

IEC 1792/13

Figure 24 – XML for example RequestMessage

The ‘any ##other’ element should be used when more complex request parameters are needed in order to qualify a request so that the resulting response message is appropriately filtered. Figure 25 is an example of a ‘GetMeterReadings’ element that is used to provide qualifiers for get MeterReadings requests.

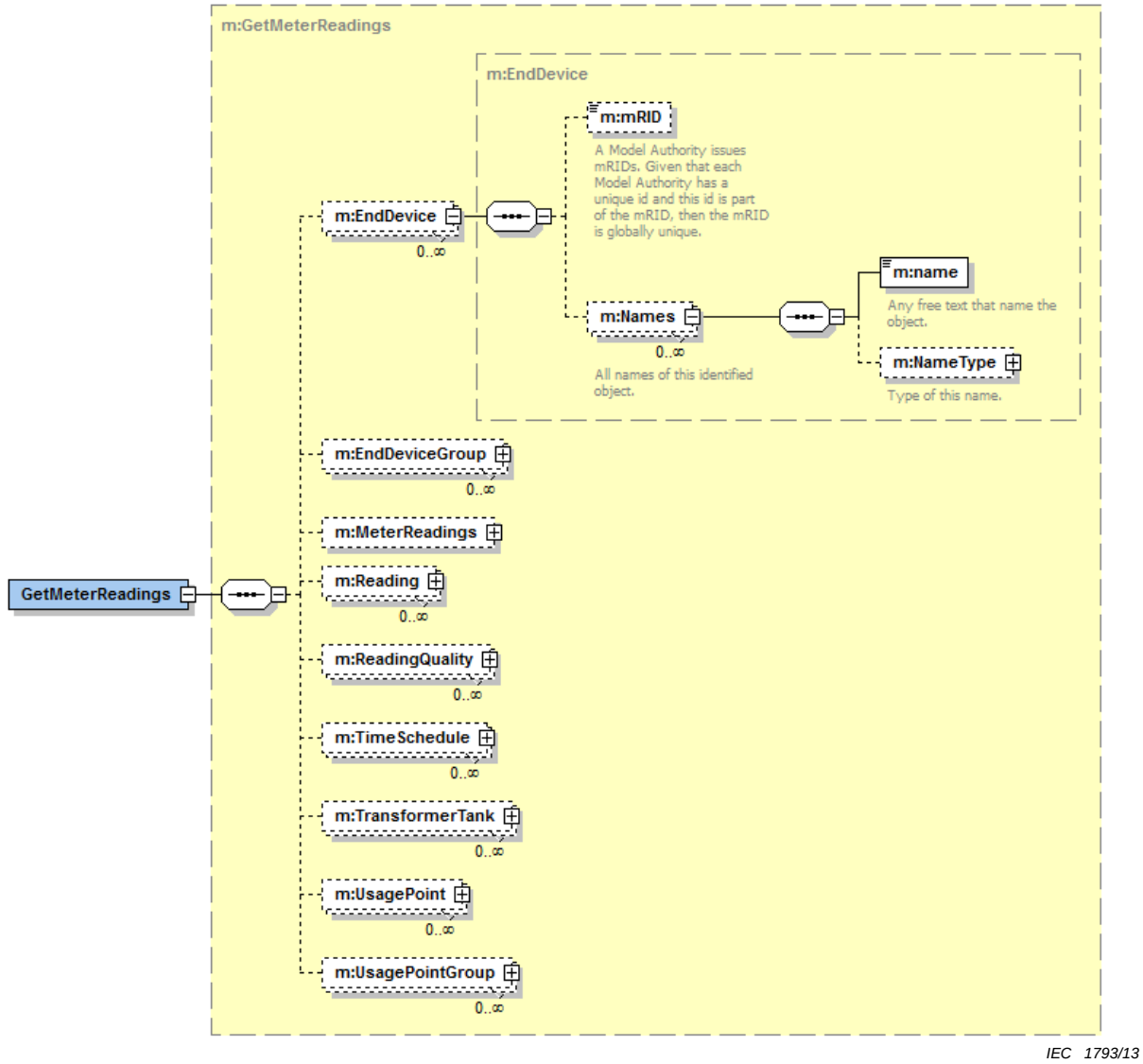


Figure 25 – Example 'Get<Noun>' profile

6.3.4 Response Message Structures

Figure 26 describes the structure of a response message that would be used in conjunction with a message or WSDL operation, as a response to the request message.

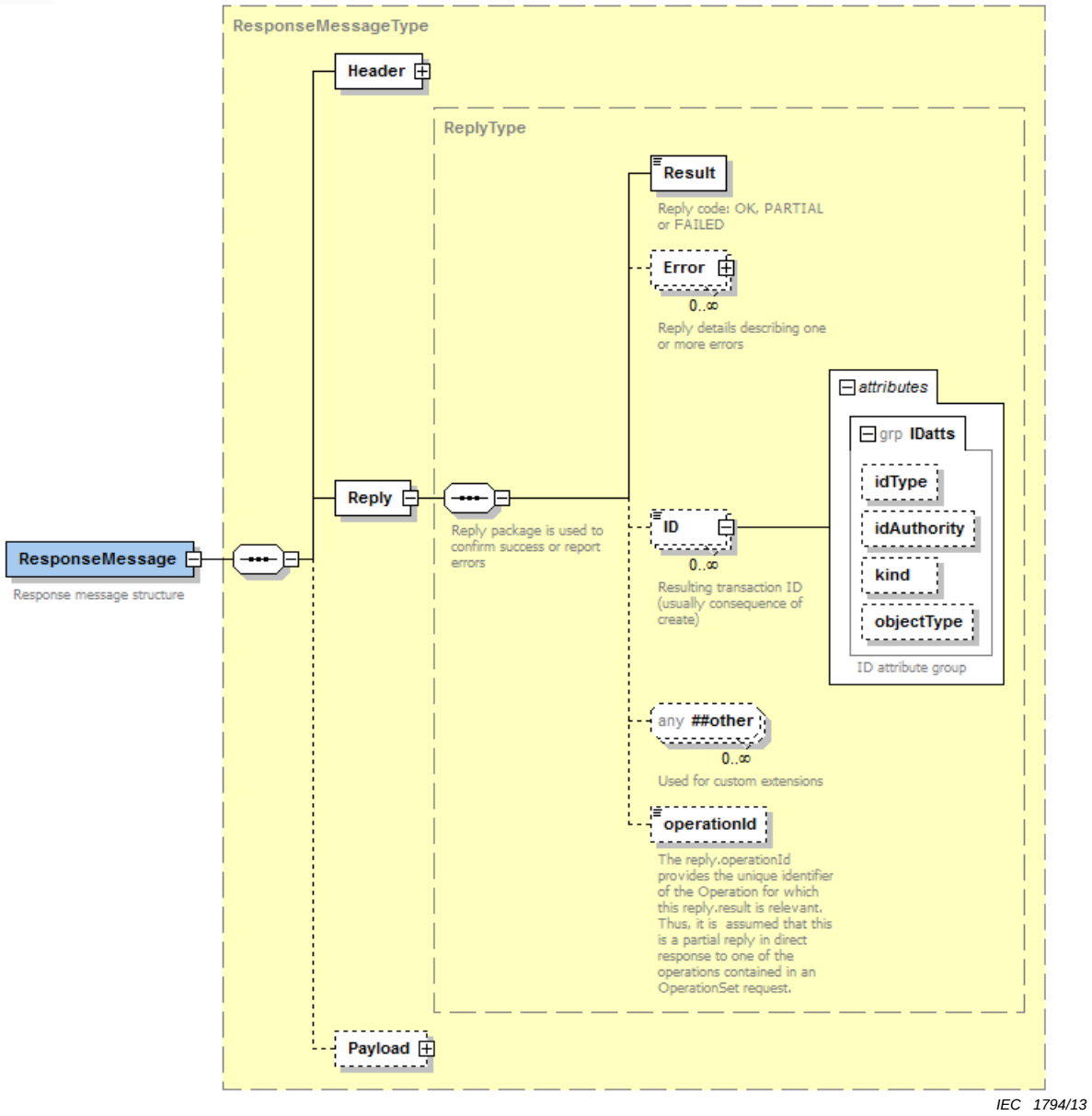


Figure 26 – ResponseMessage structure

The Reply.result value is enumerated in Message.xsd, and would be populated in the following manner:

- "OK" if there are no errors and all results have been returned. There is no requirement that a Reply.Error element be present.
- "PARTIAL" if only a partial set of results has been returned, with or without errors. Existence of errors is indicated with one or more Reply.Error.code elements.
- "FAILED" if no result can be returned due to one or more errors, indicated with one or more Reply.Error elements, each with a mandatory application level ‘code’ ‘.

This is represented by the state transition diagram shown in Figure 27.

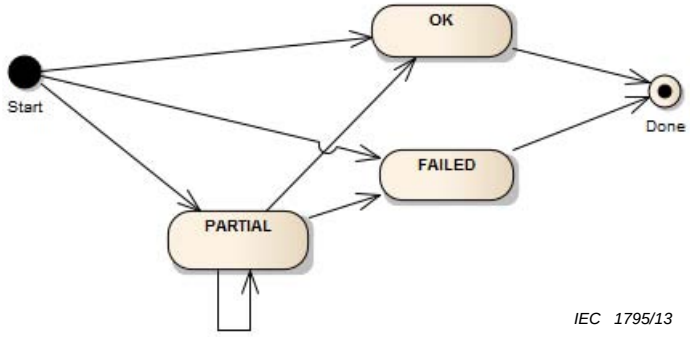


Figure 27 – Reply message states

There may also be more specific error information provided within the payload itself. Figure 28 describes the details of the Error element, which allows for both human and machine readable error messages. The structure allows the use of XPath to allow specific references to elements within an XML document.

If the entire response to a request message is being provided in a single response message and the response message contains no fatal errors, then the Reply.Result is set to "OK" and the Reply.Error.code is set with a value of "0.0".

Otherwise, if the entire response to a request message is being returned in a single message and the response message contains at least one fatal error then the Reply.Result is set to "FATAL". Such a message may contain a mixture of data items and error notifications. The Reply.Error.code, Reply.Error.ID, and other associated Reply.Error structure attributes are then set appropriately for each fatal error or informational condition being reported.

If the responding system is sending multiple response messages to a request message the Reply.Result is set to "PARTIAL". Such messages may contain a mixture of data items and error notifications. There shall be at least one Reply.Error.code of "0.2" or "0.1", depending upon whether a given response message is the last in a sequence or not (*). The Reply.Error.code, Reply.Error.ID, and other associated Reply.Error structure attributes are then set appropriately for each fatal error or informational condition being reported.

In the case where the responding system cannot determine last message in a set of response messages, then all messages in the set are to be sent with a Reply.Error.code = "0.1".

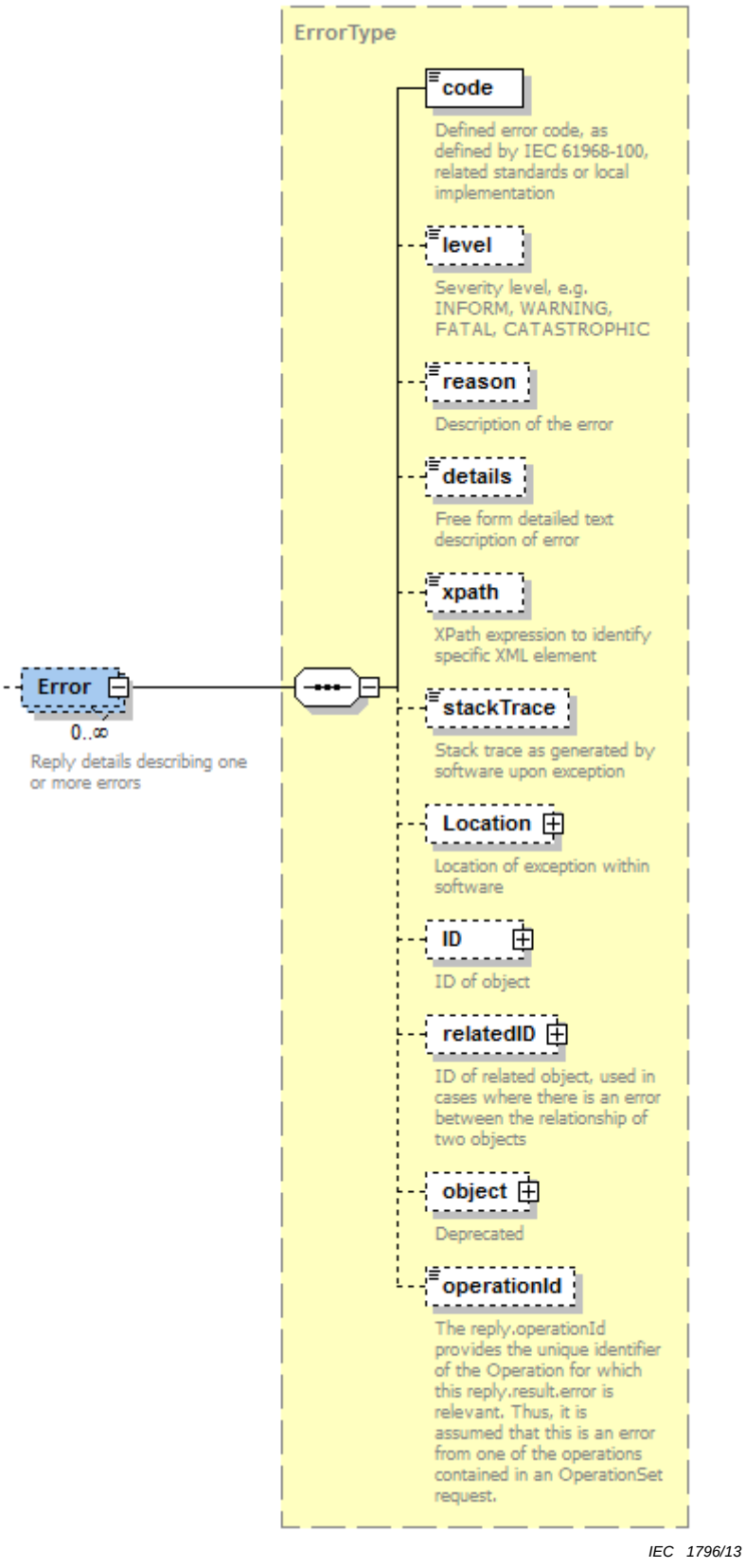


Figure 28 – Error structure

Figure 29 is an example of a ResponseMessage:

```
<ns0:ResponseMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>reply</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Reply>
   <ns0:Result>OK</ns0:Result>
</ns0:Reply>
<ns0:Payload>
   <m:Switches xsi:schemaLocation="http://iec.ch/TC57/2012/Switches# Switches.xsd"
xmlns:m="http://iec.ch/TC57/2012/Switches#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <m:Switch>
         <m:mRID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>e6d957ba-792a-4fcf-9f33-fd176a66dee8</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>567fdc86-0ccd-4a96-a318-bdc1a3015643</m:mRID>
         <m:normalOpen>false</m:normalOpen>
      </m:Switch>
   </m:Switches>
</ns0:Payload>
</ns0:ResponseMessage>
```

IEC 1797/13

Figure 29 – XML for example ResponseMessage

Figure 30 is an example of a ResponseMessage where the payload is compressed:

```
<ns0:ResponseMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>reply</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Reply>
   <ns0:Result>OK</ns0:Result>
</ns0:Reply>
<ns0:Payload>
   <ns0:Compressed>dghuywqeiwihn353218u23hb2b3b3bhu</ns0:Compressed>
   <ns0:format>XML</ns0:format>
</ns0:Payload>
</ns0:ResponseMessage>
```

IEC 1798/13

Figure 30 – XML example of payload compression

Figure 31 is an example ResponseMessage that returned an error:

```
<ns0:ResponseMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>reply</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:Revision>1</ns0:Revision>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Reply>
   <ns0:Result>FAILED</ns0:Result>
   <ns0:Error>
      <ns0:code>2.15</ns0:code>
      <ns0:level>WARNING</ns0:level>
      <ns0:details>Unknown object: e6d957ba-792a-4fcf-9f33-fd176a66dee8</ns0:details>
   </ns0:Error>
</ns0:Reply>
<ns0:Payload>
   <m:Switches xsi:schemaLocation="http://iec.ch/TC57/2012/Switches# Switches.xsd"
xmlns:m="http://iec.ch/TC57/2012/Switches#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <m:Switch>
         <m:mRID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>567fdc86-0ccd-4a96-a318-bdc1a3015643</m:mRID>
         <m:normalOpen>false</m:normalOpen>
      </m:Switch>
   </m:Switches>
</ns0:Payload>
</ns0:ResponseMessage>
```

IEC 1799/13

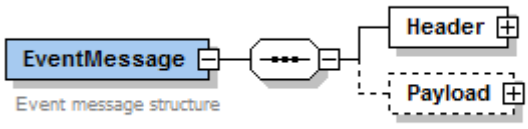
Figure 31 – XML example for error ResponseMessage

An important advantage of payload compression over the use of SOAP attachments is for signing, as a SOAP signature does NOT sign the contents of the attachment, only the message body. Using payload compression the signature covers the payload, providing for non-repudiation.

6.3.5 Event message structures

An EventMessage is typically published to report a condition of potential interest. The verbs used in an event message are past tense, e.g. created, changed, canceled, etc. An EventMessage will not include request or reply parameters, just a header and usually a payload.

Figure 32 describes the structure of an EventMessage.



IEC 1800/13

Figure 32 – EventMessage structure

Figure 33 is an example of an EventMessage:

```
<ns0:EventMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>changed</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:Revision>1</ns0:Revision>
</ns0:Header>
<ns0:Payload>
   <m:Switches xsi:schemaLocation="http://iec.ch/TC57/2012/Switches# Switches.xsd"
xmlns:m="http://iec.ch/TC57/2012/Switches#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <m:Switch>
         <m:mRID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</m:mRID>
         <m:normalOpen>false</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>567fdc86-0ccd-4a96-a318-bdc1a3015643</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
   </m:Switches>
</ns0:Payload>
</ns0:EventMessage>
```

IEC 1801/13

Figure 33 – XML example for EventMessage

NOTE In an EventMessage the ‘verb’ will be past tense, e.g. ‘created’, ‘changed’, ‘canceled’, etc.

6.3.6 Fault message structures

A FaultMessage is typically used within the definition of a WSDL and implemented by a web service to report a fault condition as a consequence of a failed attempt to process a RequestMessage (e.g. detection of a SOAP fault). It only uses a reply element (i.e. no header), as it may not have been able to interpret even the header of the RequestMessage. The Fault Message Structure is shown in Figure 34.

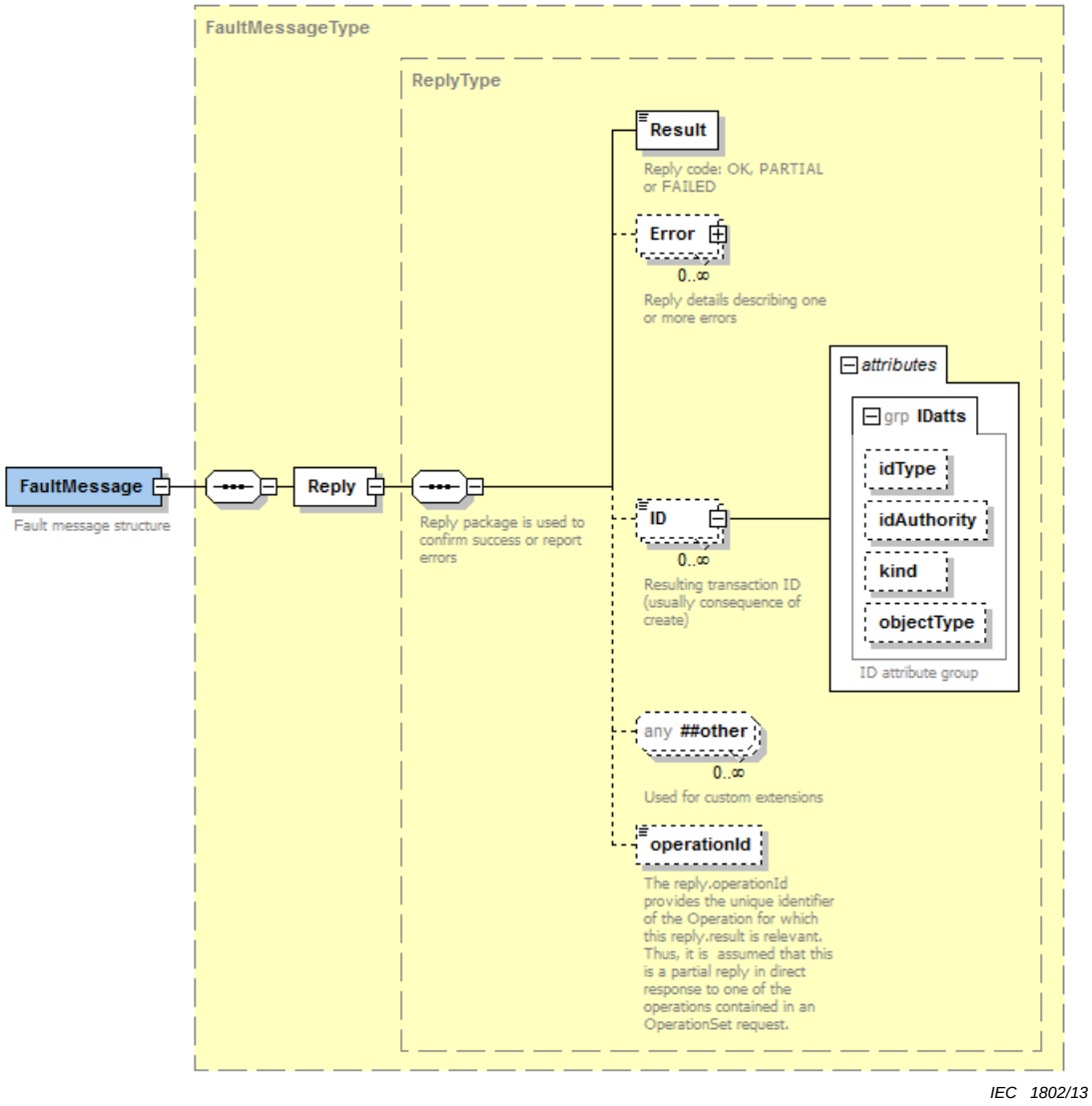


Figure 34 – Fault message structure

6.4 Payload structures

Subclause 6.4 describes two forms of payload structures: generic and type-specific. Where the common message envelope defines the payload as being generic (or type-independent), which is the common usage with both JMS and generic web services. However, the definition of WSDLs for strongly-typed web services may require a payload definition of a variant message envelope that is type-specific.

There are some types of messages where a Payload must be provided, as would be the case for a request message with a verb of ‘create’ or ‘change’, some response messages and some event messages. Payloads typically contain XML documents that conform to a defined XML schema. However, there are exceptions to this rule. Some XML payloads could potentially not have useful XML schemas, as in the case of RDF files or dynamic query results, as well as non-XML formats such as CSV and PDF.

There may also be cases where a large payload must be compressed, in the event that it would become very large and otherwise consume significant network bandwidth. In order to accommodate a variety of payload format options the generic payload structure shown in Figure 35 is used.

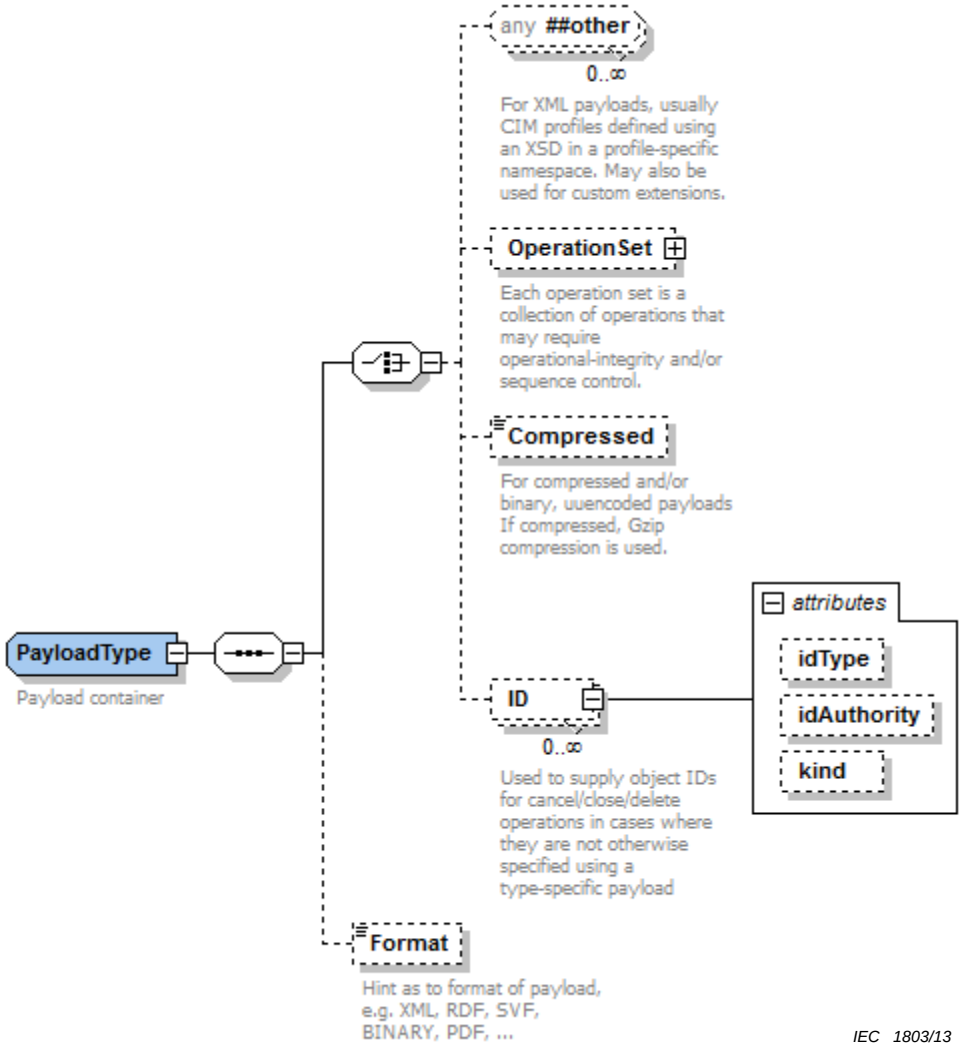


Figure 35 – Message payload container – Generic

Subclause 6.10 discusses the use of the OperationSet element for complex transactions in more detail.

When the generic message payload container is used, any type of XML document may be included, using the XML 'any' structure. While this provides options for loose-coupling, specific complex types defined by XML schemas (XSDs) can be used as well.

There are also some cases where a zipped, base64 encoded string is necessary, and would be passed using the 'Compressed' tag within the message. The base64 encoding must always be performed after compression. The 'Compressed' element is used even in caes where binary data is not compressed. The Gnu Zip compression shall be used in order to provide compatibility within both Java and Microsoft .Net implementations. A Java example is provided in Annex F. Specific examples of the usage of payload compression would be where:

- An XML payload, conforming to a recognized XML schema exceeds a predefined size (e.g. 1 MB, 5 MB, 10 MB, etc.). This would be common for model exchanges and energy market transactions.
- A payload has a non-XML format, such as PDF, Excel spread sheet, CSV file or binary image.

- A payload is formatted using XML, but has no XML schema and exceeds a predefined size (e.g. 1 MB, 5 MB, 10 MB, etc.). An example of this would be the case of dynamic XML generated as a consequence of a SQL XML query that would return an XML result set

When a payload is compressed and base64 encoded, it is stored within the Payload/Compressed message element as a string. Additionally, in order to support efficient transfer of binary formatted data, data can be base64 encoded but not compressed. This would be used for data classified as being ‘high speed’, where XML formatting would not meet performance needs.

The ID element and associated attributes can be used to supply object or transaction identifiers. This is useful in cases where a payload does not otherwise provide object identifiers as may commonly be the case for cancel, close or delete requests, responses to create requests or events using the canceled, closed or deleted verbs.

The Format element can be used to identify specific data formats, such as XML, RDF, SVG, BINARY, PDF, DOC, CSV, etc. This is especially useful if the payload is base64 encoded and potentially compressed. The use of this tag is optional, and would typically only be used when the payload is stored using the Payload/Compressed message element. Table 2 describes the relationships between elements in the Payload, showing a wide variety of payload options.

Table 2 – Payload usages

Any	*OperationSet*	*Compressed*	*Format*	*Interpretation*
XML	null	null	Null or 'XML'	Message payload is contained within the 'any', where the contents are described by the header Noun. This is the most common usage.
null	null	string	'XML'	Message payload is gzipped and base64 encoded in the Compressed element, where the contents are described by the header Noun.
null	XML	null	Null or 'XML'	Complex transaction is being conveyed using the OperationSet element, where the header Verb is 'execute' or 'executed'
XML	null	null	'RDF'	Message payload is an RDF document as conveyed using the 'any' element
null	null	string	'RDF'	Message payload is a compressed RDF document as conveyed in the Compressed element
null	null	string	'PDF'	Message payload is a compressed PDF document as conveyed in the Compressed element
null	null	string	'GZIP'	Message payload is a gzip archive of one or more files as conveyed in the Compressed element
null	null	string	'CSV'	Message payload is a CSV file being conveyed in the Compressed element
null	null	string	'XLS'	Message payload is an Excel file being conveyed in the Compressed element
null	null	string	'DOC'	Message payload is a Word document being conveyed in the Compressed element
null	null	string	'TEXT'	Message payload is a compressed text document as conveyed in the Compressed element
null	null	string	'JSON'	Message payload is a compressed JSON object as conveyed in the Compressed element
null	null	string	'BINARY'	Message payload is a binary structure that has been base64 encoded but not compressed. Any further aspects are application specific.
null	null	string	*other*	Message payload is a compressed file of some 'other' format as conveyed in the Compressed element. This allows the definition of custom Format values.

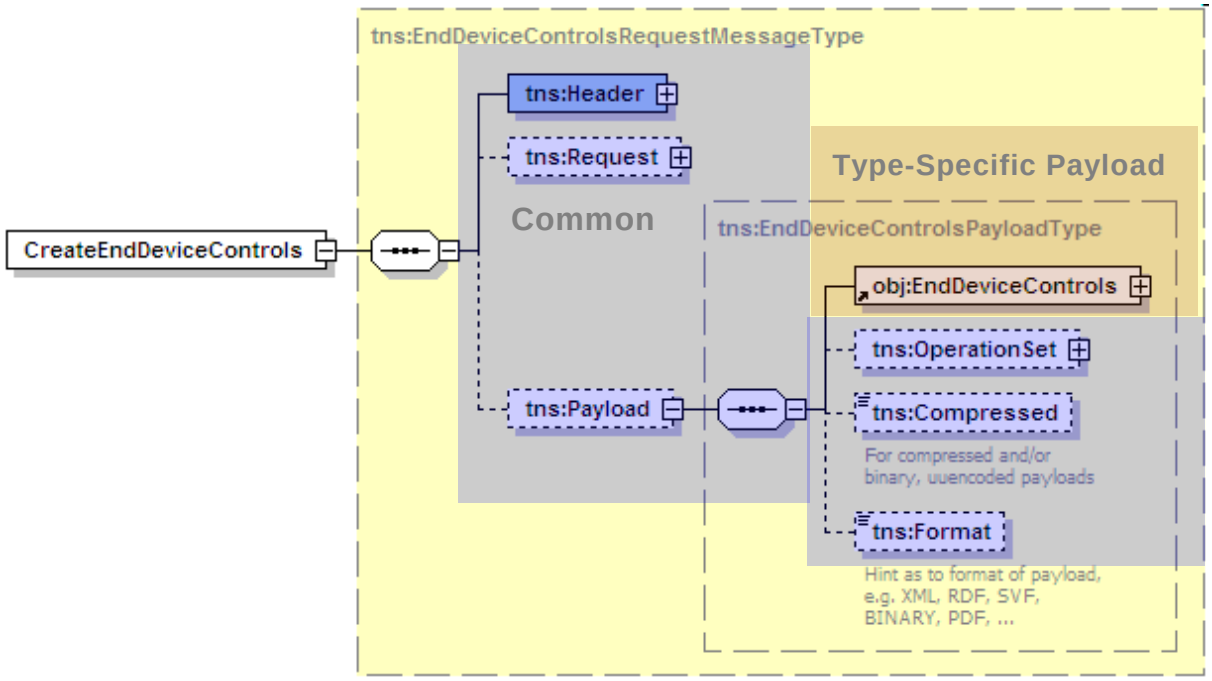
These payload options provide an alternative to the use of SOAP attachments. SOAP attachments are more difficult to secure since the SOAP envelope signature signs the SOAP body but does not sign the attachment. This also requires that the payload is processed separately from the rest of the SOAP message (e.g. the message is parsed to extract the payload, and then the payload is parsed and processed). However, we believe this implementation approach is less complex than using SOAP attachments.

6.5 Strongly-typed payloads

In cases where strongly-typed WSDLs are to be defined with operations specific for combinations of verb and noun, the common message envelope is redefined with the following two substitutions:

- Root element “Message” replaced with a WSDL operation name such as “CreateEndDeviceControls”. This address a Web service “wire signature” issue if multiple operations reference a same XSD element.
- Payload contains a concrete message type element such as EndDeviceControls.

As a result, a message structure for CreateEndDeviceControl request service operation is then redefined as shown in Figure 36:



IEC 1804/13

Figure 36 – Message payload container – Type specific example

NOTE The Message element is renamed to CreateEndDeviceControls for this service definition, and the Payload element contains a concrete object “EndDeviceControls” which is based on a CIM profile (or payload type). A key difference to be noted is the strong typing of the payload element, as opposed to use of the ‘any’ in the standard Message.xsd. This results in a type-specific version of Message.xsd per each IEC 61968 profile in use. See section 8.3 and annex C for further details.

6.6 SOAP message envelope

SOAP has been widely used as a standard protocol specification for exchanging XML information using web services. It provides an envelope that contains a header and a body. How a SOAP message is structured can be defined in WSDL binding section as an example listed in Figure 37:

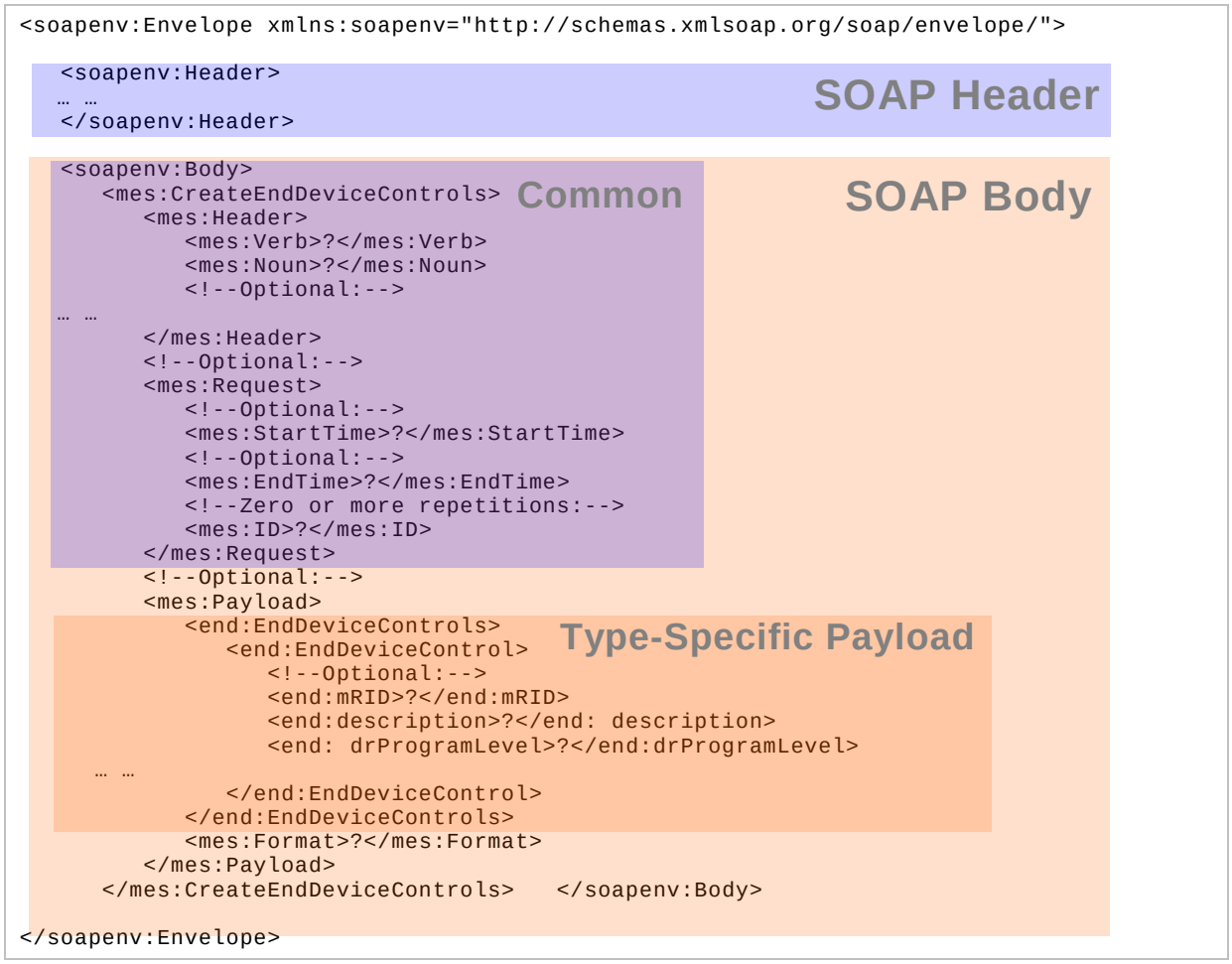
```
<wsdl:binding name="EndDeviceControl_Binding" type="tns:EndDeviceControl">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="CreatedEndDeviceControl">
      <wsdl:documentation>CreatedEndDeviceControl binding</wsdl:documentation>
<soap:operation
soapAction="http://iec.ch/TC57/2010/EndDeviceControls/CreatedEndDeviceControls"
style="document"/>
      <wsdl:input name="CreatedEndDeviceControlRequest">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CreatedEndDeviceControlResponse">
         <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="CreatedEndDeviceControlFault">
         <soap:fault name="CreatedEndDeviceControlFault" use="literal"/>
      </wsdl:fault>
   </wsdl:operation>
   …
```

IEC 1805/13

Figure 37 – SOAP bindings

In this WSDL, it is specified that where an input / output payload is located (soap:body – highlighted) and what binding style (=document – highlighted) it follows.

Based on the WSDL binding information, a SOAP message (see below) can be constructed. When using SOAP the message structure will appear within the context of the SOAP Body. This is shown in Figure 38.



IEC 1806/13

Figure 38 – SOAP envelope example for strong typing

6.7 Request processing

A request message is sent from a client to a service to initiate a query or transaction, where a response message is typically expected. The basic sequence of request processing is as follows:

1) Client constructs a request message using a common message envelope and specifying a verb and a noun. The noun identifies the type of the payload
2) The client sends the request message to the appropriate service interface. This can use transport technologies such as JMS or web services. The ESB implementation can transparently use intermediaries such as proxies, routers and adapters to transmit the request to the appropriate service instance.
3) The server accepts the message.
4) If the request message is invalid (e.g. incomplete, XML not well formed, etc.), a fault message may be returned to the client and processing terminates.
5) The service looks at the verb and noun combination, determining if the request can (or should) be processed, if not an error response message is sent to the client and

processing terminates. This step can also consider checks for authentication and authorization.

6) The service performs the desired processing, parsing the payload as needed. The service may parse the payload using an appropriate XML schema (as would define the payload type described by the message noun), using XPath expressions, using XSL transformations or other mechanisms. The service may also consider parameters provided in the message header and request packages for processing.

7) A response message is constructed, where a payload is rendered of the type identified by the noun as needed. This would commonly be the case in response to a 'get' request. The message reply element should be used to convey either a Result of 'OK' or an error code as appropriate.

8) The response message is returned to the client. This can use web services, JMS or other transport technologies as expected by the client.

9) The client processes the response message, parsing the payload as needed. The client should examine the reply element of the message to see if the request was successful (i.e. Result='OK') or encountered one or more errors.

10) Processing is completed

It is important to note that there are a variety of failure scenarios that can occur between steps 2 to 8, where a client should be able to handle a fault or time out when waiting for the reply to a request.

6.8 Event processing

An event message is a message that is published by a service (or more generally any event publisher) to potentially many listeners. Events may also be referred to as 'notifications'. Event listeners are consumers that have subscribed to one or more JMS topics of potential interest. In the case of web services, there would be an intermediary to send the events to subscribers listening via web services.

The basic sequence of event processing is as follows:

1) A service constructs an event message using a common message envelope and specifying a verb and a noun. The noun identifies the type of the payload, although not all event messages will have a payload.

2) The service sends the event message to the appropriate JMS topic. The ESB implementation can also transparently use intermediaries such as routers and adapters to transmit the event to the appropriate event listeners.

3) The listener accepts the message.

4) If the event message is invalid (e.g. incomplete, XML not well formed, etc.), processing terminates (typically after an error is logged).

5) The listener looks at the verb and noun combination, determining if the event can (or should) be processed, if not, processing terminates.

6) The listener performs the desired processing, parsing the payload as needed. The service may parse the payload using an appropriate XML schema (as would define the payload type described by the message noun), using XPath expressions, using XSL transformations or other mechanisms.

When transactional request messages (see 4.5) are processed on the bus that use the verb 'create', 'change, 'close', 'cancel' or 'delete', a corresponding event should be published using the corresponding past tense verb (e.g. 'created', 'changed, 'closed', 'canceled', 'deleted' or 'executed'). This should be issued after successful execution of the transaction by either the service that processed the request or a bus component. In these cases, the payload is identical to the payload used for the corresponding request message used to invoke the transaction.

The delivery guarantees are a consequence of the definition of the specific JMS topic or queue, as well as the means by which a listener subscribes (e.g. durable subscription).

6.9 Message correlation

One important aspect of asynchronous messaging patterns is the need to be able to correlate a request with a reply. The Header.CorrelationID is key to the association of requests with asynchronous replies. The following rules should be applied to messages to allow necessary 'linking':

- When a client provides a CorrelationID on a request, the value should be a either a hexadecimal UUID (e.g. '`D921A053-80C1-4DB6-960E-2603127B7B92`') or a generated sequence number (e.g. 100023, 100024, 100025,...) that is effectively unique within the client making the request.
- If a request message includes a CorrelationID, the response message should return the same CorrelationID.
- If no CorrelationID is provided on a request message but a MessageID is provided, the response message should set the CorrelationID to the value of the MessageID that was provided on the request. MessageID should also be UUID or generated sequence numbers.
- If no MessageID or CorrelationID is provided on a request message, there is no way to correlate an asynchronous response to a specific request. Consequentially the CorrelationID cannot be set in the response message in a manner that identifies a linkage to a specific request.
- If a service is generating events as a direct consequence of a specific request, the CorrelationID should be set on the corresponding event message as per the previous rules if possible, noting that this may not always be possible and it is therefore not a requirement. This would provide a correlation between the event and the transaction that caused it.

Refer to the discussion and CorrelationID usage example provided in 5.2.5.

6.10 Complex transaction processing using OperationSet

6.10.1 General

The purpose of Subclause 6.10 is to describe the use of the OperationSet element provided by Message.xsd. This provides support for transactions. The Message.xsd message envelope has been extended to accommodate an OperationSet construct in both the payload and reply portions of Message.xsd. The OperationSet element is shown within Figure 39.

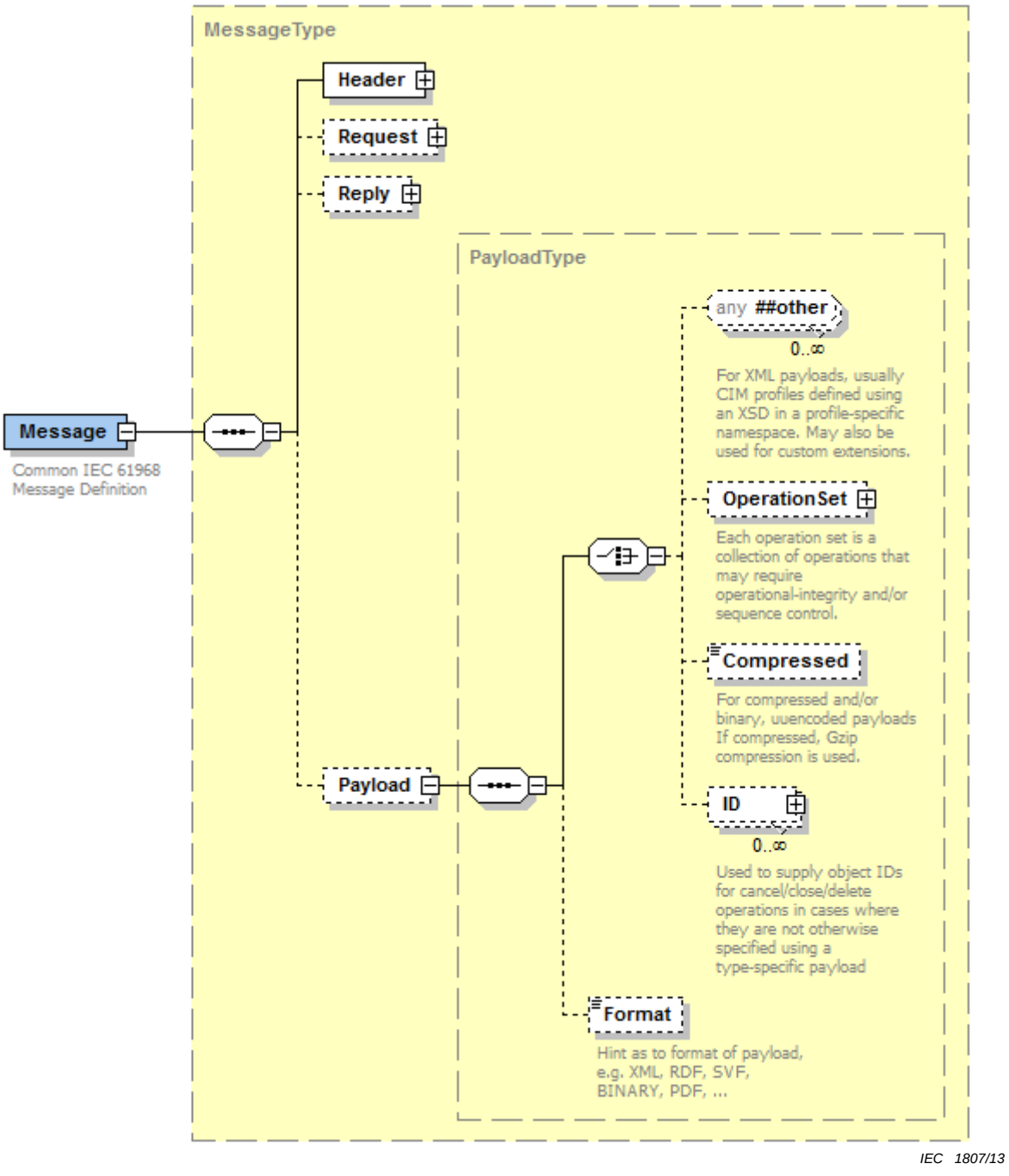


Figure 39 – Message OperationSet Element

There are two circumstances where the use of OperationSet might be necessary:

a) When modifying the configuration of a CIM object and the modification involves deleting one or more attributes or one or more instances of associated CIM objects. An example is removing a Register configuration from a Meter.

b) When performing two or more related actions that must be handled in a specific sequence and/or with overall transactional integrity (i.e., either all actions must succeed or all must be rolled back).

A message utilizing the OperationSet construct always has a Header verb of either ‘execute’ or ‘executed’ and a noun of ‘OperationSet’. An OperationSet in turn contains one or more Operation elements, and each OperationSet.Operation has an operationId which supplements the overall message CorrelationID to provide a fine-grained ability to correlate the contents of one or more reply messages with the individual operations in an OperationSet. Individual

Operation elements within an OperationSet have OperationSet-level verbs and nouns. Allowable verbs are create, created, change, changed, delete and deleted.

To support circumstance a) above, each Operation in an OperationSet also includes an elementOperation boolean. This Boolean is to be set to ‘true’ when the Operation verb is either ‘delete’ or ‘deleted’ and the intent is to delete individual attributes or individual instances of associated CIM classes from the object specified by the OperationSet noun (as opposed to deleting the entire CIM object specified by the Operation noun. If omitted, elementOperation is assumed to be ‘false’. It is emphasized that in this case, use of the Operation verb “delete” or “deleted” in combination with an elementOperation boolean set to ‘true’ effectively modifies (and does not delete) the CIM object specified by the Operation noun.

To support circumstance b) above, each OperationSet may have either an enforceMsgSequence boolean or an enforceTransactionalIntegrity boolean, or both. The enforceMsgSequence Boolean is to be set to ‘true’ when the Operations in the Operation set shall be executed in ascending order of their operationID. The enforceTransactionalIntegrity boolean is to be set to ‘true’ if all Operations in the OperationSet shall succeed. In this case, if all such Operations do not succeed, all shall be rolled back. If either or both of these booleans are omitted, they are assumed to be ‘false’.

When modifying the configuration of a CIM object using any of the verbs ‘change’, ‘changed’, ‘delete’ or ‘deleted’, only the ID of the object being changed and the information that is being changed is to be included. This is true whether or not an OperationSet is being used. It is for this reason that almost all elements within the IEC 61968-9 Master Data Management Profiles are optional in the profiles.

It is recommended that only one OperationSet be used, as multiple OperationSets would place more burden upon consumers and potentially involve unnecessarily large messages.

It should also be noted that while this provides the means to convey transactions using XML schema-based data structures, it is also technically possible to leverage IEC 61970-552[1] for transactions based upon RDF.

6.10.2 OperationSet Element

Figure 40 describes the OperationSet element in more detail. An OperationSet can:

- Require that each operation is sequentially executed by setting the enforceMsgSequence flag to ‘true’
- Require that transactional integrity be maintained (i.e. all or nothing), by setting the enforceTransactionalIntegrity flag to ‘true’
- Have one or more Operations, where each operation has a noun, verb and payload.

1 To be published.

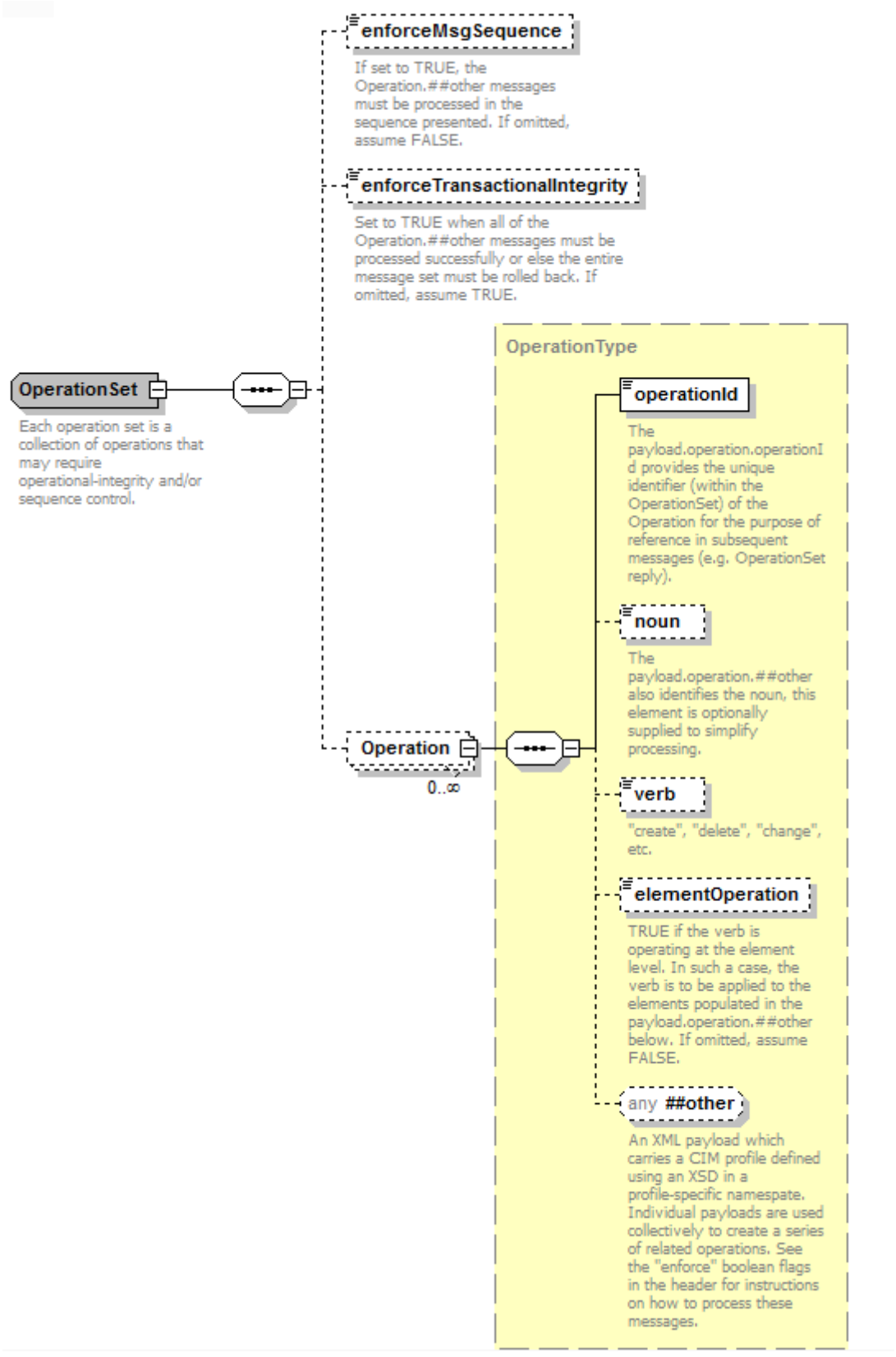


Figure 40 – OperationSet details

Within the Operation element, the noun will identify the type of the any element. The elementOperation value will cause the transaction to either act upon the object or elements within the object. Examples provided will further describe usage.

6.10.3 Patterns

Any given transaction may be executed using either a request-response message pattern (Request stereotype and Response stereotype messages) or a published event message pattern (Event stereotype messages). The four example sequence diagrams of Figure 41 illustrate the possible variations.

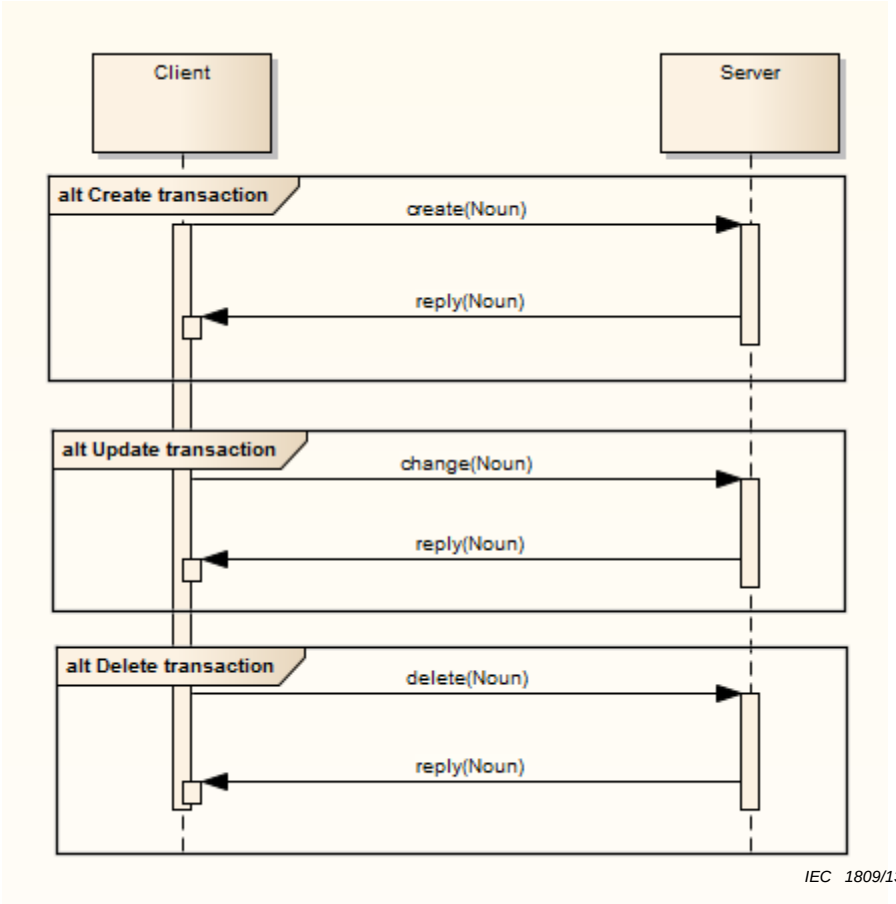


Figure 41 – Transactional Request/Response (non-OperationSet)

This request / response pattern can be used for transactions. Allowable verbs are ‘create’, ‘change’ and ‘delete’. Depending upon the scenario, there can be multiple replies to a given ‘create’, ‘change’, or ‘delete’ message. For example, a single create message can be issued to create multiple meters. In this case, the responding system can send a single reply message for all meters or multiple reply messages with the reply data for one or more meters in each message.

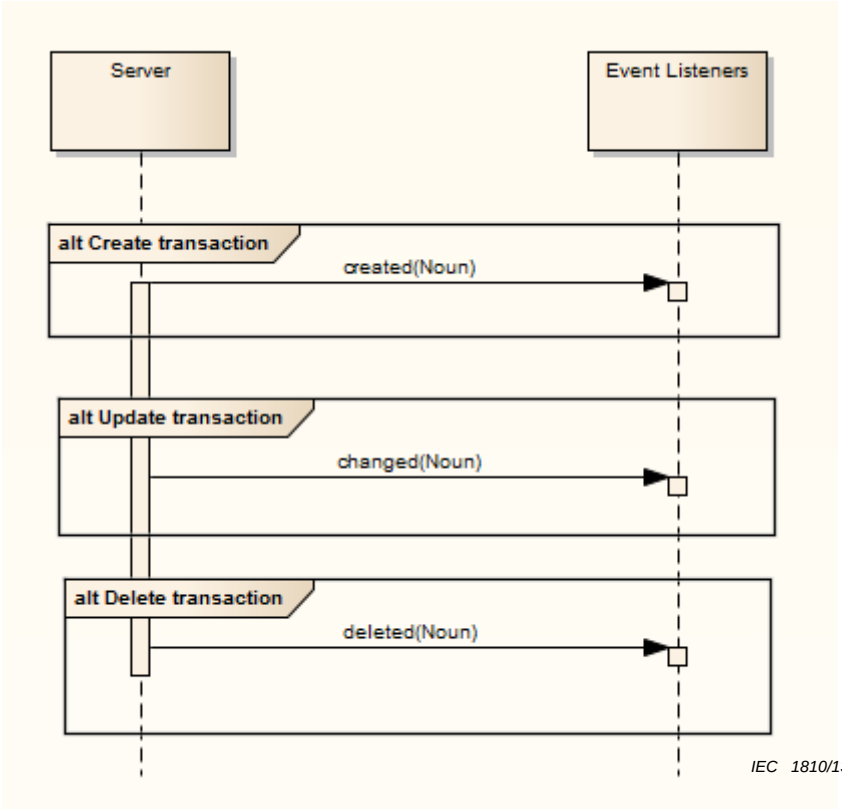


Figure 42 – Published events (non-OperationSet)

The published event pattern can also be used for transactions, as shown by the sequence diagrams of Figure 42. Allowable verbs are ‘created’, ‘changed’ and ‘deleted’. Using this pattern, an enterprise system may notify one or more other enterprise systems of events without requiring any acknowledgment or confirmation of successful processing.

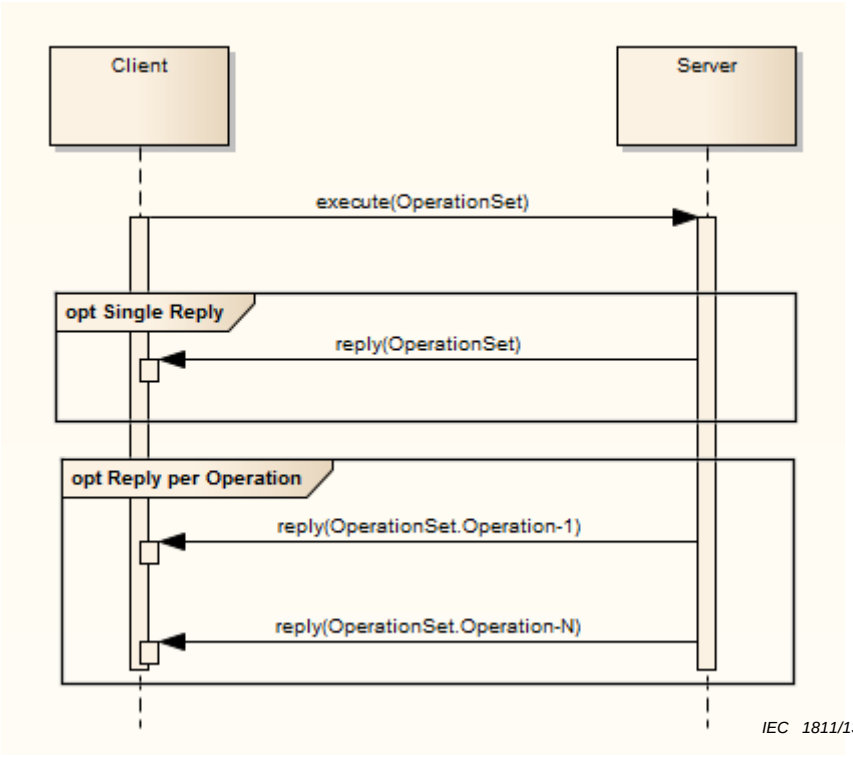


Figure 43 – Transactional Request/Response (OperationSet)

This request/response pattern can be used for any transaction involving an OperationSet, as shown by the sequence diagram of Figure 43. The verb in the message Header is always ‘execute’. The individual Operation(s) within the Operation set can have verbs and nouns consistent with the request / response transaction in Pattern 1. Depending upon the scenario, there can be multiple replies to a given execute / OperationSet transaction. For example, a single reply message can be sent for the entire OperationSet, or multiple reply messages can be sent, each with the reply data for one or more Operations in each message. The operationID element for each Operation in the request message is supplied in the reply message(s). This is used, in conjunction with the overall CorrelationID in the message Header(s) to correlate replies with their corresponding requests.

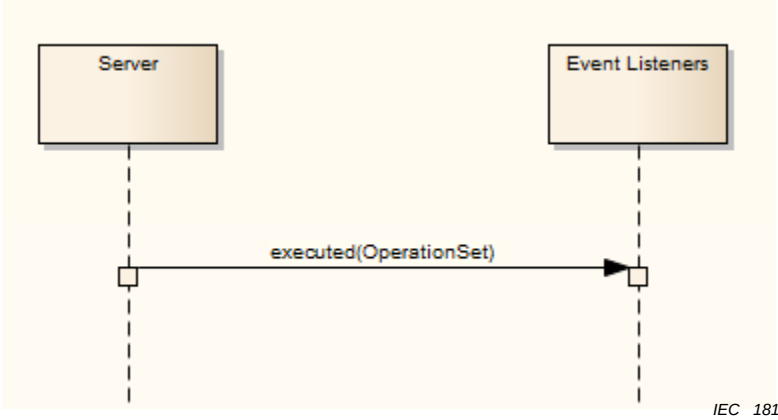


Figure 44 – Published event (OperationSet)

The published event pattern can also be used for any transaction involving an OperationSet, as shown in the sequence diagram of Figure 44. The verb in the message Header is always “executed”. The individual Operation(s) within the Operation set can have verbs and nouns consistent with the published event transaction in Pattern 2. Using this pattern, an enterprise system may notify one or more other enterprise systems of OperationSet events without requiring any acknowledgment or confirmation of successful processing.

6.10.4 OperationSet example

The following XML provides an example of a complex transaction that uses the Payload.OperationSet element.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestMessage
    xmlns = "http://iec.ch/TC57/2011/schema/message"
    xmlns:m = "http://iec.ch/TC57/2011/MeterConfig#"
    xmlns:up = "http://iec.ch/TC57/2011/UsagePointConfig#"
    xmlns:mdlc = "http://iec.ch/TC57/2011/MasterDataLinkageConfig#"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://iec.ch/TC57/2011/schema/message Message.xsd">
    <Header>
        <Verb>execute</Verb>
        <Noun>OperationSet</Noun>
        <Revision>2.0</Revision>
        <Timestamp>2012-12-20T09:30:47Z</Timestamp>
        <Source>CIS</Source>
        <AckRequired>true</AckRequired>
        <MessageID>D921A053-80C1-4DB6-960E-2603127B7B92</MessageID>
        <CorrelationID>D921A053-80C1-4DB6-960E-2603127B7B92</CorrelationID>
    </Header>
    <Payload>
        <OperationSet>
            <enforceMsgSequence>true</enforceMsgSequence>
            <enforceTransactionalIntegrity>true</enforceTransactionalIntegrity>
            <Operation>
                <operationId>1</operationId>
                <noun>MeterConfig</noun>
                <verb>create</verb>
                <mdlc:MeterConfig>
```

```
                    <mdlc:Meter>
                        <mdlc:formNumber>2S</mdlc:formNumber>
                        <mdlc:ConfigurationEvents>
                            <mdlc:createdDateTime>2012-12-
20T09:30:47Z</mdlc:createdDateTime>
                            <mdlc:effectiveDateTime>2012-12-
21T00:00:00Z</mdlc:effectiveDateTime>
                            <mdlc:Names>
                                <mdlc:name>C34531</mdlc:name>
                                <mdlc:NameType>
                                    <mdlc:name>MeterBadgeNumber</mdlc:name>
                                    <mdlc:NameTypeAuthority>
                                        <mdlc:name>UtilityXYZ</mdlc:name>
                                    </mdlc:NameTypeAuthority>
                                </mdlc:NameType>
                            </mdlc:Names>
                        </mdlc:ConfigurationEvents>
                    </mdlc:Meter>
                </mdlc:MeterConfig>
            </Operation>
            <Operation>
                <operationId>2</operationId>
                <noun>UsagePointConfig</noun>
                <verb>create</verb>
                <up:UsagePointConfig>
                    <up:UsagePoint>
                        <up:amiBillingReady>amiCapable</up:amiBillingReady>
                        <up:connectionState>connected</up:connectionState>
                        <up:isSdp>true</up:isSdp>
                        <up:isVirtual>false</up:isVirtual>
                        <up:phaseCode>B</up:phaseCode>
                        <up:readCycle>ReadCycleJ</up:readCycle>
                        <up:ConfigurationEvents>
                            <up:createdDateTime>2012-12-
20T09:30:47Z</up:createdDateTime>
                            <up:effectiveDateTime>2012-12-
21T00:00:00Z</up:effectiveDateTime>
                        </up:ConfigurationEvents>
                        <up:Names>
                            <up:name>UP43639</up:name>
                            <up:NameType>
                                <up:name>ServiceDeliveryPointID</up:name>
                                <up:NameTypeAuthority>
                                    <up:name>UtilityXYZ</up:name>
                                </up:NameTypeAuthority>
                            </up:NameType>
                        </up:Names>
                    </up:UsagePoint>
                </up:UsagePointConfig>
            </Operation>
            <Operation>
                <operationId>3</operationId>
                <noun>MasterDataLinkageConfig</noun>
                <verb>create</verb>
                <mdlc:MasterDataLinkageConfig>
                    <mdlc:ConfigurationEvent>
                        <mdlc:createdDateTime>2012-12-
17T09:30:47Z</mdlc:createdDateTime>
                        <mdlc:effectiveDateTime>2012-12-
21T00:00:00Z</mdlc:effectiveDateTime>
                    </mdlc:ConfigurationEvent>
                    <mdlc:Meter>
                        <mdlc:Names>
                            <mdlc:name>C34531</mdlc:name>
                            <mdlc:NameType>
                                <mdlc:name>MeterBadgeNumber</mdlc:name>
                                <mdlc:NameTypeAuthority>
                                    <mdlc:name>UtilityXYZ</mdlc:name>
                                </mdlc:NameTypeAuthority>
                            </mdlc:NameType>
                        </mdlc:Names>
                    </mdlc:Meter>
                    <mdlc:UsagePoint>
                        <mdlc:Names>
                            <mdlc:name>UP43639</mdlc:name>
                            <mdlc:NameType>
                                <mdlc:name>ServiceDeliveryPointID</mdlc:name>
                                <mdlc:NameTypeAuthority>
```

```
                                    <mdlc:name>UtilityXYZ</mdlc:name>
                                </mdlc:NameTypeAuthority>
                            </mdlc:NameType>
                        </mdlc:Names>
                    </mdlc:UsagePoint>
                </mdlc:MasterDataLinkageConfig>
            </Operation>
        </OperationSet>
    </Payload>
</RequestMessage>
```

The example complex transaction has three operations that do the following:

- Perform a ‘create MeterConfig’
- Perform a ‘create UsagePointConfig’
- Performs ‘create MasterDataLinkageConfig’

The XML identifies namespaces for MeterConfig, UsagePointConfig and MasterDataLinkageConfig as defined by IEC 61968-9.

6.11 Representation of time

The ISO 8601 standard is used to define the representations of time values that are conveyed through interfaces. This avoids issues related to time zones and daylight savings time changes.

Timestamps in messages published by a server process should use a prevailing time, using the following example format: *2007-03-27T14:00:00-05:00* (as time changes from CDT to CST, the *-05:00* would change to *-06:00*).

Timestamps in messages sent by a client process could use any ISO 8601 compliant timestamp.

It is extremely important to note that the use of ISO 8601 timestamps within message definitions for the external interfaces defined by this document in no way constrains other representations of time that may include:

- User interfaces, where local time or market hours may be used as desired
- Reports, where reports would be generated using an appropriate local time
- Internal integration, where an application may internally require some other time structure

6.12 Other conventions and best practices

The following are other conventions that shall be followed by this specification:

- Within XML definitions, tags should be namespace qualified. For example, an XML tag of ‘<tag>’ should be prefixed by a specific namespace reference, e.g. ‘<ns:tag>’. This will help to eliminate ambiguity. *(Note that many examples in this document are not namespace qualified for brevity and to aid legibility)*
- Quantities should be expressed using SI units where appropriate.

6.13 Technical interoperability

Open standards are a key part of the strategy to achieve technical interoperability. Standards of particular interest include:

- W3C standards
- OASIS WS-* standards

- IEC Common Information Model and related standards (e.g. IEC 61970-301 and IEC 61968-11)
- Java Message Service

It is very important that the implementation of Web Service interfaces not be dependent upon any specific proprietary, third party products. Another key requirement is that implementation of web service clients shall be possible using both Java and .Net development tools.

6.14 Service level agreements

Different categories of services will have different service level agreements (SLAs). The SLAs for some services are directly impacted by the variability in the amount of data that can be transferred.

The response time periods specified for each interface covered by an SLA typically will vary to some degree, based upon factors such as network and system loading. Consequentially, each SLA should be stated in a manner such that each SLA will be honoured X% of the time where X is often in the range of 90 to 100%.

One use of SLAs is to identify timeout periods for request handling.

6.15 Auditing, monitoring and management

The ESB will typically have capabilities for auditing, monitoring and management. There may also be common services that are used for the implementation of integration components within the ESB. Example functionality would often include:

- Logging
- Generation of unique identifiers
- Generation of signatures
- User authentication and authorization
- Identification of on-line service instances (where there may be multiple instances)

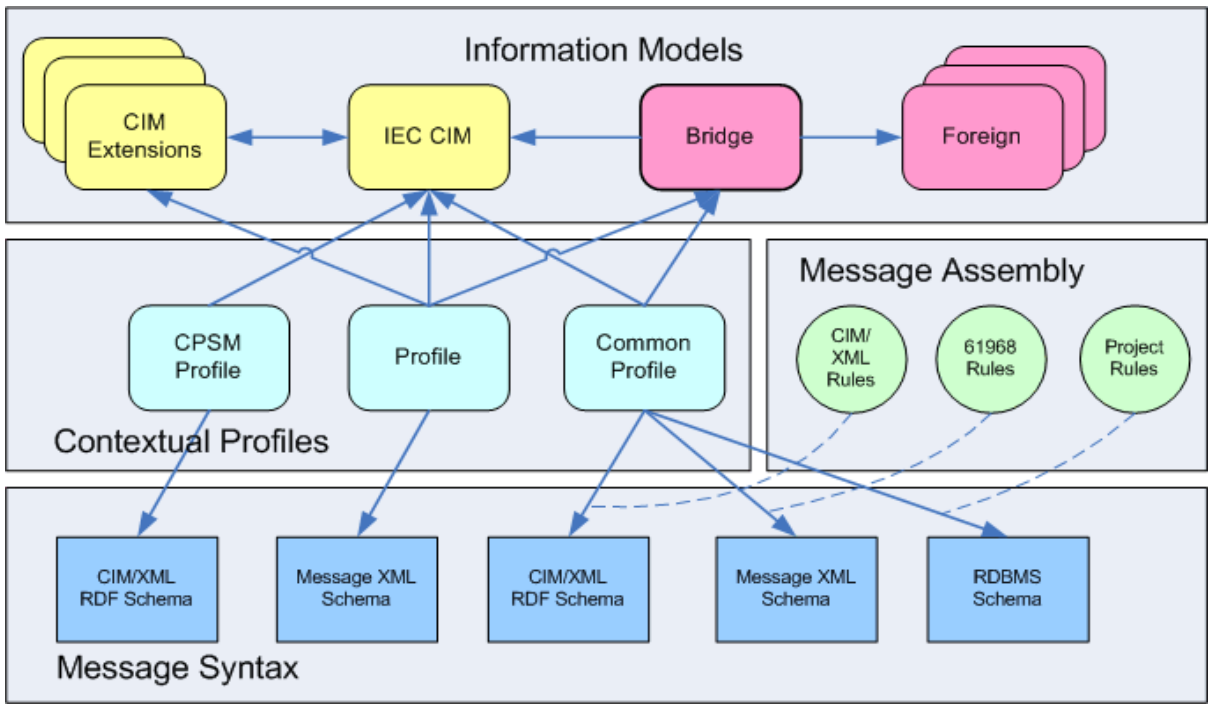
7 Payload specifications

Each noun used in a message identifies a payload type. Payload types are typically derived from the IEC CIM or other semantic models. Payload types used by the parts of IEC 61968 are always derived from the IEC CIM and have design artefacts (e.g. XSDs) that describe their structure. Cases where XSDs are not required include:

- Messages using RDF payloads as defined by IEC 61968-13 and IEC 61970-452.
- Messages using payloads as defined by IEC 61970-453.
- Response messages from services that dynamically generate XML (as in the case of SQL XML result sets).
- Non-XML compressed and encoded payloads.
- Encoded binary data (where XML formatting is not efficient as in the case of ‘high speed data’)

If an XSD is not available to describe the payload, it is the responsibility of the sender and receiver(s) to agree upon the specific formatting.

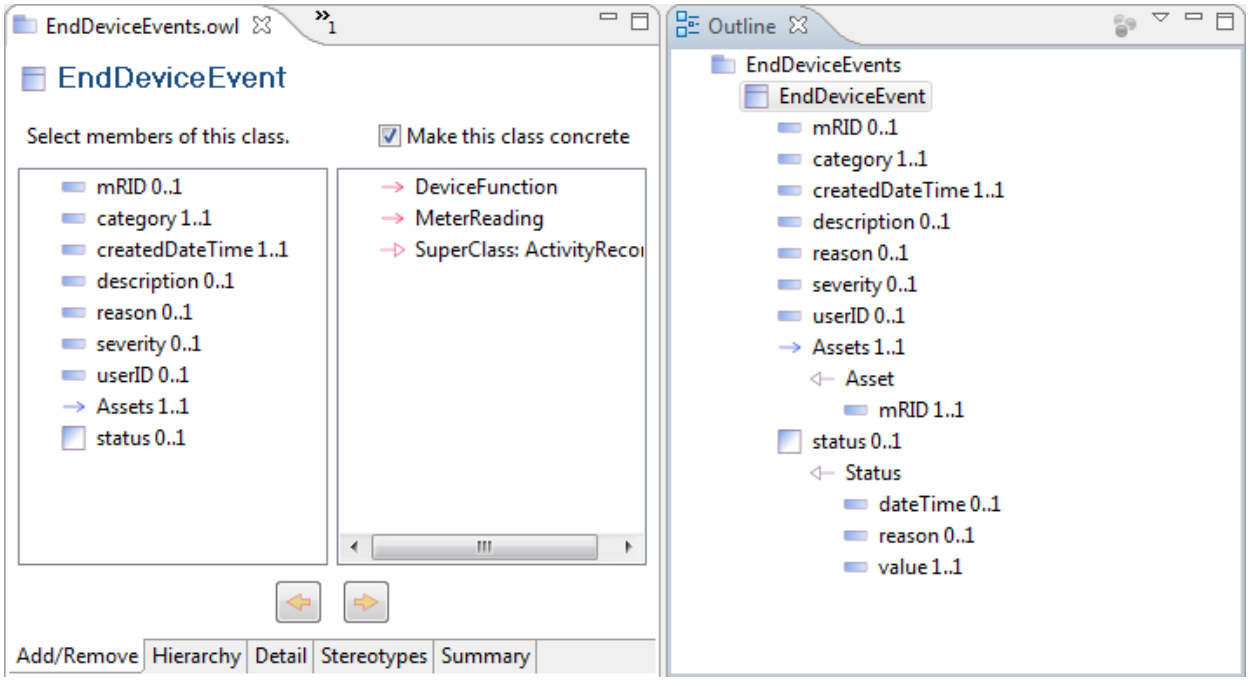
The CIM logical information model is described as a set of UML packages. The diagram in Figure 45 shows the use of the CIM from the perspectives of UML modelling and generation of design artefacts needed by integration tools. It illustrates the relationships between information models and contextual profiles that are used in conjunction with assembly rules in order to derive design artefacts.



IEC 1813/13

Figure 45 – Information Models, Profiles and Messages

Figure 46 shows an example contextual profile design within CIMTool[2].



IEC 1814/13

Figure 46 – Contextual Profile Design in CIMTool

When realizing a profile as a design artefact in the form of an XML Schema, it is important to recognize that there are many options related to the realization that may affect interoperability, these include:

2 Available at www.cimtool.org.

- Use of namespaces
- Object references ‘by value’ or ‘by reference’
- Flat or hierarchical complex type definitions
- Required vs. Optional elements
- Enumerations

Another important point is that the noun shall not be a name of a CIM UML class, otherwise it is not possible to have a valid XSD that includes that UML class in the profile/payload type.

The diagram in Figure 47 describes the structure of a simple example payload as described by an XML Schema that could be conveyed within the ‘any’ of the payload.

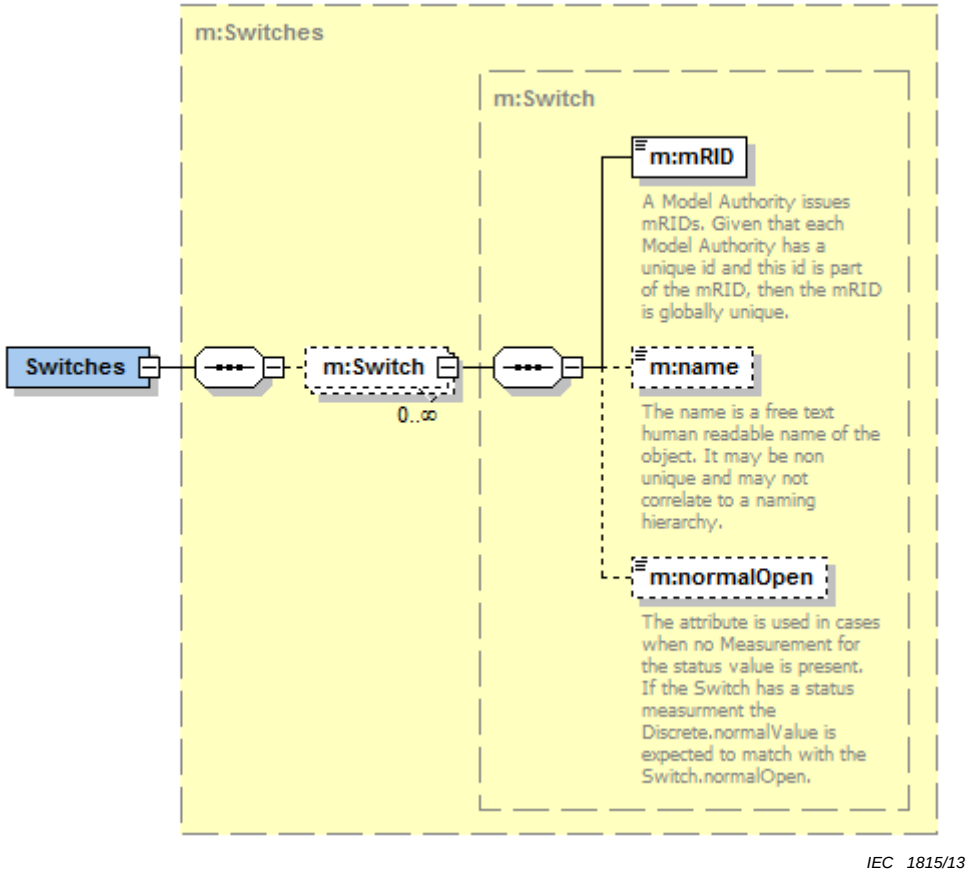


Figure 47 – Example message payload schema

The example payload of Figure 47 is described by the XML Schema definition provided by Figure 48. XML Schemas for payloads can be generated in a variety of ways. One example is the use of CIMTool, where the CIM UML model is used as the domain model for the message definition. Note that the XML Schema for a message payload minimally defines a top level element.

The important point is that the name of the top level element shall be the same as the noun that is used in the message header. In the following XSD the payload definition would be used in conjunction with the noun ‘Switches’.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:a="http://www.iec.ch/2008/Message#"
targetNamespace="http://iec.ch/TC57/2011/CIM-schema-cim12#"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns="http://iec.ch/TC57/2011/Message#" xmlns:m="http://iec.ch/TC57/2007/CIM-schema-
cim12#">
<xs:element name="Switches" type="m:Switches"/>
   <xs:complexType name="Switches">
      <xs:sequence>
         <xs:element name="Switch" type="m:Switch" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Switch">
      <xs:annotation>
         <xs:documentation>A generic device  designed to close, or open, or both, one
or more electric circuits.</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="mRID" minOccurs="1" maxOccurs="1" type="xs:string">
            <xs:annotation>
               <xs:documentation>A Model Authority issues mRIDs. Given that each Model
Authority has a unique id and this id is part of the mRID, then the mRID is globally
unique.</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="name" minOccurs="0" maxOccurs="1" type="xs:string">
            <xs:annotation>
               <xs:documentation>The name is a free text human readable name of the
object. It may be non unique and may not correlate to a naming
hierarchy.</xs:documentation>
             </xs:annotation>
         </xs:element>
         <xs:element name="normalOpen" minOccurs="0" maxOccurs="1" type="xs:boolean">
            <xs:annotation>
               <xs:documentation>The attribute is used in cases when no Measurement for
the status value is present. If the Switch has a status measurment the
Discrete.normalValue is expected to match with the
Switch.normalOpen.</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

IEC 1816/13

Figure 48 – Example payload XML schema

From the previous XML Schema of Figure 48, an example XML payload is provided by Figure 49 (as was used for examples in Clause 6):

```
<m:Switches xsi:schemaLocation="http://www.iec.ch/TC57/2011/CIM-schema-cim12#
Switches.xsd" xmlns:m="http://www.iec.ch/TC57/2011/CIM-schema-cim12#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <m:Switch>
      <m:mRID>35378383838</m:mRID>
      <m:name>SW1</m:name>
      <m:normalOpen>true</m:normalOpen>
   </m:Switch>
   <m:Switch>
      <m:mRID>363482488448</m:mRID>
      <m:name>SW2</m:name>
      <m:normalOpen>true</m:normalOpen>
   </m:Switch>
   <m:Switch>
      <m:mRID>894094949444</m:mRID>
      <m:name>SW3</m:name>
      <m:normalOpen>false</m:normalOpen>
   </m:Switch>
</m:Switches>
```

IEC 1817/13

Figure 49 – Example message XML

Specific payload formats should be defined by an interface specification using XML Schemas, as are provided by IEC 61968-3 to 61968-9. For implementations outside the scope of IEC 61968, payload definitions can be defined as needed. In most cases the message noun takes a simple form such as ‘Switches’, ‘BidSets’, ‘TroubleTickets’ or ‘WorkOrders’. However it is also possible to use a prefix which identifies a message context in the following form:

```
<context><noun>
```

This would allow for stereotypes of the basic noun definition to be used to define additional restrictions appropriate for the message context. Examples would be ‘GetMeterReadings’ and ‘GetEndDeviceAssets’.

8 Interface specifications

8.1 General

The purpose of Clause 8 is to describe interface definitions. There are three perspectives provided here:

- What is needed by a user of the interface to supplement the information provided by a specific definition language or design artefacts
- Web services artefacts and implementation details
- JMS implementation details

8.2 Application-level specifications

Specific interfaces are defined using a sequence of specific combinations of verbs and nouns (i.e. payload types). For example a request message with verb and noun of ‘get MeterReadings’ would result in a response message of ‘reply MeterReadings’. Given the potential complexity and options available for a given integration, the details of the interactions should be further documented. Application-level specifications based upon IEC 61968-100 will often require more detailed specifications that are typically beyond the capabilities of XML schemas and WSDLs. The following are simple examples of the documentation for messages that might be provided by an application-level specification.

The messages for a request would use the following message fields:

Message Element	Value
Header/Verb	get
Header/Noun	*Name of payload type*
Header/Source	*System or application initiating request*
Header/UserID	*Optional: ID of user*
Request/?	*Optional: Other request parameters may be specified as needed*

The corresponding response messages would use the following message fields:

Message Element	Value
Header/Verb	reply
Header/Noun	*Defined payload type name*
Reply/result	*Reply code, success=OK, partial success=PARTIAL, error=FAILED*
Reply/Error	*Optional: May be any number of error messages*
Payload	*Defined payload type*

In the cases of payloads that would otherwise be very large (as an example, over some threshold such as 1 megabyte), the payloads would be zipped, base64 encoded and stored within the 'Payload/Compressed' tag.

Specific nouns, verbs (as defined in Annex B) and payload formats should be defined by an interface specification, as are provided by IEC 61968-3 to 61968-9. Use case sequence diagrams are also commonly used to describe information exchange patterns in terms of verbs and nouns.

A more thorough description of the usage of an interface, potentially as part of a more complex business process would be described using a sequence diagram. The following sequence diagram provides an example of information exchange using verbs and nouns. The diagram convention uses '<verb>(<Noun>)' for each flow between components.

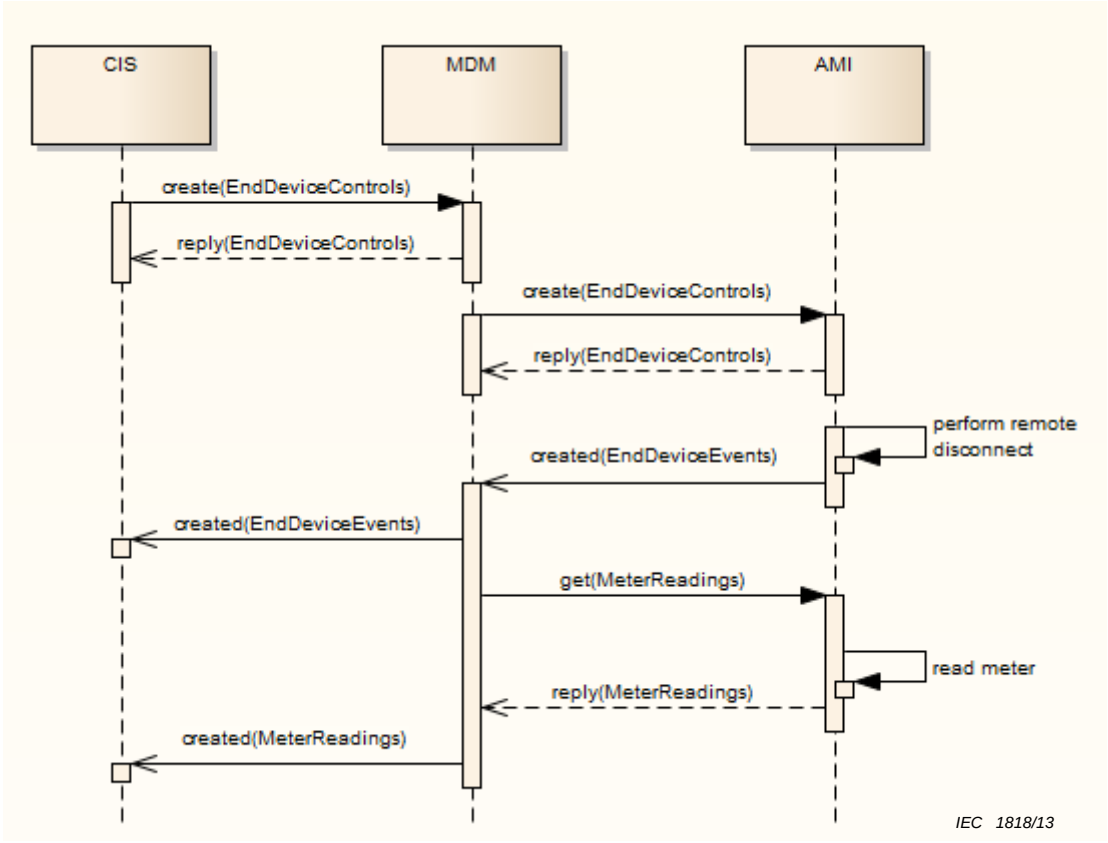


Figure 50 – Example complex business process

Figure 50 is a sequence diagram that describes a complex business process that combines several different types of messages and integration patterns.

8.3 Web service interfaces

8.3.1 General

Subclause 8.3 describes the definition of interfaces using web services. This describes the use of a document-wrapped style which maximizes interoperability. This also prescribes operations which are names using verb/noun combinations, with type-specific payload definitions.

There are two approaches described by this standard for web services:

- Strongly-typed, with procedure for WSDL generation defined in detail in Annex C
- Generic web services, where a generic WSDL is described in Annex D

8.3.2 WSDL Structure

Typically a WSDL (v1.1) is made of two parts with the tags shown in Figure 51.

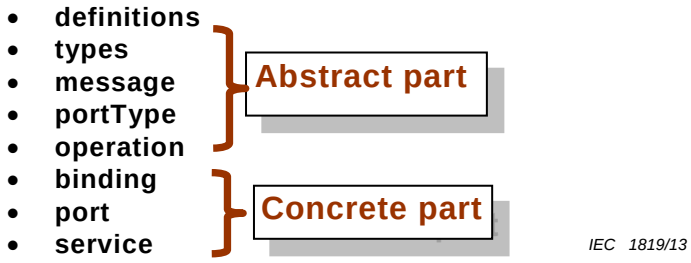


Figure 51 – WSDL structure

WSDL example document can be found in WSDL template in Annex 4.

For strongly-typed web services, the web service design practices are summarized below:

- Standard SOAP binding is used
- XSD as data type is typically imported instead of being embedded for better version control
- Wire signature issue is avoided by redefining element names such as CreateEndDeviceControl and ChangeEndDeviceControl using a single XSD complexType
- Wrapped Document style is used
- Operation name follows the Verb + Noun naming convention which allows avoiding contend-based routing

8.3.3 Document style SOAP binding

The document style using SOAP body is the most common practice in WSDL design. It can fully utilize the benefits of an XML schema for payload validation. Below is an example of the binding section in a Document style WSDL for the EndDeviceControl information exchange:

```
<wsdl:binding name="EndDeviceControls_Binding" type="tns:EndDeviceControls_Port">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="CreateEndDeviceControls">
      <soap:operation
soapAction="http://iec.ch/TC57/2010/EndDeviceControls/CreateEndDeviceControls"
style="document"/>
      <wsdl:input name="CreateEndDeviceControlsRequest">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CreateEndDeviceControlsResponse">
         <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="CreateEndDeviceControlsFault">
         <soap:fault name="CreateEndDeviceControlsFault" use="literal"/>
      </wsdl:fault>
   </wsdl:operation>
…
```

Note that both <soap:binding> and <soap:operation> styles are defined as “document” and are highlighted in gray. Also <soap:body> is used for both input and output operations. A “document” style means an XML document is included in a soap message. In this case, it is directly placed in the <soap:body>.

If a wsdl:*operation* name is the same as the input element name, this WSDL becomes a wrapped document style WSDL. Wrapped document style originates from Microsoft to mimic a RPC style. In a RPC style, a payload is always wrapped by its operation name.

The characteristics of the wrapped pattern are listed below:

- The input message has a single *part*
- The *part* is an element
- The element has the same name as the *operation*
- The element's complex type has no attributes.

Any WSDL that does not meet the criterion above is an unwrapped WSDL. There are pros and cons for both wrapped and unwrapped patterns, but wrapped document style is recommended in this profile for the sake of interoperability.

Here is a sample WSDL on the wrapped document style:

```
  … …

<wsdl:message name="CreateEndDeviceControlsRequestMessage">
   <wsdl:part name="CreateEndDeviceControlsRequestMessage"
element="message:CreateEndDeviceControls"/>
</wsdl:message>
… …

<wsdl:portType name="EndDeviceControls_Port">

   <wsdl:operation name="CreateEndDeviceControls">
      <wsdl:input name="CreateEndDeviceControlsRequest"
message="tns:CreateEndDeviceControlsRequestMessage"/>
      <wsdl:output name="CreateEndDeviceControlsResponse"
message="tns:ResponseMessage"/>
      <wsdl:fault name="CreateEndDeviceControlsFault" message="tns:FaultMessage"/>
   </wsdl:operation>
   … …
   </wsdl:operation>
</wsdl:portType>
```

8.3.4 Strongly-typed web services

8.3.4.1 General

The strongly-typed web service integration pattern is intended for use to implement semantic-based interfaces in support of a SOA integration strategy. The strongly-typed pattern has the following characteristics:

1) Uses SOAP-based web services, where fine-grained WSDLs are used to define a contract.
2) Enables stronger payload validation by defining operation messages using strongly typed payloads.

8.3.4.2 Service and operation naming

In the IEC 61968-100 strongly-typed web service implementation, the following service names are used to reflect the role of the service in the enterprise:

- Send

  To provide (send) information (business object) for public (enterprise) consumption. To be invoked by the system of record for the business object and only when the state of the business object has changed. This is used in conjunction with the verbs created, changed, closed, canceled and deleted.

- Receive

  To consume (receive) information (business object) from an external source. This is used in conjunction with the verbs created, changed, closed, canceled and deleted.

- Request

To request another party to perform a specific service. This is used in conjunction with the verbs get, create, change, close, cancel and delete.

- Execute

  To run a service provided to the public, which may include a state change request or a query request. This is used in conjunction with the verbs create, change, close, cancel and delete.

- Reply

  To reply with the result of the execution of a service (by the Execute service). This is used in conjunction with the verbs created, changed, closed, canceled and deleted.

- Show

  To provide (show) information (business object) for public (enterprise) consumption, when the state of the business object is not changed, by the system of record or other system that has a copy of the same business object.

- Retrieve

  To request specific data of a business object to be provided.

Using the service name and operation patterns, information objects and verbs, a service/operation naming convention for strongly-typed web services is described as below:

- Service name:

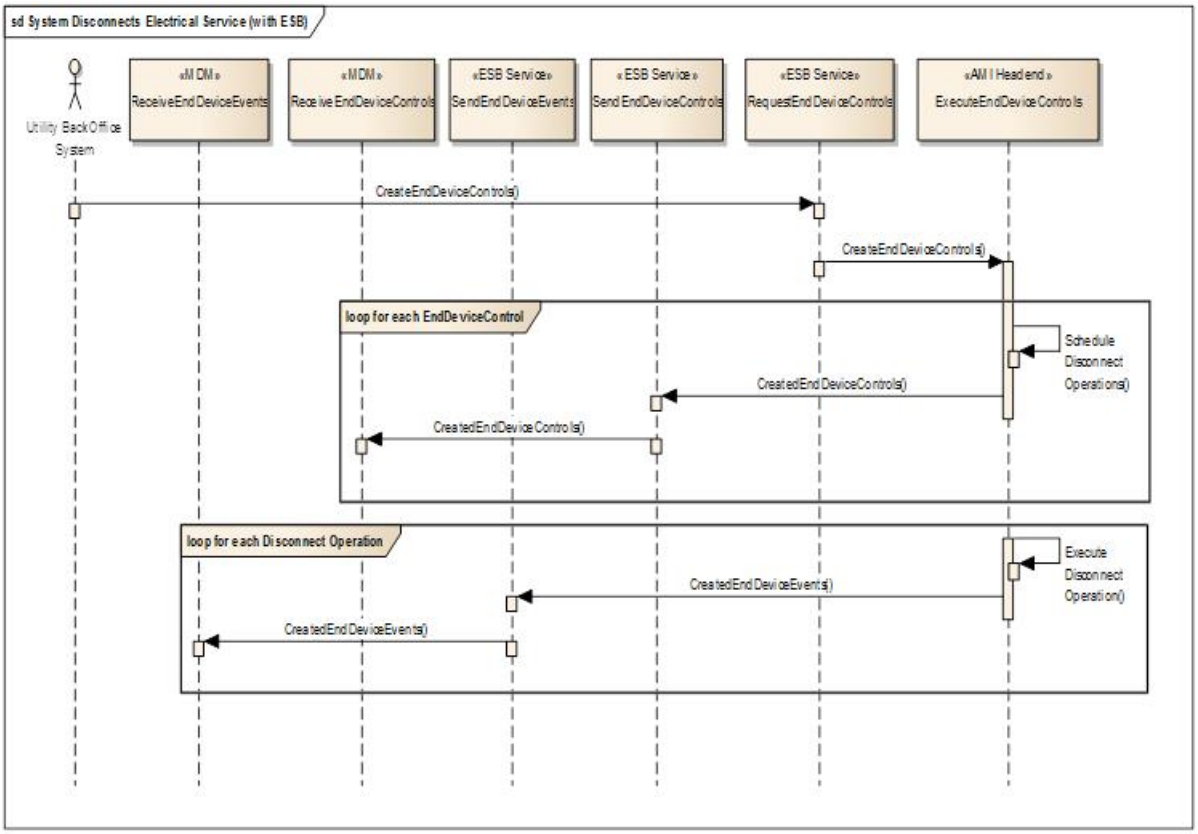
  To follow <*Service pattern name*>+<*Information Object*> such as ExecuteEndDeviceControls

- Operation name:

  To follow <*Verb*>+<*Information Object*> such as CreatedEndDeviceControls

8.3.4.3 Strongly-typed Web Service Integration Example

Figure 52 gives an example usage of the strongly-typed web services to implement connect/disconnect functionality between an MDM and AMI system utilizing an ESB.



IEC 1820/13

Figure 52 – Web service usage example

1) Meter Data Management System
   a) Services Implemented
      i) ReceiveEndDeviceControls: processes event stereotype messages (notifications) of EndDeviceControls activity.
      ii) ReceiveEndDeviceEvents: processes event stereotype messages (notifications) of EndDeviceEvent activity.
2) Enterprise Service Bus
   a) Services Implemented
      i) RequestEndDeviceControls: processes requests to perform some activity with EndDeviceControls. Routes to appropriate endpoint(s) for execution.
      ii) ReplyEndDeviceControls: processes replys regarding event stereotype messages (notifications) of EndDeviceControls activity. Publishes to the appropriate endpoint(s).
      iii) SendEndDeviceEvents: processes event stereotype messages (notifications) of EndDeviceEvent activity. Publishes to the appropriate endpoint(s).
3) AMI System
   a) Services Implemented
      i) ExecuteEndDeviceControls: acts on (executes) sets of EndDeviceControls.

8.4 JMS

8.4.1 General

Subclause 8.4 describes the use of JMS. Messages communicated using JMS will use topics and/or queues. The differences and similarities between topics and queues are summarized as follows:

- Topics are used when the destination of a message is potentially more than one process

- Queues are used when the destination of a message is at most one process
- If supported by the JMS provider, topics and queues may be organized and named hierarchically
- Except for the use of a durable subscription, a process can only receive a copy of a message published to a topic if it is running and has an active subscription
- Message published to queues will remain on the queue until de-queued by a receiving process (noting that there may be options for expiration and queue persistence by the specific JMS implementation)
- A Queue is in effect a special case of a durable topic subscription, where only one process consumes a message.

8.4.2 Topic and queue naming

When naming a topic or queue, the top-level should identify 'context'. Examples of context can be production, testing, development or training. The purpose is to insure that messages of different contexts are logically segregated if not physically segregated. For example, it is critical that there is no opportunity for a message related to a training activity to be injected into a production activity. The use of both physical and logical segregation is desirable.

Figure 53 describes a possible organization of topics and queues. This organisation is an example only and not normative.

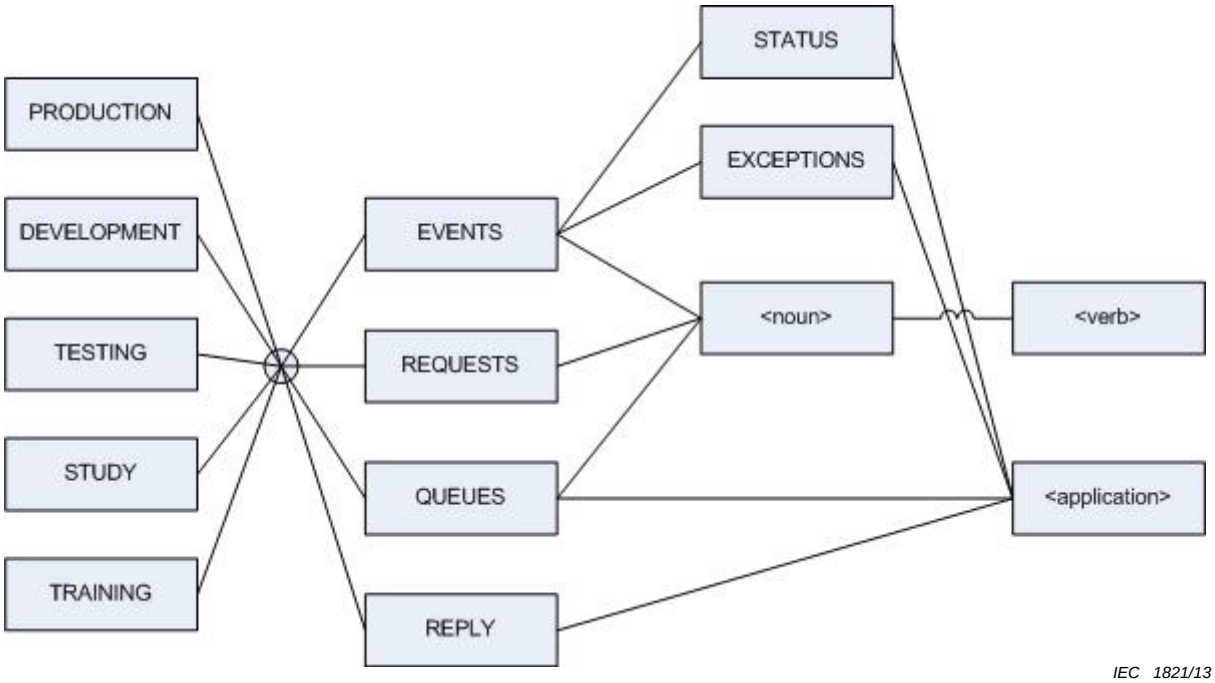


Figure 53 – Example Organization of Topics and Queues

This topic/queue organization would result in names such as:

- PRODUCTION.EVENTS.STATUS
- PRODUCTION.EVENTS.BidSet.created
- PRODUCTION.REQUESTS.BidSet.create
- STUDY.EVENTS.Contingency.created

It is important to note that JMS implementations typically allow for the use of wild cards in subscriptions. The intent of the described organization is meant to allow that feature to be leveraged. Additionally, it is possible to extend the topic definitions to provide for more

granular subscriptions. For example, a topic in the form <context>.EVENTS.<Noun> could be augmented to include a specific object ID for a specific project implementation.

8.4.3 JMS message fields

It is also important to note that some JMS header fields are related to field in the IEC 61968 message header. The following table shows where some JMS header fields may be mapped to IEC 61968-100 header fields as a best practice, but it is not required.

JMS header field	Set by	61968-100 header field
JMSDestination	send or publish method	NA
JMSDeliveryMode	send or publish method	NA
JMSExpiration	send or publish method	NA
JMSPriority	send or publish method	NA
JMSMessageID	send or publish method	MessageID
JMSTimestamp	send or publish method	TimeStamp
JMSCorrelationID	Client	CorrelationID
JMSReplyTo	Client	ReplyAddress
JMSType	Client	NA
JMSRedelivered	JMS provider	NA

9 Security

Security is a key issue for most implementations. Security requirements may be different depending upon the specific integration scenario. Some of the different example scenarios include:

- Intra-application integration of components within a controlled environment
- Inter-application integration within a controlled environment
- Inter-application integration across an enterprise
- Business-to-business integration between trusted partners using a trusted infrastructure
- Extra-enterprise integration, including enterprise application to device integration
- Publicly accessible services

There are many approaches and mechanisms that can be employed, depending upon the requirements.

The use of a SOAP envelope (which can be used with JMS as well as web services) provides benefits, where many products will leverage SOAP Headers for security purposes.

In cases where messages use a public network, security is a significant concern, although there are other situations where security can be a significant concern. Security can include authentication, authorization, encryption and non-repudiation. The details of the implementation of security are outside of the scope of this standard.

There are two basic steps in securing messaging interactions. First, the transport layer is secured. The second step is to secure the message itself. The transport layer is typically secured through the use of Secure Socket Layer (SSL) and Transport Layer Security (TLS). Besides creating a secure communication channel between a client and a service, message exchanges require that security information be embedded within the message itself. This is often the case when a message needs to be processed by several intermediary nodes before

it reaches the target service or when a message must be passed among several services to be processed.

It is important to note that message-level security is very useful in XML document-centric applications, since different sections of the XML document may have different security requirements or be intended for different users.

10 Version control

It is important to recognize that new versions of interfaces may be provided over time, largely as a consequence of:

- Staging of initial implementation
- New requirements
- Upgrades to vendor products

Wherever possible, interfaces will be evolved through augmentation, where a newer version of an interface is compatible with a previous version of an interface. However, this will not always be possible. New versions of interfaces will be manifested by:

- Changes to WSDLs
- Changes to XML Schemas
- Changes to software implementations

In line with OASIS guidelines for namespaces, it is strongly desirable preserve namespaces, especially when definitions have backward compatibility. New namespaces should only be created when a definition cannot be backwards compatible. There are two types of updates in terms of version control:

- Major version update:

  In this case major update has been made in an XSD and its backward compatibility has been broken as a result.

- Minor version update:

  In this case backward compatibility is intact. One example of such minor update is a new element added but as an optional field.

A naming convention for version control of message payloads is proposed here to use XSD targetNamespace, version attribute, and annotation as below:

- targetNamespace=“http://iec.ch/TC57/yyyy/<Payload Type Name>”
- version="<Major version>.<Minor version>".
- Annotation added for detail description such as “Version 1.0 created in 2009/02”.

Here are two examples for major and minor XSD updates, respectively.

In this example a 2009/02 version has a major update, its targetNamespace and version can be changed from:

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2010/EndDeviceControls"
version="1.0">
   <xs:annotation>
      <xs:documentation>
         Major version 1.0 created in 2010/11
      </xs:documentation>
   </xs:annotation>
```

To

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2011/EndDeviceControls"
version="2.0">
   <xs:annotation>
      <xs:documentation>
         Major version 2.0 created in 2011/03
      </xs:documentation>
   </xs:annotation>
```

However if an update is minor, its targetNamespace and version can be changed as follows, from:

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2010/EndDeviceControls"
version="1.0">
   <xs:annotation>
      <xs:documentation>
         Major version 1.0 created in 2010/11
      </xs:documentation>
   </xs:annotation>
```

To the following example with a minor version:

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2010/EndDeviceControls"
version="1.1">
   <xs:annotation>
      <xs:documentation>
         Major version 1.0 created in 2010/11
Minor version 1.1 created in 2010/12
      </xs:documentation>
   </xs:annotation>
```

The “version” attribute does not apply to XML validation against an XSD so its content change (2nd example, minor change) does not break validation against previous XSD version.

For versioning of Message.xsd, similar rules would apply.

Annex A
(normative)

XML schema for common message envelope

The following XML schema is used to define a common message envelope (CME) for request, response and event messages as referenced throughout this standard.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Common Message Specification for IEC 61968 -->
<!--            Change Log                      -->
<!-- 2010/12/15 Added OperationSet to Payload   -->
<!-- 2011/03/09 Corrected FaultMessageType      -->
<!-- 2011/03/09 Baseline for version control    -->
<!-- 2011/03/10 Created type definitions for OperationSet and Operation to improve
compatibility with SoapUI  -->
<!-- 2011/05/06 Removed deprecated verbs, added 'executed'  -->
<!-- 2011/05/06 Changed base namespace to follow WG14 convention of 'iec.ch'  -->
<!-- 2012/02/10 Added relatedObject to Error element  -->
<!-- 2012/02/11 Created a new ObjectType for use in Error element  -->
<!-- 2012/02/11 Removed enumeration for Header.Context  -->
<!-- 2012/02/12 Added note that Error.object.Name elements are deprecated  -->
<!-- 2012/02/12 Added more comments to message elements  -->
<!-- 2012/02/16 Corrected comment for Reply.Error.level  -->
<!-- 2012/02/16 Revised comment for Reply.Error.code  -->
<!-- 2012/02/22 Added ID to Payload for optional use by close/cancel/delete  -->
<!-- 2012/02/22 Extended ID elements to have attributes for idType, idAuthority,
iSmRID -->
<!-- 2012/02/22 Extended ErrorType elements to use ID and relatedID elements, with
deprecation of object -->
<!-- 2012/02/24 Added kind attribute to ID elements in place of iSmRID -->
<!-- 2012/03/19 Corrected ID and relatedID definitions in ErrorType -->
<!-- 2012/03/20 Revised ID elements to use an attribute group -->
<!-- 2012/03/21 Corrected Payload.ID elements -->
<!-- 2012/04/03 Corrected Reply.Error.object.Name -->
<!-- 2012/04/03 Corrected Header.User.Organization made optional -->
<!-- 2012/06/08 Updated IDatts attribute group to include objectType attribute as
string -->
<!-- 2012/10/14 corrections and revisions to annotations for FDIS -->
<xs:schema xmlns="http://iec.ch/TC57/2011/schema/message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://iec.ch/TC57/2011/schema/message"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
  <xs:complexType name="RequestType">
    <xs:annotation>
      <xs:documentation>Request type definition</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:annotation>
        <xs:documentation>Request package is typically used to supply parameters for
'get' requests</xs:documentation>
      </xs:annotation>
      <xs:element name="StartTime" type="xs:dateTime" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Start time of interest</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="EndTime" type="xs:dateTime" minOccurs="0">
        <xs:annotation>
          <xs:documentation>End time of interest</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Option" type="OptionType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Request type specialization</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ID" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Object ID for request</xs:documentation>
        </xs:annotation>
        <xs:complexType>
```

```
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attributeGroup ref="IDatts"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>This can be a CIM profile defined as an XSD with a CIM-
specific namespace This may also be used for custom extensions.</xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ReplyType">
    <xs:annotation>
      <xs:documentation>Reply type definition</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:annotation>
        <xs:documentation>Reply package is used to confirm success or report
errors</xs:documentation>
      </xs:annotation>
      <xs:element name="Result">
        <xs:annotation>
          <xs:documentation>Reply code: OK, PARTIAL or FAILED</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="OK"/>
            <xs:enumeration value="PARTIAL"/>
            <xs:enumeration value="FAILED"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Error" type="ErrorType" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Reply details describing one or more
errors</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ID" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Resulting transaction ID (usually consequence of
create)</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attributeGroup ref="IDatts"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Used for custom extensions</xs:documentation>
        </xs:annotation>
      </xs:any>
      <xs:element name="operationId" type="xs:integer" minOccurs="0">
        <xs:annotation>
          <xs:documentation>The reply.operationId provides the unique identifier of
the Operation for which this reply.result is relevant. Thus, it is  assumed that this
is a partial reply in direct response to one of the operations contained in an
OperationSet request.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="PayloadType">
    <xs:annotation>
      <xs:documentation>Payload container</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:choice>
```

```
        <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>For XML payloads, usually CIM profiles defined using an
XSD in a profile-specific namespace. May also be used for custom
extensions.</xs:documentation>
          </xs:annotation>
        </xs:any>
        <xs:element name="OperationSet" type="OperationSet" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Each operation set is a collection of operations that
may require operational-integrity and/or sequence control.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="Compressed" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>For compressed and/or binary, uuencoded payloads If
compressed, Gzip compression is used.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="ID" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>Used to supply object IDs for cancel/close/delete
operations in cases where they are not otherwise specified using a type-specific
payload</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attributeGroup ref="IDatts"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs:element name="Format" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Hint as to format of payload, e.g. XML, RDF, SVF, BINARY,
PDF, ...</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OperationType">
    <xs:annotation>
      <xs:documentation>For master data set synchronization XML
payloads.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="operationId" type="xs:integer">
        <xs:annotation>
          <xs:documentation>The payload.operation.operationId provides the unique
identifier (within the OperationSet) of the Operation for the purpose of reference in
subsequent messages (e.g. OperationSet reply).</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="noun" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>The payload.operation.##other also identifies the noun,
this element is optionally supplied to simplify processing.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="verb" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>"create", "delete", "change", etc.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="elementOperation" type="xs:boolean" default="false"
minOccurs="0">
        <xs:annotation>
          <xs:documentation>TRUE if the verb is operating at the element level. In
such a case, the verb is to be applied to the elements populated in the
payload.operation.##other below. If omitted, assume FALSE.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:any namespace="##other" processContents="skip" minOccurs="0">
        <xs:annotation>
```

```
          <xs:documentation>An XML payload which carries a CIM profile defined using
an XSD in a profile-specific namespace. Individual payloads are used collectively to
create a series of related operations. See the "enforce" boolean flags in the header
for instructions on how to process these messages.</xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OperationSet">
    <xs:annotation>
      <xs:documentation>Each operation set is a collection of operations that may
require operational-integrity and/or sequence control.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="enforceMsgSequence" type="xs:boolean" minOccurs="0">
        <xs:annotation>
          <xs:documentation>If set to TRUE, the Operation.##other messages must be
processed in the sequence presented. If omitted, assume FALSE.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="enforceTransactionalIntegrity" type="xs:boolean"
minOccurs="0">
        <xs:annotation>
          <xs:documentation>Set to TRUE when all of the Operation.##other messages
must be processed successfully or else the entire message set must be rolled back. If
omitted, assume FALSE.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Operation" type="OperationType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ReplayDetectionType">
    <xs:annotation>
      <xs:documentation>Used to detect and prevent replay attacks</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Nonce" type="xs:string"/>
      <xs:element name="Created" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="UserType">
    <xs:annotation>
      <xs:documentation>User type definition</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="UserID" type="xs:string">
        <xs:annotation>
          <xs:documentation>User identifier</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Organization" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>User parent organization identifier</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="HeaderType">
    <xs:annotation>
      <xs:documentation>Message header type definition</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:annotation>
        <xs:documentation>Message header contains control and descriptive information
about the message.</xs:documentation>
      </xs:annotation>
      <xs:element name="Verb">
        <xs:annotation>
          <xs:documentation>This enumerated list of verbs that can be used to form
message types in compliance with the IEC 61968 standard.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="cancel"/>
            <xs:enumeration value="canceled"/>
            <xs:enumeration value="change"/>
            <xs:enumeration value="changed"/>
```

```
            <xs:enumeration value="create"/>
            <xs:enumeration value="created"/>
            <xs:enumeration value="close"/>
            <xs:enumeration value="closed"/>
            <xs:enumeration value="delete"/>
            <xs:enumeration value="deleted"/>
            <xs:enumeration value="get"/>
            <xs:enumeration value="reply"/>
            <xs:enumeration value="execute"/>
            <xs:enumeration value="executed"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Noun" type="xs:string">
        <xs:annotation>
          <xs:documentation>The Noun of the Control Area identifies the main subject
of the message type, typically a real world object defined in the
CIM.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Revision" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Revision level of the message type.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ReplayDetection" type="ReplayDetectionType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Use to introduce randomness in the message to enhance
effectiveness of encryption</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Context" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Intended context for information usage, e.g. PRODUCTION,
TESTING, TRAINING, ...</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Timestamp" type="xs:dateTime" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Application level relevant time and date for when this
instance of the message type was produced. This is not intended to be used by
middleware for message management.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Source" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Source system or application that sends the
message</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="AsyncReplyFlag" type="xs:boolean" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Indicates whether or not reply should be
asynchronous</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ReplyAddress" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Address to be used for asynchronous replies, typically a
URL/topic/queue.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="AckRequired" type="xs:boolean" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Indicates whether or not an acknowledgement is
required</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="User" type="UserType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>User information of the sender</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="MessageID" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Unique message ID to be used for tracking
messages</xs:documentation>
        </xs:annotation>
```

```
      </xs:element>
      <xs:element name="CorrelationID" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>ID to be used by applications for correlating
replies</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Comment" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Optional comment</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Property" type="MessageProperty" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Message properties can be used to identify information
needed for extended routing and filtering capabilities</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Used to allow custom extensions</xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Message" type="MessageType">
    <xs:annotation>
      <xs:documentation>Common IEC 61968 Message Definition</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="MessageProperty">
    <xs:annotation>
      <xs:documentation>Message properties can be used for extended routing and
filtering</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Value" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="RequestMessage" type="RequestMessageType">
    <xs:annotation>
      <xs:documentation>Request message structure</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ResponseMessage" type="ResponseMessageType">
    <xs:annotation>
      <xs:documentation>Response message structure</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="EventMessage" type="EventMessageType">
    <xs:annotation>
      <xs:documentation>Event message structure. </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="MessageType">
    <xs:annotation>
      <xs:documentation>Generic Message Type</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Header" type="HeaderType"/>
      <xs:element name="Request" type="RequestType" minOccurs="0"/>
      <xs:element name="Reply" type="ReplyType" minOccurs="0"/>
      <xs:element name="Payload" type="PayloadType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="RequestMessageType">
    <xs:annotation>
      <xs:documentation>Request Message Type, which will typically result in a
ResponseMessage to be returned. This is typically used to initiate a transaction or a
query request.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Header" type="HeaderType"/>
      <xs:element name="Request" type="RequestType" minOccurs="0"/>
      <xs:element name="Payload" type="PayloadType" minOccurs="0"/>
```

```
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ResponseMessageType">
    <xs:annotation>
      <xs:documentation>Response MessageType, typically used to reply to a
RequestMessage</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Header" type="HeaderType"/>
      <xs:element name="Reply" type="ReplyType"/>
      <xs:element name="Payload" type="PayloadType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="FaultMessageType">
    <xs:annotation>
      <xs:documentation>Fault Message Type, which is used in cases where the incoming
message (including the header) cannot be parsed</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Reply" type="ReplyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="EventMessageType">
    <xs:annotation>
      <xs:documentation>Event Message Type, which is used to indicate a condition of
potential interest. Note that the Payload may be required in the
future.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="Header" type="HeaderType"/>
      <xs:element name="Payload" type="PayloadType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ErrorType">
    <xs:annotation>
      <xs:documentation>Error Structure</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="code" type="xs:string">
        <xs:annotation>
          <xs:documentation>Defined error code, as defined by IEC 61968-100, related
standards or local implementation</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="level" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Severity level, e.g. INFORM, WARNING, FATAL,
CATASTROPHIC</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="INFORM"/>
            <xs:enumeration value="WARNING"/>
            <xs:enumeration value="FATAL"/>
            <xs:enumeration value="CATASTROPHIC"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="reason" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Description of the error</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="details" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Free form detailed text description of
error</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="xpath" type="xs:QName" minOccurs="0">
        <xs:annotation>
          <xs:documentation>XPath expression to identify specific XML
element</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="stackTrace" type="xs:string" minOccurs="0">
        <xs:annotation>
```

```
          <xs:documentation>Stack trace as generated by software upon
exception</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Location" type="LocationType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Location of exception within software</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ID" minOccurs="0">
        <xs:annotation>
          <xs:documentation>ID of object</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attributeGroup ref="IDatts"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="relatedID" minOccurs="0">
        <xs:annotation>
          <xs:documentation>ID of related object, used in cases where there is an
error between the relationship of two objects</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attributeGroup ref="IDatts"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="object" type="ObjectType" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Deprecated</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="operationId" type="xs:integer" minOccurs="0">
        <xs:annotation>
          <xs:documentation>The reply.operationId provides the unique identifier of
the Operation for which this reply.result.error is relevant. Thus, it is  assumed that
this is an error from one of the operations contained in an OperationSet
request.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OptionType">
    <xs:annotation>
      <xs:documentation>Request options</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="value" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LocationType">
    <xs:annotation>
      <xs:documentation>Process location where error was
encountered</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="node" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Name of the pipeline/branch/route node where error
occurred</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="pipeline" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Name of the pipeline where error occurred (if
applicable)</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="stage" type="xs:string" minOccurs="0">
        <xs:annotation>
```

```
          <xs:documentation>Name of the stage where error occurred (if
applicable)</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ObjectType">
    <xs:annotation>
      <xs:documentation>Used to identify an object of interest</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="mRID" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>A UUID-based name for the object</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Name" type="Name" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>The Name structure is deprecated. It will be completely
removed in the next edition</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="objectType" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Type of object</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="NameType">
    <xs:annotation>
      <xs:documentation>From CIM</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string" minOccurs="0"/>
      <xs:element name="NameTypeAuthority" type="NameTypeAuthority" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Name">
    <xs:annotation>
      <xs:documentation>From CIM</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="NameType" type="NameType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="NameTypeAuthority">
    <xs:annotation>
      <xs:documentation>From CIM</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="FaultMessage" type="FaultMessageType">
    <xs:annotation>
      <xs:documentation>Fault message structure</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:simpleType name="IDKindType">
    <xs:annotation>
      <xs:documentation>ID Kind Type</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="name"/>
      <xs:enumeration value="uuid"/>
      <xs:enumeration value="transaction"/>
      <xs:enumeration value="other"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:attributeGroup name="IDatts">
    <xs:annotation>
      <xs:documentation>ID attribute group</xs:documentation>
    </xs:annotation>
    <xs:attribute name="idType" type="xs:string"/>
```

```
    <xs:attribute name="idAuthority" type="xs:string"/>
    <xs:attribute name="kind" type="IDKindType"/>
    <xs:attribute name="objectType" type="xs:string"/>
  </xs:attributeGroup>
</xs:schema>
```

Annex B
(normative)

Verbs

Table B.1 provides normative definitions of verbs to be used in message headers, as defined by the IEC 61968-1 standard. These are realized as enumerated values within Messsage.xsd.

Table B.1 – Normative definitions of verbs

Verbs	Meaning	Message structure
create	The 'create' verb is used to publish a request to the master system to create a new object. The master system may in turn publish the new object as an event using the verb 'created'. The master system may also use the verb 'reply' to respond to the 'create' request, indicating whether the request has been processed successfully or not.	Request message will include HeaderType and Payload structures.
change	The 'change' verb is used to publish a request to the master system to make a change to an object based on the information in the message. The master system may in turn publish the changed object as an event using the verb 'changed' to notify that the object has been changed since last published. The master system may also use the verb 'reply' to respond to the 'change' request, indicating whether the request has been processed successfully or not.	Request message will include HeaderType, RequestType and optionally Payload structures. The requestType structure will potentially identify specific object IDs.
cancel	The'cancel' verb is used to publish a request to the master system to cancel the object, most commonly in the cases where the object represents a business document. The master system may in turn publish the canceled message as an event using the verb 'canceled' to notify that the document has been canceled since last published. The master system may also use the verb 'reply' to respond to the 'cancel' request, indicating whether the request has been processed successfully or not. The 'cancel' verb is used when the business content of the document is no longer valid due to error(s).	Request message will include HeaderType, RequestType and optionally Payload structures. The requestType structure will potentially identify specific object IDs.
close	The 'close' verb is used to publish a request to the master system to close the object, most commonly in cases where the object represents a business document. The master system may in turn publish the closed message as an event using the verb 'closed' to notify that the document has been closed since last published. The master system may also use the verb 'reply' to respond to the 'close' request, indicating whether the request has been processed successfully or not. The 'close' verb is used when the business document reaches the end of its life cycle due to successful completion of a business process.	Request message will include HeaderType, RequestType and optionally Payload structures. The requestType structure will potentially identify specific object IDs.
delete	The 'delete' verb is used to publish a request to the master system to delete one or more objects. The master system may in turn publish the closed message as an event using the verb 'deleted' to notify that the object has been deleted since last published. The master system may also use the verb 'reply' to respond to the 'delete' request, indicating whether the request has been processed successfully or not. The 'delete' verb is used when the business object should no longer be kept in the integrated systems either due to error(s) or due to archiving needs. However, the master system will most likely retain a historical record of the object after deletion.	Request message will include HeaderType, RequestType and optionally Payload structures. The requestType structure will potentially identify specific object IDs.
execute	This is used when the message is conveying a complex transaction that involves a variety of create, delete and/or change operations through the use of the Payload.OperationSet element..	See Payload.OperationSet in Message.xsd.
get	The 'get' verb is used to issue a query request to the master system to return a set of zero or more objects that meet a specified criteria. The master system may in turn return zero or more objects using the 'reply' verb in a response message.	Request message will include HeaderType and RequestType structures. The requestType structure will potentially identify specific parameters to qualify the request, such as object IDs.

Verbs	Meaning	Message structure
created	The ‘created’ verb is used to publish an event that is a notification of the creation of a object as a result of either an external request or an internal action within the master system of that object. This message type is usually subscribed by interested systems and could be used for mass updates. There is no need to reply to this message type.	Event message will include HeaderType and Payload structures.
changed	The ‘changed’ verb is used to publish an event that is a notification of the change of an objectt as a result of either an external request or an internal action within the master system of that object. This could be a generic change in the content of the object or a specific status change such as “approved”, “issued” etc. This message type is usually subscribed by interested systems and could be used for mass updates. There is no need to reply to this message type.	Event message will include HeaderType and Payload structures.
closed	The ‘closed’ verb is used to publish an event that is a notification of the normal closure of an objectt as a result of either an external request or an internal action within the master system of that object. This message type is usually subscribed by interested systems and could be used for mass updates. There is no need to reply to this message type.	Event message will include HeaderType and Payload structures.
canceled	The ‘canceled’ verb is used to publish an event that is a notification of the cancellation of an objectt as a result of either an external request or an internal action within the master system of that object. This message type is usually subscribed by interested systems and could be used for mass updates. There is no need to reply to this message type.	Event message will include HeaderType and Payload structures.
deleted	The ‘deleted’ verb is used to publish an event that is a notification of the deletion of an object as a result of either an external request or an internal action within the master system of that object. This message type is usually subscribed by interested systems and could be used for mass updates. There is no need to reply to this message type.	Event message will include HeaderType and Payload structures.
executed	This provides for an event that indicates the execution of a complex transaction that uses the Payload.OperationSet element.	See Payload.OperationSet in Message.xsd.
reply	There are two primary usages of the ‘reply’ verb, but in both cases it is only used in response to request messages, whether the pattern used is synchronous or asynchronous. The first usage is to indicate the success, partial success or failure of a transactional request to the master system to create, change, delete, cancel, or close a document. The second usage is in response to a ‘get’ request, where objects of interest may be returned in the response.	Used only for response messages. For responses to transactional requests, the message will contain HeaderType and ReplyType structures. For responses to get requests, the message will contain HeaderType, ReplyType and potentially Payload structures.

Annex C
(normative)

Procedure for strongly typed WSDL generation

C.1 General

The purpose of this annex is to describe the process for the generation of WSDLs and related artifacts. Figure C.1 provides an overview of the process as needed to create and reference specific design artifacts.

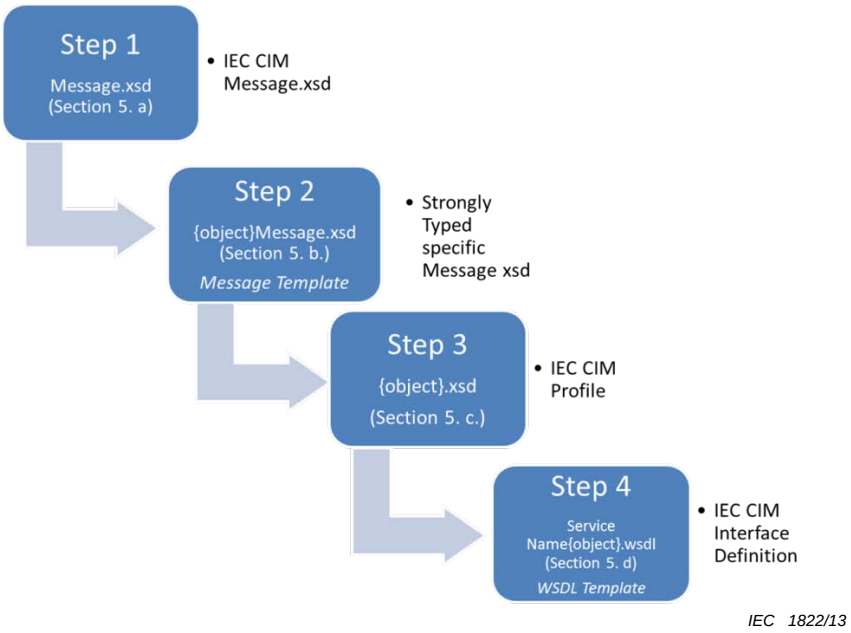


Figure C.1 – Process for WSDL Generation

Using templates, the process allows the creation of a WSDL that defines a set of operations for a given type of object (i.e. a specific noun). Each operation represents a combination of verb and noun. This WSDL will then reference a type specific message envelope, which references both the standard 61968 envelope structure definitions and a profile definition for the given noun.

C.2 WSDL definition steps

Step 1) Sequence diagram based on use case

The sequence diagram of Figure C.2 shows message flow and service providers and consumers based on a use case. It presents messages in sequence base on integration requirements. The following diagram shows an EndDeviceControl message flow from CIS to HeadEnd via an intermediary ESB. CIS, in this case, is a requestor of EndDeviceControl so its message to ESB has a present tense verb “Create”. The message name follows the operation name convention <Verb>+<Information Object> as described in section 8.3.4 . The verb is “Create” and the information object is “EndDeviceControl” in this case.

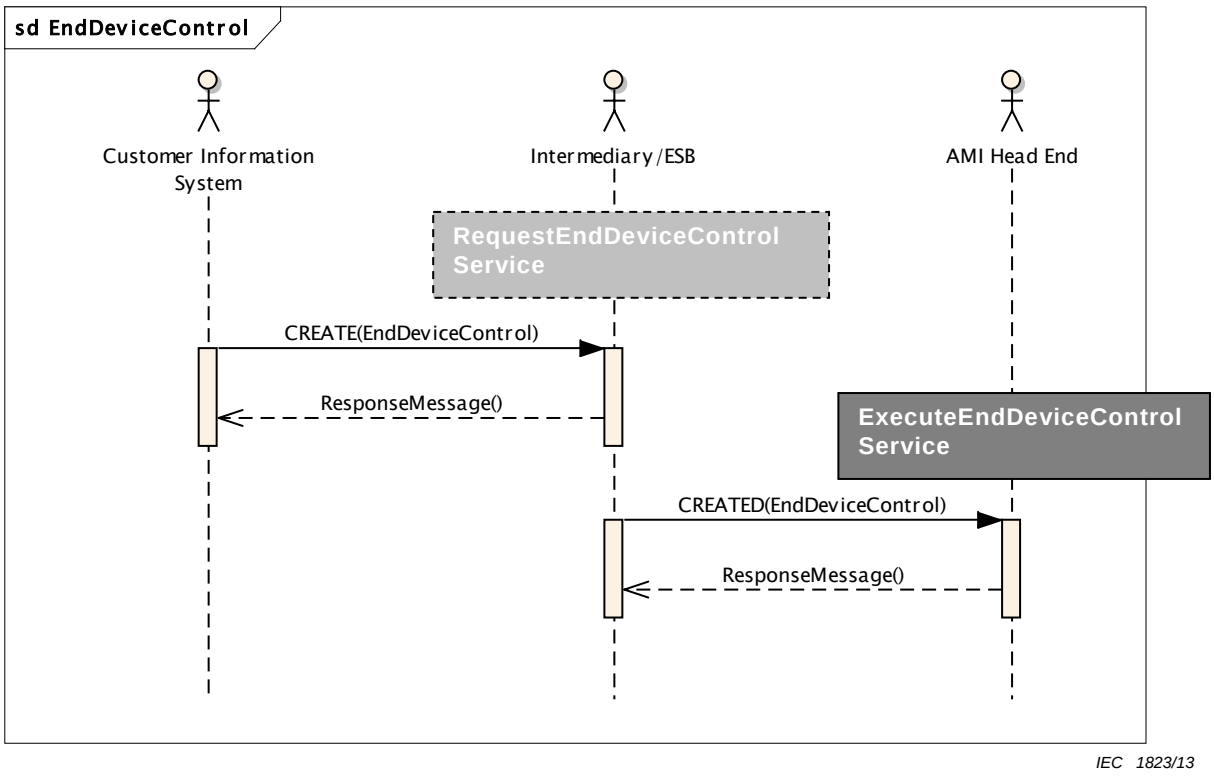


Figure C.2 –Example sequence diagram

Step 2) Service semantics

Based on the sequence diagram, two services are provided for the entire message flow, one by ESB and one by HeadEnd. The naming of the two services are based on their service pattern (see secion 8.3.4) such as the role of service provided by the ESB is to “Request” end device control and the pattern of the service provided by HeadEnd is to “Execute” end device control. Each service will have the same operation as indicated in the sequence diagram as messages. As a result, both services have operation “CreatedEndDeviceControl” but have different service name, one is called “RequestEndDeviceControl” and the other named “ExecuteEndDeviceControl”.

Step 3) Create folder structure

Under the preferred root folder, create an xsd folder for the xsd templates and CIM profiles. The WDSL’s artifacts will be one-level above relative to the xsd folder.

Figure C.3 shows the WSDL folder structure.

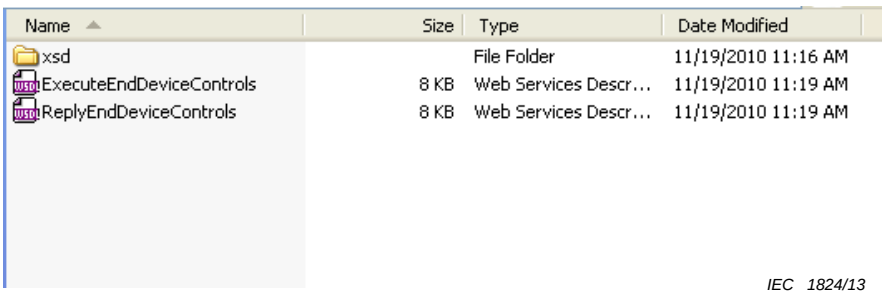


Figure C.3 – WSDL folder structure

The WSDLs are located in the root directory. The referenced XSD files are located in the XSD directory. A CIM profile xsd (EndDeviceControls.xsd), common message.xsd and {object}Message.xsd are located in this folder. The schema location is specified in wsdl:types section as the example for EndDeviceControlMessage listed in Figure C.4.

```
<wsdl:types>

   <xs:schema targetNamespace="http://iec.ch/TC57/2011/schema/message"
              elementFormDefault="qualified">

              <xs:include schemaLocation="xsd/EndDeviceControlsMessage.xsd"/>

   </xs:schema>

</wsdl:types>
```

IEC 1825/13

Figure C.4 – WSDL type definitions

The common message envelope, Message.xsd, can be found in Annex A.

Step 4) Message payload definition & WSDL generation

Service definition follows clause 8. Document literal style is used in SOAP binding. As for a large payload MTOM is can be utilized. The template for a wrapped document style WSDL definition can be found in WSDL Template section in this Appendix.

This example demonstrates how to generate a WSDL for the *Execute* integration pattern. The same process can be used for any integration pattern by replacing Execute with the service naming pattern needed.

a) Copy the Message template xsd (see Message Template subclause in this Annex A) and place it in the correct folder directory

b) Copy the Message template (see Message Template subclause in this Annex C) and replace the {Information_Object_Name} variable with the correct noun.

   – Save as {Information_Object_Name}Message.xsd (i.e. EndDeviceControlsMessage.xsd). This file is saved in the /xsd directory.

     There are two types of Object Message xsd templates, these include:

     - Send/Receive/Reply/Request/Execute
     - Get/Reply

c) Place the IEC CIM Profile xsd into the xsd folder that was created in Step 3.

d) Copy the Execute wsdl template (see WSDL Template subclause in this Annex) and replace the {Pattern_Name} variable with the correct pattern and replace the {Information_Object_Name} variable with the correct noun

   – Save as Execute{Information_Object_Name}.wsdl (i.e. ExecuteEndDeviceControls.wsdl). This file is saved in the root directory.

     There are two types of Object Message xsd templates, these include:

     - Request/Execute
     - Send/Receive/Reply

Note that this is an example for *Execute* pattern but the steps are identical for other service patterns such as Reply for example.

C.3 Message templates

The WSDL will reference a type-specific set of message structures, which in turn leverage the standard type-independent Message.xsd as described in Annex A. Occurrences of {Information_Object_Name} within the template would be replaced with a specific profile name.

Two message templates are provided in Subclause C.3.

1) Message XSD template for:

- **Send/Receive/Reply**
- **Request/Execute**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msg="http://iec.ch/TC57/2011/schema/message"
xmlns:obj="http://iec.ch/TC57/2011/{Information_Object_Name}#"
targetNamespace="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
  <!-- Base Message Definitions -->
  <xs:import namespace="http://iec.ch/TC57/2011/schema/message"
schemaLocation="Message.xsd"/>
  <!-- CIM Information Object Definition -->
  <xs:import namespace="http://iec.ch/TC57/2011/{Information_Object_Name}#"
schemaLocation="{Information_Object_Name}.xsd"/>
  <!-- PayloadType Definition -->
  <xs:complexType name="{Information_Object_Name}PayloadType">
    <xs:sequence>
      <xs:element ref="obj:{Information_Object_Name}"/>
      <xs:element name="OperationSet" type="msg:OperationSet" minOccurs="0"/>
      <xs:element name="Compressed" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>For compressed and/or binary, uuencoded
payloads</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Format" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Hint as to format of payload, e.g. XML, RDF, SVF, BINARY,
PDF, ...</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- Message Types -->
  <!-- RequestMessageType -->
  <xs:complexType name="{Information_Object_Name}RequestMessageType">
    <xs:sequence>
      <xs:element name="Header" type="msg:HeaderType"/>
      <xs:element name="Request" type="msg:RequestType" minOccurs="0"/>
      <xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <!-- ResponseMessageType -->
  <xs:complexType name="{Information_Object_Name}ResponseMessageType">
    <xs:sequence>
      <xs:element name="Header" type="msg:HeaderType"/>
      <xs:element name="Reply" type="msg:ReplyType"/>
      <xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <!-- EventMessageType -->
  <xs:complexType name="{Information_Object_Name}EventMessageType">
    <xs:sequence>
      <xs:element name="Header" type="msg:HeaderType"/>
      <xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
```

```
  <!-- FaultMessageType -->
  <xs:complexType name="{Information_Object_Name}FaultMessageType">
    <xs:sequence>
      <xs:element name="Reply" type="msg:ReplyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Create{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
  <xs:element name="Change{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
  <xs:element name="Cancel{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
  <xs:element name="Close{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
  <xs:element name="Delete{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
  <xs:element name="Created{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
  <xs:element name="Changed{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
  <xs:element name="Canceled{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
  <xs:element name="Closed{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
  <xs:element name="Deleted{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
  <xs:element name="{Information_Object_Name}ResponseMessage"
type="tns:{Information_Object_Name}ResponseMessageType"/>
  <xs:element name="{Information_Object_Name}FaultMessage"
type="tns:{Information_Object_Name}FaultMessageType"/>
</xs:schema>
```

2) Message XSD template for

- **Get and Reply**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msg="http://iec.ch/TC57/2011/schema/message"
xmlns:obj1="http://iec.ch/TC57/2011/{Information_Object_Name}#"
xmlns:obj2="http://iec.ch/TC57/2011/Get{Information_Object_Name}#"
targetNamespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
  <!-- Base Message Definitions -->
  <xs:import namespace="http://iec.ch/TC57/2011/schema/message"
schemaLocation="Message.xsd"/>
  <!-- CIM Information Object Definition -->
  <xs:import namespace="http://iec.ch/TC57/2011/{Information_Object_Name}#"
schemaLocation="{Information_Object_Name}.xsd"/>
  <!-- Remove this Import if there is no "Get" Profile associated with this Object. --
>
  <xs:import namespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}#"
schemaLocation="Get{Information_Object_Name}.xsd"/>
  <!-- RequestType Definition -->
  <xs:complexType name="Get{Information_Object_Name}RequestType">
    <xs:sequence>
      <xs:element name="StartTime" type="xs:dateTime" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Start time of interest</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="EndTime" type="xs:dateTime" minOccurs="0">
        <xs:annotation>
          <xs:documentation>End time of interest</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Option" type="msg:OptionType" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Request type specialization</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="ID" type="xs:string" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Object ID for request</xs:documentation>
```

```
        </xs:annotation>
      </xs:element>
      <!-- Remove this Element if there is no "Get" Profile associated with this
Object. -->
      <xs:element ref="obj2:Get{Information_Object_Name}"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>This can be a CIM profile defined as an XSD with a CIM-
specific namespace</xs:documentation>
        </xs:annotation>
      </xs:any>
    </xs:sequence>
  </xs:complexType>
  <!-- PayloadType Definition -->
  <xs:complexType name="{Information_Object_Name}PayloadType">
    <xs:sequence>
      <xs:element ref="obj1:{Information_Object_Name}" minOccurs="0"/>
      <xs:element name="OperationSet" type="msg:OperationSet" minOccurs="0"/>
      <xs:element name="Compressed" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>For compressed and/or binary, uuencoded
payloads</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Format" type="xs:string" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Hint as to format of payload, e.g. XML, RDF, SVF, BINARY,
PDF, ...</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- Message Types -->
  <!-- RequestMessageType -->
  <xs:complexType name="Get{Information_Object_Name}RequestMessageType">
    <xs:sequence>
      <xs:element name="Header" type="msg:HeaderType"/>
      <xs:element name="Request" type="tns:Get{Information_Object_Name}RequestType"/>
      <xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <!-- ResponseMessageType -->
  <xs:complexType name="{Information_Object_Name}ResponseMessageType">
    <xs:sequence>
      <xs:element name="Header" type="msg:HeaderType"/>
      <xs:element name="Reply" type="msg:ReplyType"/>
      <xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <!-- FaultMessageType -->
  <xs:complexType name="{Information_Object_Name}FaultMessageType">
    <xs:sequence>
      <xs:element name="Reply" type="msg:ReplyType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Get{Information_Object_Name}"
type="tns:Get{Information_Object_Name}RequestMessageType"/>
  <xs:element name="{Information_Object_Name}ResponseMessage"
type="tns:{Information_Object_Name}ResponseMessageType"/>
  <xs:element name="{Information_Object_Name}FaultMessage"
type="tns:{Information_Object_Name}FaultMessageType"/>
</xs:schema>
```

NOTE The following strings need to be replaced for both templates

{Information_Object_Name}

Replaced with CIM profile that is being used. For example, EndDeviceControls. We use the plural form of the information object to avoid collisons within the XSD.

C.4 WSDL templates

This section provides WSDL templates that would be edited in order to create design artifacts for strongly typed web services.

WSDL for 'Get' requests

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   name="Get{Information_Object_Name}"
         targetNamespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}"
         xmlns:tns="http://iec.ch/TC57/2011/Get{Information_Object_Name}"
         xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
         xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"

xmlns:infoMessage="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message">
         <wsdl:types>

         <xs:schema
targetNamespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}"
                        elementFormDefault="qualified">

                  <xs:import
namespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message"
schemaLocation="xsd/Get{Information_Object_Name}Message.xsd"/>

         </xs:schema>

      </wsdl:types>

      <!-- Message Definitions -->
      <wsdl:message name="Get{Information_Object_Name}RequestMessage">
         <wsdl:part name="Get{Information_Object_Name}RequestMessage"
element="infoMessage:Get{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="ResponseMessage">
         <wsdl:part name="ResponseMessage"
element="infoMessage:{Information_Object_Name}ResponseMessage"/>
      </wsdl:message>

      <wsdl:message name="FaultMessage">
         <wsdl:part name="FaultMessage"
element="infoMessage:{Information_Object_Name}FaultMessage"/>
      </wsdl:message>

      <!-- Port Definitions -->
      <wsdl:portType name="Get{Information_Object_Name}_Port">

         <wsdl:operation name="Get{Information_Object_Name}">
            <wsdl:input name="Get{Information_Object_Name}Request"
message="tns:Get{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Get{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Get{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

      </wsdl:portType>

      <wsdl:binding name="Get{Information_Object_Name}_Binding"
type="tns:Get{Information_Object_Name}_Port">

         <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

         <wsdl:operation name="Get{Information_Object_Name}">
```

```
            <soap:operation
soapAction="http://iec.ch/TC57/2011/Get{Information_Object_Name}/Get{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Get{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Get{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Get{Information_Object_Name}Fault">
               <soap:fault name="Get{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
      </wsdl:binding>

      <wsdl:service name="Get{Information_Object_Name}">
         <wsdl:port name="Get{Information_Object_Name}_Port"
binding="tns:Get{Information_Object_Name}_Binding">
            <soap:address
location="http://iec.ch/TC57/2011/Get{Information_Object_Name}"/>
         </wsdl:port>
      </wsdl:service>

</wsdl:definitions>
```

WSDL for Send, Receive, Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
         name="{Send | Receive | Reply}{Information_Object_Name}"
         targetNamespace="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"
         xmlns:tns="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"
         xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
         xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
         xmlns:infoMessage="http://iec.ch/TC57/2011/{Information_Object_Name}Message">

      <wsdl:types>

         <xs:schema targetNamespace="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"
                        elementFormDefault="qualified">

                  <xs:import
namespace="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
schemaLocation="xsd/{Information_Object_Name}Message.xsd"/>

         </xs:schema>

      </wsdl:types>

      <!-- Message Definitions -->

      <wsdl:message name="Created{Information_Object_Name}EventMessage">
         <wsdl:part name="Created{Information_Object_Name}EventMessage"
element="infoMessage:Created{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Changed{Information_Object_Name}EventMessage">
         <wsdl:part name="Changed{Information_Object_Name}EventMessage"
element="infoMessage:Changed{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Closed{Information_Object_Name}EventMessage">
         <wsdl:part name="Closed{Information_Object_Name}EventMessage"
element="infoMessage:Closed{Information_Object_Name}"/>
      </wsdl:message>
```

```
      <wsdl:message name="Canceled{Information_Object_Name}EventMessage">
         <wsdl:part name="Canceled{Information_Object_Name}EventMessage"
element="infoMessage:Canceled{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Deleted{Information_Object_Name}EventMessage">
         <wsdl:part name="Deleted{Information_Object_Name}EventMessage"
element="infoMessage:Deleted{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="ResponseMessage">
         <wsdl:part name="ResponseMessage"
element="infoMessage:{Information_Object_Name}ResponseMessage"/>
      </wsdl:message>

      <wsdl:message name="FaultMessage">
         <wsdl:part name="FaultMessage"
element="infoMessage:{Information_Object_Name}FaultMessage"/>
      </wsdl:message>

      <!-- Port Definitions -->
      <wsdl:portType name="{Information_Object_Name}_Port">

         <wsdl:operation name="Created{Information_Object_Name}">
            <wsdl:input name="Created{Information_Object_Name}Event"
message="tns:Created{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Created{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Created{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Changed{Information_Object_Name}">
            <wsdl:input name="Changed{Information_Object_Name}Event"
message="tns:Changed{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Changed{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Changed{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Canceled{Information_Object_Name}">
            <wsdl:input name="Canceled{Information_Object_Name}Event"
message="tns:Canceled{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Canceled{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Canceled{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Closed{Information_Object_Name}">
            <wsdl:input name="Closed{Information_Object_Name}Event"
message="tns:Closed{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Closed{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Closed{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Deleted{Information_Object_Name}">
            <wsdl:input name="Deleted{Information_Object_Name}Event"
message="tns:Deleted{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Deleted{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Deleted{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

      </wsdl:portType>

      <wsdl:binding name="{Information_Object_Name}_Binding"
type="tns:{Information_Object_Name}_Port">

         <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

         <wsdl:operation name="Created{Information_Object_Name}">
```

```
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Created{Information_Obje
ct_Name}" style="document"/>
            <wsdl:input name="Created{Information_Object_Name}Event">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Created{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Created{Information_Object_Name}Fault">
               <soap:fault name="Created{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Changed{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Changed{Information_Obje
ct_Name}" style="document"/>
            <wsdl:input name="Changed{Information_Object_Name}Event">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Changed{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Changed{Information_Object_Name}Fault">
               <soap:fault name="Changed{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Canceled{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Canceled{Information_Obj
ect_Name}" style="document"/>
            <wsdl:input name="Canceled{Information_Object_Name}Event">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Canceled{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Canceled{Information_Object_Name}Fault">
               <soap:fault name="Canceled{Information_Object_Name}Fault"
use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Closed{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Closed{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Closed{Information_Object_Name}Event">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Closed{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Closed{Information_Object_Name}Fault">
               <soap:fault name="Closed{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Deleted{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Deleted{Information_Obje
ct_Name}" style="document"/>
            <wsdl:input name="Deleted{Information_Object_Name}Event">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Deleted{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Deleted{Information_Object_Name}Fault">
               <soap:fault name="Deleted{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
      </wsdl:binding>

      <wsdl:service name="{Send | Receive | Reply}{Information_Object_Name}">
         <wsdl:port name="{Information_Object_Name}_Port"
binding="tns:{Information_Object_Name}_Binding">
            <soap:address location="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"/>
         </wsdl:port>
```

```
      </wsdl:service>

</wsdl:definitions>
```

NOTE {Send | Receive | Reply} should be replaced with a proper service patter name such as Receive.

WSDL for Request, Execute

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
      name="{Request | Execute}{Information_Object_Name}"
         targetNamespace="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"
         xmlns:tns="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"
         xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
         xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
         xmlns:infoMessage="http://iec.ch/TC57/2011/{Information_Object_Name}Message">

      <wsdl:types>

         <xs:schema targetNamespace="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"
                        elementFormDefault="qualified">

                  <xs:import
namespace="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
schemaLocation="xsd/{Information_Object_Name}Message.xsd"/>

         </xs:schema>

      </wsdl:types>
      <xs:schema
targetNamespace="http://iec.ch/TC57/2011/ExecuteEndDeviceControlsMessage"
                        elementFormDefault="qualified">
                  <xs:import
namespace="http://iec.ch/TC57/2011/EndDeviceControlsMessage"
schemaLocation="xsd/EndDeviceControlsMessage.xsd"/>
                  <!--<xs:include schemaLocation="xsd/EndDeviceControlsMessage.xsd"/>-->

         </xs:schema>

      <!-- Message Definitions -->

      <wsdl:message name="Create{Information_Object_Name}RequestMessage">
         <wsdl:part name="Create{Information_Object_Name}RequestMessage"
element="infoMessage:Create{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Change{Information_Object_Name}RequestMessage">
         <wsdl:part name="Change{Information_Object_Name}RequestMessage"
element="infoMessage:Change{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Close{Information_Object_Name}RequestMessage">
         <wsdl:part name="Close{Information_Object_Name}RequestMessage"
element="infoMessage:Close{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Cancel{Information_Object_Name}RequestMessage">
         <wsdl:part name="Cancel{Information_Object_Name}RequestMessage"
element="infoMessage:Cancel{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Delete{Information_Object_Name}RequestMessage">
         <wsdl:part name="Delete{Information_Object_Name}RequestMessage"
element="infoMessage:Delete{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="ResponseMessage">
```

```
        <wsdl:part name="ResponseMessage"
element="infoMessage:{Information_Object_Name}ResponseMessage"/>
     </wsdl:message>

     <wsdl:message name="FaultMessage">
        <wsdl:part name="FaultMessage"
element="infoMessage:{Information_Object_Name}FaultMessage"/>
     </wsdl:message>

     <!-- Port Definitions -->
     <wsdl:portType name="{Information_Object_Name}_Port">

        <wsdl:operation name="Create{Information_Object_Name}">
           <wsdl:input name="Create{Information_Object_Name}Request"
message="tns:Create{Information_Object_Name}RequestMessage"/>
           <wsdl:output name="Create{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
           <wsdl:fault name="Create{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
        </wsdl:operation>

        <wsdl:operation name="Change{Information_Object_Name}">
           <wsdl:input name="Change{Information_Object_Name}Request"
message="tns:Change{Information_Object_Name}RequestMessage"/>
           <wsdl:output name="Change{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
           <wsdl:fault name="Change{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
        </wsdl:operation>

        <wsdl:operation name="Cancel{Information_Object_Name}">
           <wsdl:input name="Cancel{Information_Object_Name}Request"
message="tns:Cancel{Information_Object_Name}RequestMessage"/>
           <wsdl:output name="Cancel{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
           <wsdl:fault name="Cancel{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
        </wsdl:operation>

        <wsdl:operation name="Close{Information_Object_Name}">
           <wsdl:input name="Close{Information_Object_Name}Request"
message="tns:Close{Information_Object_Name}RequestMessage"/>
           <wsdl:output name="Close{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
           <wsdl:fault name="Close{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
        </wsdl:operation>

        <wsdl:operation name="Delete{Information_Object_Name}">
           <wsdl:input name="Delete{Information_Object_Name}Request"
message="tns:Delete{Information_Object_Name}RequestMessage"/>
           <wsdl:output name="Delete{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
           <wsdl:fault name="Delete{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
        </wsdl:operation>

     </wsdl:portType>

     <wsdl:binding name="{Information_Object_Name}_Binding"
type="tns:{Information_Object_Name}_Port">

        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

        <wsdl:operation name="Create{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Create{Information_Objec
t_Name}" style="document"/>
           <wsdl:input name="Create{Information_Object_Name}Request">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Create{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Create{Information_Object_Name}Fault">
              <soap:fault name="Create{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
```

```
        </wsdl:operation>
        <wsdl:operation name="Change{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Change{Information_Objec
t_Name}" style="document"/>
           <wsdl:input name="Change{Information_Object_Name}Request">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Change{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Change{Information_Object_Name}Fault">
              <soap:fault name="Change{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Cancel{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Cancel{Information_Objec
t_Name}" style="document"/>
           <wsdl:input name="Cancel{Information_Object_Name}Request">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Cancel{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Cancel{Information_Object_Name}Fault">
              <soap:fault name="Cancel{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Close{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Close{Information_Object
_Name}" style="document"/>
           <wsdl:input name="Close{Information_Object_Name}Request">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Close{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Close{Information_Object_Name}Fault">
              <soap:fault name="Close{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Delete{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Delete{Information_Objec
t_Name}" style="document"/>
           <wsdl:input name="Delete{Information_Object_Name}Request">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Delete{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Delete{Information_Object_Name}Fault">
              <soap:fault name="Delete{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
     </wsdl:binding>

     <wsdl:service name="{Request | Execute}{Information_Object_Name}">
        <wsdl:port name="{Information_Object_Name}_Port"
binding="tns:{Information_Object_Name}_Binding">
           <soap:address location="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"/>
        </wsdl:port>
     </wsdl:service>

</wsdl:definitions>
```

NOTE 1 {Request | Execute} should be replaced with a proper service patter name such Execute.

NOTE 2 For all templates, the following string needs to be replaced.

{Information_Object_Name}

Replaced with CIM profile that is being used, for example, EndDeviceControls. The plural form of the information object is used to avoid collisions within the XSD.

Annex D
(normative)

Generic WSDL

The purpose of this annex is to describe a generic WSDL that is not strongly typed to a specific verb and noun combination . Instead, this WSDL provides the ability to convey messages that may use any valid verb/noun and noun combinations, where the common message envelope as defined by Message.xsd is used without modification. The generic WSDL provides three operations:

- Request: to issue requests, where a response may be returned
- Response: to issue asynchronous responses
- PublishEvent: to sent event messages, where the assumption is that an intermediary is responsible for publication of the event to all potentially interested 'listeners'

This approach has the benefit of avoiding the need to construct variations of Message.xsd.

Figure D.1 provides an overview of the operations and messages.

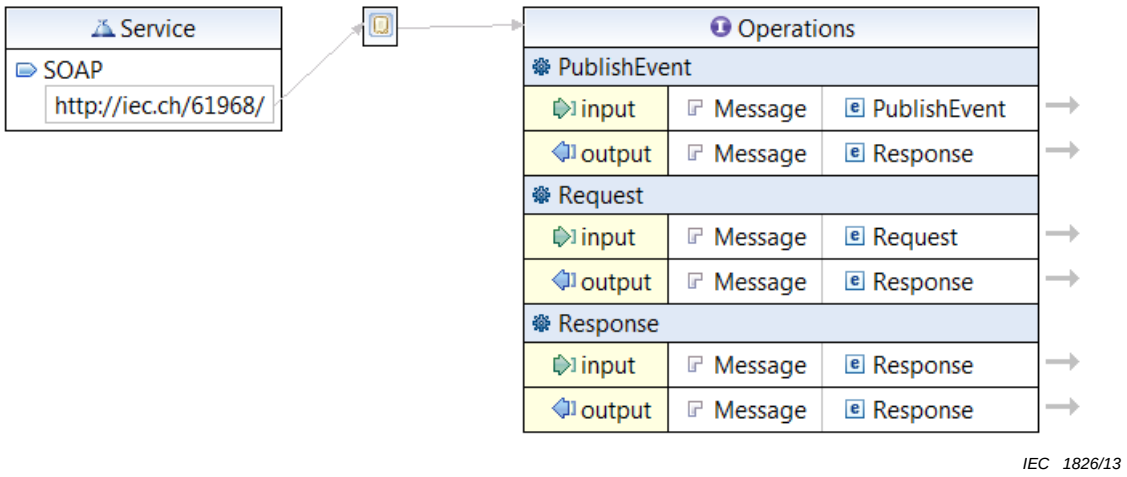


Figure D.1 – Generic WSDL structure

The following is XML that defines the abstract WSDL for the implementation of a generic IEC 61968-100 web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- IEC 61968 WSDL for Generic, Type-Independent Web Services -->
<!-- Uses document wrapped WSDL style -->
<wsdl:definitions xmlns:ns="http://iec.ch/TC57/2011/abstract"
xmlns:ns2="http://iec.ch/TC57/2011/schema/message"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://iec.ch/TC57/2011/abstract">
   <wsdl:types>
      <xsd:schema targetNamespace="http://iec.ch/TC57/2011/schema/message">
         <xsd:include schemaLocation="xsd/Message.xsd"/>
         <xsd:element name="PublishEvent" type="ns2:EventMessageType"/>
         <xsd:element name="Request" type="ns2:RequestMessageType"/>
         <xsd:element name="Response" type="ns2:ResponseMessageType"/>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="EventMessage">
      <wsdl:part name="Message" element="ns2:EventMessage"/>
   </wsdl:message>
```

```
   <wsdl:message name="RequestMessage">
      <wsdl:part name="Message" element="ns2:RequestMessage"/>
   </wsdl:message>
   <wsdl:message name="ResponseMessage">
      <wsdl:part name="Message" element="ns2:ResponseMessage"/>
   </wsdl:message>
   <wsdl:portType name="Operations">
      <wsdl:operation name="PublishEvent">
         <wsdl:input message="ns:EventMessage"/>
         <wsdl:output message="ns:ResponseMessage"/>
      </wsdl:operation>
      <wsdl:operation name="Request">
         <wsdl:input message="ns:RequestMessage"/>
         <wsdl:output message="ns:ResponseMessage"/>
      </wsdl:operation>
      <wsdl:operation name="Response">
         <wsdl:input message="ns:ResponseMessage"/>
         <wsdl:output message="ns:ResponseMessage"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="SOAP" type="ns:Operations">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
                <!-- Operation for publication of events -->
      <wsdl:operation name="PublishEvent">
         <soap:operation soapAction="http://iec.ch/61968/PublishEvent"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
                <!-- Operation for request/reply interactions -->
      <wsdl:operation name="Request">
         <soap:operation soapAction="http://iec.ch/61968/Request"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
                <!-- Operation for asynchronous responses -->
      <wsdl:operation name="Response">
         <soap:operation soapAction="http://iec.ch/61968/Response"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="Service">
      <wsdl:port name="SOAP" binding="ns:SOAP">
         <soap:address location="http://iec.ch/61968/"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Annex E
(informative)

AMQP

The Advanced Message Queueing Protocol (AMQP[3]) defines an open 'wire' protocol for queue-based messaging. Products that use the AMQP protocol are becoming common in the marketplace, as well as the availability of open source offerings. This is contrasted by, but also complementary of the fact that JMS provides a standardized API but JMS implementations do not have a standardized wire protocol.

The use of AMQP with IEC 61968-100 is identical to the use of JMS using queues in cases where the clients use the JMS API. From the perspective of the client application code, the use of AMQP may be completely transparent. The application data conveyed within messages is simply conveyed using the common message envelope as defined by Message.xsd. The same general approach can be taken with other AMQP APIs.

3 Details on AMQP are available at http://amqp.org.

Annex F
(informative)

Payload Compression Example

The purpose of this annex is to provide an example of the code required to compress and encode a payload, where the payload is then passed as the contents of the Payload/Compressed element. Payload compression can be used for any messaging technology, including JMS, generic web services and strongly typed web services.

The following is a Java class example that leverages commonly used classes for compression and base64 encoding.

```
package soap.test;

import java.io.*;
import java.util.zip.*;
import org.apache.commons.codec.binary.Base64;

public class CompressionClientCompressandEncode{
     private byte[] input = null;

     public CompressionClientCompressandEncode(String xmlInput) {
       this.input = xmlInput.getBytes();
     }

     public CompressionClientCompressandEncode(byte[] input) {
       this.input = input;
     }

     // Returns the Compressed and Encoded byte[]

     private byte[] compressAndEncode() throws Exception {

       // GZIP the contents..
         ByteArrayOutputStream outstream =
           new ByteArrayOutputStream();
         GZIPOutputStream gzipOutstream =
           new GZIPOutputStream(outstream);
         gzipOutstream.write(input);
         gzipOutstream.close();

         // Encode the compressed byte array from the stream
         byte[] b =
           Base64.encodeBase64(outstream.toByteArray());
         return b;
     }
}
```

The following program (when combined with the above class) shows example usage, where sample input XML is zipped and encoded using the above code, and then decoded and unzipped to get the original input:

```
public static void main ( String args[]) {
  String s = "<root>" +
             "<name>raju</name>" +
```

```
                "</root>";

        System.out.println("Original Xml");
        System.out.println(s);

        byte[] b = s.getBytes();

        CompressionClientCompressandEncode cc =
            new CompressionClientCompressandEncode(b);

        try {

              byte[] compressedAndEncoded = cc.compressAndEncode();

                      System.out.println("Compressed And Encoded");
                      System.out.println(
                        new String(compressedAndEncoded));

                    // Validate ..
                    byte[] unencoded =
                          Base64.decodeBase64(compressedAndEncoded);

                    ByteArrayInputStream bi1 =
                          new ByteArrayInputStream(unencoded);
                    GZIPInputStream g1 = new GZIPInputStream(bi1);


                    System.out.println("Unencoded and Uncompressed..");
                    for (int c = g1.read(); c != -1; c = g1.read()) {
                        System.out.write(c);
                    }
                    System.out.flush();

              } catch (Exception e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
              }
    }
```

An important note is that if compression is used, either both the source and target systems shall support compression, or any mismatches be addressed by intermediary processes within the ESB. The Gzip compression algoroithm should be used, as is demonstrated in this example.

Annex G
(informative)

XMPP

The Extensible Messaging and Presence Protocol (XMPP) defines an application level protocol for near real-time communications using XML. XMPP is standardized by IETF RFCs 6120, 6121 and 6122. There are many functional similarities to JMS, and as a consequence it is possible to map IEC 61968-100 onto XMPP.

Using XMPP, clients connect to an XMPP server just as JMS clients would connect to a JMS server. Messages are sent using XML 'stanzas', where each stanza conveys an XML element that is logically a fragment of an XML document that is representative of a session. There are three fundamental types of stanzas:

- <message> – used for pushing messages
- <iq> – Info/Query, used for request/response patterns, where an IQ stanza may be of one of four types: get, set, result or error
- <presence> – Used to convey the status of a contact

The IEC 61968-100 message envelope can be placed within an XMPP stanza. The type of stanza that should be used is dependent upon the messaging pattern. Request/reply patterns should use the <iq> stanza type. Publish/subscribe message patterns should use the <message> stanza type. The following is an example of an IEC 61968-9 event message being conveyed within an XMPP stanza.

```
<message from='meter76943@myUtility.com' to='headend@myUtility.com'>
        <ns0:Message xmlns:ns0="http://www.iec.ch/TC57/2011/schema/message">
          <ns0:Header>
            <ns0:Verb>created</ns0:Verb>
            <ns0:Noun>EndDeviceEvents</ns0:Noun>
            <ns0:Revision>1</ns0:Revision>
            <ns0:Timestamp>2009-11-04T18:52:50.001-05:00</ns0:Timestamp>
            <ns0:Source>Metering System</ns0:Source>
          </ns0:Header>
          <ns0:Payload>
            <ns1:EndDeviceEvents xmlns:ns1="http://iec.ch/TC57/2011EndDeviceEvents#">
              <ns1:EndDeviceEvent ref='3.26.1.185'>
                <ns1:mRID>76943</ns1:mRID>
                <ns1:createdDateTime>2009-11-04T18:52:50.001-05:00</ns1:createdDateTime>
                <ns1:description>Power off alarm</ns1:description>
                <ns1:severity>1</ns1:severity>
                <ns1:Assets>
      <ns1:mRID>AC761473800C7B0417481114A11348C16111911121B46C016BF012C68121106</ns1:mRID>
                </ns1:Assets>
                  </ns1:EndDeviceEvent>
              </ns1:EndDeviceEvents>
          </ns0:Payload>
        </ns0:Message>
</message>
```

XMPP messages are addressed using the 'from' and 'to' attributes within the <message> and <iq> elements. Where JMS topic or queue names would be used for addressing with JMS, addressing for XMPP is defined by IETF RFC 6122.

Bibliography

IEC 61968-9, *Application integration at electric utilities – System interfaces for distribution management – Part 9: Interfaces for meter reading and control*

IEC 61968-13, *Application integration at electric utilities – System interfaces for distribution management – Part 13: CIM RDF Model exchange format for distribution*

IEC 61970-452, *Energy management system application program interface (EMS-API) – Part 452: CIM Statis Transmission Network Model Profiles*

IEC 61970-453[4], *Energy management system application program interface (EMS-API) – Part 453: Diagram Layout Profile*

IEC 62361-100: *Harmonization of Quality Codes across TC 57 – Part 100: Naming and design rules for CIM profiles to XML schema mapping*[5]

EIP: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison Wesley, October 2003; ISBN-10: 0321200683, ISBN-13: 978-0321200686 (http://www.eaipatterns.com).

SOAP over Java Message Service Proposed Recommendation: *http://www.w3.org/TR/soapjms/*

XSL: Extensible Stylesheet Language (XSL): www.w3.org/TR/xsl/.

XPATH: XML Path Language (XPATH): *www.w3.org/TR/xpath/*.

IETF RFC 1738, *Univeral Resource Locators*

IETF RFC 2616, *Hyper-Text Transport Protocol 1.1*

IETF RFC 4122, *A Universally Unique Identifier (UUID) URN Namespace*

Extensible Markup Language (XML): *http://www.w3.org/TR/REC-xml*

XML Schema: *http://www.w3.org/XML/Schema*

Simple Object Access Protocol (SOAP) 1.2: *http://www.w3.org/TR/soap12-part1/*

WS-I Basic Profile Version 1.0: *http://www.oasis.org*

JMS API: JSR-914, *Java Message Service (JMS) API*

---

4 To be published.

5 To be published.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

INTÉGRATION D'APPLICATIONS POUR LES SERVICES ÉLECTRIQUES – INTERFACES SYSTÈME POUR LA GESTION DE DISTRIBUTION –

Partie 100: Profils de mise en oeuvre

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale CEI 61968-100 a été établie par le comité d'études 57 de la CEI: Gestion des systèmes de puissance et échanges d'informations associés.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
57/1358/FDIS	57/1382/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61968, publiées sous le titre général *Intégration d'applications pour les services électriques – Interfaces système pour la gestion de distribution*, peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

INTRODUCTION

Cette partie de la CEI 61968 a pour objet de définir un ensemble de profils de mise en œuvre pour la norme CEI 61968 avec des technologies régulièrement utilisées pour l'intégration en entreprise. De manière spécifique, le présent document décrit la façon dont les charges de messages définies par les Parties 3-9 de la norme CEI 61968 sont acheminées via des services Web et Java Messaging System. Des recommandations sont également fournies pour l'utilisation des technologies Enterprise service Bus (ESB). L'objectif est de fournir des détails qui suffisent à rendre les mises en œuvre de la norme CEI 61968 interopérables. Le présent document est en outre destiné à décrire les modèles d'intégration, ainsi que les méthodologies qui peuvent être utilisées avec des technologies d'intégration existantes et à venir.

La série des normes CEI 61968 est prévue pour faciliter *l'intégration inter-applications,* par opposition à *l'intégration intra-applications*. L'intégration intra-applications est destinée aux programmes d'un même système, qui communiquent habituellement les uns avec les autres via des intergiciels (middleware) intégrés dans leur environnement d'exécution sous-jacent et tendent à être optimisés pour des connexions proches, en temps réel et synchrones, ainsi que des interrogations / réponses interactives ou des modèles de communication conversationnels. La CEI 61968, en revanche, est prévue pour supporter l'intégration inter-applications d'une entreprise de distribution qui a besoin de relier des systèmes disparates existants ou futurs (applications héritées ou achetées), chacun supporté par des environnements d'exécution différents. Par conséquent, ces normes d'interface sont appropriées pour les applications faiblement couplées avec une plus grande hétérogénéité dans le langage, les systèmes d'exploitation, les protocoles et des outils de gestion. Cette série de normes, qui est destinée à être mise en œuvre avec des services d'intergiciel, qui échangent des messages parmi des applications, complétera, mais ne remplacera pas, les entrepôts de données de l'entreprise de distribution, les passerelles de base de données et les archives opérationnelles.

La présente norme est basée sur le rapport technique EPRI 1018795 et sur d'autres contributions.

La série CEI 61968, prise dans son ensemble, définit les interfaces pour les éléments principaux d'une architecture d'interface pour les systèmes de distribution dans une entreprise de distribution d'électricité. La Partie 1: Architecture des interfaces et recommandations générales, identifie et établit des exigences pour des interfaces normalisées, basées sur un Modèle d'Interface de Référence (IRM, Interface Reference Model). Les Parties 3 à 9 de la norme CEI 61968 définissent les interfaces relatives à chacune des principales activités fonctionnelles décrites par le Modèle d'Interface de Référence.

Comme décrit dans la norme CEI 61968, il existe plusieurs composants d'application distribués utilisés par l'entreprise de distribution pour contrôler les réseaux de distribution électriques. Ces fonctions incluent la surveillance et la commande des équipements de fourniture d'énergie, les processus de gestion qui assurent la fiabilité du système, la gestion de la tension électrique, la gestion de la demande, la gestion des interruptions de service, la gestion des travaux, la mise en correspondance automatisée, le relevé de compteurs, la commande des compteurs et la gestion des équipements. Cet ensemble de normes est limité à la définition des interfaces et est indépendant de sa mise en œuvre. Il pourvoit à l'interopérabilité entre les différents systèmes informatiques, plates-formes et langages de programmation. Les processus et les technologies utilisés pour mettre en application une fonctionnalité se conformant à ces interfaces sont considérés comme étant hors du domaine d'application de ces normes; seule l'interface proprement dite est spécifiée dans ces normes.

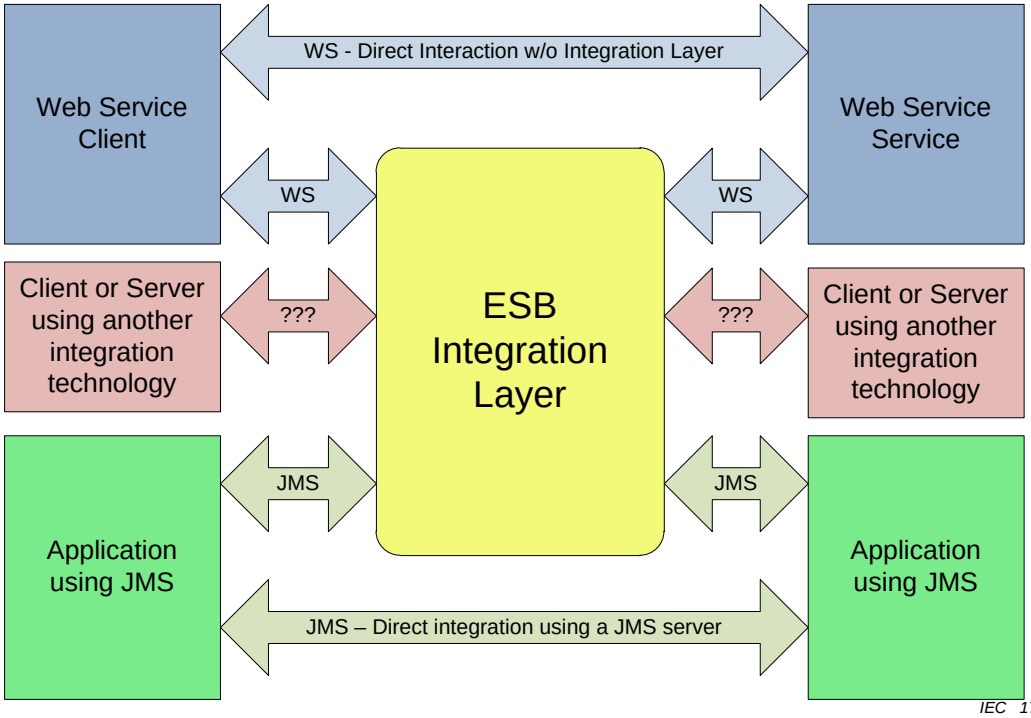
INTÉGRATION D'APPLICATIONS POUR LES SERVICES ÉLECTRIQUES – INTERFACES SYSTÈME POUR LA GESTION DE DISTRIBUTION –

Partie 100: Profils de mise en oeuvre

1 Domaine d'application

Cette partie de la CEI 61968 spécifie un profil de mise en œuvre pour l'application des autres parties de la CEI 61968 avec des technologies d'intégration communes, dont JMS et les services Web. La présente Norme internationale fournit également des lignes directrices pour l'utilisation des technologies Enterprise Service Bus (ESB). Cela fournit un moyen de dériver les mises en œuvre interopérables de la CEI 61968-3 à la CEI 61968-9. Dans le même temps, la présente Norme internationale peut être utilisée au-delà des échanges d'informations définis par la CEI 61968, par exemple pour l'intégration des systèmes du marché ou l'intégration générale d'entreprise.

Le Figure 1 tente d'apporter une vue d'ensemble du domaine d'application, où les messages compatibles CEI 61968 sont acheminés via des services Web ou JMS. Grâce à une couche d'intégration ESB, l'initiateur d'un échange d'informations peut utiliser les services Web, le récepteur peut utiliser JMS et vice versa. La couche d'intégration prend également en charge un ou plusieurs échanges d'informations à l'aide de modèles d'intégration de type "publier/souscrire" (publish/subscribe) et de fonctionnalités clés telles que les garanties de livraison.



Légende

Anglais	Français
Web Service Client	Client Web Service
WS – Direct Interaction w/o Integration Layer	WS – Interaction directe sans couche d'intégration
Web Service Service	Service Web Service
Client or Server using another integration	Client ou serveur avec une autre technologie

Anglais	Français
technology	d'intégration
ESB Integration Layer	Couche d'intégration ESB (Enterprise Service Bus)
Application using JMS	Application qui utilise JMS
JMS – Direct integration using a JMS server	JMS – Intégration directe avec un serveur JMS

Figure 1 – Vue d'ensemble du domaine d'application

Le domaine d'application du présent document inclut spécifiquement les éléments suivants:

- modèles d'intégration qui prennent en charge les échanges d'informations CEI 61968
- conception des interfaces à utiliser pour les services Web fortement typés
- conception des interfaces à utiliser pour les services Web génériques
- conception des interfaces qui utilisent JMS
- définition d'artefacts de conception standard et de modèles associés
- reconnaissance que des technologies autres que JMS et les services Web peuvent être utilisées pour l'intégration de cette norme (avec des exemples spécifiques et les recommandations associées décrites dans les annexes)

Ce profil peut également s'appliquer à des problèmes d'intégration non couverts par le domaine d'application de la CEI 61968.

Il est important de noter que d'autres profils de mise en œuvre peuvent potentiellement être définis pour la CEI 61968 et que la présente ne couvre pas le seul profil de mise en œuvre possible. Ce profil peut en outre être adapté pour satisfaire à des besoins spécifiques de projets d'intégration particuliers.

Le domaine d'application du présent document ne porte en outre pas sur les spécifications de détails de mise en œuvre à utiliser pour la sécurité

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 60050-300, *Vocabulaire Electrotechnique International – Mesures et appareils de mesure électriques et électroniques – Partie 311: Termes généraux concernant les mesures – Partie 312: Termes généraux concernant les mesures électriques – Partie 313: Types d'appareils électriques de mesure – Partie 314: Termes spécifiques selon le type d'appareil*

CEI 61968-1, *Intégration d'applications pour les services électriques – Interfaces système pour la gestion de distribution – Partie 1: Architecture des interfaces et recommandations générales*

CEI/TS 61968-2, *Application integration at electric utilities – System interfaces for distribution management – Part 2: Glossary* (disponible en anglais seulement)

CEI 61968-11, *Intégration d'applications pour les services électriques – Interfaces système pour la gestion de distribution – Partie 11: Extensions du modèle d'information commun (CIM) pour la distribution*

CEI 61970-301, *Interface de programmation d'application pour système de gestion d'énergie (EMS-API) – Partie 301: Base de modèle d'information commun (CIM)*

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

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

3 Termes, définitions et abréviations

3.1 Termes et définitions

Pour les besoins de la présente spécification, les termes et définitions donnés dans la CEI 60050-300, la CEI/TS 61968-2, la CEI 62051 et la CEI 62055-31 s'appliquent.

3.2 Abréviations

Les termes et abréviations suivants sont utilisés dans le présent 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
CEI	Commission Electrotechnique Internationale
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
W3C	World-Wide Web Consortium
WS	Web Services (Services Web)
WS-*	Web Services standards (Normes de services Web)

WS-I	Web Services Interoperability (Interopérabilité des services Web)
WSDL	Web Services Definition Language
XML	eXtensible Markup Language
XSD	XML Schema (Schéma XML)
XSL	XML Stylesheet Language

3.3 Terminologie des technologies d'intégration communes

3.3.1 Généralités

En cas de différence entre les définitions de cette norme et les définitions contenues dans d'autres normes CEI référencées, les définitions contenues dans la CEI/TS 61968-2 doivent être prioritaires par rapport aux définitions répertoriées ailleurs et les définitions contenues dans le présent document doivent être prioritaires par rapport aux définitions répertoriées dans la CEI/TS 61968-2.

3.3.2 Enterprise Service Bus (ESB)

Un Enterprise Service Bus (ESB) (Bus de service d'entreprise) est un type d'architecture logicielle utilisé comme couche d'intégration. Cette architecture est généralement mise en œuvre par des technologies disponibles dans une catégorie de produits d'infrastructure d'intergiciel (middleware), qui s'appuient généralement sur des normes reconnues et qui fournissent des services essentiels pour des architectures plus complexes via un moteur de messagerie spécifique à un événement et basé sur des normes (le bus).

Un ESB fournit généralement une couche d'abstraction en plus de la mise en œuvre d'un système de messagerie d'entreprise, ce qui permet aux architectes d'intégration d'exploiter la valeur de la messagerie sans avoir à écrire du code. Contrairement à l'approche EAI (Enterprise Application Integration) plus classique, définie par une pile monolithique dans une architecture hub-and-spoke, l'ESB se fonde sur des fonctions de base décomposées selon leurs éléments constitutifs, avec un déploiement distribué si nécessaire, travaillant en harmonie.

Un ESB ne met pas en œuvre une architecture SOA (Service Oriented Architecture – architecture orientée service), mais fournit les éléments avec lesquels une telle mise en œuvre peut avoir lieu.

3.3.3 Java Messaging Service (JMS)

L'API Java Message Service (JMS) (Service de messagerie Java) est une API intergiciel Java Message Oriented (orienté message java) qui permet d'envoyer des messages entre plusieurs clients. JMS prend en charge des modèles de messagerie "demande/réponse" (request/reply), "publish/subscribe"(publication/abonnement) et "point-to-point"(point à point). Dans le cadre de la plate-forme Java, version Entreprise, JMS est défini par une spécification développée sous le processus Java Community Process JSR 914. Il est important de noter que certains fournisseurs de produits ESB proposent des liens de JMS vers les langages C, C++ et/ou C#. JMS est alors une fausse appellation. Lorsque le protocole de câblage est différent entre les mises en œuvre JMS, il est souvent trivial d'établir des liaisons entre différentes mises en œuvre JMS.

3.3.4 Service-Oriented Architecture (SOA)

Service Oriented Architecture (SOA) (Architecture orientée service) est un type d'architecture de système informatique qui permet de créer et d'utiliser des processus métiers, appelés services, dans l'ensemble du cycle de vie. De plus, SOA définit et provisionne l'infrastructure informatique afin de permettre à différentes applications d'échanger des données et de participer à ces processus métiers. Ces fonctions sont faiblement couplées avec les systèmes d'exploitation et les langages de programmation qui sous-tendent les applications. SOA sépare les fonctions en unités distinctes (les services), qui peuvent ensuite être distribuées

sur un réseau et qui peuvent être combinées et réutilisées dans le but de créer des applications pour l'entreprise. Ces services, pour communiquer entre eux, transfèrent des données d'un service à l'autre ou coordonnent une activité entre plusieurs services. Pour beaucoup, les concepts SOA ont été construits à partir d'anciens concepts liés à l'informatique distribuée et à la programmation modulaire.

3.3.5 Event-Driven Architecture (EDA)

Event Driven Architecture (EDA) (Architecture orientée évènement) est un modèle d'architecture logicielle qui favorise la production, la détection et la consommation d'événements. Un événement est toute modification d'état d'intérêt potentiel. Dans un EDA, les événements sont transmis entre des services et des composants logiciels faiblement couplés, généralement via des modèles de messageries de type "publish/subscribe". EDA complète SOA; en effet, SOA 2.0 est également appelé SOA "en fonction des événements". EDA est essentiel pour toute une gamme de modèles d'intelligence économique, dont des modèles complexes de traitement des événements.

3.3.6 Simple Object Access Protocol (SOAP)

SOAP (Simple Object Access Protocol – Protocole d'accès à des objets simples) est une norme qui définit le formatage des messages XML. SOAP sert de couche de base à la pile de protocoles des services Web. SOAP est en outre fréquemment utilisé dans JMS. Les types de transport communs pour SOAP incluent les transports HTTP, HTTPS et JMS propriétaires. SOAP est également appelé "Service Oriented Architecture Protocol" (Protocole d'architecture orientée service). SOAP est une recommandation W3C.

SOAP 1.1 et SOAP 1.2 sont deux versions fréquemment utilisées. Il convient d'utiliser SOAP 1.2 pour les nouvelles intégrations ou interfaces si cela est possible.

3.3.7 Web Services (WS)

Un Service Web (WS, Web Service) est défini par le W3C comme étant un "système logiciel conçu pour prendre en charge une interaction machine-à-machine interopérable via un réseau". Les services Web ne sont souvent que des API Web auxquelles on peut accéder via un réseau comme Internet et que l'on peut exécuter sur un système distant qui héberge les services demandés.

La définition de service Web W3C englobe de nombreux systèmes; toutefois, en règle générale, le terme fait référence aux clients et aux serveurs qui communiquent à l'aide de messages XML qui suivent la norme SOAP. Une hypothèse commune dans le champ et la terminologie: il existe également une description lisible par une machine des opérations prise en charge par le serveur et écrite au format WSDL (Web Services Description Language). Cela n'est par une exigence pour un "endpoint" (*point d'extrémité)* SOAP, mais il s'agit d'une condition préalable pour la génération automatisée de code côté-client dans de nombreux outils de développement Java et .Net.

Lorsque les services Web ont deux styles principaux, centrés sur les documents et centrés sur RPC, cette spécification est centrée sur les documents, ce qui est une aide pour SOA et pour le transport des charges utiles CEI 61968. Pour une interopérabilité accrue, la forme conditionnée (wrapped) du document est requise.

Il est important de noter que les services Web ne prennent pas facilement en charge les messageries "publish/subscribe" sauf si des mécanismes tels que WS-Eventing sont utilisés. Toutefois, cette spécification décrit également une approche pour que l'ESB route les messages de façon asynchrone vers les abonnés configurés.

Cette norme décrit deux approches de l'utilisation des services Web, où les opérations WSDL sont génériques ou fortement typées. Les services Web génériques ont des WSDL et des opérations associées qui sont définies de manière indépendante de type charge utile. Les

services Web fortement typés sont définis lorsque des WSDL et opérations sont définis individuellement pour des types de charge utile spécifiques.

Ceci fournit également des cadres automatisées pour la validation par le serveur du contenu de message basé sur des schémas de message contenus dans le document WSDL.

3.3.8 Web Services Definition Language (WSDL)

Le langage WSDL (Web Services Definition Language – Langage de définition des services Web) est un langage XML utilisé pour décrire les services Web. WSDL est souvent utilisé conjointement avec SOAP et un schéma XML afin de fournir des services Web sur Internet ou via un intranet. WSDL est une recommandation W3C.

WSDL 1.1 et WSDL 2.0 sont deux versions fréquemment utilisées. WSDL 2.0 prend mieux en charge l'interopérabilité entre les mises en œuvre Java et .Net.

Les modèles WSDL fournis dans ce document sont basés sur WSDL 1.1. WSDL 1.1 est donc une exigence. Pour mieux prendre en charge l'interopérabilité entre diverses mises en œuvre, tous les documents WSDL sont conformes au Profil de base 1.1 de WS-I.

3.3.9 XML Schema (XSD)

XML Schema, publié sous forme de recommandation W3C en mai 2001, est un des nombreux langages de schéma XML. Il a été le premier langage de schéma XML distinct à atteindre le statut Recommandation W3C.

Comme tous les langages de schéma XML, XML Schema peut être utilisé pour exprimer un schéma: un jeu de règles auxquelles un document XML doit se conformer pour être considéré comme "valide" selon ce schéma. Pourtant, contrairement à la plupart des autres langages de schéma, XML Schema a également été conçu dans le but de voir la détermination de la validité d'un document produire un ensemble d'informations adhérant à des types de données spécifiques. Une instance de XML Schema est une Définition de schéma XML (XSD) et porte généralement le nom d'extension ".xsd". Le langage lui-même est parfois nommé XSD de manière informelle.

Il est important de noter que les charges utiles (payload) du 61 968 sont définies au moyen de schémas XML. Ces schémas XML fournissent des caractéristiques normatives pour les charges utiles de l'application transmises au moyen de cette norme.

3.3.10 Representational State Transfer (REST)

Representational State Transfer (REST) (Transfert de l'état représentationnel) est une alternative à l'utilisation de services Web basés sur SOAP. REST en soi n'est pas une norme actuellement, mais plutôt un type d'architecture.

REST tire profit de HTTP, où chaque URL est la représentation d'un objet donné. Les interfaces peuvent être définies en termes de charges utiles XML pour ce qui concerne les demandes et les réponses. Aucun WSDL n'est à utiliser pour REST. De plus, les documents XML utilisés pour les demandes et réponses ne nécessitent pas d'être définis grâce à XSD, bien que XML soit communément conforme à un XSD.

Dans REST, un objet peut être récupéré (lu) grâce à une commande HTTP GET. De même, une commande HTTP POST est utilisée pour créer un objet, une commande HTTP PUT est utilisée pour modifier un objet et une commande HTTP DELETE est utilisée pour supprimer un objet.

REST n'est décrit ici qu'à des fins d'exhaustivité; il peut en effet être rencontré par un projet d'intégration. La présente norme ne fait toutefois aucune recommandation spécifique pour les mises en correspondance. REST peut être pris en charge à l'avenir.

3.3.11 Queue

Une file d'attente est une construction prise en charge par de nombreux produits de messagerie permettant de fournir une messagerie fiable avec garantie de livraison. Une API standard comprenant des modèles de messagerie basés sur les files d'attente est fournie par JMS. AMQP définit également un protocole ouvert de messagerie basée sur les files d'attente.

3.3.12 Thème

Un thème (topic) est une construction prise en charge par de nombreux produits de messagerie pour activer des modèles de messagerie de type publier/souscrire dans lesquels il peut y avoir potentiellement de nombreux consommateurs pour un message qui a été envoyé à un thème donné par un éditeur. Les thèmes sont habituellement utilisés comme destination des messages d'évènement. Les thèmes sont directement pris en charge par JMS.

3.3.13 Destination de message

Une destination de message est l'adresse cible d'un message, qu'il s'agisse d'une demande, d'une réponse ou d'un message d'évènement. Avec JMS, la destination peut être un thème ou une file d'attente. Avec des mécanismes HTTP tels que les services Web, la destination est indiquée sous la forme d'une URL.

3.3.14 Demande (Request)

Une demande est un message envoyé par un client (ou source) vers un serveur (ou cible) pour lequel une réponse est attendue. La demande peut être à la fois une requête (où les données sont renvoyées par la cible) ou une transaction (où les données sont modifiées dans la cible). Une demande utilise des verbes tels que "get" (obtenir), "create" (créer), "change" (modifier), "delete" (supprimer), "cancel" (annuler), "close" (fermer) ou "execute" (exécuter).

3.3.15 Réponse (Response)

Une réponse est un message envoyé comme conséquence d'une demande, provenant généralement de la cible de la demande à destination de la source de la demande. Les messages de réponse sont synonymes des messages "reply" (répondre). Un message de réponse utilise un verbe "reply" (répondre).

3.3.16 Requête (Query)

Une requête est un type de demande dans laquelle la cible est supposée renvoyer des informations à la source de la demande. Le message de demande d'une requête utilise le verbe "get" (obtenir). Cela est généralement mis en œuvre au moyen d'un modèle demande/réponse (request/reply).

3.3.17 Transaction

Une transaction est un type de demande dans laquelle la cible modifie généralement les informations qu'elle gère. Une demande de transaction utilise des verbes tels que "create" (créer), "change" (modifier), "delete" (supprimer), "cancel" (annuler), "close" (fermer) ou "execute" (exécuter). Cela peut être mis en œuvre au moyen d'une grande variété de modèles. Si un modèle autre que "demande/réponse" est utilisé, il convient de fournir une garantie de livraison de la part du transport.

3.3.18 Evénement (Event)

Le terme "Evènement" est particulièrement surchargé; il peut en outre avoir différentes significations dans différents contextes. La définition la plus générale d'un évènement est "quelque chose qui se produit à un endroit et un instant donnés" ou "un changement d'état présentant un intérêt potentiel". En conséquence d'un évènement, "quelque chose" peut

générer un "message d'évènement" afin de rapporter le fait selon lequel un certain type d'évènement s'est produit à un instant donné. Les messages d'évènement sont de ce fait asynchrones par nature et sont généralement envoyés aux abonnés potentiellement intéressés au moyen d'une messagerie basée sur les thèmes.

Les messages d'évènement constituent un type de messages asynchrones. Une application peut communément générer un évènement lorsqu'un état d'intérêt est détecté ou lorsqu'une transaction a été traitée. Il existe plusieurs modèles d'intégration relatifs à la prise en charge des évènements.

Les évènements sont généralement publiés de manière asynchrone sous la forme de messages d'évènement par un système ou une application afin de rapporter les états d'intérêt. Dans certains cas, un système ou une application peut identifier en interne un état d'intérêt, mais dans d'autres cas, l'état peut être détecté par une source externe comme un appareil. Les messages d'évènement utilisent des verbes au passé comme "created" (créé), "changed" (modifié), "deleted" (supprimé), "closed" (fermé), "canceled" (annulé) ou "executed" (exécuté).

4 Cas d'utilisation

4.1 Généralités

L'objet de l'Article 4 est de décrire plusieurs cas d'utilisation relatifs aux interactions entre les composants au sein d'un ensemble de systèmes coopérant pour prendre en charge un ensemble de processus commerciaux. Il est important de noter que les cas d'utilisation présentés sont observés depuis l'intégration des systèmes, contrairement aux cas d'utilisation au niveau de l'application finale. Dans les cas d'utilisation décrits dans cet article, on retrouve les acteurs suivants:

- Client
- Serveur
- ESB
- Adaptateur
- Abonné (auditeur d'événement)

Trois termes clés relatifs à la messagerie sont «request» (demande), «reply» (réponse) et «event» (évènement). Dans la terminologie de la CEI 61968, ceux-ci sont mentionnés sous la forme de verbes utilisés pour définir les flux d'informations spécifiques.

L'hypothèse selon laquelle une variété de technologies d'intégration peut être utilisée, avec JMS et les services Web comme points principaux de cette norme, est primordiale dans les cas d'utilisation.

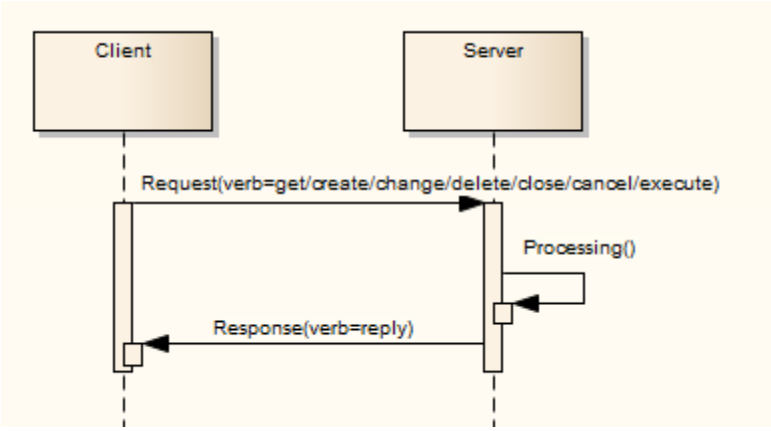
4.2 Demande/ réponse simple

Le premier cas d'utilisation est une simple demande/réponse (Request/Reply) entre un client et le serveur. C'est un synonyme de "request/response". L'initiateur de la demande est le client, alors que l'objet de la demande est traité par le serveur. La première vue est la vue simple sans utilisation d'un ESB. Cette utilisation implique l'un des deux cas:

a) Un client faisant une demande d'interrogation à un serveur, pour laquelle le serveur renvoie au client un ensemble d'objets en fonction de certains critères de filtrage.

b) Un client faisant une demande de transaction à un serveur, pour laquelle un ensemble d'objets est créé ou modifié d'une façon ou d'une autre.

Les deux cas sont illustrés par le Figure 2.



IEC 1770/13

Légende

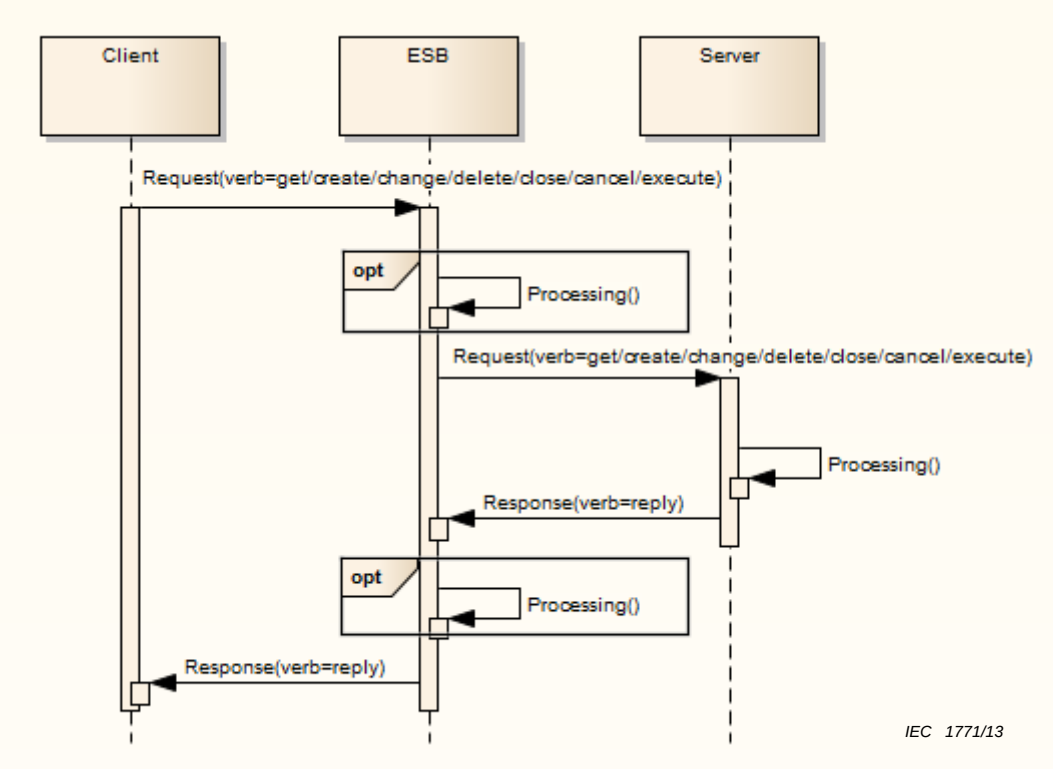
Anglais	Français
Server	Serveur

Figure 2 – Demande/Réponse simple

Selon les termes de la CEI 61968, les demandes utilisent des verbes tels que "get", "create", "change", "delete", "close", "cancel" ou "execute".

4.3 Demande/réponse au moyen d'un ESB

Le cas d'utilisation demande/réponse (request/reply) simple peut également être étendu pour tirer profit d'un ESB. Dans l'ESB, de nombreuses actions peuvent être prises par des intermédiaires si nécessaire pour faciliter l'intégration et le découplage des composants, comme des transformations et routages.



Légende

Anglais	Français
Server	Serveur

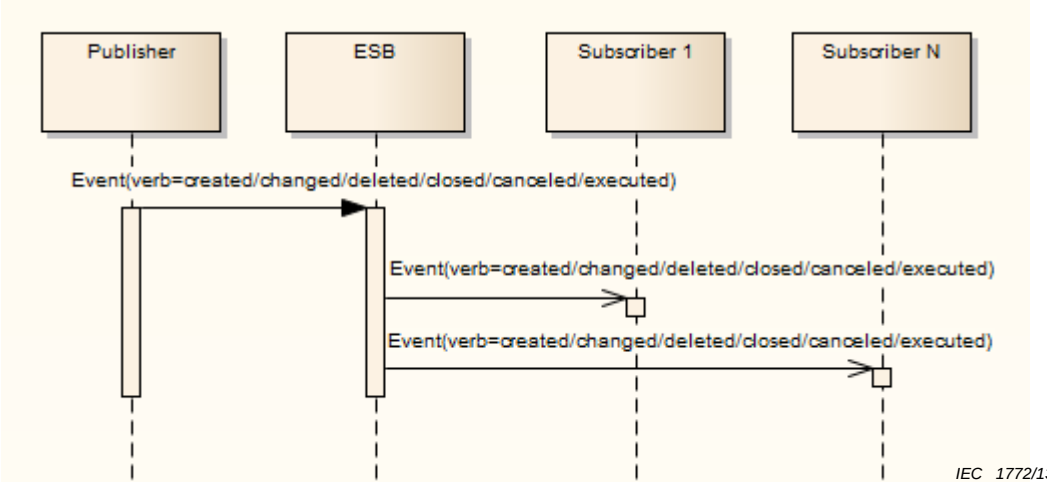
Figure 3 – Demande/réponse qui utilise des intermédiaires

Le fait de découpler le client du serveur, de sorte que le client ne puisse prendre connaissance de l'emplacement exact du serveur ou se conformer à l'interface exacte utilisée par le serveur, constitue un aspect primordial de ce cas d'utilisation. Le routage et la mise en correspondance peuvent avoir lieu dans la couche d'intégration.

Il est également recommandé de laisser les intermédiaires ESB sans état. L'utilisation d'intermédiaires sans état simplifie la mise en œuvre de l'équilibrage de charge et la haute disponibilité.

4.4 Evénements

Les processus clients ont souvent besoin d'être informés d'un évènement d'intérêt potentiel. De nombreux processus clients peuvent écouter (s'abonner à) des évènements. Par exemple, les messages EndDeviceEvents qui peuvent être envoyés par un système de comptage. On trouve d'autres exemples indiquant la mise en œuvre d'une commande ou d'une transaction, comme la création ou la mise à jour d'un ordre de travail.



Légende

Anglais	Français
Publisher	Editeur
Subscriber 1	Abonné 1
Subscriber N	Abonné N

Figure 4 – Evénement

Selon les termes de la CEI 61968, les évènements utilisent le "passé" des verbes "created", "changed", "deleted", "canceled", "closed" or "executed".

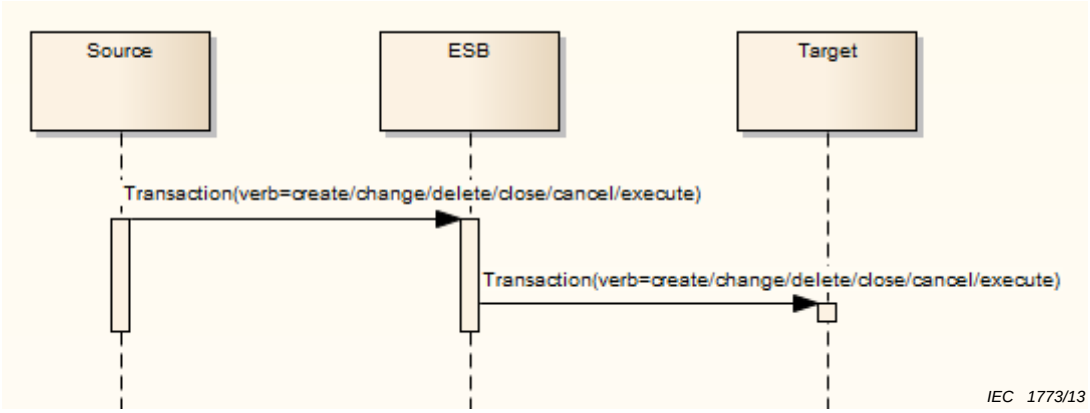
C'est un exemple de modèle unidirectionnel, dans lequel l'expéditeur n'attend pas de réponse au niveau de l'application, bien que le transport utilisé puisse permettre des acquittements de niveau inférieur afin de faciliter les garanties de livraison lorsque cela est nécessaire.

Les écouteurs qui utilisent les services Web auront besoin d'une interface exposée à une URL connue par un intermédiaire de sorte que les évènements puissent être distribués correctement. Des règles doivent également être définies pour le traitement des nouvelles tentatives lorsque la livraison doit être garantie. Les implications de l'ordonnancement du message nécessitent également d'être prises en compte. L'ordonnancement du message est un thème important en soi qui peut constituer un intérêt dans une prochaine édition.

4.5 Transactions

Le cas d'utilisation pour une transaction est généralement la combinaison d'un échange de type demande/réponse entre un client et un serveur, suivi de la publication d'évènements. Les clients et services peuvent utiliser différents mécanismes de transport dans lesquels un client peut utiliser des services Web, mais un client peut utiliser JMS, ce qui constitue un aspect important. Un intermédiaire au sein de l'ESB peut fournir les fonctions de routage nécessaires.

Les transactions sont en général mises en œuvre selon un modèle demande/réponse, mais elles peuvent également être mises en œuvre grâce à un modèle "point-to-point" (point-à-point) ou unidirectionnel. Le modèle unidirectionnel peut utiliser des verbes au présent pour les transactions ou des verbes au passé pour les événements et les besoins d'une mise en œuvre spécifique qu'elle peut nécessiter.

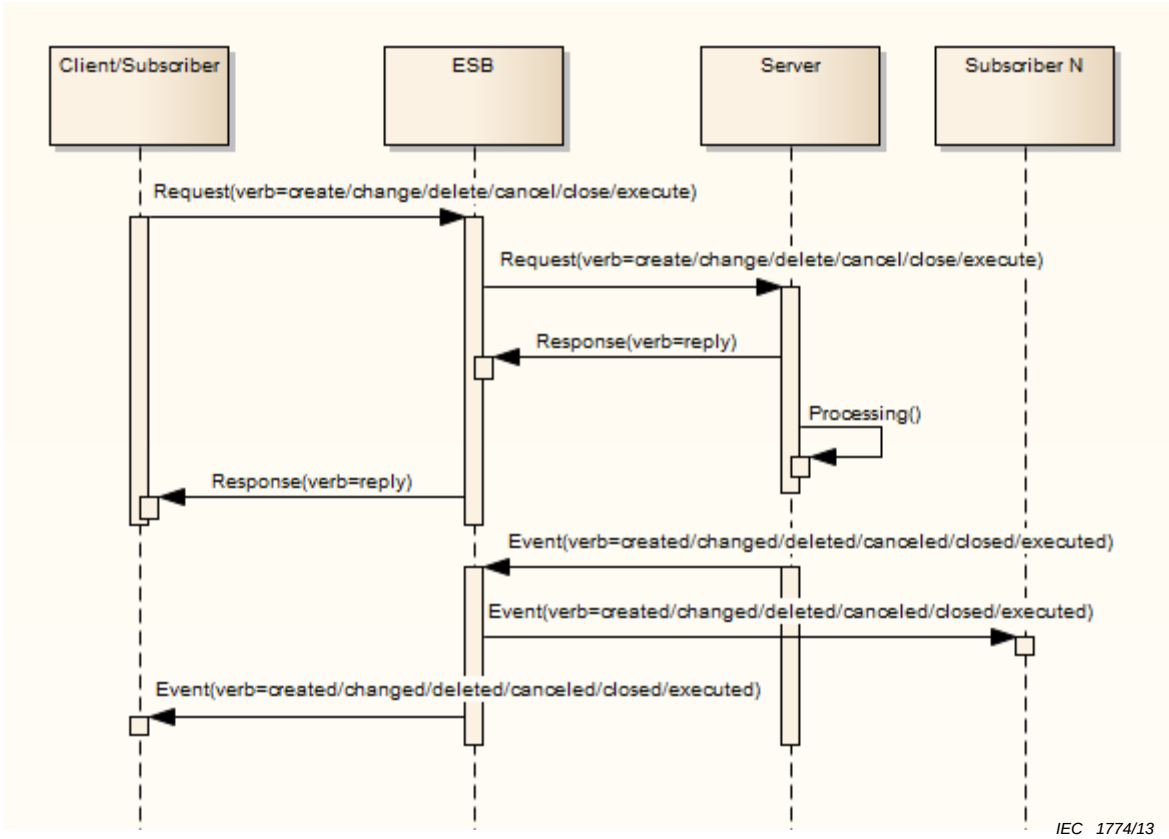


Légende

Anglais	Français
Target	Hauteur de

Figure 5 – Modèle point-to-point (unidirectionnel)

Le modèle unidirectionnel est utilisé dans les cas où un système d'enregistrement veut faire suivre une transaction vers un système pair qui doit également appliquer la transaction. Si des erreurs sont détectées par la cible, elles doivent être enregistrées en vue d'être résolues dans le système cible. La figure suivante présente un exemple dans lequel une transaction engendre des évènements propagés pour attirer l'attention des abonnés.



Légende

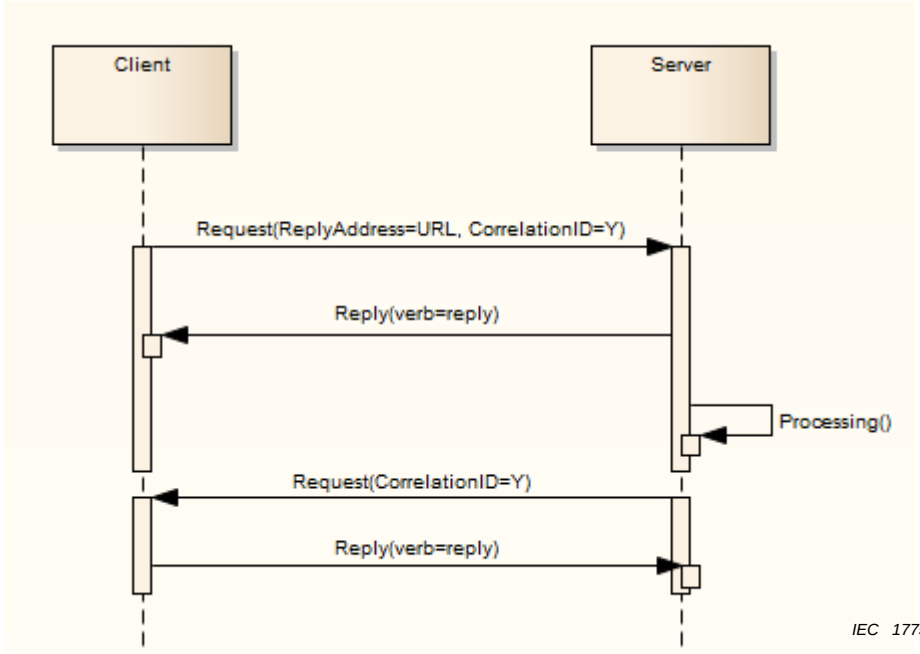
Anglais	Français
Client/Subscriber	Client/Abonné
Server	Serveur
Subscriber N	Abonné N

Figure 6 – Exemple de transaction

Selon les termes de la CEI 61968, les demandes relatives aux transactions utilisent les verbes create (créer), change (changer), delete (effacer), cancel (annuler), close (fermer) et execute (executer). Les verbes relatifs aux évènements sont au passé. L'utilisation du verbe ''execute'' implique une transaction complexe et l'utilisation de l'élément Payload.OperationSet selon la description de 6.9.

4.6 Procédure de rappel (Callback)

Une procédure de rappel est un processus asynchrone d'échange de messages. Il se compose de deux appels *request/response* (initial et final) synchrones. Les deux sont liés de telle façon que chaque partie peut identifier sans ambiguïté quelle procédure de rappel est associée à telle demande initiale. Dans ce cas, le client envoie une demande initiale au serveur. Une fois le message reçu par le serveur, celui-ci renvoie un message de réponse. A ce point, la transaction de messages initiale est terminée et l'application client n'est plus contrainte d'effectuer d'autres traitements. Une fois le traitement terminé par le serveur, ce dernier invoque la séquence request/response finale par le biais d'un message de demande. Le processus de rappel est terminé une fois que le client (de la demande initiale) répond à la demande finale. Cela est illustré dans le diagramme de séquence suivant.



Légende

Anglais	Français
Server	Serveur

Figure 7 – Procédures de rappel

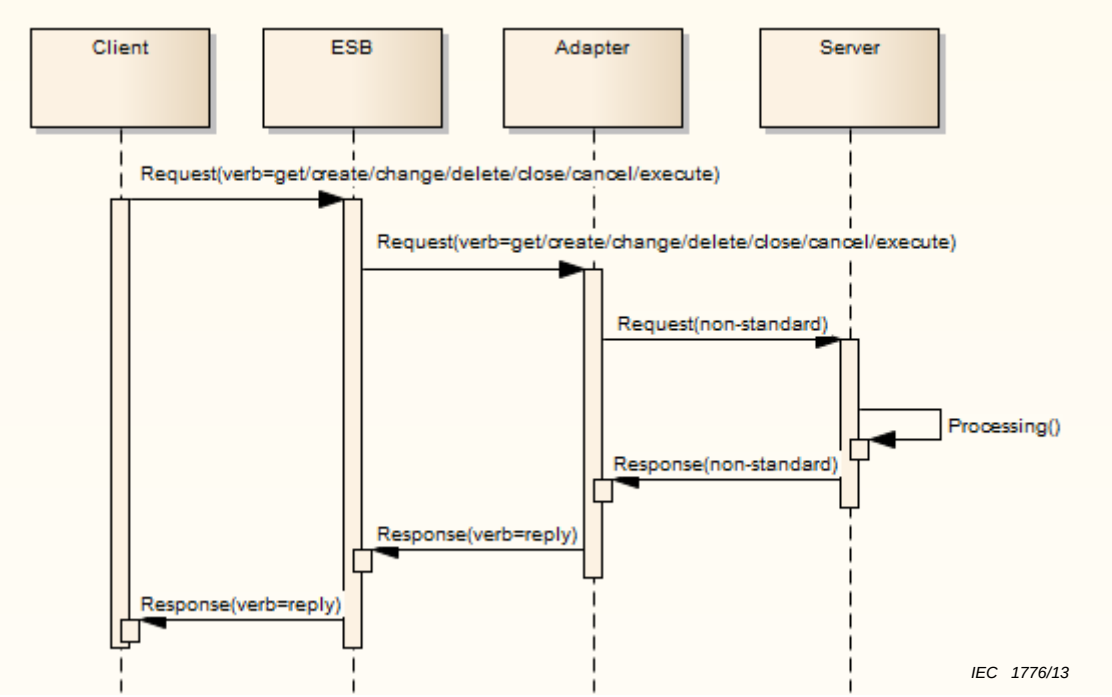
Dans le processus de rappel, le client est tenu d'informer le serveur de la destination de la réponse finale; il s'agit souvent d'une URL appelée adresse de rappel. Cette information est incluse dans le message de demande initial. Le 6.3.2 traite des éléments spécifiques des messages qui servent à contrôler ce type de dialogue;

Les procédures de rappel sont généralement mises en œuvre par des réponses asynchrones lors de l'utilisation de JMS.

4.7 Adaptateurs

Il est possible qu'une application (client ou serveur) ne puisse pas se connecter directement à un ESB ou une autre application. Dans ces cas, un adaptateur peut être utilisé pour prendre

en charge la "désadaptation d'impédance" entre l'application et l'ESB. Dans certains cas, l'application peut simplement être une base de données ou un dossier de fichiers. Dans le diagramme suivant, un adaptateur est utilisé pour connecter un serveur à l'ESB.



Légende

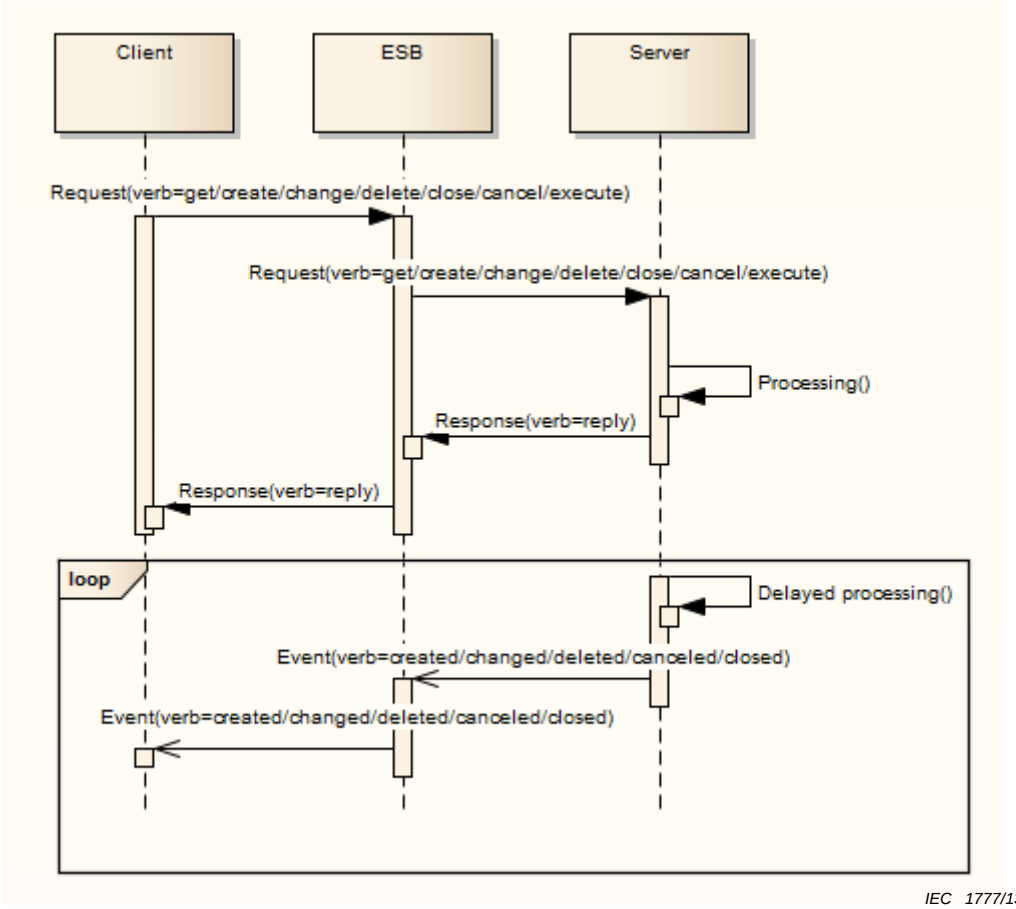
Anglais	Français
Adapter	Adaptateur
Server	Serveur

Figure 8 – Utilisation d'adaptateurs

L'adaptateur peut prendre en charge un certain nombre de fonctions et peut entreprendre des actions au nom de l'application comme la génération d'évènements en réponse au succès d'une transaction. L'activité la plus commune consiste à convertir les données entre un modèle d'application et un modèle canonique d'entreprise.

4.8 Messagerie complexe

Certains cas d'utilisation peuvent nécessiter des modèles de messagerie plus complexes. Par exemple, il se peut qu'une transaction puisse impliquer de nombreux événements induits qui peuvent être envoyés au client sous la forme d'évènements asynchrones.



Légende

Anglais	Français
Server	Serveur
Loop	Boucle

Figure 9 – Messagerie complexe

Le cas d'utilisation ci-dessus se rencontre dans des applications comme le comptage, où il peut prendre un temps significatif pour obtenir des résultats, comme dans le cas d'une régulation de charge ou de déconnexion, lorsqu'un message EndDeviceControls peut être envoyé, mais les résultats peuvent être rapportés après un certain retard grâce à un message EndDeviceEvents.

Il est important de noter qu'il s'agit d'une variation du modèle de rappel. La demande initiale peut être soit une requête, dans laquelle les résultats sont renvoyés de manière asynchrone, soit une transaction en conséquence de laquelle un ou plusieurs évènements peuvent être générés.

4.9 Orchestration

Les cas d'utilisation relatifs à l'orchestration ou la chorégraphie d'un processus commercial ainsi qu'aux transactions réparties de courte ou longue durée ne relèvent pas du domaine d'application de cette spécification. Les approches d'intégration décrites dans ce document peuvent être renforcées par des intégrations plus complexes qui traitent ces besoins.

4.10 Cas d'utilisation au niveau de l'application

Les cas d'utilisation décrits dans les Paragraphes 4.1 à 4.9 peuvent être appliqués aux cas d'utilisation au niveau de l'application finale. Dans l'exemple suivant de cas d'utilisation au

niveau de l'application, des messages sont définis au moyen de ''verbs'' (verbes) et ''nouns'' de la norme CEI 61968, sous la forme '<verb>(<Noun>)'. Les types de systèmes spécifiques (ou représentatifs) sont également identifiés comme étant opposés aux types d'acteurs plus génériques.

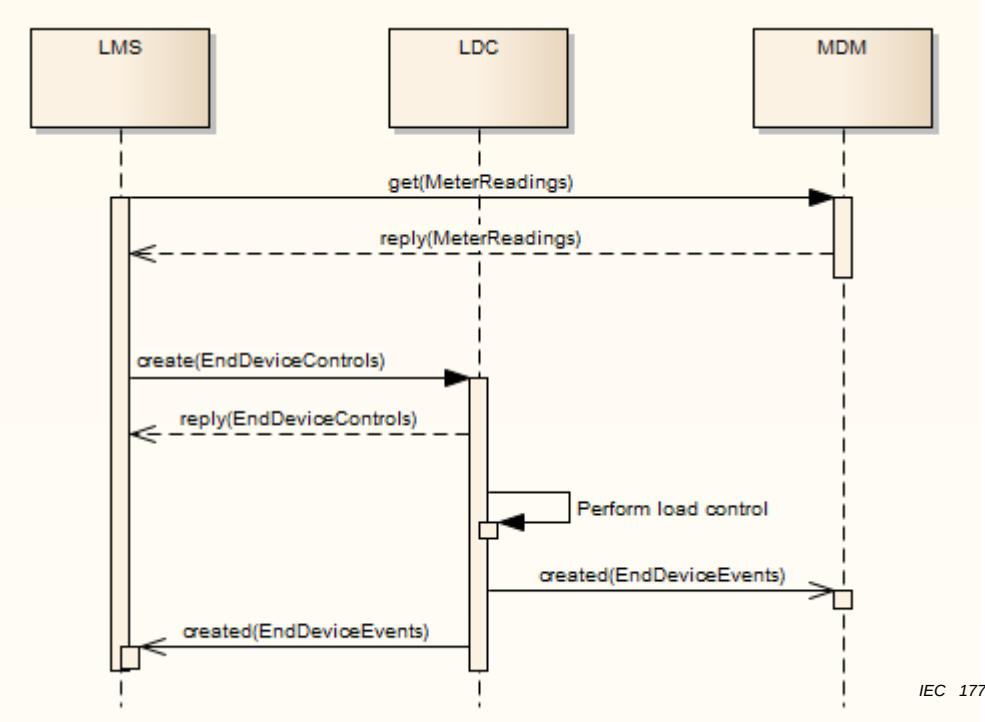


Figure 10 – Exemple de cas d'utilisation au niveau de l'application

5 Modèles d'intégration

5.1 Généralités

Le présent document identifie un ensemble de modèles d'intégration du service principal prenant en charge les cas d'utilisation décrits précédemment. Il rend également possibles des modèles d'intégration plus complexes par l'utilisation d'intermédiaires au sein d'un bus de serveurs d'entreprise (ESB), ou lorsqu'une application sert d'intermédiaire.

5.2 Points de vue du client et du serveur

5.2.1 Généralités

Du point de vue des clients et des serveurs, il existe plusieurs modèles d'intégration de base décrits dans le présent document. Ces modèles comprennent, mais sans s'y limiter:

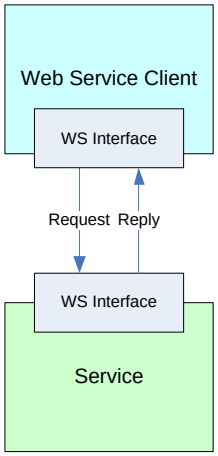
- Les demandes/réponses synchrones
  - Les mises en œuvre des services Web utilisent une opération avec des messages d'entrée et sortie
  - Les mises en œuvre des services JMS échangent des messages dans des files d'attente
- Les demandes/réponses asynchrones
  - Les mises en œuvre des services Web utilisent des opérations séparées pour les messages de demande et de réponse de rappel
  - Les mises en œuvre des services JMS échangent des messages dans des files d'attente

- Les actions de type publier/souscrire dans lesquelles de nombreuses cibles peuvent écouter les messages
  - Les mises en œuvre des services Web impliquent un client qui écoute des URL spécifiées pour les événements qui sont envoyés à plusieurs cibles par un intermédiaire ESB.
  - Les mises en œuvre JMS impliquent des cibles qui écoutent les messages d'un thème

Du point de vue du client seul ou du serveur seul, que le client communique directement ou non avec le serveur ou qu'il le fasse via des intermédiaires ne présente pas d'intérêt tant qu'ils ont choisi d'utiliser le même mécanisme de transport. La valeur d'un ESB réside dans le fait qu'une architecture d'intégration peut être fournie, dans laquelle un client peut choisir d'utiliser des services Web, des thèmes ou des files d'attente, mais aussi dans laquelle chaque service cible peut également choisir d'utiliser des services Web, thèmes ou files d'attente indépendamment des choix de tout client. Ceci peut être utilisé pour isoler les choix d'applications dans l'entreprise en fonction de la technologie de sorte que toute technologie de transition utilisée n'ait aucun impact direct sur les applications individuelles qui limiterait les options de migration d'applications.

5.2.2 Modèle de service Web de base

Le diagramme suivant illustre un modèle demande/réponse de base qui utilise des services Web.



Légende

Anglais	Français
Web Service Client	Client Web Service
WS Interface	Interface WS
Request	Demande
Reply	Réponse

Figure 11 – Demande/réponse de base qui utilise des services Web

Dans ce modèle, le client émet une demande vers l'interface d'un service Web exposé par un service donné. Cette interface est définie grâce à un WSDL. Cependant, ce modèle d'interaction peut également être effectué grâce à REST. Il convient pour le client de s'attendre à différentes sorties possibles:

- La demande est traitée avec succès; dans ce cas, un message de réponse est renvoyé en temps opportun (Résultat=OK)
- La demande est acceptée, mais génère un message de réponse renvoyant un code d'erreur au niveau de l'application (Résultat= FAILED)

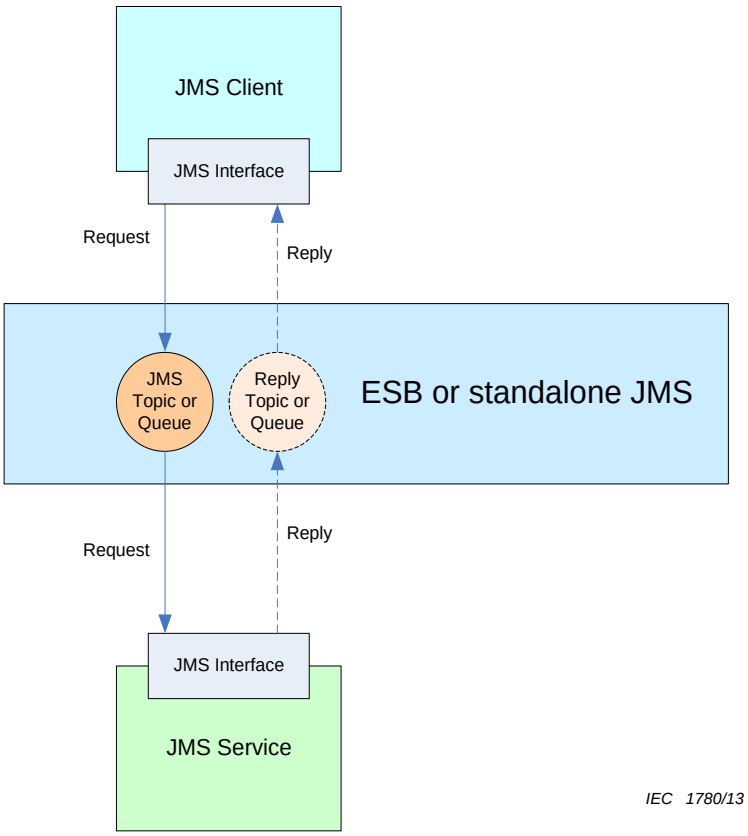
- La demande est acceptée, mais génère un message de réponse avec un jeu de résultats partiel (Résultat=PARTIAL)
- La demande génère un défaut renvoyé au client
- Après l'envoi de la demande, aucune réponse ni aucun défaut n'est renvoyé en temps opportun

Il est également important de noter qu'il peut y avoir différents niveaux de sécurité qui peuvent être requis pour la mise en œuvre. Le cas extrême se produit lorsque le client utilise un réseau public pour communiquer avec le service, tandis que l'authentification, l'autorisation, le cryptage et la signature peuvent être importants.

Si une réponse n'est pas requise, on parle de modèle "unidirectionnel". Il convient toutefois de faire attention aux garanties de livraison requises.

5.2.3 Modèle demande/réponse JMS de base

La Figure 12 décrit un modèle demande/réponse de base dans lequel le client envoie un message JMS à un thème (ou une file d'attente). Dans ce modèle, le message de réponse est optionnel et il peut exister des cas dans lesquels un message de réponse n'est jamais envoyé.



Légende

Anglais	Français
JMS Client	Client JMS
JMS Interface	Interface JMS
Request	Demande
Reply	Réponse
JMS Topic or Queue	Thème ou file d'attente JMS
ReplyTopic or Queue	Thème ou file d'attente de réponse

Anglais	Français
ESB or standalone JMS	ESB ou JMS autonome
JMS Service	Service JMS

Figure 12 – Demande/réponse de base qui utilise JMS

Un service écoutant une destination de message (thème ou file d'attente) consomme le message de demande et émet une réponse au client. La destination du message est gérée par la mise en œuvre de JMS (typiquement une partie d'un ESB). Le client peut consommer le message de réponse de façon synchrone ou asynchrone, la décision revenant à la seule discrétion de la mise en œuvre du client.

Lorsque le présent document se réfère à une "destination de message" JMS, les mises en œuvre peuvent utiliser soit des thèmes, soit des files d'attente, pour la messagerie demande/réponse. Lors de l'utilisation de thèmes, un service utilise en règle générale un abonnement durable dans les cas où les messages de demande ne doivent pas être perdus.

Le client qui initie la demande peut s'attendre à l'un des résultats suivants:

- Le message de demande est envoyé avec succès vers un thème (ou une file d'attente), d'où un message de réponse (lié à la demande par un ID) est renvoyé en temps opportun (Résultat=OK)
- Le message de demande est accepté, mais génère un message de réponse renvoyant un code d'erreur au niveau de l'application (Résultat=FAILED)
- La demande est acceptée, mais génère un message de réponse avec un jeu de résultats partiel (Résultat=PARTIAL)
- La tentative d'envoi d'un message de demande au thème (ou à la file d'attente) échoue
- Après l'envoi du message de demande, aucun message de réponse n'est jamais reçu (ce qui peut être normal ou non selon le service)

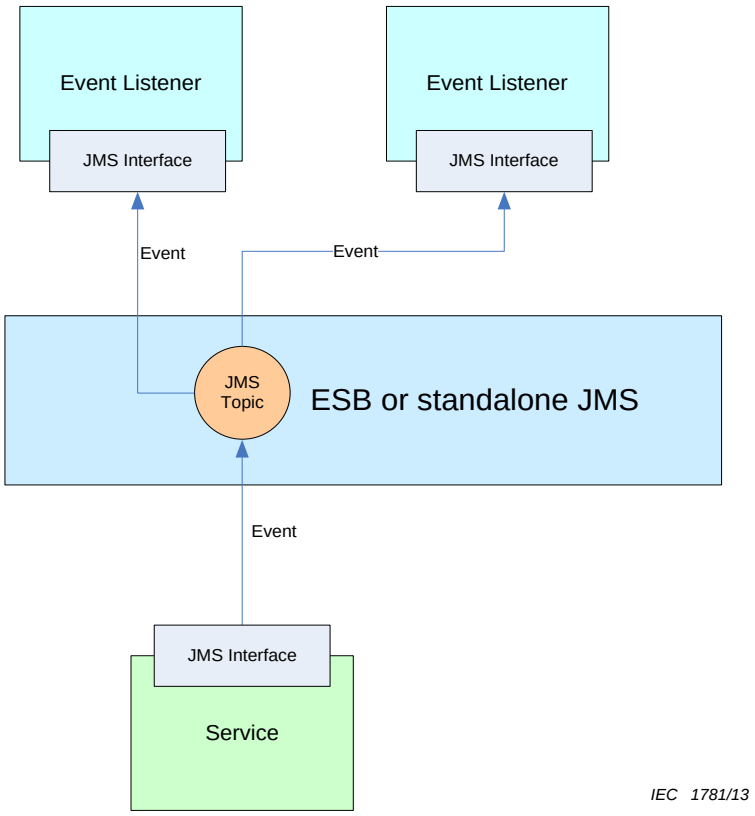
Les messages de réponse sont envoyés au moyen de thèmes ou de files d'attente, suivant le cas; les thèmes spécifiques peuvent être définis de manière statique ou dynamique. Pour des questions de simplicité, les diagrammes suivants n'identifient pas explicitement les thèmes ou les files d'attente. Les garanties de livraison offertes par les mises en œuvre de JMS laissent aux clients le choix de faire des demandes de transaction qui ne nécessitent pas de réponses. C'est ce que l'on appelle parfois un modèle unidirectionnel.

JMS est généralement utilisé au sein d'un réseau privé d'entreprise sécurisé. Cependant, dans certains cas isolés, il se peut que ce ne soit pas le cas et que la sécurité constitue une question importante. JMS peut également être configuré facilement pour utiliser SSL/TLS et/ou utiliser l'authentification du client.

Ce modèle ne demande pas en fait une mise en œuvre complète d'ESB, mais simplement une mise en œuvre de JMS. Il est également important de noter que les mises en œuvre JMS de différents fournisseurs ne sont généralement pas interopérables et qu'un "pont" peut être requis lorsque des clients ne peuvent pas utiliser une mise en œuvre JMS commune.

5.2.4 Ecouteurs d'événements

L'intégration d'un processus d'écoute des évènements qui peuvent être édités constitue un autre modèle d'intégration. Dans de nombreux cas, un service peut éditer des messages d'évènement potentiellement intéressants pour d'autres processus.



Légende

Anglais	Français
Event Listener	Ecouteur d'événement
JMS Interface	Interface JMS
Event	Evénement
JMS Topic	Thème JMS
ESB or standalone JMS	ESB ou JMS autonome

Figure 13 – Ecouteurs d'évènements qui utilisent JMS

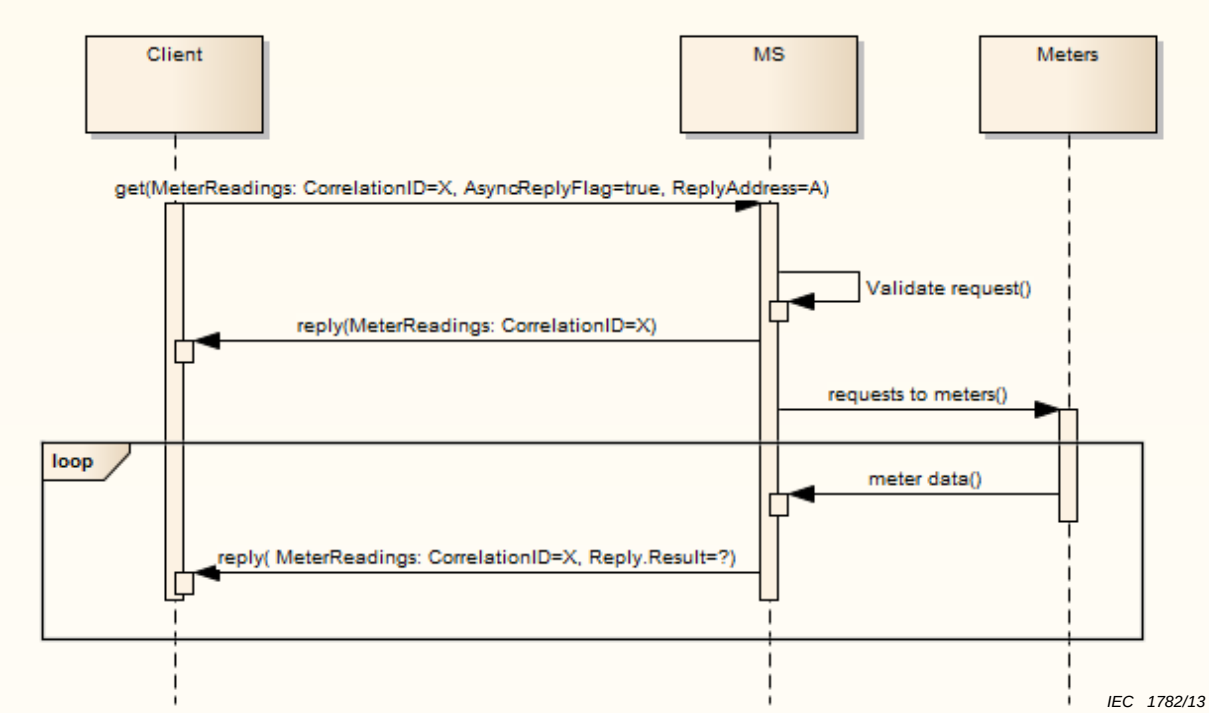
Il convient généralement de publier les événements selon des thèmes et non dans des files d'attente; en effet, il va généralement y avoir de nombreux consommateurs d'événements. L'écouteur est responsable de son abonnement à un ou plusieurs thèmes JMS potentiellement intéressants. Certains écouteurs peuvent choisir d'utiliser un abonnement durable dans les cas où les évènements ne doivent pas être perdus. Les messages d'évènement sont envoyés au consommateur de façon asynchrone. Il n'y a aucun message d'acquittement de quelque sorte renvoyé au service.

Certains problèmes relatifs au nombre de thèmes (ou files d'attente) et à la sortie (le nombre d'écouteurs de l'évènement) peuvent être pris en charge grâce à la mise en œuvre d'un JMS donné.

Il est également possible d'étendre l'architecture décrite par le présent document pour tirer profit de l'utilisation de WS-Eventing pour l'abonnement et la publication d'évènements par les clients du service Web. Il existe actuellement une version "open source" de WS_Eventing dans le projet Apache. Cependant, le présent document ne se concentre pas sur les aspects spécifiques de la mise en œuvre et de l'utilisation de WS-Eventing. Le présent document fournit également un autre exemple d'évènements de routage qui utilise des services Web.

5.2.5 Modèle Demande/réponse asynchrone

La demande/réponse asynchrone autorise l'envoi d'une ou plusieurs réponses de manière asynchrone vers un client émettant une demande. Il s'agit d'un modèle d'intégration complexe, légèrement plus difficile à mettre en œuvre à l'aide des services Web qu'avec JMS.



Légende

Anglais	Français
Meters	Compteurs
Loop	Boucle

Figure 14 – Modèle Demande/Réponse asynchrone

Dans ce modèle, un client effectue une demande à destination d'un service. Lorsque le service peut ne pas être en mesure de traiter et/ou de renvoyer les informations désirées immédiatement, le service répond immédiatement par un banal message d'acquittement. Une fois que le traitement est en mesure d'intervenir et/ou les informations souhaitées sont obtenues, le service peut fournir une ou plusieurs réponses asynchrones au client au moyen d'une URL spécifique, d'un thème ou d'une file d'attente. Les éléments principaux (tels qu'ils sont décrits dans l'Article 6) utilisés pour contrôler cet échange comprennent:

- Header.CorrelationID est utilisé pour relier tous les messages de façon logique. Lorsqu'une CorrelationID est fournie par le client au cours de la demande initiale, le serveur doit l'inclure à tous les messages de réponse et d'évènement liés.
- Header.AsyncReplyFlag est réglé sur "true" dans la demande initiale.
- Header.ReplyAddress est défini sur la demande initiale pour identifier l'endroit où il convient de définir les réponses.
- Reply.Result est utilisé dans les réponses, où "PARTIAL" indique que d'autres réponses peuvent être attendues; "OK" indique que le traitement est terminé et qu'il convient de ne pas attendre d'autres réponses. Une valeur ‘’FAILED’’ indique une condition d’erreur.

Ce modèle peut être utilisé pour obtenir des relevés de compteurs dans les cas où la tête de réseau d'un système de comptage doit demander les informations désirées à un ou plusieurs compteurs.

5.3 Point de vue du bus

5.3.1 Généralités

Les clients et serveurs peuvent soit communiquer directement, soit via des intermédiaires. Le bus de services d'entreprise (ESB) est utilisé pour la gestion des intermédiaires. L'utilisation d'un ESB fournit de nombreuses variations de modèles de communication. Cependant, le client considère toujours l'ESB comme un serveur. De même, le serveur considère l'ESB comme un client ordinaire.

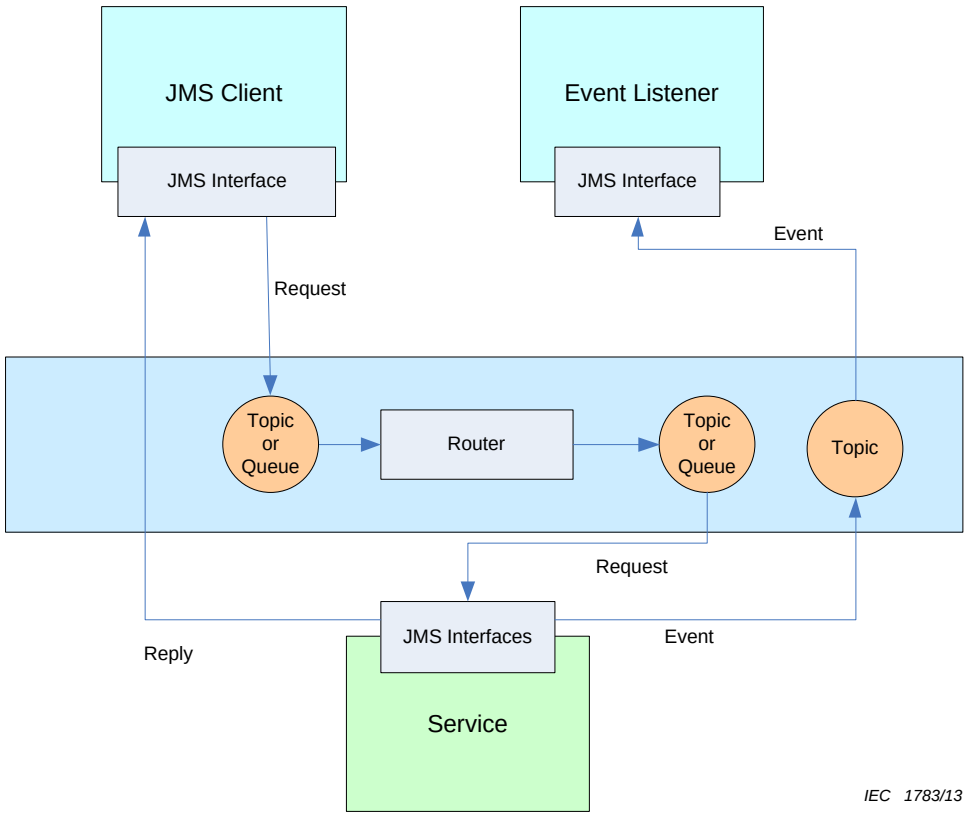
Cela est important parce que:

- Aucune connaissance de l'ESB n'est imposée lors de la mise en œuvre d'une interface CEI 61968 sur un client ou un serveur.
- Il n'existe aucune exigence pour un produit ESB placé sur une interface CEI 61968, excepté un serveur JMS lorsque des messages JMS sont à utiliser
- Une mise en œuvre de projet qui utilise la norme CEI 61968 peut utiliser un ESB et mettre en œuvre librement des modèles d'intégration appropriés au projet et aux mises en œuvre associées.

Comme un ESB n'est pas exigé pour la mise en œuvre d'une interface CEI 61968, le reste de 5.3 ne décrit que des recommandations et est fourni en qualité de matériel informatif.

5.3.2 Modèle de messagerie ESB qui utilise JMS

Le modèle de messagerie ESB de base qui utilise JMS introduit l'option de routage des demandes. Cela est utile pour le découplage ultérieur du client et du serveur; en effet, le bus (par l'utilisation d'une logique de routage, souvent appelée modèle d'intégration du "Routeur basé sur du contenu" [EIP]) peut prendre des décisions relatives au traitement de la demande. Cela est décrit dans le diagramme suivant.



Légende

Anglais	Français
JMS Client	Client JMS
JMS Interface	Interface JMS
Event Listener	Ecouteur d'événement
Request	Demande
Event	Evénement
Topic or Queue	Thème ou file d'attente
Router	Routeur
Topic	Thème
JMS Interfaces	Interfaces JMS
Reply	Réponse
Event	Evénement

Figure 15 – Routage basé sur du contenu ESB

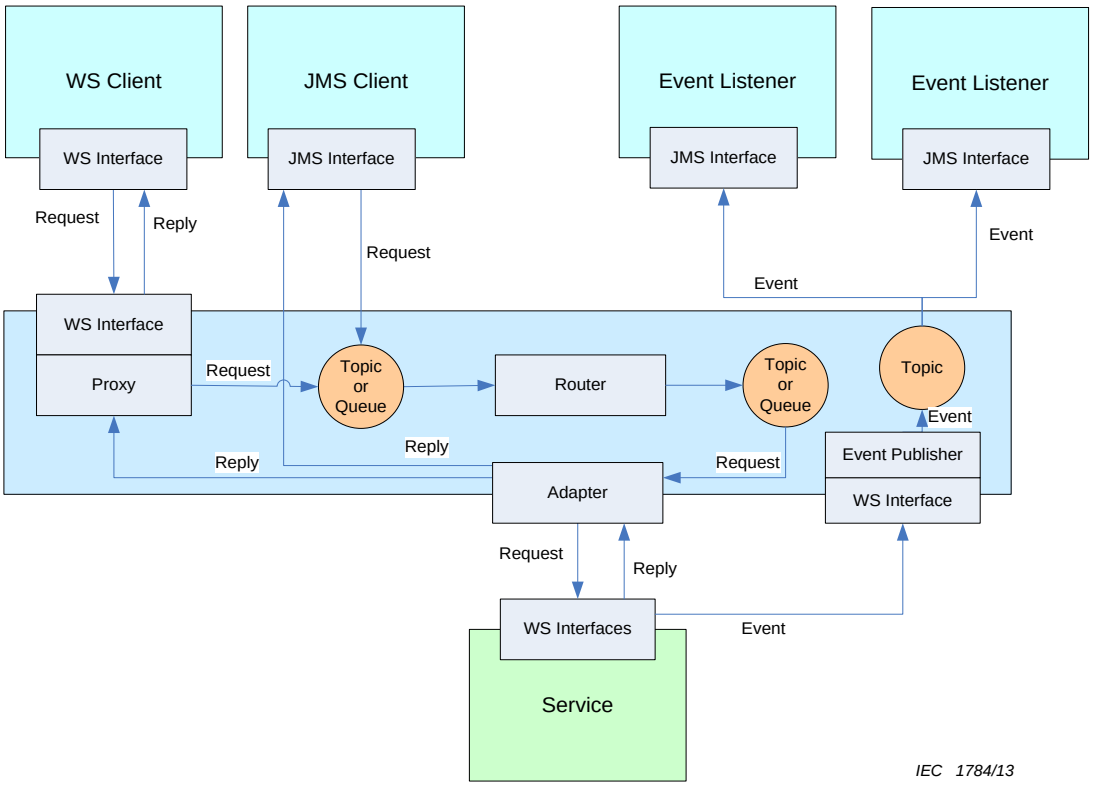
Lorsqu'un client émet une demande à destination d'un thème (ou file d'attente), un routeur sur le bus peut décider de faire suivre le message vers un autre thème (ou file d'attente). Les décisions prises par le routeur peuvent prendre en considération chacun des contenus suivants:

- Contenu de l'en-tête d'un message
- Contenu de la charge utile d'un message (même s'il convient de l'éviter)
- Le statut de l'instance d'un service de destination
- Le besoin d'équilibrer la charge

Il est important de noter qu'un des avantages de l'approche faiblement couplée décrite par le présent document réside dans le fait que les composants du routage ne sont pas liés à des messages d'un type de charge utile spécifique. Le routeur peut être configuré à l'aide d'expressions XPath pour identifier le contenu des messages et déterminer le routage réel.

5.3.3 Modèles de messagerie ESB qui utilise une demande de service Web

Le diagramme suivant étend le modèle décrit précédemment pour permettre au client d'un service Web autant qu'à un client JMS d'initier une demande.



Légende

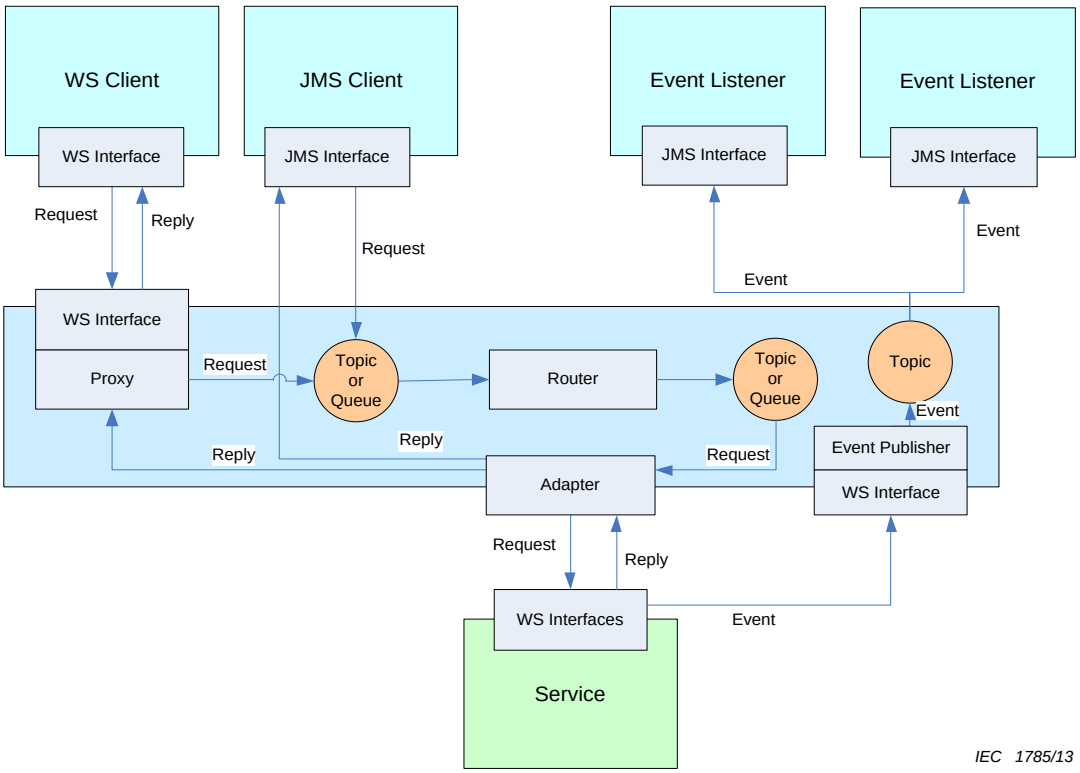
Anglais	Français
WS Client	Client WS
JMS Client	Client JMS
Event Listener	Ecouteur d'événement
WS Interface	Interface WS
JMS Interface	Interface JMS
Request	Demande
Reply	Réponse
Event	Evénement
Proxy	Proxy
Topic or Queue	Thème ou file d'attente
Router	Routeur
Adapter	Adaptateur
Event publisher	Editeur d'événement

Figure 16 – ESB avec Smart Proxy et routage basé sur du contenu

Un composant de Proxy (parfois appelé modèle d'intégration "Smart Proxy" [EIP]) est mis en œuvre sur l'ESB afin d'exposer un service Web (tel qu'il est défini par un WSDL). Le Smart Proxy peut prendre des décisions relatives à la répartition des demandes et à la corrélation des réponses. Le message acheminé dans le WSDL est simplement converti en message JMS et est ensuite routé comme il se doit.

5.3.4 Traitement de demande ESB à destination d'un service Web

Le diagramme suivant étend le modèle précédent pour permettre à un service d'exposer son interface comme un service Web avec un WSDL défini de façon appropriée.



Légende

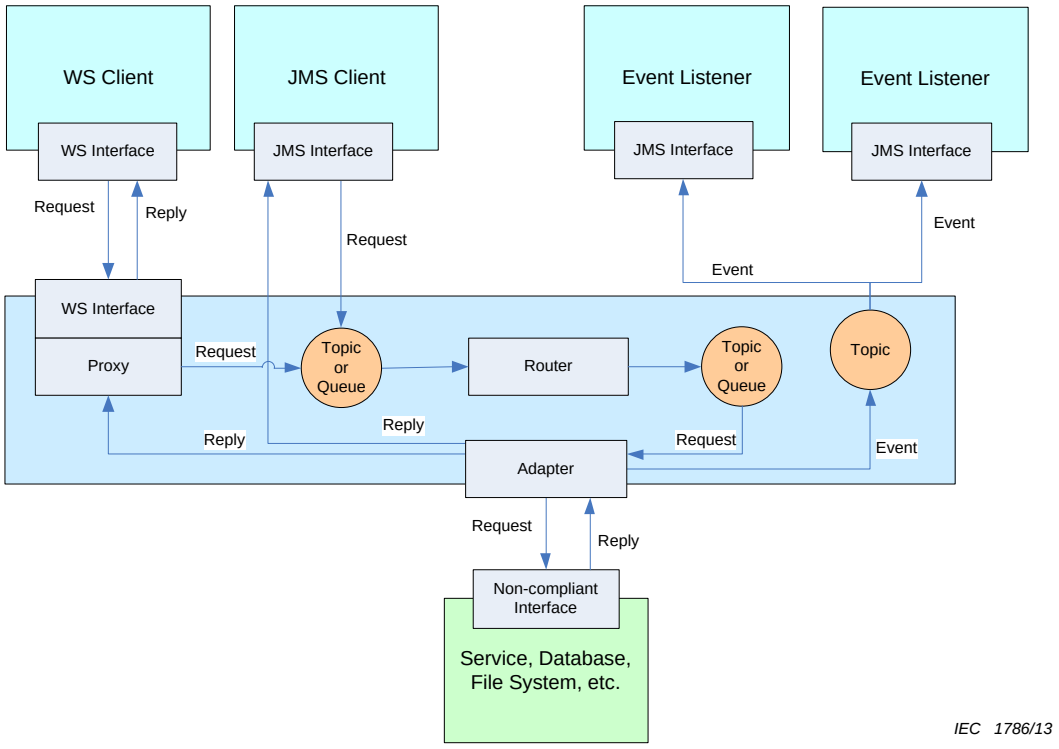
Anglais	Français
WS Client	Client WS
JMS Client	Client JMS
Event Listener	Ecouteur d'événement
WS Interface	Interface WS
JMS Interface	Interface JMS
Request	Demande
Reply	Réponse
Event	Evénement
Proxy	Proxy
Topic or Queue	Thème ou file d'attente
Router	Routeur
Topic	Thème
Adapter	Adaptateur
Event Publisher	Editeur d'événement
WS Interfaces	Interfaces WS

Figure 17 – ESB avec proxies, routeurs et adaptateurs

Dans ce modèle, un adaptateur est mis en œuvre dans l'ESB pour convertir le message JMS interne en demande de service Web appropriée.

5.3.5 Traitement de demande ESB via un adaptateur

Le diagramme suivant est une variation du modèle d'intégration précédent, dans lequel le serveur utilise une interface qui ne serait pas en d'autres cas conforme au profil d'interface décrit par le présent document. Cela montre qu'une interface conforme à la norme CEI 61968 peut être utilisée pour s'intégrer à un serveur, une base de données, un système de fichiers ou autre source ou puits de données autrement non conforme à la CEI 61968 via l'utilisation d'un adaptateur dans l'ESB. Les adaptateurs peuvent aussi ne pas dépendre d'un ESB.



Légende

Anglais	Français
WS Client	Client WS
JMS Client	Client JMS
Event Listener	Ecouteur d'événement
WS Interface	Interface WS
JMS Interface	Interface JMS
Request	Demande
Reply	Réponse
Event	Evénement
Proxy	Proxy
Topic or Queue	Thème ou file d'attente
Router	Routeur
Topic	Thème
Adapter	Adaptateur
Event Publisher	Editeur d'événement
WS Interfaces	Interfaces WS
Topic	Thème
Non-compliant Interface	Interface non conforme
Service, Database, File System, etc.	Service, Base de données, Système de fichiers, etc.

Figure 18 – Intégration d'ESB à des ressources non conformes

L'intégration entre l'adaptateur et une interface non conforme peut utiliser une grande variété de mécanismes d'intégration en fonction des capacités du produit ESB en question. Ces mécanismes peuvent comprendre, mais sans s'y limiter:

- JMS
- Services Web

- HTTP
- Java Database Connectivity (JDBC)
- File Transfer Protocol (FTP)
- Lecture et/ou écriture de fichiers
- Accès à la base de données propriétaire

Lorsque la spécification de JMS et JDBC implique l'utilisation d'une trame JEE (Java Enterprise Edition), aucune exigence réelle n'est imposée. La plupart des bases de données prenant en charge JBDC peuvent également être atteintes au moyen d'une connectivité ODBC (Open Database Connectivity), que ce soit directement ou via des produits de pont. De nombreux produits ESB qui prennent en charge JMS ont également des API qui peuvent être utilisées pour envoyer et recevoir des messages JMS avec un langage autre que le Java (C, C++, par exemple). Cependant, il est important de reconnaître que l'utilisation du cadre JEE fournit à la plate-forme un haut degré d'indépendance.

5.3.6 Modèles d'intégration personnalisés

Typiquement, un projet d'intégration implique la mise en œuvre d'une variété de modèles d'intégration personnalisés. Les Paragraphes 5.3.1 à 5.3.5 faisaient allusion à l'existence et l'utilisation potentielles de certains de ces modèles mis en œuvre en qualité de processus intermédiaires au sein de l'ESB. Ceux-ci incluraient potentiellement, mais sans s'y limiter, des modèles tels que [EIP]:

- Routeur basé sur le contenu, dans lequel les messages sont acheminés en fonction du contenu du message qui est généralement référencé grâce à des expressions XPath
- Smart Proxy, dans lequel les messages peuvent être réacheminés vers un service de destination spécifique, dans lequel les réponses provenant du service sont acceptées et renvoyées vers le client
- Ticket, dans lequel une copie d'un fichier souvent très imposant est conservée sous la forme d'un document à disposition des autres processus, dans lequel le suivi du statut courant du document est assuré, mais dans lequel le document est en général transporté par des moyens autres que la messagerie
- Transformation, dans lequel les transformations habituellement définies par XSL sont utilisées pour reformater le contenu des messages
- Pont, dans lequel un message publié sur un thème ou une file d'attente peut être transmis à ou reçu d'une autre infrastructure de messagerie (ce modèle peut parfois être mis en œuvre au moyen de produits tiers ou simplement via la configuration)

Cependant, il est important de noter que la présente norme ne mandate pas la mise en œuvre ni l'utilisation de quelque modèle d'intégration personnalisé spécifique. Il est également important de noter deux philosophies de première importance pour la mise en œuvre de modèles d'intégration:

1) Les modèles sont mis en œuvre en qualité de définition de processus unique qui peut être instanciée une ou plusieurs fois pour prendre en charge potentiellement de nombreux flux d'informations lorsqu'il n'y a aucune contrainte de type
2) Les modèles sont des gabarits utilisés pour mettre en œuvre un processus dans le but de prendre en charge un flux d'informations spécifique menant à une mise en œuvre "spécifique au modèle"

Dans les deux cas, il existe des compromis significatifs. Le point d'intérêt de la présente spécification est de recommander la première option par l'utilisation d'une enveloppe de message commune qui prend facilement en charge les mises en œuvre communes profitables de modèles d'intégration spécifiques par opposition aux instances spécifiques au modèle d'un modèle d'intégration donné.

6 Organisation du message

6.1 Généralités

Chaque interface de service est construite pour accepter un message présentant un verbe et un noun. Le noun identifie le type de charge utile qui peut être fourni dans le message de demande, de réponse ou d'événement. Cela permet aux interfaces d'être faiblement couplées.

Les interfaces des services sont définies au moyen d'une ou des deux possibilités suivantes:

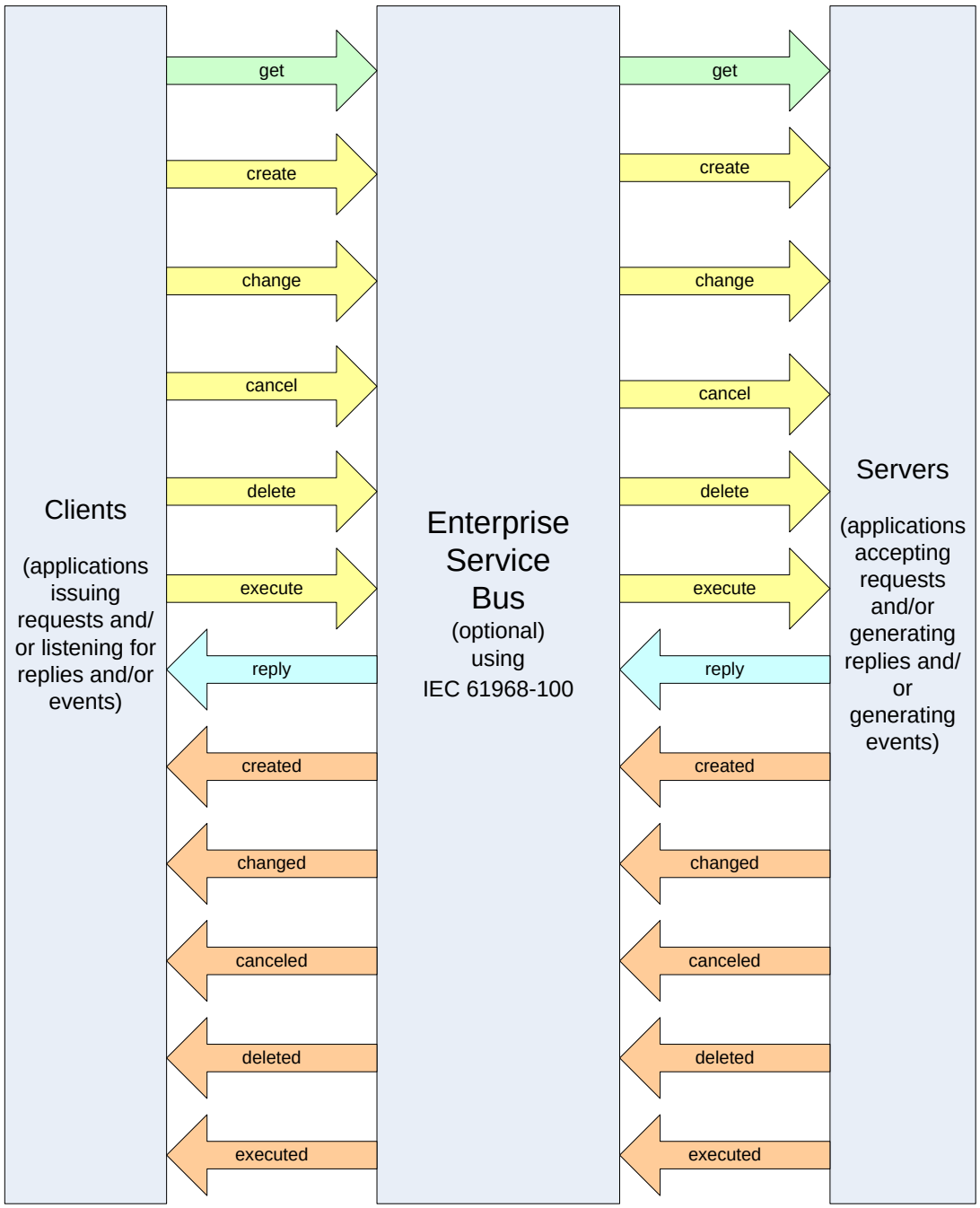
- Web Services Definition Language (WSDL), dans lequel les messages de demande, de réponse et de défaut sont définis pour une ou plusieurs opérations
- Définition de message JMS

Dans tous les cas, les Schémas XML (XSD) sont utilisés pour définir la structure des enveloppes de message. Dans la plupart des cas, les XSD sont utilisés pour définir la structure des charges utiles des messages. Le contenu des charges utiles des messages est décrit dans l'Article 7.

6.2 Messages CEI 61968

6.2.1 Généralités

La norme CEI 61968-1 spécifie un échange d'informations sous la forme d'un verbe, d'un noun et d'une charge utile. La Figure 19 montre le flux directionnel des messages entre les clients, les serveurs et l'ESB basé sur le verbe.



IEC 1787/13

Légende

Anglais	Français
Clients (applications issuing requests and/or listening for replies and/or events)	Clients (applications qui émettent des demandes et/ou qui écoutent les réponses et/ou les événements)
Enterprise Service Bus (optional) using IEC 61968-100)	Enterprise Service Bus (facultatif) utilisant la CEI 61968-100)
Servers (applications accepting requests and/or generating replies and/or events)	Serveurs (applications qui acceptent les demandes et/ou qui génèrent les réponses et/ou les événements)

Figure 19 – Messagerie entre clients, serveurs et un ESB

6.2.2 Verbes (Verbs)

La norme CEI 61968-1 identifie un ensemble de verbes et l'Annexe B de cette norme définit une liste normative. Le présent paragraphe entend fournir plus de précisions sur l'utilisation de chaque verbe et identifier la dépréciation de certains verbes ainsi que des synonymes. Dans Tableau 1, les verbes utilisés pour les demandes sont associés au verbe qu'il convient d'utiliser dans un message de réponse et qui serait utilisé pour la publication d'un événement, alors que les événements sont souvent la conséquence de la réussite d'une transaction initiée par une demande.

Table 1 – Verbes et leur utilisation

Verbe de demande	Verbe de réponse	Verbe d'évènement	Utilisation
get	reply	(none)	query
create	reply	created	transaction
change	reply	changed	transaction
cancel	reply	canceled	transaction
close	reply	closed	transaction
delete	reply	deleted	transaction
execute	reply	executed	transaction

L'utilisation des verbes de demande se fait comme suit:

- "get" est utilisé pour demander des objets du type précisé par le nom du message
- "create" est utilisé pour créer des objets du type précisé par le nom
- "delete" est utilisé pour supprimer des objets, bien que parfois un objet n'est jamais réellement supprimé dans le système cible afin de conserver un historique
- "close" et "cancel" impliquent des actions relatives aux processus commerciaux comme la fermeture d'un ordre d'exécution ou l'annulation d'une demande de contrôle
- "change" est utilisé pour modifier des objets, mais il est important de noter qu'il peut y avoir des ambiguïtés à traiter dans les règles commerciales, particulièrement dans le cas de jeux de données complexes (les jeux de données complexes ont typiquement des relations N:1 et il est important d'être clair lorsque les relations s'ajoutent ou sont à remplacer par une mise à jour).
- "execute" est utilisé lorsqu'une transaction complexe est en cours de transmission au moyen d'un OperationSet, qui peut contenir plus d'un verbe.

La réponse à chacune des demandes ci-dessus utilise le verbe "reply". Les verbes d'évènements sont souvent la conséquence d'une demande; ainsi, "create" peut générer l'évènement "created". Les verbes utilisés pour les évènements utilisent la "forme passée" du verbe de la demande associée. Il n'existe aucune exigence qu'un évènement doive être initié par une demande; certains évènements peuvent être générés indépendamment de toute demande spécifique.

Il peut être nécessaire de définir les règles commerciales et de validation pour l'application de verbes dans certains cas particuliers. Cela est en partie vrai dans le sens où de nombreuses règles se trouvent hors des limites de l'UML et du Schéma XML

Il est également important de noter que les énumérations pour les verbes contenues dans le schéma de message XML standard s'écrivent en **minuscules**. Les majuscules sont par ailleurs pratiques pour des fins de documentation.

La norme CEI 61968-1 identifiait auparavant les verbes "update", "updated", "show", "subscribe", "unsubscribe" et "publish". Tous ont été dépréciés. La raison en est que "show"

est un synonyme de "reply" et que les verbes "subscribe", "unsubscribe" et "publish" font référence à des fonctions effectuées au sein de la couche transport (au moyen de JMS, par exemple).

6.2.3 Nouns

Les nouns sont utilisés pour identifier le type des informations qui sont échangées. Ils sont aussi appelés communément profils. Chaque noun a typiquement une définition de Schéma XML correspondante définie par un espace de noms qui lui est propre. Les nouns sont typiquement identifiés par les cas d'utilisation. Dans un message, le noun est utilisé pour identifier le type de charge utile ou le type d'objet à satisfaire ou ayant été satisfait. Parmi les nouns communs extraits de CEI 61968-9, on trouve:

- EndDeviceControls
- EndDeviceEvents
- MeterReadings

Si nécessaire, les nouns peuvent être définis pour distinguer le contenu des différents flux d'informations. Ils peuvent ne pas être définis en tant que classes dans un modèle UML, mais le contenu et la structure du noun sont définis au moyen de classes, d'attributs et de relations à partir d'un modèle UML.

6.2.4 Charges utiles

Chaque noun identifie une structure de charge utile qui est typiquement transmise au moyen d'un document XML conforme à un Schéma XML. La structure de la charge utile est généralement définie comme un profil contextuel à partir d'un modèle UML. C'est l'approche retenue pour définir les structures de messages par la CEI 61968-9.

La Figure 20 illustre la structure d'une charge utile résultant de la définition du profil contextuel:

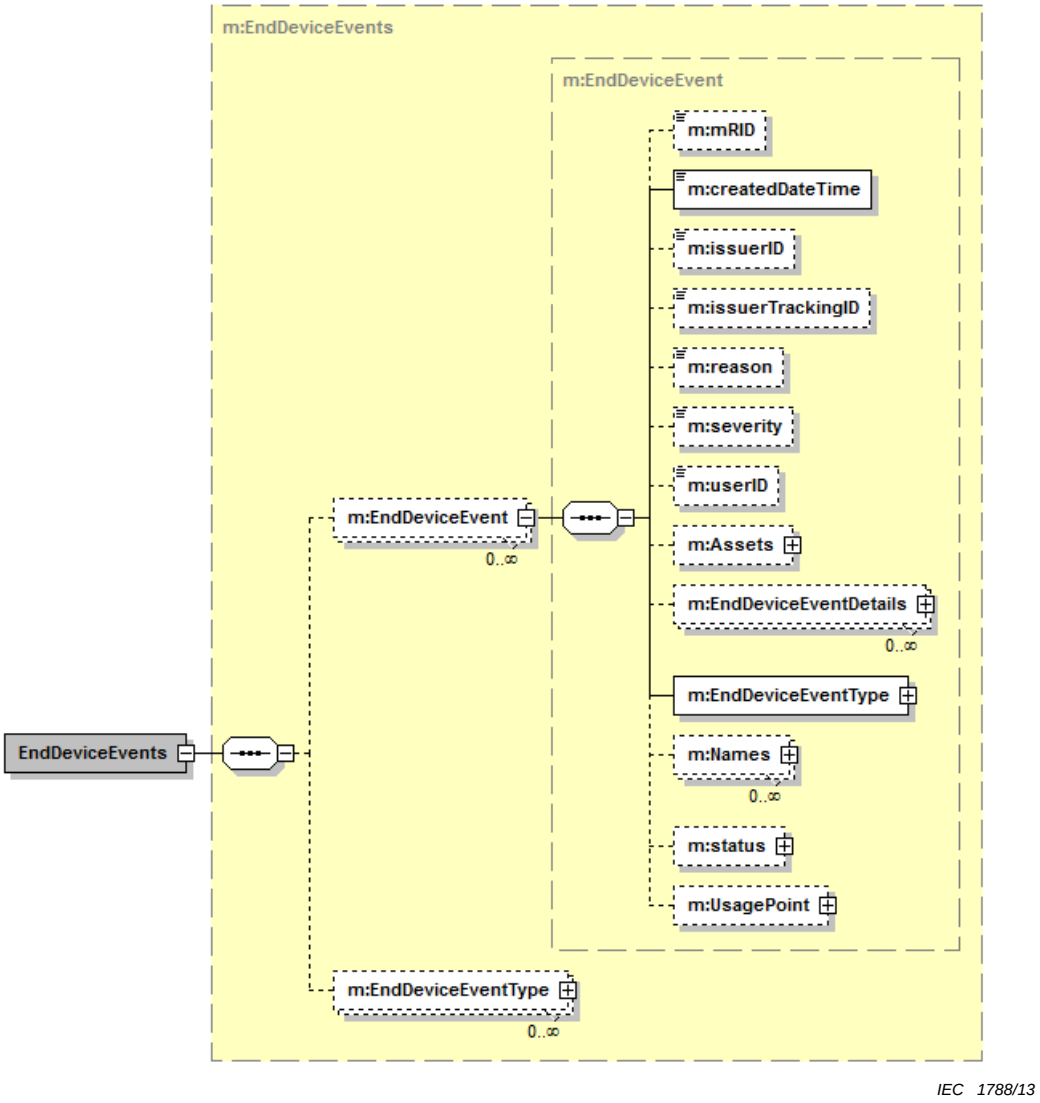


Figure 20 – Exemple de schéma de charge utile

Selon la situation, une charge utile peut être ou non requise dans un message. Une charge utile de message doit être utilisée dans les cas suivants:

- Lors de l'émission d'une demande "create"
- Lors de l'émission d'une demande "change"
- Lors de l'émission d'une demande "execute"
- Dans un message de réponse à une demande "get" réussie
- Pour tous les messages d'évènement ("created", "changed", "deleted" "closed", "canceled" ou "executed")

Dans les cas où un événement est la conséquence d'une demande transactionnelle et le noun de la demande est le même que celui de l'événement, la charge utile de la demande est copiée comme la charge utile de l'événement.

Il convient de ne pas utiliser la charge utile d'un message dans les cas suivants:

- Dans un message de réponse à une demande "get" échouée, dans lequel aucun résultat n'est envoyé
- Dans une réponse à une demande "create", "change", "delete" "close", "cancel" ou "execute"

- Dans une demande "delete" "close" ou "cancel", à condition que l'ID de l'objet ou des objets soit indiquée au moyen des éléments Request.ID
- Dans une demande "get", alors que les paramètres utilisés pour filtrer la demande sont fournis dans l'élément Request du message, de façon optionnelle au moyen d'un profil "Get" dans l'élément Request.any.

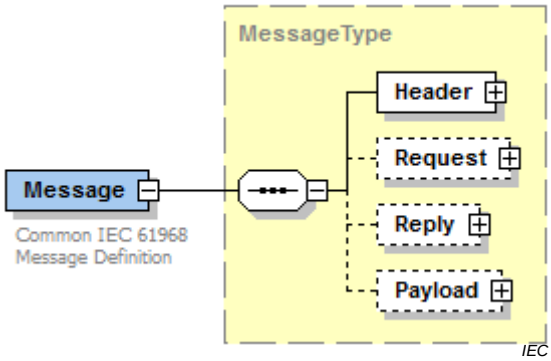
6.3 Enveloppe de message commune

6.3.1 Généralités

Sauf spécification contraire, tous les messages utilisent une enveloppe de message commune (CME, Common Message Envelope) dans laquelle un stéréotype prédéfini est utilisé pour les demandes et un autre stéréotype est utilisé pour les réponses. Il y a également des stéréotypes pour les événements et les défauts. Cette structure est basée sur les recommandations de la CEI 61968-1. Les messages sont construits autour de plusieurs sections, dont:

- Header: (en-tête) Requis pour tous les messages (à l'exception des messages de réponse de défaut), qui utilisent une structure commune à toutes les interfaces de service.
- Request: (demande) Optionnelle, elle définit les paramètres communément utilisés qui sont nécessaires pour qualifier les demandes de requêtes "get" ou pour identifier les objets spécifiques pour les demandes "delete", "cancel" ou "close". Une disposition permet l'inclusion d'une structure complexe au moyen d'un élément Payload.any. Dans le cas d'une demande visant à obtenir des MeterReadings, un profil "GetMeterReadings" peut être défini afin de passer les qualificateurs de demande. Dans un tel cas, il convient de nommer le profil selon la convention "Get<Noun>". Non utilisé pour les messages d'évènement ou de réponse.
- Reply: (réponse) Requise uniquement pour les messages de réponse afin d'indiquer le succès, l'échec et les détails des erreurs. Non utilisé pour les messages de demande ou d'évènements.
- Payload: (charge utile) Utilisée pour transmettre les informations du message en réponse à la combinaison "Verb" et "Noun" de l'En-tête du message. Requise pour les demandes "create", "change" et "execute". Elle est également requise pour les messages d'événement. Optionnelle dans les autres cas, conformément à la description qui en est faite plus loin dans le présent document, particulièrement dans l'Annexe B. La structure de la charge utile fournit des options pour la compression de la charge utile.

La Figure 21 propose une vue générique de la structure du message de haut niveau:



Légende

Anglais	Français
Common IEC 61968 Message Definition	Définition de message CEI 61968 commune

Figure 21 – Enveloppe de message commune

A partir de cette enveloppe de message commune, quatre stéréotypes identifient le sous-ensemble d'éléments spécifiques utilisés pour les demandes, les réponses, les évènements et les défauts:

- RequestMessage
- ResponseMessage
- EventMessage
- FaultMessage

Lorsque ces stéréotypes sont utiles lors de la définition des interfaces pour différencier clairement les demandes des réponses, des évènements et des défauts, il est courant d'utiliser en interne la structure de Message la plus générique au sein d'un logiciel tel que les services et intermédiaires communs.

6.3.2 Structure de l'en-tête du message

La structure de l'en-tête est commune aux messages de demande, de réponse et d'évènements. L'en-tête a actuellement deux champs requis devant être remplis. Ce sont:

- Verb (verbe), afin d'identifier l'action spécifique à entreprendre. Il existe un ensemble énuméré de verbes valides, dont les valeurs communément utilisées comprennent "get", "create", "change", "cancel", "close", "execute" et "reply". Dans les messages de notification d'évènements, on utilise des verbes au passé, parmi lesquels se trouvent "created", "changed", "canceled", "closed" et "executed", Il convient de vérifier que les mises en œuvre traitent les verbes déconseillés "update" et "updated" comme des synonymes de "change" et de "changed".
- Noun: pour identifier le sujet de l'action et/ou le type de charge utile, comme MeterReadings, Notification, etc.

Les champs qui peuvent être fournis en option comprennent les valeurs suivantes:

- Revision: Pour indiquer la révision de la définition du message. Il convient de définir cette option sur '1' par défaut.
- ReplayDetection: Elément complexe avec un marqueur temporel et un nonce utilisés pour prévenir les attaques de relecture. Le marqueur temporel est généré par le système source pour indiquer à quel moment le message a été créé. Le nonce est un numéro ou une chaîne de séquence générée aléatoirement (par exemple UUID) qui n'est pas répété par le système source pendant un jour au moins. Cela sert à améliorer le cryptage.
- Context: Une chaîne qui peut servir à identifier le contexte du message. Cela peut aider à fournir une protection du niveau d'application contre la consommation incorrecte de messages dans les configurations où il peut y avoir des environnements système multiples tournant sur la même infrastructure de messagerie. PRODUCTION, TESTING, STUDY et TRAINING en sont quelques exemples.
- Timestamp: Une chaîne certifiée ISO-8601 qui identifie le moment où le message a été envoyé. Cela est identique au JMSTimestamp fourni par JMS. Il peut s'agir de l'heure zulu ("z") ou d'une heure présentant un décalage horaire.
- Source: Identification de la source du message, qu'il convient de définir sur le nom du système ou de l'organisation.
- AsyncReplyFlag: Un booléen ("vrai" ou "faux") qui indique si un message de réponse sera envoyé de manière asynchrone. On considère par défaut que les réponses sont envoyées de manière synchrone.
- ReplyAddress: L'adresse à laquelle il convient d'envoyer les réponses, qui est généralement utilisée pour les réponses asynchrones. Il convient de lui donner la forme d'une URL, d'un nom de thème ou d'un nom de file d'attente. Ce champ ressemble au champ JMSReplyTo fourni par JMS. Ignoré lors de l'utilisation de modèles d'intégration unidirectionnels (par exemple AckRequired=false). Si l'adresse de réponse est un thème, il convient d'ajouter le préfixe 'topic:' au nom du thème. Si l'adresse de réponse est une

file d'attente, il convient d'ajouter le préfixe 'queue:' au nom de la file d'attente. Si l'adresse de réponse est un service Web, il convient de faire commencer l'URL qui représente l'adresse de réponse par 'http://' ou par 'https://'.

- AckRequired: Booléen ("vrai" ou "faux") qui indique si un acquittement est ou non requis. Si faux est choisi, cela indiquerait que le modèle d'intégration unidirectionnel est utilisé pour envoyer des messages transactionnels.
- User: Structure complexe identifiant l'utilisateur et l'organisation associée. Il convient de fournir cet élément qui peut être requis par certaines interfaces, en fonction des mises en œuvre sous-jacentes. Cela autorise de considérer la chaîne UsersID et la chaîne facultative Organization comme des sous-éléments.
- MessageID: Chaîne identifiant de manière unique un message. Il convient d'utiliser un UUID ou un numéro de séquence. Cela est identique au JMSMessageID fourni par JMS. Il convient pour un processus de ne pas émettre deux messages avec la même valeur MessageID.
- CorrelationID: Utilisé pour "relier" des messages entre eux. Peut être fourni dans une demande, de sorte que le client puisse corréler un message de réponse correspondant. Le serveur placera la valeur CorrelationID entrante comme CorrelationID de la réponse sortante. S'il n'est pas fourni dans la demande, il convient de définir le CorrelationID de la réponse sur la valeur de MessageID qui a été utilisée dans la demande, le cas échéant. Cela est identique au JMSCorrelationID fourni par JMS. Compte-tenu du fait que le CorrelationID est utilisé pour "relier" des messages, il peut être réutilisé dans plus d'un message. Il convient d'utiliser un UUID ou un numéro de séquence.
- Comment: Tout texte descriptif, mais ne doit jamais être utilisé pour une logique de traitement.
- Property: Type complexe permettant de transmettre des paires nom/valeur personnalisées. La source et la cible nécessitent de s'entendre sur l'utilisation. Identiques à une Propriété telle que définie par JMS.
- any: Peut être utilisé pour des extensions personnalisées

La Figure 22 décrit la structure de l'en-tête utilisée pour les messages de demande, de réponse et d'évènement.

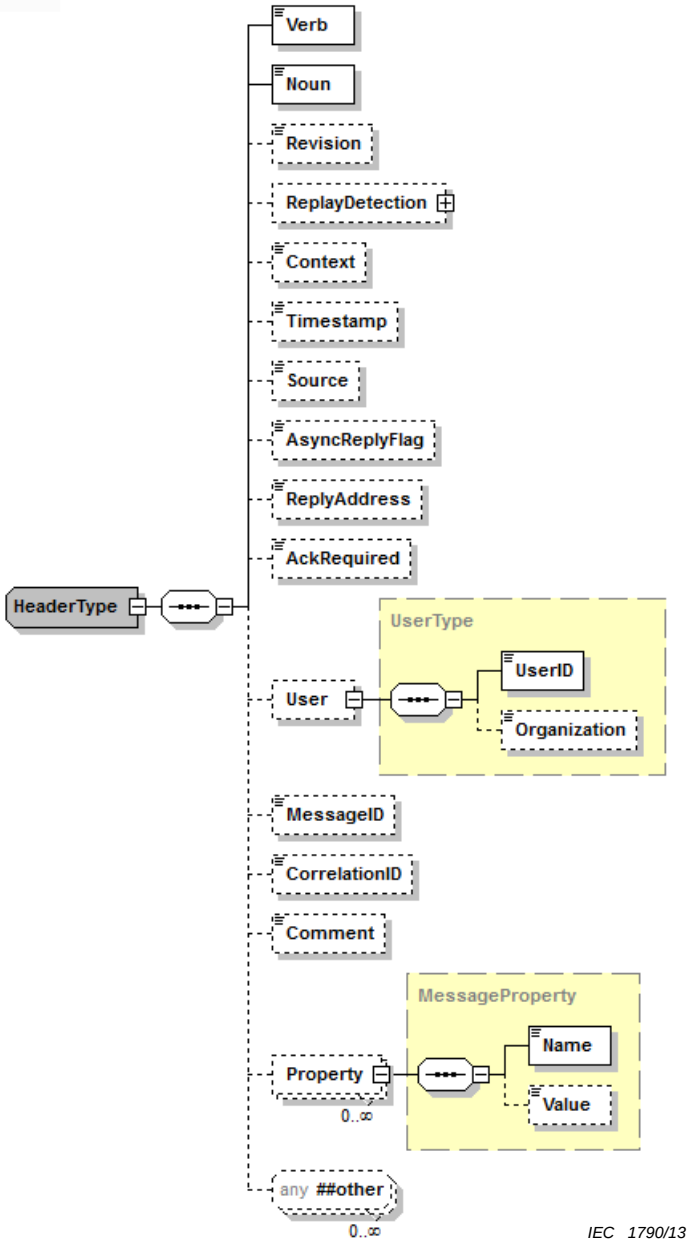


Figure 22 – Structure commune de l'en-tête de message

Bien que seuls deux éléments soient requis, verb et noun, de nombreux éléments optionnels peuvent être ajoutés. Dans la Figure 22, les éléments optionnels sont représentés par des bordures pointillées.

Ce qui suit est un exemple en XML d'un message remplissant tous les champs de l'en-tête.

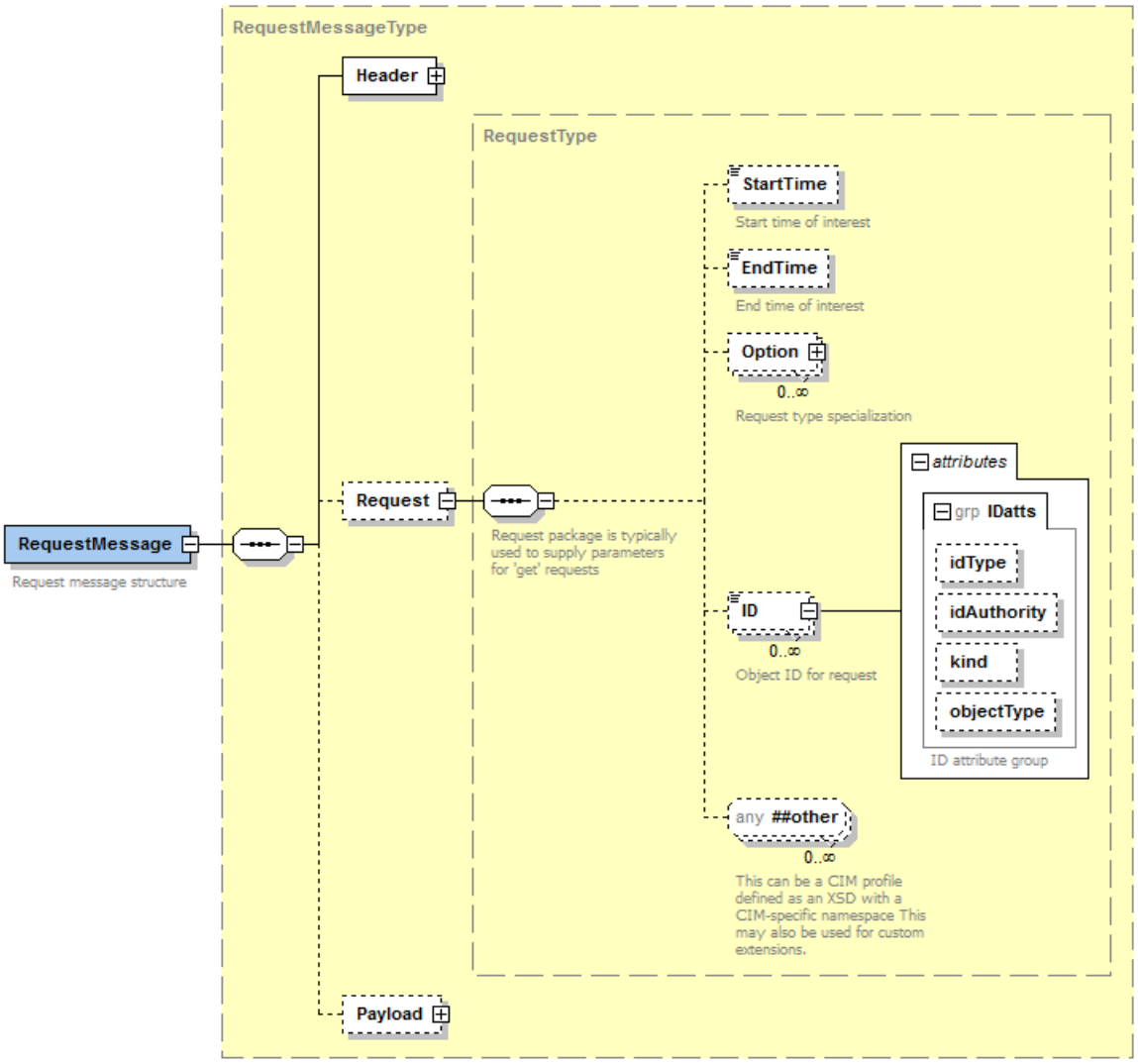
```
<?xml version="1.0" encoding="UTF-8"?>
<RequestMessage xsi:schemaLocation="http://iec.ch/TC57/2011/schema/message
Message.xsd" xmlns="http://iec.ch/TC57/2011/schema/message"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <Header>
              <Verb>get</Verb>
              <Noun>LoadForecast</Noun>
              <Revision>1</Revision>
              <ReplayDetection>
                     <Nonce>dcd98b7102dd2f0e8b11d0f600bfb0c093</Nonce>
                     <Created>2012-12-16T09:30:47.0Z</Created>
              </ReplayDetection>
              <Context>PRODUCTION</Context>
              <Timestamp>2001-12-16T09:30:47.0Z</Timestamp>
              <Source>EMS</Source>
```

```
                <AsyncReplyFlag>false</AsyncReplyFlag>
                <ReplyAddress>queue:EMS.ReplyQueue</ReplyAddress>
                <AckRequired>true</AckRequired>
                <User>
                        <UserID>Bob</UserID>
                        <Organization>Scheduling</Organization>
                </User>
                <MessageID>3432626</MessageID>
                <CorrelationID>3432626</CorrelationID>
                <Comment>Example message</Comment>
                <Property>
                        <Name>timeout</Name>
                        <Value>10</Value>
                </Property>
        </Header>
        <Request>
                <StartTime>2012-12-17T00:00:00.0Z</StartTime>
                <EndTime>2012-12-17T24:00:00.0Z</EndTime>
        </Request>
</RequestMessage>
```

Dans les cas où le message est transmis par un moyen comme SOAP ou JMS, il existe une certaine redondance entre les champs optionnels de l'enveloppe du message et l'en-tête au niveau du transport. Dans ces cas, les deux champs peuvent simplement être ajustés sur la même valeur. Dans les cas où ils présentent des différences, ils doivent être utilisés comme il convient pour l'enveloppe de message au niveau du transport et de l'application.

6.3.3 Structures de RequestMessage (message de demande)

La Figure 23 décrit la structure d'un message de demande devant être utilisé conjointement à un message ou une opération WSDL.



Légende

Anglais	Français
Request message structure	Structure de demande de message
Start time of interest	Début de la période étudiée
End time of interest	Fin de la période étudiée
Request type specialization	Spécialisation du type de demande
Request package is typically used to supply parameters for 'get' requests	Le paquetage de demande est généralement utilisé pour fournir les paramètres des demandes "get"
Object ID for request	ID d'objet de la demande
This can be a CIM profile defined as an XSD with a CIM-specific namespace. This may also be used for custom extensions.	Il peut s'agir d'un profil CIM défini en tant que XSD avec un espace de nom CIM. Cela peut également être utilisé pour les extensions personnalisées.

Figure 23 – Structure d'un RequestMessage

Le RequestMessage peut également contenir en option un élément dont les paramètres sont pertinents pour la demande, appelé Request. Une des utilisations principales du RequestType est d'éviter le placement de paramètres de demande spécifiques à l'application dans l'en-tête ou dans les définitions de charge utile.

Aucun élément requis ne se trouve dans l'élément Request. L'utilisation des éléments dans l'élément Request est décrite comme suit:

- StartTime: Utilisé lorsqu'une requête nécessite de préciser une heure de début comme filtre, mais un tel paramètre n'est pas fourni dans un profil ''Get''. Si les deux existent, cela est ignoré.
- EndTime: Utilisé lorsqu'une requête nécessite de préciser une heure de fin comme filtre, mais un tel paramètre n'est pas fourni dans un profil ''Get''. Si les deux existent, cela sera ignoré
- Option: Utilisé lorsque des paires nom/valeur sont utiles pour filtrer une requête ou transmettre des options de demande générale ou personnalisée, Des exemples d'utilisation concernent la spécification d'une valeur de temporisation d'une transaction ou la spécification d'un mode de réponse comme étant "Agrégé" ou "En flux". A l'heure actuelle, il n'existe aucune énumération normative de ces valeurs.
- ID: Utilisé lorsque l'ID d'un ou plusieurs objets est nécessaire pour filtrer une demande de requête. Peut également servir à identifier les objets spécifiques dans le cadre de transactions "delete", "cancel" ou "close". Chaque ID peut spécifier des attributs, en premier lieu pour identifier le type d'ID, qui peut être un nom, un UUID, une transaction ou autre. L'UUID par défaut est utilisé pour les valeurs mRID. S'il s'agit d'un nom, l'idType et l'idAuthority peuvent être précisés.
- any: Utilisé pour tout élément d'un profil "Get<Noun>" (par exemple, GetMeterReadings) à transmettre avec des paramètres qui peuvent qualifier une demande. Peut également être utilisé pour d'autres extensions non standard. Dans les cas où un profil "Get" est utilisé, les éléments définis dans le profil "Get" prennent la priorité sur les éléments StartTime, EndTime et ID. Ceci reconnaît l'asymétrie entre les informations requises pour qualifier une demande et les informations qui sont renvoyées dans une réponse.

Des situations qui peuvent utiliser la paire Nom/valeur optionnelle peuvent être décrites comme une partie d'autres normes. Dans certains cas, il peut être décidé qu'il convient de modifier d'autres éléments de demande standard (par exemple, les éléments Get<Noun> décrits dans la Figure 25) afin de faciliter une telle demande.

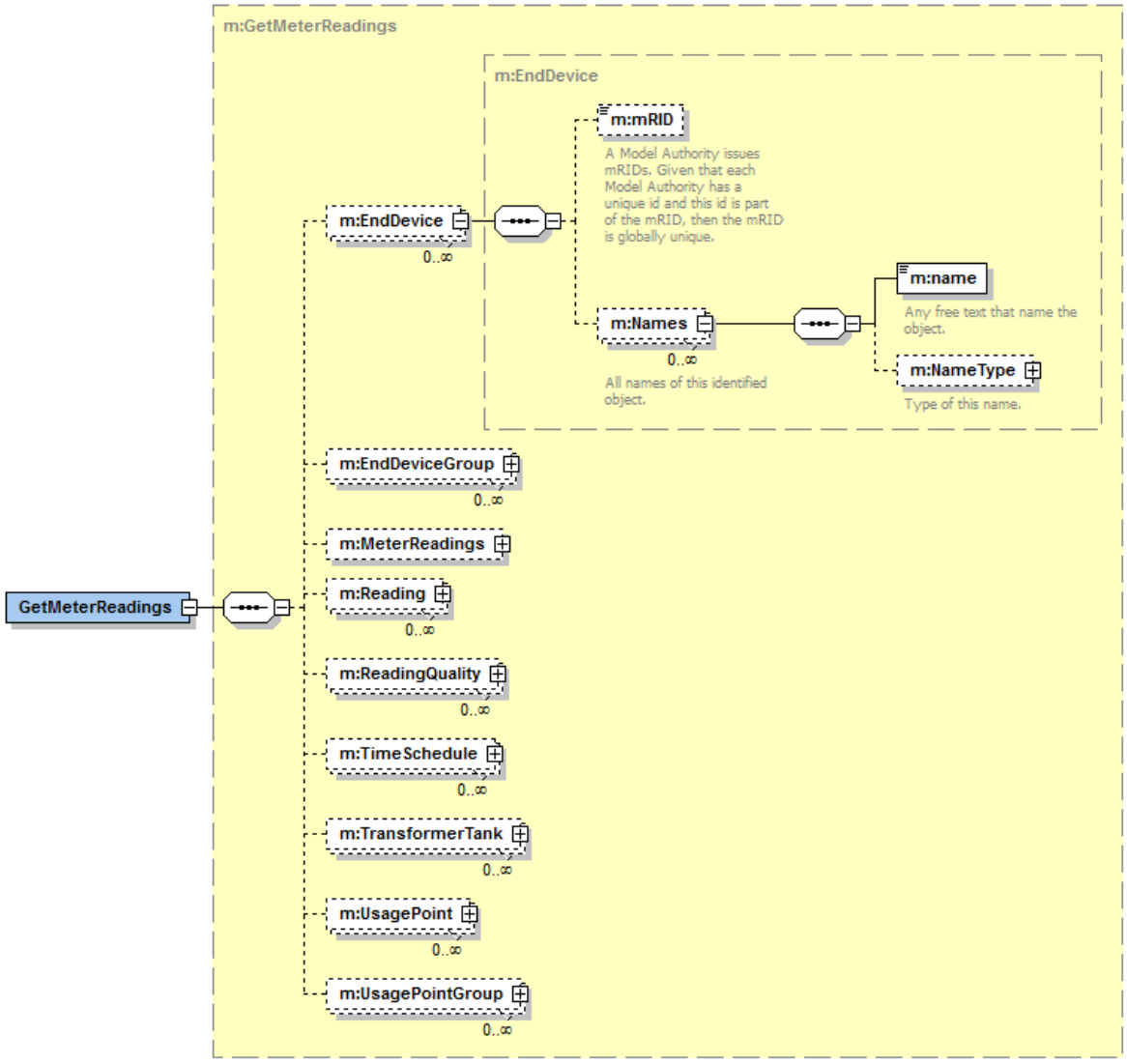
La Figure 24 présente un exemple de RequestMessage où les éléments Request.ID sont utilisés pour identifier les objets intéressants:

```
<ns0:RequestMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>get</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:Revision>1</ns0:Revision>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Request>
   <ns0:ID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</ns0:ID>
   <ns0:ID>e6d957ba-792a-4fcf-9f33-fd176a66dee8</ns0:ID>
   <ns0:ID>567fdc86-0ccd-4a96-a318-bdc1a3015643</ns0:ID>
</ns0:Request>
</ns0:RequestMessage>
```

IEC 1792/13

Figure 24 – Exemple XML de RequestMessage

Il convient d'utiliser l'élément 'any ##other' lorsque des paramètres de demande plus complexes sont nécessaires pour qualifier une demande afin que le message de réponse qui en résulte soit correctement filtré. La Figure 25 présente un exemple d'élément "GetMeterReadings" utilisé pour fournir des qualificateurs aux demandes MeterReadings.



IEC 1793/13

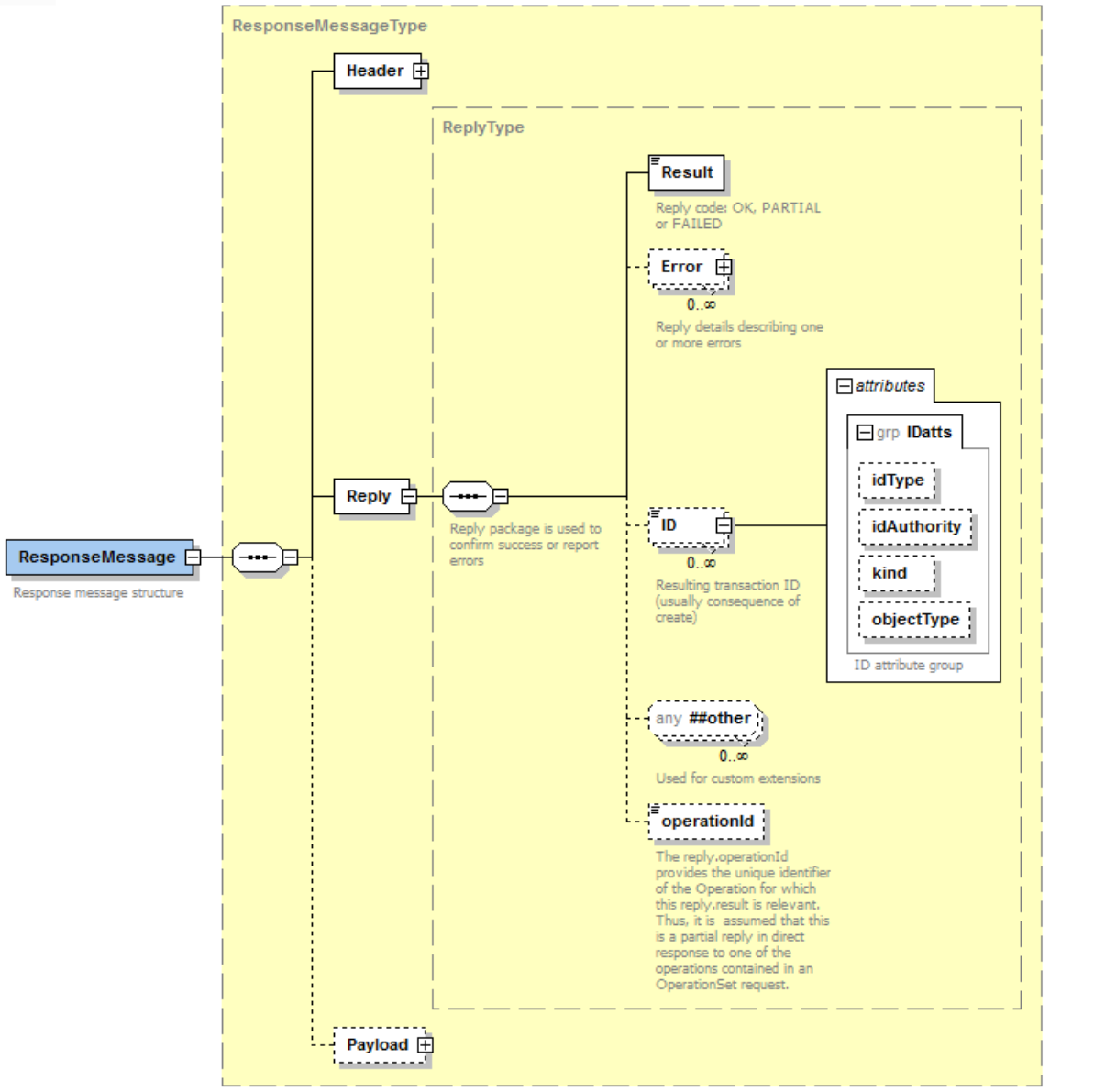
Légende

Anglais	Français
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part or the mRID, then the mRID is globally unique.	Un ModelingAuthority (c'est-à-dire une autorité de modélisation) émet des mRID. Sachant que chaque ModelingAuthority a un identificateur unique et que cet identificateur fait partie du mRID, donc le mRID est unique au niveau global
All names of this identified object.	Tous les noms de cet objet identifié.
Any free text that name the object.	Tout texte libre qui désigne l'objet.
Type of this name.	Type de ce nom.

Figure 25 – Exemple de profil ''Get<Noun>''

6.3.4 Structures de ResponseMessage (message de réponse)

La Figure 26 décrit la structure d'un message de réponse devant être utilisé conjointement à un message ou une opération WSDL, en réponse au message de demande.



Légende

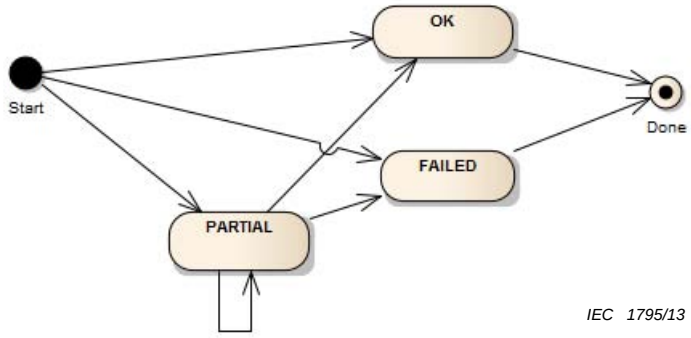
Anglais	Français
Response message structure	Structure de réponse de message
Reply code: OK, PARTIAL or FAILED	Code de réponse: ok, partiel ou en échec
Reply details describing one or more errors	Détails de réponse décrivant une ou plusieurs erreurs.
Reply package is used to confirm success or report issues	Le paquetage de réponse est utilisé pour confirmer le succès ou rapporter des problèmes
Resulting transaction ID (usually consequence of create)	Identificateur de transaction obtenu (habituellement une conséquence de create)
Used for custom extensions	Utilisé pour les extensions personnalisées
The reply-operationId provides the unique identifier of the Operation for which this reply.Result is relevant. Thus, it is assumed that this is a partial reply in direct response to one of the operations contained in the OperationSet request.	reply.operationId fournit l'identifiant unique de l'opération pour laquelle ce reply.result est adapté. Il est donc supposé qu'il s'agit d'une réponse partielle qui répond directement à une des opérations contenues dans une demande OperationSet.

Figure 26 – Structure d'un ResponseMessage

Le résultat de la Réponse est énuméré dans le Message.xsd devant être rempli de la façon suivante:

- "OK" s'il n'y a aucune erreur et si tous les résultats ont été renvoyés. Il n'existe aucune exigence qu'un élément Reply.Error soit présent.
- "PARTIAL" (partiel) si seul un ensemble partiel des résultats a été renvoyé, avec ou sans erreurs. L'existence d'erreurs est indiquée par un ou plusieurs éléments de code Reply.Error.
- "FAILED" (échec) si aucun résultat ne peut être renvoyé en raison d'une ou plusieurs erreurs. Indiqué par un ou plusieurs éléments Reply.Error, chacun comprenant un "code" de niveau d'application obligatoire.

Cela est représenté par le diagramme de transition d'état montré à la Figure 27.



Légende

Anglais	Français
Start	Début
Done	Fin
PARTIAL	PARTIEL
ERROR	ERREUR

Figure 27 – Etats du message de réponse

Il peut également y avoir des informations d'erreurs plus spécifiques fournies au sein de la charge utile elle-même. La Figure 28 décrit les détails d'un élément Error, ce qui permet la création de messages d'erreur lisibles à la fois par l'homme et la machine. La structure permet l'utilisation de XPath pour autoriser les références spécifiques à des éléments contenus dans un document XML.

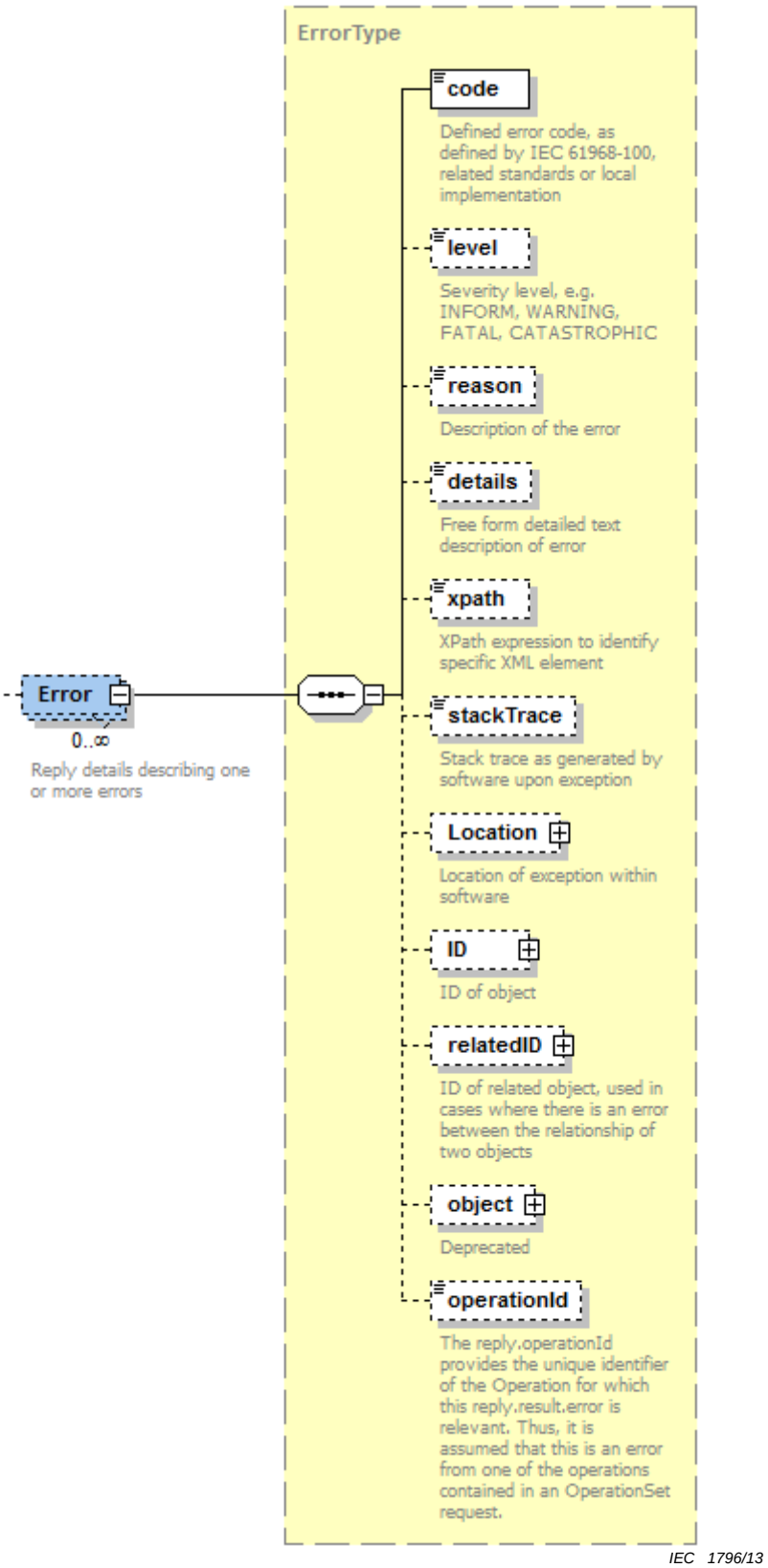
Si toute la réponse à un message de demande est fournie dans un seul message de réponse et si ce dernier ne contient aucune erreur fatale, alors le résultat Reply.Result est réglé sur ‘’OK’’ et le code Reply.Error.code est ajusté sur la valeur "0.0".

Autrement, si toute la réponse à un message de demande est renvoyée dans un message unique et si le message de réponse contient au moins une erreur fatale, alors Reply.Result est réglé sur ‘’FATAL’’ Un tel message peut contenir une association d’éléments de données et de notifications d’erreurs. Le Reply.Error.code, Reply.Error.ID et d’autres attributs de la structure associée Reply.Error sont donc réglés de façon appropriée pour chaque erreur fatale ou condition informationnelle rapportée.

Si le système répondant envoie des messages de réponse multiples à un message de demande, le Reply.Result est réglé sur "PARTIAL" De tels messages peuvent contenir une association d’éléments de données et de notifications d’erreurs. Il doit y avoir au moins un

Reply.Error.code de "0.2" ou "0.1", selon qu'un message de réponse donné est le dernier dans une séquence ou non (*).Le Reply.Error.code, Reply.Error.ID et d'autres attributs de la structure associée Reply.Error sont donc réglés de façon appropriée pour chaque erreur fatale ou condition informationnelle rapportée.

Dans le cas où le système répondant ne peut déterminer le dernier message dans un jeu de messages de réponse, alors tous les messages du jeu sont à envoyer avec un Reply.Error.code = "0.1".



Légende

Anglais	Français
Defined error code, as defined by IEC 61968-100 related standards or local implementation	Code d'erreur défini, comme défini par la CEI 61968-100, les normes associées ou la mise en œuvre locale.
Severity level, e.g. INFORM, WARNING, FATAL, CATASTROPHIC	Niveau de gravité, par exemple INFORM, WARNING, FATAL, CATASTROPHIC
Description of the error	Description de l'erreur
Free form detailed text description of error	Description sous forme de texte libre de l'erreur
XPath expression to identify specific XML element	Expression XPath pour définir un élément XML spécifique
Stack trace as generated by software upon exception	Trace de pile générée par le logiciel en cas d'exception
Location of exception within software	Emplacement de l'exception dans le logiciel
ID of object	ID de l'objet
ID of related object, used in cases where there is an error between the relationship of two objects	ID de l'objet associé, utilisé en cas d'erreur dans la relation entre les deux objets
Deprecated	Déconseillée
The reply.operationId provides the unique identifier of the Operation for which this reply.result.error is relevant. Thus, it is assumed that this is an error from one of the operations contained in an OperationSetrequest.	reply.operationId fournit l'identifiant unique de l'opération pour laquelle ce reply.result.error est adapté. Il est donc supposé qu'il s'agit d'une erreur de l'une des opérations contenues dans une demande OperationSet.

Figure 28 – Structure de Error (erreur)

La Figure 29 est un exemple de ResponseMessage:

```
<ns0:ResponseMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>reply</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Reply>
   <ns0:Result>OK</ns0:Result>
</ns0:Reply>
<ns0:Payload>
   <m:Switches xsi:schemaLocation="http://iec.ch/TC57/2012/Switches# Switches.xsd"
xmlns:m="http://iec.ch/TC57/2012/Switches#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <m:Switch>
         <m:mRID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>e6d957ba-792a-4fcf-9f33-fd176a66dee8</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>567fdc86-0ccd-4a96-a318-bdc1a3015643</m:mRID>
         <m:normalOpen>false</m:normalOpen>
      </m:Switch>
   </m:Switches>
</ns0:Payload>
</ns0:ResponseMessage>
```

IEC 1797/13

Figure 29 – Exemple XML de ResponseMessage

La Figure 30 est un exemple de ResponseMessage dans lequel la charge utile est compressée:

```
<ns0:ResponseMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>reply</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Reply>
   <ns0:Result>OK</ns0:Result>
</ns0:Reply>
<ns0:Payload>
   <ns0:Compressed>dghuywqeiwihn353218u23hb2b3b3bhu</ns0:Compressed>
   <ns0:format>XML</ns0:format>
</ns0:Payload>
</ns0:ResponseMessage>
```

IEC 1798/13

Figure 30 – Exemple XML de Compression de charge utile

La Figure 31 est un exemple de ResponseMessage ayant renvoyé une erreur:

```
<ns0:ResponseMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>reply</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:Revision>1</ns0:Revision>
   <ns0:CorrelationID>1729363b5b7d9c6a0a88d02ae97c64b0</ns0:CorrelationID>
</ns0:Header>
<ns0:Reply>
   <ns0:Result>FAILED</ns0:Result>
   <ns0:Error>
      <ns0:code>2.15</ns0:code>
      <ns0:level>WARNING</ns0:level>
      <ns0:details>Unknown object: e6d957ba-792a-4fcf-9f33-fd176a66dee8</ns0:details>
   </ns0:Error>
</ns0:Reply>
<ns0:Payload>
   <m:Switches xsi:schemaLocation="http://iec.ch/TC57/2012/Switches# Switches.xsd"
xmlns:m="http://iec.ch/TC57/2012/Switches#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <m:Switch>
         <m:mRID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>567fdc86-0ccd-4a96-a318-bdc1a3015643</m:mRID>
         <m:normalOpen>false</m:normalOpen>
      </m:Switch>
   </m:Switches>
</ns0:Payload>
</ns0:ResponseMessage>
```

IEC 1799/13

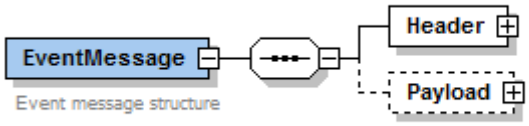
Figure 31 – Exemple XML d'un Error ResponseMessage

Un des avantages importants de la compression de charge utile par rapport à l'utilisation de pièces jointes SOAP réside dans la signature, car la signature SOAP ne signe PAS le contenu de la pièce jointe mais seulement le corps du message. Grâce à la compression de la charge utile, la signature couvre la charge utile, en empêchant le rejet.

6.3.5 Structures de EventMessage (message d'événement)

Un EventMessage est typiquement publié pour rapporter une condition présentant un intérêt potentiel. Les verbes utilisés dans un message d'évènement sont au passé (created, changed, canceled, etc.). Un EventMessage n'inclut pas les paramètres de demande ou de réponse, simplement un en-tête et habituellement une charge utile.

La Figure 32 décrit la structure d'un EventMessage.



IEC 1800/13

Légende

Anglais	Français
Event message structure	Structure d'événement de message

Figure 32 – Structure de EventMessage

La Figure 33 est un exemple de EventMessage:

```
<ns0:EventMessage xmlns:ns0 = "http://www.iec.ch/TC57/2011/schema/message">
<ns0:Header>
   <ns0:Verb>changed</ns0:Verb>
   <ns0:Noun>Switches</ns0:Noun>
   <ns0:Revision>1</ns0:Revision>
</ns0:Header>
<ns0:Payload>
   <m:Switches xsi:schemaLocation="http://iec.ch/TC57/2012/Switches# Switches.xsd"
xmlns:m="http://iec.ch/TC57/2012/Switches#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <m:Switch>
         <m:mRID>b9cd8d2a-56a2-45e3-89d0-caaabb9e2985</m:mRID>
         <m:normalOpen>false</m:normalOpen>
      </m:Switch>
      <m:Switch>
         <m:mRID>567fdc86-0ccd-4a96-a318-bdc1a3015643</m:mRID>
         <m:normalOpen>true</m:normalOpen>
      </m:Switch>
   </m:Switches>
</ns0:Payload>
</ns0:EventMessage>
```

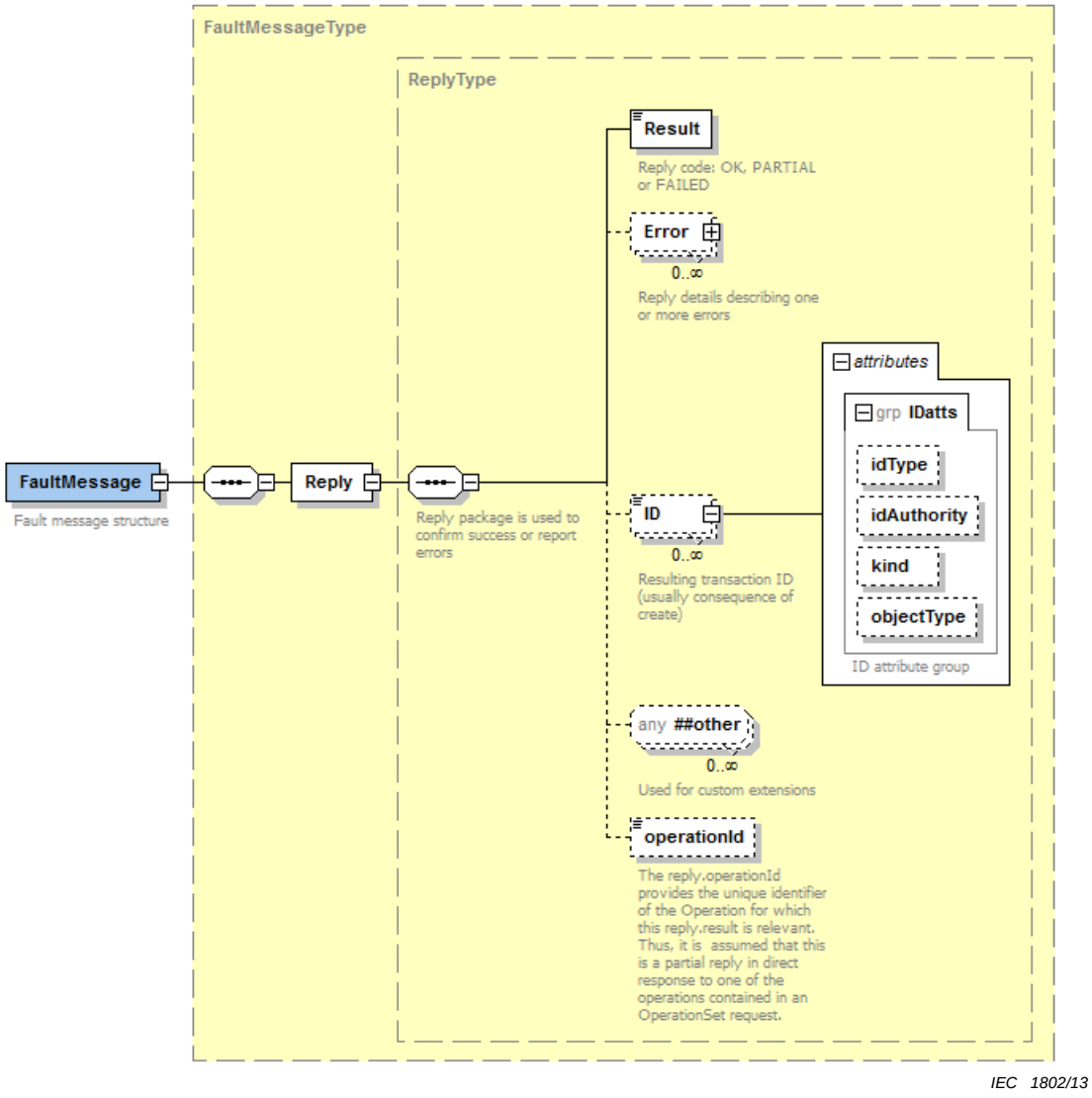
IEC 1801/13

Figure 33 – Exemple XML de EventMessage

NOTE Dans un EventMessage, le ‘’verb’’ est au passé comme, par exemple, ‘’created’’, ‘’changed’’, ‘’canceled’’, etc.

6.3.6 Structures de FaultMessage (message de défaut)

Un FaultMessage est typiquement utilisé dans la définition d'un WSDL et mis en œuvre par un service Web pour rapporter une condition défaillante en conséquence d'un échec de traitement d'un RequestMessage (par exemple détection d'un défaut SOAP). Il n'utilise qu'un élément de réponse (sans en-tête), car il peut ne pas avoir été capable d'interpréter même l'en-tête du RequestMessage. La Fault Message Structure est visualisée dans la Figure 34.



Légende

Anglais	Français
Fault message structure	Structure d'erreur de message
Reply code: OK, PARTIAL or FAILED	Code de réponse: OK, PARTIEL ou EN ECHEC
Reply details describing one or more errors	Détails de réponse décrivant une ou plusieurs erreurs.
Reply package is used to confirm success or report issues	Le paquetage de réponse est utilisé pour confirmer le succès ou rapporter des problèmes
Resulting transaction ID (usually consequence of create)	Identificateur de transaction obtenu (habituellement une conséquence de create)
Used for custom extensions	Utilisé pour les extensions personnalisées
The reply-operationId provides the unique identifier of the Operation for which this reply.result is relevant. Thus, it is assumed that this is a partial reply in direct response to one of the operations contained in the OperationSet request.	reply.operationId fournit l'identifiant unique de l'opération pour laquelle ce reply.result est adapté. Il est donc supposé qu'il s'agit d'une réponse partielle qui répond directement à une des opérations contenues dans une demande OperationSet.

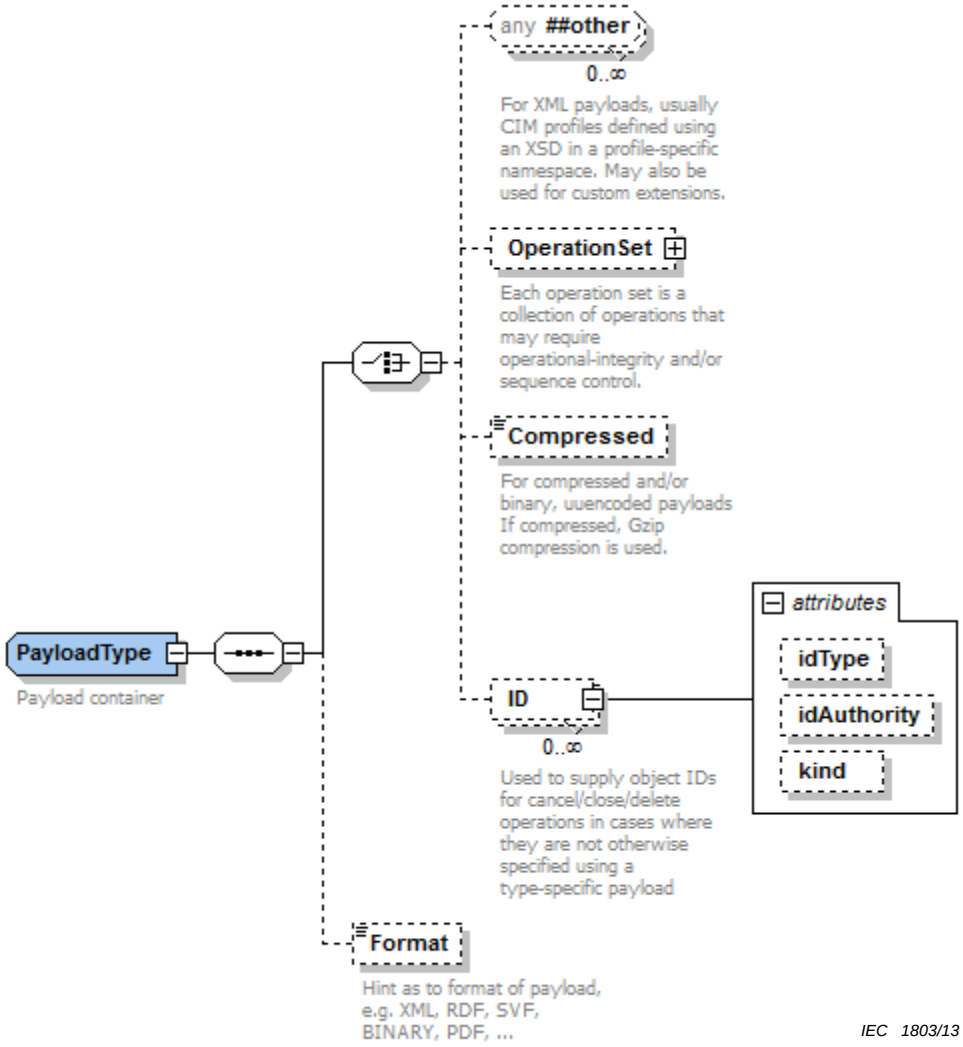
Figure 34 – Structure de FaultMessage

6.4 Structures de Payload (charge utile)

Ce paragraphe décrit deux formes de structures de charges utiles: générique et spécifique au modèle. Lorsque l'enveloppe de message commune définit la charge utile comme étant générique (ou indépendante du modèle), qui est l'utilisation commune avec à la fois JMS et les services Web génériques. Cependant, la définition des WSDL pour les services Web fortement typés peut nécessiter une définition de la charge utile d'une enveloppe de message variante spécifique au modèle.

Dans certains types de messages, la charge utile doit être fournie, comme ce serait le cas pour un message de demande avec un verbe "create" ou "change", certains messages de réponse et certains messages d'évènement. Les charges utiles contiennent typiquement des documents XML conformes à un schéma XML défini. Cependant, il y a des exceptions à cette règle. Certaines charges utiles XML peuvent ne pas avoir de schémas XML utiles, comme c'est le cas des fichiers RDF ou des résultats de requêtes dynamiques, ainsi que les formats non XML comme CSV et PDF.

Il peut également y avoir des cas dans lesquels une charge utile imposante doit être compressée, pour éviter qu'elle ne devienne très imposante et consomme une part significative de la bande passante du réseau. Afin de concilier une variété d'options de formats de la charge utile, on utilise la structure d'un élément générique de charge utile visualisée dans la Figure 35.



Légende

Anglais	Français
For XML payloads, usually CIM profiles defined using an XSD in a profile-specific namespace. May also be used for custom extensions.	Pour les charges utiles XML, généralement des profils CIM définis à l'aide d'un XSD dans un espace de nom spécifique au profil. Peut également être utilisé pour les extensions personnalisées.
Each operation set is a collection of operations that may require operational-integrity and/or sequence control.	Chaque OperationSet (jeu d'opération) est un ensemble d'opérations qui peut nécessiter une intégrité opérationnelle et/ou un contrôle de séquence.
For compressed and/or binary, uuencoded payloads. If compressed, Gzip compression is used.	Pour charges utiles compressées et/ou binaires sous forme uuencode. Le cas échéant, la compression Gzip est utilisée.
Payload container	Conteneur de charge utile
Used to supply object IDs for cancel/close/delete operations in cases where they are not otherwise specified using a type-specific payload	Utilisé pour fournir des ID d'objet pour les opérations d'annulation/de fermeture/de suppression lorsqu'ils ne sont pas spécifiés d'une autre manière avec une charge utile spécifique pour un type.
Hint as to format of payload, e.g. XML, RDF, SVF, BINARY, PDF, ...	Astuce sur le format de la charge utile, par exemple XML, RDF, SVF, BINARY, PDF, , etc.

Figure 35 – Conteneur de charge utile du message – Générique

Le Paragraphe 6.10 traite en détail de l'utilisation de l'élément OperationSet pour les transactions complexes.

Lorsque le conteneur de la charge utile du message générique est utilisé, tout type de document XML peut être inclus grâce à la structure XML "any". Alors que cela fournit des options de couplage lâche, des modèles complexes spécifiques définis par des schémas XML (XSD) peuvent également être utilisés.

Il y a également certains cas dans lesquels une chaîne encodée en base64 zippée est nécessaire et est filtrée grâce à la balise "Compressed" inclue dans le message. Le codage en base64 doit toujours être réalisé après la compression. L'élément "Compressed" est utilisé même lorsque des données binaires ne sont pas compressées. La compression Gnu Zip doit être utilisée afin de fournir la compatibilité avec les mises en œuvre Java et Microsoft .Net. Un exemple de Java est fourni dans l'Annexe F. Quelques exemples spécifiques de l'utilisation de la compression de la charge utile:

- Une charge utile XML, conforme à un schéma XML reconnu, dépasse une taille prédéfinie (par exemple 1 Mo, 5 Mo, 10 Mo, etc.). Cela est courant avec les échanges de modèles et les transactions du marché énergétique.
- Une charge utile présente un format non XML, comme un PDF, une feuille de calcul Excel, un fichier CSV ou une image binaire.
- Une charge utile est formatée au moyen d'XML, mais ne dispose pas de schéma XML et dépasse une taille prédéfinie (par exemple 1 Mo, 5 Mo, 10 Mo, etc.). Le cas d'un XML dynamique généré en réponse à une requête SQL XML devant renvoyer un ensemble de résultats XML constitue un exemple.

Lorsqu'une charge utile est compressée et encodée en base64, elle est stockée dans l'élément Payload/Compressed sous la forme d'une chaîne de caractères. De plus, afin de prendre en charge le transfert efficace des données binaires formatées, les données peuvent être encodées en base64 mais pas compressées. Cela est utilisé pour les données classées "grande vitesse", pour lesquelles le formatage XML ne satisferait pas aux exigences de performance.

L'élément ID et ses attributs associés peuvent être utilisés pour fournir des identificateurs aux transactions ou objets. Cela est utile dans les cas où une charge utile ne fournit pas d'identificateurs d'objets, comme cela peut être communément le cas des demandes ''cancel'', ''close'' ou ''delete'', les réponses aux demandes ''create'' ou les évènements qui utilisent les verbes au passé ''canceled'', ''closed'' ou ''deleted''.

L'élément Format peut être utilisé pour identifier des formats de données spécifiques comme XML, RDF, SVG, BINARY, PDF, DOC, CSV, etc. Cela est particulièrement utile si la charge utile est encodée en base64 et potentiellement compressée. L'utilisation de cette balise est optionnelle et ne devrait typiquement intervenir que lorsque la charge utile est stockée grâce à l'élément de message Payload/Compressed. Le Tableau 2 décrit les relations entre les éléments de la Charge utile, montrant une importante variété d'options de charges utiles.

Table 2 – Utilisations de charge utile

Tous	*OperationSet*	*Compressé*	*Format*	*Interprétation*
XML	Nul	Nul	Null ou 'XML'	La charge utile du message est contenue dans le "any", où le contenu est décrit par le Noun de l'en-tête. C'est l'utilisation la plus courante.
nul	Nul	String	'XML'	La charge utile du message est gzippée en encodée en base64 dans l'élément Compressed, où le contenu est décrit par le Noun de l'en-tête.
nul	XML	Nul	Null ou 'XML'	La transaction complexe est en cours de transmission grâce à l'élément OperationSet dans lequel le Verbe de l'en-tête est "execute" ou "executed"
XML	Nul	Nul	'RDF'	La charge utile du message est un document RDF tel que transmis grâce à l'élément "any"
nul	Nul	String	'RDF'	La charge utile du message est un document RDF compressé tel que transmis dans l'élément Compressed
nul	Nul	String	'PDF'	La charge utile du message est un document PDF compressé tel que transmis dans l'élément Compressed
nul	Nul	String	'GZIP'	La charge utile du message est une archive gzip d'un ou plusieurs fichiers telle que transmise dans l'élément Compressed
nul	Nul	String	"CSV"	La charge utile du message est un fichier CSV en cours de transmission dans l'élément Compressed
nul	Nul	String	'XLS'	La charge utile du message est un fichier Excel en cours de transmission dans l'élément Compressed
nul	Nul	String	'DOC'	La charge utile du message est un document Word en cours de transmission dans l'élément Compressed
nul	Nul	String	'TEXT'	La charge utile du message est un document texte compressé tel que transmis dans l'élément Compressed
nul	Nul	String	'JSON'	La charge utile du message est un objet JSON compressé tel que transmis dans l'élément Compressed
nul	Nul	String	"BINARY"	La charge utile du message est une structure binaire encodée en base64, mais non compressée. Tout autre aspect est spécifique à l'application.
nul	Nul	String	*Other*	La charge utile du message est un fichier compressé de tout "autre" format tel que transmis dans l'élément Compressed. Cela rend possible la définition de valeurs de Format personnalisées.

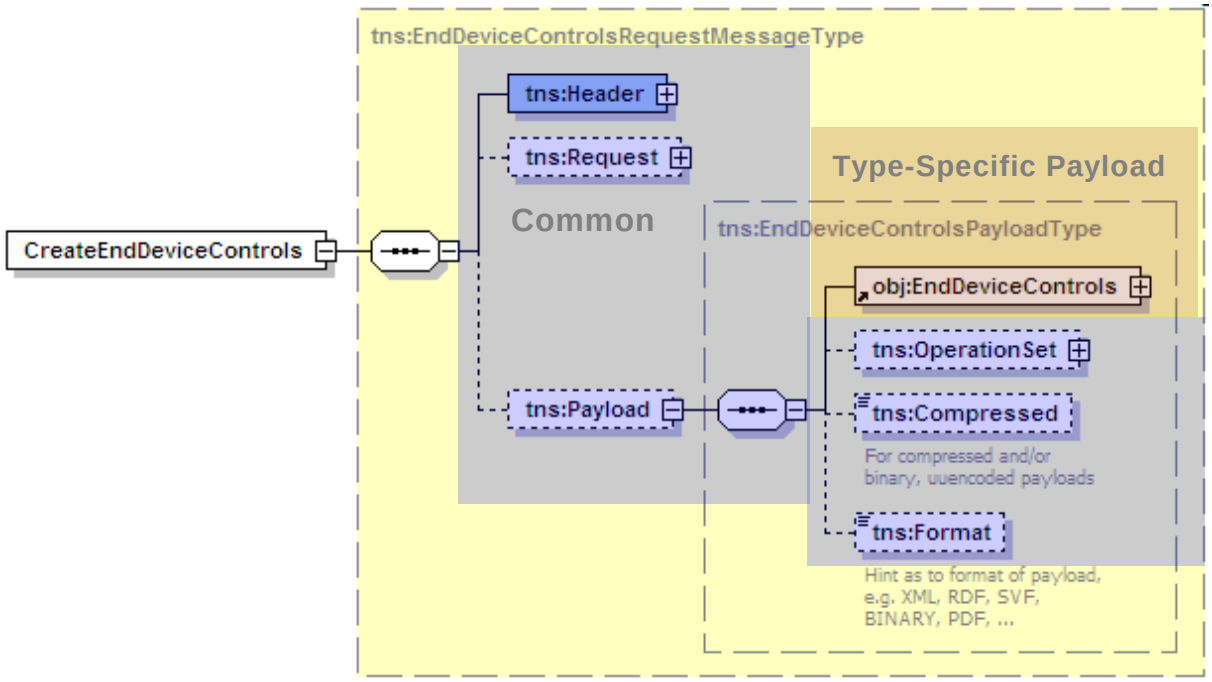
Ces options de charges utiles fournissent une alternative à l'utilisation des pièces jointes SOAP. Les pièces jointes SOAP sont plus difficiles à sécuriser car la signature de l'enveloppe SOAP signe le corps SOAP mais ne signe pas la pièce jointe. Cela requiert également que la charge utile soit traitée séparément du reste du message SOAP (par exemple le message est décomposé pour en extraire la charge utile. Celle-ci est ensuite séparée et traitée). Cependant, nous pensons que cette approche de la mise en œuvre est moins complexe qu'avec les pièces jointes SOAP.

6.5 Charges utiles fortement typées

Dans les cas où des WSDL fortement typés sont à définir avec des opérations spécifiques pour les combinaisons de verbe et de noun, l'enveloppe de message commune est redéfinie avec les deux changements suivants:

- Elément racine "Message" remplacé par un nom d'opération WSDL comme "CreatedEndDeviceControl". Cela prend en charge un problème de "signature de câble" du service Web si plusieurs opérations font référence à un élément XSD.
- La charge utile contient un élément type de message concret comme EndDeviceControls.

En conséquence, une structure de message pour l'opération de service de la demande CreatEndDeviceControls est souvent redéfinie comme l'indique la Figure 36:



Légende

Anglais	Français
For compressed and/or binary, uuencoded payloads	Pour charges utiles compressées et/ou binaires sous forme uuencode
Hint as to format of payload, e.g. XML, RDF, SVF, BINARY, PDF, ...	Astuce sur le format de la charge utile, par exemple XML, RDF, SVF, BINARY, PDF, , etc.
Type-Specific Payload	Charge utile selon le type

Figure 36 – Conteneur de charge utile du message – Spécifique au Type

NOTE L'élément Message est renommé en CreateEndDeviceControls pour cette définition de service et l'élément Payload contient un objet "EndDeviceControls" concret basé sur un profil CIM (ou un modèle de charge utile). Une différence essentielle à noter est le caractère fortement typé de l'élément de charge utile, contrairement à l'utilisation de ''any'' dans le Message.xsd normal isé. Ceci donne lieu à une version spécifique au type du Message.xsd par chaque profil conforme CEI 61968. Voir 8.3 et l'Annexe C pour plus de détails.

6.6 Enveloppe de message SOAP

SOAP a été largement utilisé comme spécification du protocole standard pour l'échange d'informations XML au moyen de services Web. Il fournit une enveloppe contenant un en-tête et un corps. La façon dont un message SOAP est structuré peut être définie dans la section de liaison WSDL comme dans l'exemple donné par la Figure 37:

```
<wsdl:binding name="EndDeviceControl_Binding" type="tns:EndDeviceControl">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="CreatedEndDeviceControl">
      <wsdl:documentation>CreatedEndDeviceControl binding</wsdl:documentation>
<soap:operation
soapAction="http://iec.ch/TC57/2010/EndDeviceControls/CreatedEndDeviceControls"
style="document"/>
      <wsdl:input name="CreatedEndDeviceControlRequest">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CreatedEndDeviceControlResponse">
         <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="CreatedEndDeviceControlFault">
         <soap:fault name="CreatedEndDeviceControlFault" use="literal"/>
      </wsdl:fault>
   </wsdl:operation>
   …
```

IEC 1805/13

Figure 37 – Liens SOAP

Dans ce WSDL est spécifié l'emplacement d'une charge utile d'entrée / sortie (soap:body – souligné) et le style de liaison (=document – souligné) qu'elle suit.

Un message SOAP (voir ci-dessous) peut être construit sur la base des informations de liaison WSDL. Avec SOAP, la structure du message apparaît dans le contexte du SOAP Body. Cela apparaît dans la Figure 38.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Header>                                              SOAP Header
   … …
</soapenv:Header>

<soapenv:Body>                                                SOAP Body
<mes:CreateEndDeviceControls>          Common
<mes:Header>
<mes:Verb>?</mes:Verb>
<mes:Noun>?</mes:Noun>
<!--Optional:-->
   … …
</mes:Header>
<!--Optional:-->
<mes:Request>
<!--Optional:-->
<mes:StartTime>?</mes:StartTime>
<!--Optional:-->
<mes:EndTime>?</mes:EndTime>
<!--Zero or more repetitions:-->
<mes:ID>?</mes:ID>
</mes:Request>
<!--Optional:-->
<mes:Payload>
<end:EndDeviceControls>                Type-Specific Payload
<end:EndDeviceControl>
<!--Optional:-->
<end:mRID>?</end:mRID>
<end:description>?</end:description>
<end:drProgramLevel>?</end:drProgramLevel>
      … …
</end:EndDeviceControl>
</end:EndDeviceControls>
<mes:Format>?</mes:Format>
</mes:Payload>
</mes:CreateEndDeviceControls></soapenv:Body>

</soapenv:Envelope>
```

IEC 1806/13

Figure 38 – Exemple d'enveloppe SOAP à caractère fortement typé

6.7 Traitement de la demande

Un message de demande est envoyé d'un client à un service pour initier une requête ou une transaction là où un message de réponse est généralement attendu. La séquence de base de traitement de la demande est la suivante:

1) Le client construit un message de demande au moyen d'une enveloppe de message commune et en indiquant un verbe et un noun. Le noun identifie le type de charge utile
2) Le client envoie le message de demande à l'interface de service appropriée. Il peut pour cela utiliser des technologies de transport comme JMS ou des services Web. La mise en œuvre d'ESB peut de manière transparente utiliser des intermédiaires comme des ''proxies'', des routeurs et des adaptateurs pour transmettre la demande à l'instance du service approprié.
3) Le serveur accepte le message.
4) Si le message de demande n'est pas valide (par exemple incomplet, XML mal constitué, etc.), un message de défaut peut être renvoyé au client et le traitement prend fin.
5) Le service regarde la combinaison de verbe et de noun, afin de déterminer si la réponse peut être traitée (ou s'il convient de le faire); dans le cas contraire, un message de réponse d'erreur est envoyé au client et le traitement prend fin. Cette étape peut aussi prendre en compte des contrôles d'authentification et d'autorisation.
6) Le service effectue le traitement désiré, décomposant la charge utile si nécessaire. Le service peut analyser la charge utile à l'aide d'un schéma XML adapté (comme défini par le type de charge décrit par le nom du message), d'expressions XPath, de transformations XSL ou d'autres mécanismes. Le service peut également prendre en compte les paramètres fournis dans l'en-tête du message et les paquets de demandes en vue d'être traités.
7) Un message de réponse est construit dans lequel une charge utile prend la forme identifiée par le noun si nécessaire. Cela est communément le cas en réponse à une demande "get". Il convient d'utiliser l'élément de réponse du message pour acheminer un résultat ''OK'' ou un code d'erreur le cas échéant.
8) Le message de réponse est renvoyé au client. Il peut pour ce faire utiliser les services Web, JMS ou d'autres technologies de transport comme prévu avec le client.
9) Le client traite le message de réponse, décomposant la charge utile si nécessaire. Il convient que le client examine l'élément de réponse du message pour voir si la demande a réussi (si elle a renvoyé ''OK'') ou si des erreurs se sont produites.
10) Le traitement est terminé

Il est important de noter que différents scénarios de panne peuvent se produire pendant l'exécution des étapes 2 à 8, où il convient de vérifier que le client est en mesure de traiter une panne ou une temporisation pendant qu'il attend la réponse à une demande.

6.8 Traitement de l'événement

Un message d'évènement est un message publié par un service (ou plus généralement tout éditeur d'évènement) à destination de nombreux écouteurs potentiels. Les évènements peuvent également être appelés "notifications". Les écouteurs d'évènements sont des consommateurs abonnés à un ou plusieurs thèmes JMS potentiellement intéressants. Dans le cas des services Web, il y aura un intermédiaire chargé d'envoyer les événements aux abonnés qui écoutent les thèmes par le biais des services Web.

La séquence de base de traitement de l'évènement est la suivante:

1) Un service construit un message d'évènement au moyen d'une enveloppe de message commune et en indiquant un verbe et un noun. Le noun identifie le type de charge utile, bien que tous les messages d'évènement ne disposent pas de charge utile.
2) Le service envoie le message d'évènement au thème JMS approprié. La mise en œuvre d'ESB peut également utiliser de manière transparente des intermédiaires comme des

routeurs et des adaptateurs pour transmettre l'évènement aux écouteurs d'évènement appropriés.

3) L'écouteur accepte le message.

4) Si le message d'évènement est invalide (par exemple incomplet, XML mal formulé...), le traitement prend fin (typiquement après l'enregistrement d'une erreur).

5) L'écouteur regarde la combinaison de verbe et de noun, afin de déterminer si l'événement peut être traité (ou s'il convient de le faire); si ce n'est pas le cas, le traitement prend fin.

6) L'écouteur effectue le traitement désiré, décomposant la charge utile si nécessaire. Le service peut analyser la charge utile à l'aide d'un schéma XML adapté (comme défini par le type de charge décrit par le noun du message), d'expressions XPath, de transformations XSL ou d'autres mécanismes.

Lorsque des messages de demande transactionnels (voir 4.5) sont traités sur le bus qui utilise le verbe 'create', 'change', 'close', 'cancel' ou 'delete', il convient de publier un événement correspondant avec ce verbe au passé ('created', 'changed', 'closed', 'canceled', 'deleted' ou 'executed'). Il convient de l'émettre après la réussite de la transaction par le service qui a traité la demande ou par un composant du bus. Dans ces cas, la charge utile est identique à celle du message de demande correspondant utilisé pour invoquer la transaction.

Les garanties de livraison sont une conséquence de la définition de la file d'attente ou du thème JMS spécifiques, ainsi que les moyens par lesquels un écouteur souscrit (par exemple abonnement durable).

6.9 Corrélation de messages

Un aspect important des modèles de messagerie asynchrones concerne la nécessité de pouvoir corréler une demande et une réponse. Le Header.CorrelationID est primordial pour l'association de demandes et de réponses asynchrones. Il convient d'appliquer les règles suivantes aux messages pour autoriser les "liaisons" nécessaires.

- Lorsqu'un client fournit un CorrelationID dans une demande, il convient de définir la valeur sur un UUID hexadécimal (comme '`D921A053-80C1-4DB6-960E-2603127B7B92`') ou sur un numéro de séquence généré (par exemple, 100023, 100024, 100025, ...) effectivement unique pour le client qui a émis la demande.
- Si un message de demande inclut un CorrelationID, il convient que le message de réponse renvoie le même CorrelationID.
- Si aucun CorrelationID n'est fourni dans un message de demande mais si un MessageID est fourni, il convient que le message de réponse mette le CorrelationID à la valeur de MessageID fournie dans la demande. Il convient de définir la valeur de MessageID sur un UUID ou sur un numéro de séquence généré.
- Si le message de demande ne contient ni MessageID ni CorrelationID, il n'est pas possible de corréler une réponse asynchrone à une demande particulière. De ce fait, le CorrelationID ne peut pas être défini dans le message de réponse de manière à identifier le lien à une demande particulière.
- Si un service génère des événements comme une conséquence directe à une demande spécifique, il convient de définir CorrelationID sur le message d'événement correspondant selon les règles précédentes si possible, même si cela peut ne pas être toujours possible et il n'est pas donc une exigence. Ceci peut engendrer une corrélation entre l'événement et la transaction qui l'a causé.

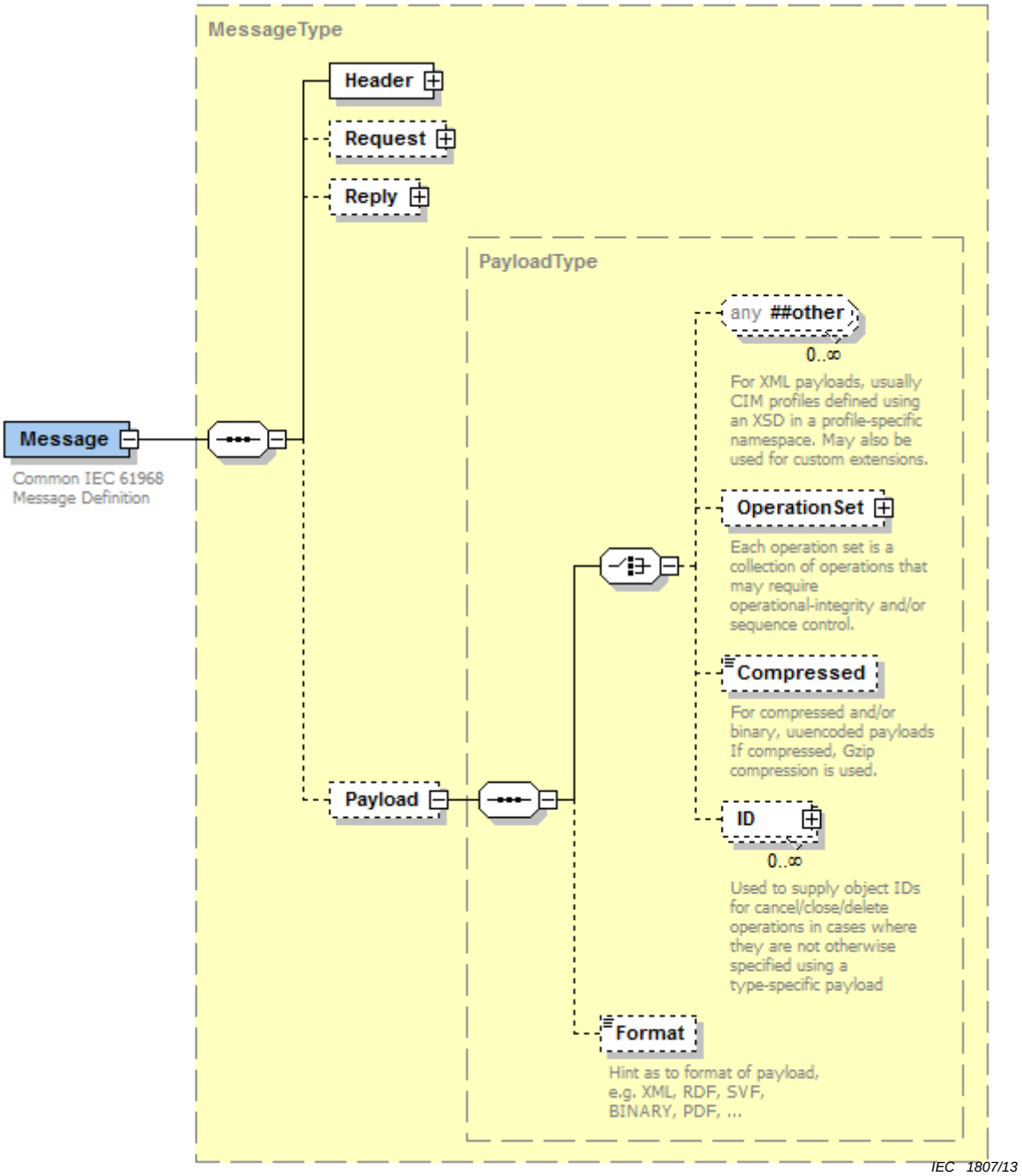
Se reporter à l'exemple d'utilisation de la discussion et de CorrelationID fourni en 5.2.5.

6.10 Traitement de transactions complexes au moyen d'OperationSet

6.10.1 Généralités

Le but du Paragraphe 6.10 est de décrire l'utilisation de l'élément OperationSet fournit par Message.xsd. Cela permet la prise en charge des transactions. L'enveloppe du message

Message.xsd a été étendue pour intégrer une construction OperationSet à la fois dans les parties charge utile et réponse de Message.xsd. L'élément OperationSet est visualisé au travers de la Figure 39.



Légende

Anglais	Français
Common IEC 61968 Message Definition	Définition de message CEI 61968 commune
For XML payloads, usually CIM profiles defined using an XSD in a profile-specific namespace. May also be used for custom extensions.	Pour les charges utiles XML, généralement des profils CIM définis à l'aide d'un XSD dans un espace de nom spécifique au profil. Peut également être utilisé pour les extensions personnalisées.
Each operation set is a collection of operations that may require operational-integrity and/or sequence control.	Chaque OperationSet (jeu d'opération) est un ensemble d'opérations qui peut nécessiter une intégrité opérationnelle et/ou un contrôle de séquence.

Anglais	Français
For compressed and/or binary, uuencoded payloads. If compressed, Gzip compression is used.	Pour charges utiles compressées et/ou binaires sous forme uuencode. Le cas échéant, la compression Gzip est utilisée.
Used to supply object IDs for cancel/close/delete operations in cases where they are not otherwise specified using a type-specific payload	Utilisé pour fournir des ID d'objet pour les opérations d'annulation/de fermeture/de suppression lorsqu'ils ne sont pas spécifiés d'une autre manière avec une charge utile spécifique pour un type.
Hint as to format of payload, e.g. XML, RDF, SVF, BINARY, PDF, ...	Astuce sur le format de la charge utile, par exemple XML, RDF, SVF, BINARY, PDF, , etc.

Figure 39 – Elément OperationSet du message

L'utilisation d'OperationSet peut être nécessaire dans deux circonstances:

a) Lors de la modification de la configuration d'un objet CIM impliquant la suppression d'un ou plusieurs attributs ou une ou plusieurs instances des objets CIM associés. Le retrait d'une configuration de registre d'un compteur en est un exemple.

b) Lors de la réalisation d'une ou plusieurs actions liées qui doivent être traitées dans un ordre particulier et/ou avec une intégrité transactionnelle générale (toutes les actions doivent réussir ou toutes doivent être différées).

Un message qui utilise la construction OperationSet a toujours dans son En-tête le verbe "execute" ou "executed" et le noun "OperationSet". Un OperationSet contient à son tour un ou plusieurs éléments Operation; chaque OperationSet.Operation a un OperationID qui remplace le CorrelationID général du message afin de fournir une capacité à granularité fine à corréler le contenu d'un ou plusieurs messages de réponse avec les opérations individuelles d'un OperationSet. Les éléments Operation individuels d'un OperationSet disposent de verbes et de nouns du niveau de l'OperationSet. Les verbes admissibles sont ‘’create’’, ‘’created’’, ‘’change’’, ‘’changed’’, ‘’delete’’ et ‘’deleted’’.

Pour prendre en charge la circonstance a) ci-dessus, chaque Operation d'un OperationSet comprend également un booléen elementOperation. Ce booléen doit être défini sur "true" lorsque le verbe d'Operation est "delete" ou "deleted"; l'intention est de supprimer les attributs individuels ou les instances individuelles des classes CIM associées provenant de l'objet spécifié par le noun d'OperationSet (par opposition à la suppression de l'intégralité de l'objet CIM spécifié par le noun d'Operation. S'il est négligé, elementOperation est considéré "false". Il est à souligner que dans ce cas, l'utilisation du verbe d'Operation "delete" ou "deleted" combiné à un booléen elementOperation défini sur "true" modifie effectivement (et ne supprime pas) l'objet CIM spécifié par le noun d'Operation.

Pour prendre en charge la circonstance b) ci-dessus, chaque OperationSet peut disposer d'un booléen enforceMsgSequence, d'un booléen enforceTransactionalIntegirty ou des deux. Le booléen enforceMsgSequence est à définir sur "true" lorsque les Operations de l'OperationSet doivent être exécutées dans l'ordre croissant de leur operationID. Le booléen enforceTransactionalIntegrity est à définir sur "true" si toutes les Operations de l'OperationSet doivent réussir. Dans ce cas, si toutes les Operations ne sont pas couronnées de succès, elles doivent toutes être différées. Si l'un ou l'autre de ces booléens est négligé, il est considéré "false".

Lors de la modification de la configuration d'un objet CIM qui utilise l'un des verbes "change", "changed", "delete" ou "deleted", seul l'ID de l'objet et des informations en cours de modification sont à inclure. Cela est vrai qu'un OperationSet soit utilisé ou non. C'est pour cette raison que presque tous les éléments au sein des Profils de gestion des données maîtres CEI 61968-9 sont optionnels dans les profils.

Il est recommandé de n'utiliser qu'un seul OperationSet, car plusieurs OperationSets placeraient un fardeau plus important sur les consommateurs et pourraient impliquer des messages inutilement imposants.

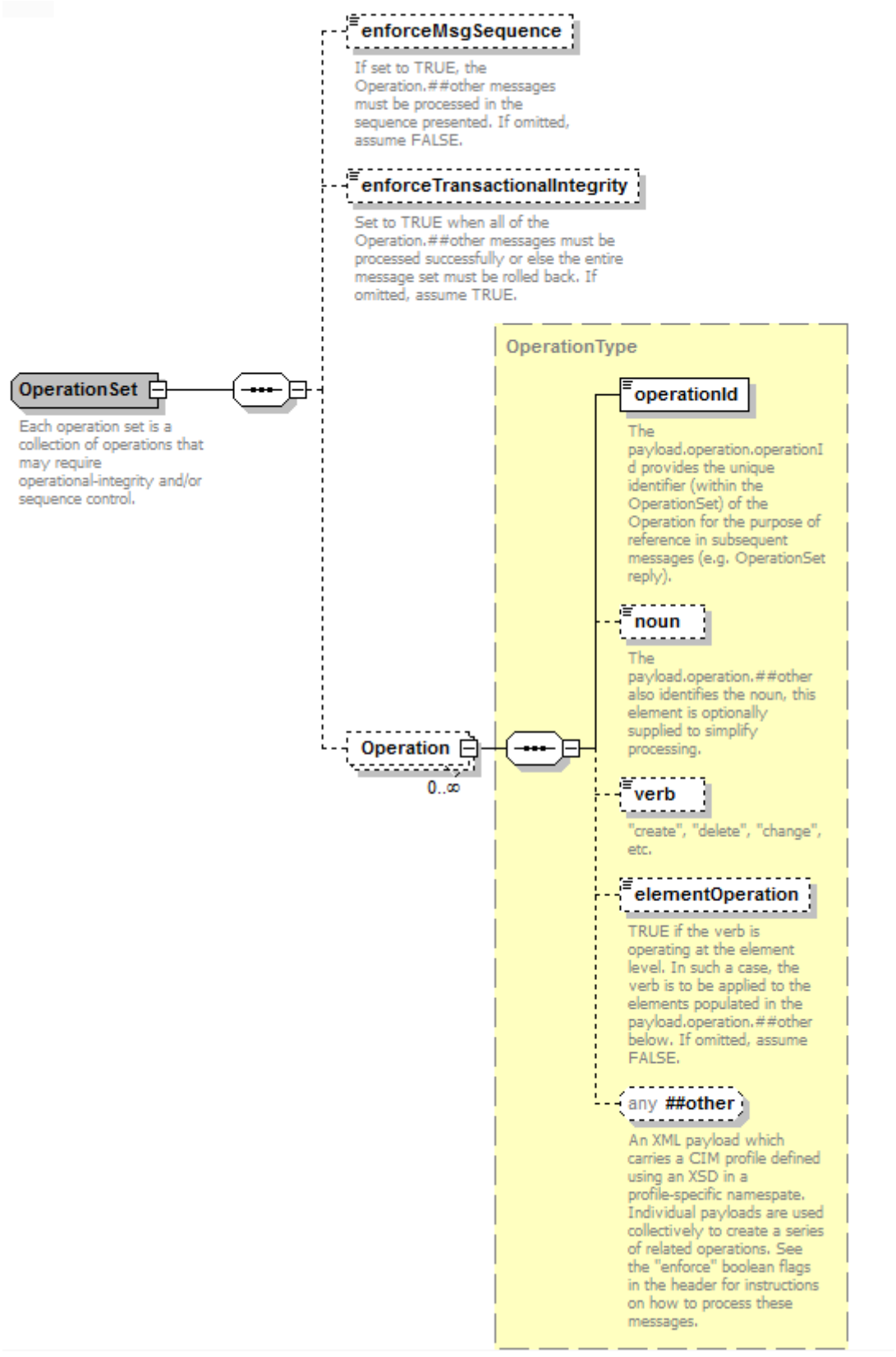
Il convient de noter que, même si cela permet d'acheminer des transactions avec des structures de données basées sur des schémas XML, il est également possible techniquement d'utiliser la norme CEI 61970-552[1] pour des transactions basées sur RDF.

6.10.2 Elément OperationSet

La Figure 40 décrit l'élément OperationSet plus en détail. Un OperationSet peut:

- Exiger que chaque opération soit exécutée dans un ordre précis en donnant à l'indicateur enforceMsgSequence la valeur "true"
- Exiger que l'intégrité transactionnelle soit maintenue (tout ou rien) en donnant à l'indicateur enforceTransactionalIntegrity la valeur "true"
- Avoir une ou plusieurs Operations, où chaque operation a un noun, un verbe et une charge utile.

1 A publier.



IEC 1808/13

Légende

Anglais	Français
Each operation set is a collection of operations that may require operational-integrity and/or sequence control.	Chaque jeu d'opération est un ensemble d'opérations qui peut nécessiter une intégrité opérationnelle et/ou un contrôle de séquence.
If set to TRUE, the Operation.##other messages	Si TRUE, les messages Operation.##other

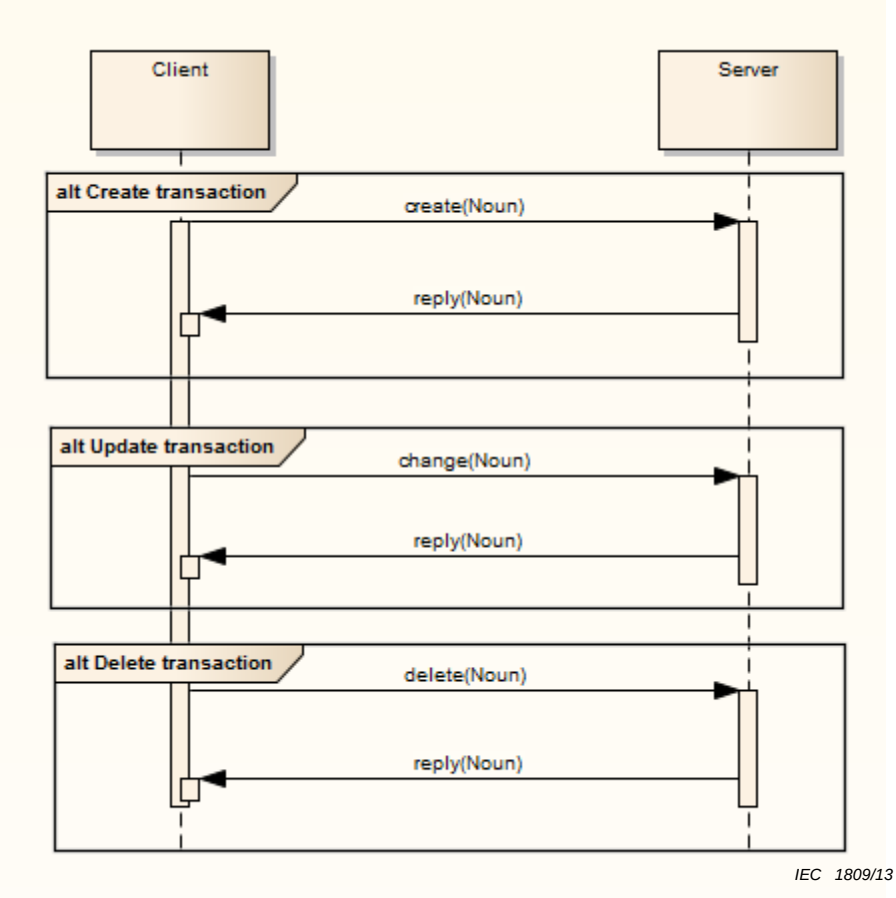
Anglais	Français
must be processed in the sequence presented. If omitted, assume FALSE.	doivent être traités dans l'ordre présenté. S'il est omis, supposer FALSE.
Set to TRUE when all of the Operation.##other messages must be processed successfully or else the entire set must be rolled back. If omitted, assume TRUE.	Définir sur TRUE lorsque tous les messages Operation.##other doivent être traités avec succès ou si tout le jeu doit être remis à zéro. S'il est omis, supposer TRUE.
The paload.operation.operationId provides the unique identifier (within the OperationSet) of the Operation for the purpose of reference in subsequent messages (e.g. OperationSet reply).	payload.operation.operationId fournit l'identifiant unique (dans OperationSet) de l'opération à des fins de référence dans les messages suivants (par exemple, réponse OperationSet reply).
The payload.operation.##other also identifies the noun, this element is optionally supplied to simplify processing.	payload.operation.##other identifie également le nom, cet élément est fourni de façon facultative pour simplifier le traitement.
"create", "delete", "change", etc.	"créer", "supprimer", "modifier", etc.
TRUE if the verb is operating at the element level. In such a case, the verb is to be applied to the elements populated in the payload.operation.##other below. If omitted, assume FALSE.	TRUE si le verbe fonctionne au niveau de l'élément. Dans ce cas, le verbe doit être appliqué aux éléments intégrés dans payload.operation.##other ci-dessous. S'il est omis, supposer FALSE.
An XML payload which carried a CIM profile defined using an XSD in a profile-specific namespace. Individual payloads are used collectively to create a series of related operations. See the "enforce" boolean flags in the header for instructions on how to process these messages.	Une charge utile XML qui porte un profil CIM défini à l'aide d'un XSD dans un espace de nom spécifique au profil. Les charges utiles individuelles sont utilisées collectivement pour créer une série d'opérations associées. Voir les indicateurs booléens "enforce" dans l'en-tête pour des instructions sur la façon de traiter ces messages.

Figure 40 – Détails de OperationSet

Au sein d'un élément Operation, le noun identifie le type de chaque élément. La valeur de elementOperation détermine si la transaction agit sur l'objet ou les éléments compris dans l'objet. Les exemples fournis décrivent l'utilisation de façon plus précise.

6.10.3 Modèles (patterns)

Toute transaction donnée peut être exécutée soit grâce à un modèle de message demande-réponse (messages stéréotype de Demande et stéréotype de Réponse) ou un modèle de message d'évènement publié (messages stéréotype d'Evénement). Les quatre diagrammes comportant un exemple de séquence de la Figure 41 illustrent les variations possibles.

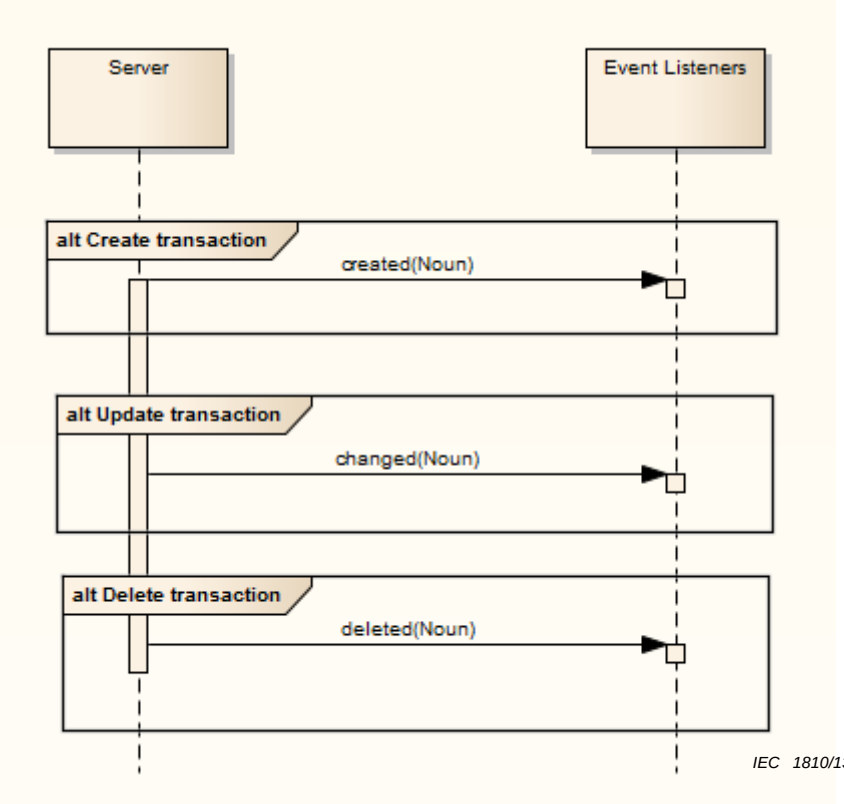


Légende

Anglais	Français
Server	Serveur
alt Create transaction	arrêter la transaction de création
alt Update transaction	arrêter la transaction de mise à jour
alt Delete transaction	arrêter la transaction de suppression

Figure 41 – Demande/réponse transactionnelle (non OperationSet)

Ce modèle de demande / réponse peut être utilisé pour les transactions. Les verbes admissibles sont "create", "change" et "delete". En fonction du scénario, il peut y avoir des réponses multiples à un message "create", "change" ou "delete" donné. Par exemple, un seul message ‘’create’’ peut être envoyé pour créer plusieurs mesures. Dans ce cas, le système répondant peut envoyer un message de réponse unique à tous les compteurs ou des messages de réponse multiples avec les données de réponse d'un ou plusieurs compteurs dans chaque message.

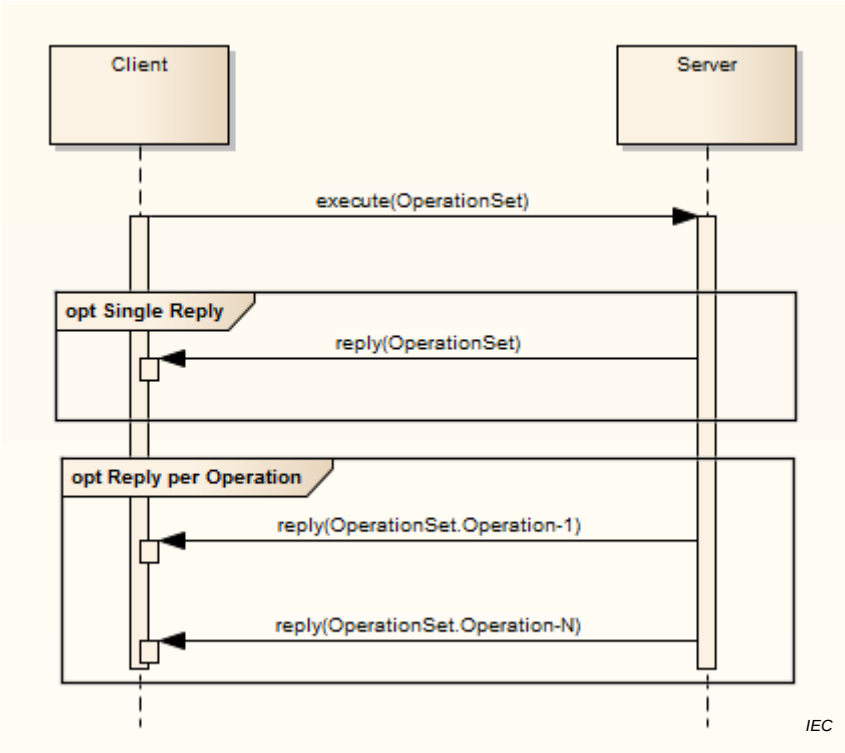


Légende

Anglais	Français
Event Listeners	Écouteurs d'événements
alt Create transaction	arrêter la transaction de création
alt Update transaction	arrêter la transaction de mise à jour
alt Delete transaction	arrêter la transaction de suppression

Figure 42 – Evènements publiés (non OperationSet)

Le modèle d'évènement publié peut aussi être utilisé pour des échanges, comme indiqué par les diagrammes de séquence de la Figure 42. Les verbes admissibles sont "created", "changed" et "deleted". Grâce à ce modèle, un système d'entreprise peut notifier un ou plusieurs autres systèmes d'évènements d'entreprise sans avoir besoin d'acquittement ou de confirmation que le traitement a été couronné de succès.

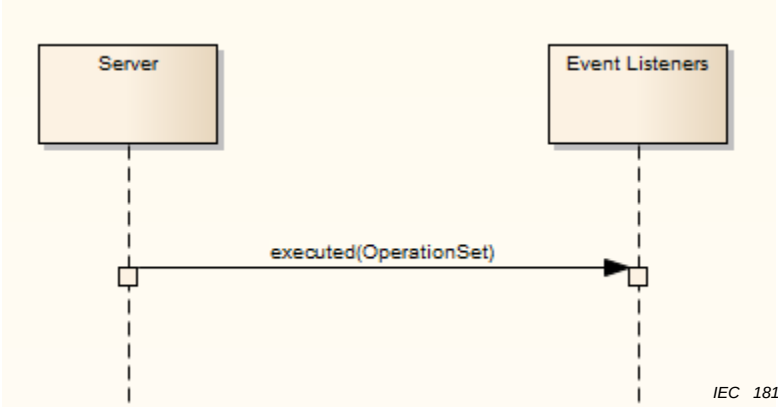


Légende

Anglais	Français
Server	Serveur
opt Single Reply	opt Réponse unique
opt Reply per Operation	opt Réponse par opération

Figure 43 – Demande/réponse transactionnelle (OperationSet)

Ce modèle de demande / réponse peut être utilisé pour tout échange faisant intervenir un OperationSet, comme indiqué par le diagramme de séquence de la Figure 43. Le verbe de l'En-tête du message est toujours "execute". Les opérations individuelles de l'OperationSet peuvent avoir des verbes et des nouns cohérents avec la transaction de demande / réponse du Modèle 1. En fonction du scénario, il peut y avoir des réponses multiples à une transaction execute / OperationSet donnée. Par exemple, un message de réponse unique peut être envoyé pour tout l'OperationSet ou plusieurs messages de réponse peuvent être envoyés, chacun comprenant les données de réponse d'une ou de plusieurs opérations dans chaque message. L'élément OperationID de chaque Operation du message de demande est fourni dans les messages de réponse. Cela est utilisé conjointement au CorrelationID général contenu dans les en-têtes de messages afin de corréler les réponses avec leurs demandes correspondantes.



Légende

Anglais	Français
Server	Serveur
Event Listeners	Ecouteurs d'événements

Figure 44 – Evènements publiés (OperationSet)

Le modèle d'évènement publié peut aussi être utilisé pour tout échange faisant intervenir un OperationSet, comme indiqué par le diagramme de séquence de la Figure 44. Le verbe de l'En-tête du message est toujours "executed". Les Operations individuelles de l'OperationSet peuvent avoir des verbes et des nouns cohérents avec la transaction d'évènement publié du Modèle 2. Grâce à ce modèle, un système d'entreprise peut notifier un ou plusieurs autres systèmes d'évènements d'entreprise sans avoir besoin d'acquittement ou de confirmation que le traitement a été couronné de succès.

6.10.4 Exemple de OperationSet

Le XML suivant fournit un exemple de transaction complexe qui utilise l'élément Payload.OperationSet

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestMessage
    xmlns = "http://iec.ch/TC57/2011/schema/message"
    xmlns:m = "http://iec.ch/TC57/2011/MeterConfig#"
    xmlns:up = "http://iec.ch/TC57/2011/UsagePointConfig#"
    xmlns:mdlc = "http://iec.ch/TC57/2011/MasterDataLinkageConfig#"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation = "http://iec.ch/TC57/2011/schema/message Message.xsd">
<Header>
<Verb>execute</Verb>
<Noun>OperationSet</Noun>
<Revision>2.0</Revision>
<Timestamp>2012-12-20T09:30:47Z</Timestamp>
<Source>CIS</Source>
<AckRequired>true</AckRequired>
<MessageID>D921A053-80C1-4DB6-960E-2603127B7B92</MessageID>
<CorrelationID>D921A053-80C1-4DB6-960E-2603127B7B92</CorrelationID>
</Header>
<Payload>
<OperationSet>
<enforceMsgSequence>true</enforceMsgSequence>
<enforceTransactionalIntegrity>true</enforceTransactionalIntegrity>
<Operation>
<operationId>1</operationId>
<noun>MeterConfig</noun>
<verb>create</verb>
<mdlc:MeterConfig>
<mdlc:Meter>
<mdlc:formNumber>2S</mdlc:formNumber>
<mdlc:ConfigurationEvents>
<mdlc:createdDateTime>2012-12-20T09:30:47Z</mdlc:createdDateTime>
<mdlc:effectiveDateTime>2012-12-21T00:00:00Z</mdlc:effectiveDateTime>
<mdlc:Names>
```

```
<mdlc:name>C34531</mdlc:name>
<mdlc:NameType>
<mdlc:name>MeterBadgeNumber</mdlc:name>
<mdlc:NameTypeAuthority>
<mdlc:name>UtilityXYZ</mdlc:name>
</mdlc:NameTypeAuthority>
</mdlc:NameType>
</mdlc:Names>
</mdlc:ConfigurationEvents>
</mdlc:Meter>
</mdlc:MeterConfig>
</Operation>
<Operation>
<operationId>2</operationId>
<noun>UsagePointConfig</noun>
<verb>create</verb>
<up:UsagePointConfig>
<up:UsagePoint>
<up:amiBillingReady>amiCapable</up:amiBillingReady>
<up:connectionState>connected</up:connectionState>
<up:isSdp>true</up:isSdp>
<up:isVirtual>false</up:isVirtual>
<up:phaseCode>B</up:phaseCode>
<up:readCycle>ReadCycleJ</up:readCycle>
<up:ConfigurationEvents>
<up:createdDateTime>2012-12-20T09:30:47Z</up:createdDateTime>
<up:effectiveDateTime>2012-12-21T00:00:00Z</up:effectiveDateTime>
</up:ConfigurationEvents>
<up:Names>
<up:name>UP43639</up:name>
<up:NameType>
<up:name>ServiceDeliveryPointID</up:name>
<up:NameTypeAuthority>
<up:name>UtilityXYZ</up:name>
</up:NameTypeAuthority>
</up:NameType>
</up:Names>
</up:UsagePoint>
</up:UsagePointConfig>
</Operation>
<Operation>
<operationId>3</operationId>
<noun>MasterDataLinkageConfig</noun>
<verb>create</verb>
<mdlc:MasterDataLinkageConfig>
<mdlc:ConfigurationEvent>
<mdlc:createdDateTime>2012-12-17T09:30:47Z</mdlc:createdDateTime>
<mdlc:effectiveDateTime>2012-12-21T00:00:00Z</mdlc:effectiveDateTime>
</mdlc:ConfigurationEvent>
<mdlc:Meter>
<mdlc:Names>
<mdlc:name>C34531</mdlc:name>
<mdlc:NameType>
<mdlc:name>MeterBadgeNumber</mdlc:name>
<mdlc:NameTypeAuthority>
<mdlc:name>UtilityXYZ</mdlc:name>
</mdlc:NameTypeAuthority>
</mdlc:NameType>
</mdlc:Names>
</mdlc:Meter>
<mdlc:UsagePoint>
<mdlc:Names>
<mdlc:name>UP43639</mdlc:name>
<mdlc:NameType>
<mdlc:name>ServiceDeliveryPointID</mdlc:name>
<mdlc:NameTypeAuthority>
<mdlc:name>UtilityXYZ</mdlc:name>
</mdlc:NameTypeAuthority>
</mdlc:NameType>
</mdlc:Names>
</mdlc:UsagePoint>
</mdlc:MasterDataLinkageConfig>
</Operation>
</OperationSet>
</Payload>
</RequestMessage>
```

L'exemple de transaction complexe comporte trois opérations qui permettent:

- d'exécuter 'create MeterConfig'
- d'exécuter 'create UsagePointConfig'
- d'exécuter 'create MasterDataLinkageConfig

L'XML identifie les espaces de nom pour MeterConfig, UsagePointConfig et MasterDataLinkageConfig tels qu'ils sont définis dans la norme CEI 61968-9.

6.11 Représentation de l'heure

La norme ISO 8601 est utilisée pour définir les représentations des valeurs temporelles transmises dans les interfaces. Cela permet d'éviter les problèmes relatifs aux fuseaux horaires et aux changements d'heure.

Il convient que les marqueurs temporels des messages publiés par un processus de serveur utilisent une heure prédominante, selon le format suivant: *2007-03-27T14:00:00-05:00* (quand l'heure passe de CDT à CST, le *-05:00* devient *-06:00*).

Les marqueurs temporels des messages envoyés par un processus client peuvent utiliser n'importe quel marqueur temporel conforme à la norme ISO 8601.

Il est extrêmement important de noter que l'utilisation des marqueurs temporels conformes ISO 8601 dans les définitions de message pour les interfaces externes définies par le présent document ne limite en aucun cas les autres représentations de l'heure, qui peuvent inclure:

- User interfaces (Interfaces utilisateur), où l'heure locale ou les heures d'ouverture de la bourse peuvent être utilisées au choix
- Reports (rapports), où les rapports sont générés à l'aide d'une heure locale appropriée
- Internal integration (Intégration interne), où une application peut exiger en interne une autre structure temporelle

6.12 Autres conventions et meilleures pratiques

Ci-après se trouvent d'autres conventions qui doivent être suivies par cette spécification:

- Dans les définitions XML, il convient de qualifier les balises par espace de nom. Par exemple, dans la balise XML '<tag>', il convient d'ajouter un préfixe contenant la référence d'espace de nom, par exemple '<ns:tag>'. Cela aide à éliminer toute ambiguïté. *(A noter que de nombreux exemples contenus dans le présent document ne sont pas qualifiés par espace de nom pour des raisons de brièveté et pour faciliter la lisibilité.)*
- Il convient d'exprimer les grandeurs avec des unités SI, si possible.

6.13 Interopérabilité technique

Les normes ouvertes constituent un élément primordial de la stratégie visant à atteindre l'interopérabilité technique. Parmi ces normes présentant un intérêt particulier, on trouve:

- Les normes W3C
- Les normes OASIS WS-*
- La norme CEI Modèle d'Information Commun et les normes associées (par exemple CEI 61970-301 et CEI 61968-11)
- Java Message Service

Il est très important que la mise en œuvre des interfaces de Service Web ne dépende d'aucun produit tiers spécifique. Une autre exigence primordiale est que la mise en œuvre de clients de services Web doit être possible aussi bien avec des outils de développement Java que .Net.

6.14 Contrats sur les niveaux de service

Différentes catégories de services ont différents contrats sur les niveaux de service (SLA). Les SLA de certains services sont directement affectés par la variabilité de la quantité de données qui peut être transférée.

Les périodes de temps de réponse spécifiées pour chaque interface couverte par un SLA varient en général dans une certaine mesure en fonction de facteurs tels que la charge du réseau et du système. Il convient donc d'indiquer chaque SLA de telle manière que chaque SLA sera utilisé X % du temps, où X est souvent compris entre 90 % et 100 %.

Les SLA peuvent être utilisés pour identifier les périodes de temporisation pour le traitement des demandes.

6.15 Audit, surveillance et gestion

L'ESB dispose en général de capacités d'audit, de surveillance et de gestion. Il peut également y avoir des services communs utilisés pour la mise en œuvre de composants d'intégration au sein de l'ESB. Les exemples de fonctionnalités incluent souvent les fonctionnalités suivantes:

- Consignation
- Génération d'identificateurs uniques
- Génération de signatures
- Authentification et autorisation de l'utilisateur
- Identification des instances de services en ligne (où il peut y avoir plusieurs instances)

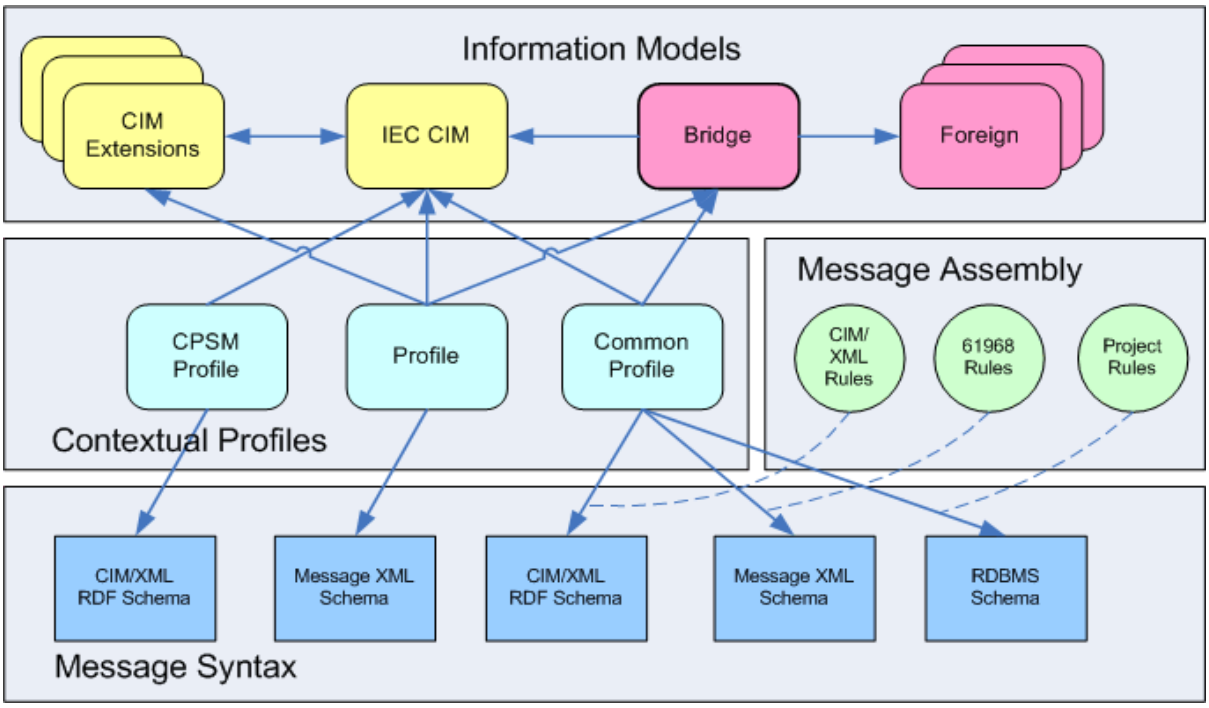
7 Spécifications de la charge utile

Chaque noun utilisé dans un message identifie un type de charge utile. Les types de charges utiles sont généralement dérivés du CIM CEI ou d'autres modèles sémantiques. Les types de charges utiles utilisés par les parties de la norme CEI 61968 sont toujours dérivés du CIM CEI et ont des artéfacts de conception (par exemple XSD) qui décrivent leur structure. Cas dans lesquels les XSD ne sont pas requis:

- Messages qui utilisent des charges utiles RDF telles qu'elles sont définies par les normes CEI 61968-13 et CEI 61970-452.
- Messages qui utilisent des charges utiles telles qu'elles sont définies dans la norme CEI 61970-453.
- Messages de réponse provenant de services qui génèrent XML de façon dynamique (comme dans le cas de jeux de résultats SQL XML).
- Charges utiles encodées et compressées non XML.
- Données binaires encodées (lorsque le formatage XML n'est pas efficace comme dans le cas de "données à grande vitesse")

Si un XSD n'est pas disponible pour décrire la charge utile, il incombe à l'expéditeur et au(x) destinataire(s) de s'entendre sur le formatage spécifique.

Le modèle d'informations logiques CIM est décrit comme un ensemble de paquets UML. Le diagramme de la Figure 45 décrit l'utilisation du CIM du point de vue de la modélisation et de la génération UML d'artéfacts de conception nécessaires aux outils d'intégration. Il illustre les relations entre les modèles d'information et les profils contextuels utilisés conjointement aux règles d'assemblage afin de déterminer les artéfacts de conception.



IEC 1813/13

Légende

Anglais	Français
Information Models	Modèles d'informations
CIM Extensions	Extensions CIM
IEC CIM	CIM CEI
Bridge	Pont
Foreign	Etranger
CPSM Profile	Profil CPSM
Profile	Profil
Common Profile	Profil commun
Message Assembly	Assemblage de messages
CIM/XML Rules	Règles CIM/XML
61968 Rules	Règles 61968
Project Rules	Règles de projet
Contextual Profiles	Profils contextuels
CIM/XML RDF Schema	Schéma CIM/XML RDF
Message XML Schema	Schéma de message XML
RDBMS Schema	Schéma RDBMS
Message Syntax	Syntaxe de message

Figure 45 – Modèles, profils et messages d'information

La Figure 46 montre un exemple de conception de profil contextuel dans CIMTool[2].

2 Disponible à l'addresse www.cimtool.org.

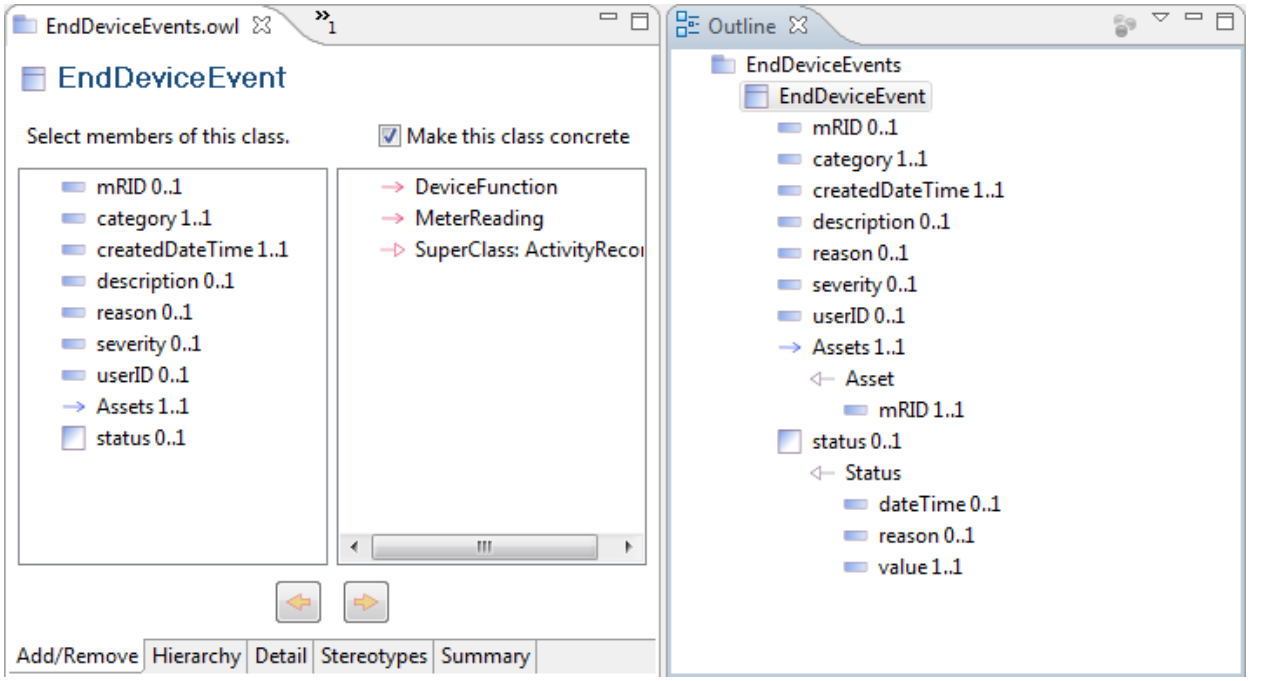


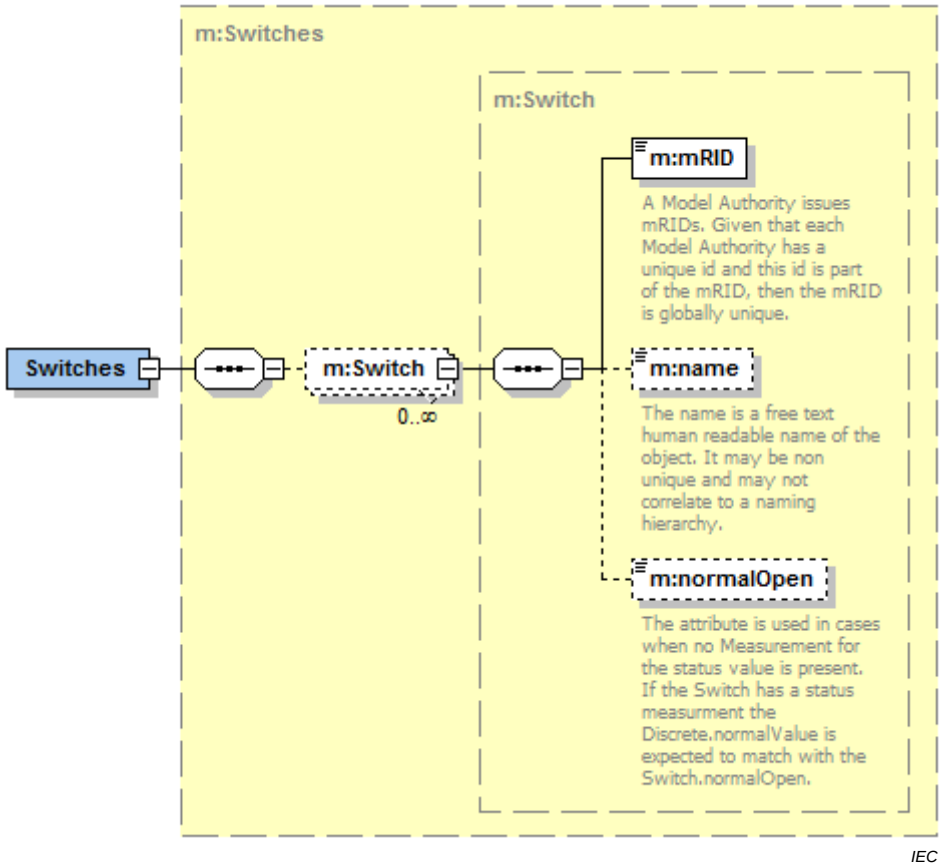
Figure 46 – Conception de profil contextuel dans CIMTool

Lors de la réalisation d'un profil comme artefact de conception sous la forme d'un Schéma XML, il est important de noter que de nombreuses options relatives à la réalisation peuvent affecter l'interopérabilité, comme:

- L'utilisation des espaces de noms
- Les références aux objets "par valeur" ou "par référence"
- Les définitions hiérarchiques ou à plat de modèles complexes
- Eléments requis / optionnels
- Enumérations

Un autre point important concerne le fait que le noun ne doit pas être un nom de classe CIM UML, sinon, il n'est pas possible d'avoir un XSD valide incluant cette classe UML dans le type de profil/charge utile.

Le diagramme de la Figure 47 décrit la structure d'un exemple de charge utile simple telle qu'elle est décrite par un Schéma XML qui peut être transmis dans le "any" de la charge utile.



Légende

Anglais	Français
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part or the mRID, then the mRID is globally unique.	Un Modeling Authority (c'est-à-dire une autorité de modélisation) émet des mRID. Sachant que chaque Modeling Authority a un identificateur unique et que cet identificateur fait partie du mRID, donc le mRID est unique au niveau global
The name is a free text human readable name of the object. It may be non unique and may not correlate to a naming hierarchy.	Le name (c'est-à-dire nom) est un nom en texte libre de l'objet lisible par l'homme. Il peut ne pas être unique et peut ne pas se corréler à une hiérarchie de dénomination.
The attribute is used in cases when no Measurement for the status value is present. If the Switch has a status measurement, the Discrete.normalValue is expected to match with the Switch.normalOpen.	L'attribut est utilisé dans des cas où aucune mesure pour la valeur de statut n'est présente. Si le Switch a une mesure de statut, alors Discrete.normalValue est censé correspondre à Switch.normalOpen.

Figure 47 – Exemple de schéma de charge utile du message

L'exemple de charge utile de la Figure 47 est décrit par la définition du Schéma XML fournie par la Figure 48. Les Schémas XML des charges utiles peuvent être générés de bien des façons. L'utilisation de CIMTool, par exemple, où le modèle CIM UML est utilisé comme modèle de domaine pour la définition de message. A noter que le schéma XML de la charge utile d'un message définit au minimum un élément de haut niveau.

Le point important est que le nom de l'élément de haut niveau doit être le même que le noun qui est utilisé dans l'en-tête du message. Dans le XSD suivant, la définition de charge utile peut être utilisée conjointement au nom "Switches".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:a="http://www.iec.ch/2008/Message#"
targetNamespace="http://iec.ch/TC57/2011/CIM-schema-cim12#"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns="http://iec.ch/TC57/2011/Message#" xmlns:m="http://iec.ch/TC57/2007/CIM-schema-
cim12#">
<xs:element name="Switches" type="m:Switches"/>
   <xs:complexType name="Switches">
      <xs:sequence>
         <xs:element name="Switch" type="m:Switch" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Switch">
      <xs:annotation>
         <xs:documentation>A generic device  designed to close, or open, or both, one
or more electric circuits.</xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="mRID" minOccurs="1" maxOccurs="1" type="xs:string">
            <xs:annotation>
               <xs:documentation>A Model Authority issues mRIDs. Given that each Model
Authority has a unique id and this id is part of the mRID, then the mRID is globally
unique.</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="name" minOccurs="0" maxOccurs="1" type="xs:string">
            <xs:annotation>
               <xs:documentation>The name is a free text human readable name of the
object. It may be non unique and may not correlate to a naming
hierarchy.</xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="normalOpen" minOccurs="0" maxOccurs="1" type="xs:boolean">
            <xs:annotation>
               <xs:documentation>The attribute is used in cases when no Measurement for
the status value is present. If the Switch has a status measurment the
Discrete.normalValue is expected to match with the
Switch.normalOpen.</xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

IEC 1816/13

Figure 48 – Exemple de schéma de charge utile XML

A partir du schéma XML de la Figure 48, un exemple de charge utile XML est fourni par la Figure 49 (telle qu'utilisé à titre d'exemple à l'Article 6):

```
<m:Switches xsi:schemaLocation="http://www.iec.ch/TC57/2011/CIM-schema-cim12#
Switches.xsd" xmlns:m="http://www.iec.ch/TC57/2011/CIM-schema-cim12#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <m:Switch>
      <m:mRID>35378383838</m:mRID>
      <m:name>SW1</m:name>
      <m:normalOpen>true</m:normalOpen>
   </m:Switch>
   <m:Switch>
      <m:mRID>363482488448</m:mRID>
      <m:name>SW2</m:name>
      <m:normalOpen>true</m:normalOpen>
   </m:Switch>
   <m:Switch>
      <m:mRID>894094949444</m:mRID>
      <m:name>SW3</m:name>
      <m:normalOpen>false</m:normalOpen>
   </m:Switch>
</m:Switches>
```

IEC 1817/13

Figure 49 – Exemple de message XML

Il convient de définir les formats de charge utile spécifiques selon une spécification d'interface à l'aide de schémas XML, tels que fournis par la norme CEI 61968-3 à 61968-9. Pour les mises en œuvre qui ne relèvent pas du domaine d'application de la norme CEI 61968, les charges utiles peuvent être définies comme souhaité. Dans la plupart des cas, le nom du message prend une forme simple comme "Switches", "BidSets", "TroubleTickets" ou "WorkOrders". Cependant, il est également possible de réutiliser un préfixe identifiant un contexte de message sous la forme suivante:

```
<context><noun>
```

Cela permet l'utilisation de stéréotypes de la définition du nom de base pour définir les restrictions supplémentaires appropriées au contexte du message. On trouve par exemple "GetMeterReadings" et "GetEndDeviceAssets".

8 Spécifications d'interface

8.1 Généralités

Le présent article a pour objet de décrire les définitions d'interface. Trois points de vue sont fournis ici:

- Ce qu'il faut à un utilisateur de l'interface pour compléter les informations fournies par un langage de définition spécifique ou des artefacts de conception
- Artefacts et détails de mise en œuvre des services Web
- Détails de mise en œuvre de JMS

8.2 Spécifications au niveau de l'application

Des interfaces spécifiques sont définies au moyen d'une séquence de combinaisons spécifiques de verbes et de nouns (types de charges utiles). Par exemple, un message de demande avec pour verbe et noun "getMeterReadings" entraîne l'envoi du message de réponse "reply MeterReadings". Etant donné la complexité potentielle et les options disponibles pour une intégration donnée, il convient de documenter de façon plus détaillée les interactions. Les spécifications au niveau de l'application basées sur la norme CEI 61968-100 nécessitent souvent des spécifications plus détaillées qui en général dépassent les capacités des schémas XML et des WSDL. Voici quelques exemples simples de la documentation pour des messages qui pourraient être fournis par une spécification au niveau de l'application.

Les messages relatifs à une demande utilisent les champs de message suivants:

Elément de message	Valeur
Header/Verb	get
Header/Noun	*Nom du type de charge utile*
Header/Source	*Demande d'initialisation de système ou d'application*
Header/UserID	*Facultatif: ID de l'utilisateur*
Request/?	*Facultatif: D'autres paramètres de demande peuvent être spécifiés si nécessaire*

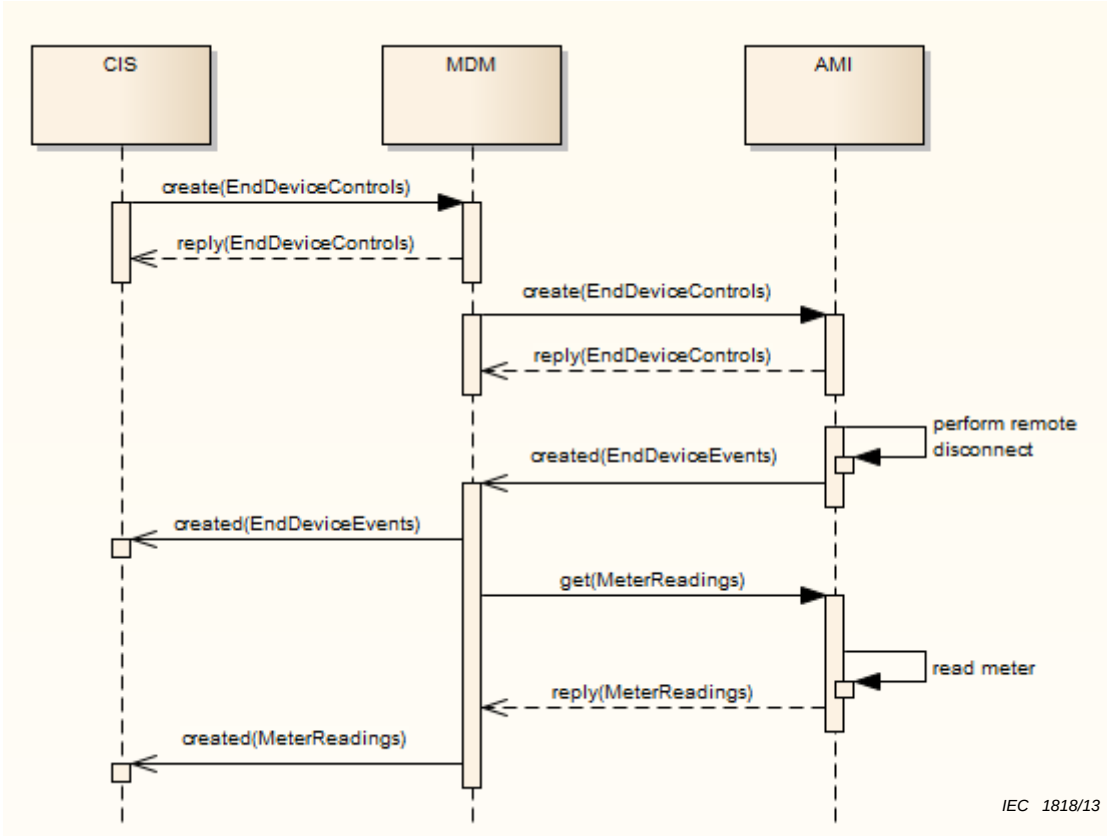
Les messages de réponse correspondants utilisent les champs de message suivants:

Elément de message	Valeur
Header/Verb	reply
Header/Noun	*Nom du type de charge utile définie*
Reply/result	*Code de réponse, succès=OK, succès partiel=PARTIAL, erreur=FAILED*
Reply/Error	*Facultatif: Peut être n'importe quel numéro de messages d'erreur*
Charge utile	*Type de charge utile définie*

Dans le cas où des charges utiles seraient autrement très imposantes (au-dessus d'un seuil comme 1 Mégaoctet, par exemple), les charges utiles seraient zippées, encodées en base64 et stockées dans la balise "Payload/Compressed».

Il convient de définir les nouns, les verbes (comme défini dans l'Annexe B) et les formats de charge utile spécifiques dans une spécification d'interface, comme indiqué dans la norme CEI 61968-3 à 61968-9. Les diagrammes de séquence de cas d'utilisation sont eux aussi souvent utilisés pour décrire les modèles d'échange d'informations en termes de verbes et de nouns.

Une description plus complète de l'utilisation d'une interface faisant potentiellement partie d'un processus commercial plus complexe serait décrite au moyen d'un diagramme de séquence. Le diagramme de séquence suivant fournit un exemple d'échange d'informations grâce à des verbes et des nouns. La convention du diagramme utilise "<verb>(<Noun>)" pour chaque flux entre les composants.



Légende

Anglais	Français
perform remote disconnect	effectuer un débranchement à distance
read meter	relever le compteur

Figure 50 – Exemple de processus commercial complexe

La Figure 50 est un diagramme de séquence qui décrit un processus commercial complexe combinant plusieurs types de messages et de modèles d'intégration différents.

8.3 Interfaces de services Web

8.3.1 Généralités

Le Paragraphe 8.3 décrit la définition des interfaces au moyen de services Web. Il décrit l'utilisation d'un style conditionné dans un document qui maximise l'interopérabilité. Il spécifie également les opérations qui sont des noms qui utilisent des combinaisons verb/noun avec des définitions de charges utiles spécifiques au modèle.

Il y a deux approches décrites dans la présente norme pour les services Web:

- Services Web fortement typés, dont la procédure pour la génération de WSDL est définie en détail dans l'Annexe C
- Services Web génériques, dont un WSDL générique est décrit dans l'Annexe D.

8.3.2 Structure WSDL

De manière générale, un WSDL (v1.1) est composé de deux parties comprenant les balises données en Figure 51.

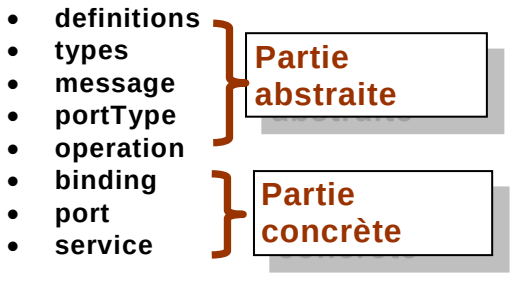


Figure 51 – Structure WSDL

Un exemple de document WSDL peut être trouvé dans le modèle WSDL de l'Annexe 4.

Pour les services Web fortement typés, les pratiques de conception de services Web sont résumées ci-dessous:

- Le lien SOAP standard est utilisé
- XSD en tant que type de données est en général importé plutôt que d'être intégré pour un meilleur contrôle de la version
- Le problème de signature de câble est évité en redéfinissant les noms des éléments comme CreateEndDeviceControl et ChangeEndDeviceControl au moyen d'un XSD complexType unique
- Le style Wrapped document (document conditionné) est utilisé
- Le nom de l'opération suit la convention de dénomination Verb + Noun qui permet d'éviter le routage basé sur le contenu.

8.3.3 Lien SOAP de style document

Le style de document qui utilise le corps de SOAP est le plus couramment utilisé en conception WSDL. Il peut utiliser complètement les avantages d'un schéma XML pour valider la charge utile. Ci-dessous se trouve un exemple de la section liante d'un WSDL de style de document pour l'échange d'informations EndDeviceControl:

```
<wsdl:binding name="EndDeviceControls_Binding" type="tns:EndDeviceControls_Port">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="CreateEndDeviceControls">
      <soap:operation
soapAction="http://iec.ch/TC57/2010/EndDeviceControls/CreateEndDeviceControls"
style="document"/>
      <wsdl:input name="CreateEndDeviceControlsRequest">
         <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CreateEndDeviceControlsResponse">
         <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="CreateEndDeviceControlsFault">
         <soap:fault name="CreateEndDeviceControlsFault" use="literal"/>
      </wsdl:fault>
   </wsdl:operation>
…
```

A noter que les styles <soap:binding> et <soap:operation> sont tous deux définis comme un "document" et sont soulignés en gris. De même, <soap:body> est utilisé pour les opérations d'entrée et de sortie. Un style "document" signifie qu'un document XML est inclus dans un message SOAP. Dans ce cas, il est directement placé dans le <soap:body>.

Si un nom wsdl:*operation* est le même que le nom d'entrée d'un élément, ce WSDL devient un WSDL de style conditionné dans un document. Le style conditionné dans un document provient de Microsoft pour imiter un style RPC. Dans un style RPC, une charge utile est toujours conditionnée par son nom d'opération.

Les caractéristiques du modèle conditionné (wrapped) sont énumérées ci-dessous:

- Le message d'entrée a une seule *partie*
- La *partie* est un élément
- L'élément a le même nom que *l'opération*
- Le type complexe de l'élément n'a pas d'attributs.

Tout WSDL ne satisfaisant pas au critère indiqué ci-dessus est un WSDL déconditionné. Il y a des pour et des contre les modèles conditionnés et déconditionnés, mais le style de document conditionné est recommandé dans ce profil pour les besoins de l'interopérabilité.

Voici un échantillon de WSDL sur le style de document conditionné:

```
  … …

<wsdl:message name="CreateEndDeviceControlsRequestMessage">
   <wsdl:part name="CreateEndDeviceControlsRequestMessage"
element="message:CreateEndDeviceControls"/>
</wsdl:message>
… …

<wsdl:portType name="EndDeviceControls_Port">

   <wsdl:operation name="CreateEndDeviceControls">
      <wsdl:input name="CreateEndDeviceControlsRequest"
message="tns:CreateEndDeviceControlsRequestMessage"/>
      <wsdl:output name="CreateEndDeviceControlsResponse"
message="tns:ResponseMessage"/>
      <wsdl:fault name="CreateEndDeviceControlsFault" message="tns:FaultMessage"/>
   </wsdl:operation>
   ……
   </wsdl:operation>
</wsdl:portType>
```

8.3.4 Services Web fortement typés

8.3.4.1 General

Le modèle d'intégration des services Web fortement typés sert à mettre en œuvre des interfaces basées sur la sémantique venant à l'appui d'une stratégie d'intégration SOA. Le modèle fortement typé présente les caractéristiques suivantes:

1) Il utilise les services Web basés sur SOAP pour lesquels les WSLD à granularité fine sont utilisés pour définir un contrat.
2) Il permet une validation plus poussée de la charge utile par la définition de messages d'opération à l'aide de charges utiles fortement typées.

8.3.4.2 Désignation des services et opérations

Dans la mise en œuvre des services Web fortement typés de la CEI 61968-100, les noms de service suivants sont utilisés pour refléter le rôle du service dans l'entreprise:

- Send

  Fournir (envoyer) des informations (objet commercial) pour la consommation publique (entreprise). Invoqué par le système d'enregistrement pour l'objet commercial et uniquement lorsque l'état de l'objet commercial a changé. Ce modèle est utilisé conjointement aux verbes created, changed, closed, canceled et deleted.

- Receive

  Consommer (recevoir) des informations (objet commercial) d'une source externe. Ce modèle est utilisé conjointement aux verbes created, changed, closed, canceled et deleted.

- Request

  Demander à une autre partie de réaliser un service spécifique. Ce modèle est utilisé conjointement aux verbes get, create, change, close, cancel et delete.

- Execute

  Exécuter un service fourni au public qui peut inclure une demande de changement d'état ou une demande de requête. Ce modèle est utilisé conjointement aux verbes create, change, close, cancel et delete.

- Reply

  Répondre avec le résultat de l'exécution d'un service (par la demande Execute service). Ce modèle est utilisé conjointement aux verbes created, changed, closed, canceled et deleted.

- Show

  Fournir (montrer) des informations (objet commercial) pour la consommation publique (entreprise), lorsque l'état de l'objet commercial est inchangé, par le biais du système d'enregistrement ou de tout autre système disposant d'une copie dudit objet commercial.

- Retrieve

  Demander des données spécifiques d'un objet commercial à fournir.

Une convention de désignation des services/opérations des services Web fortement typés qui utilisent les modèles de noms de services/opérations, les objets d'information et les verbes est décrite comme suit:

- Nom du service:

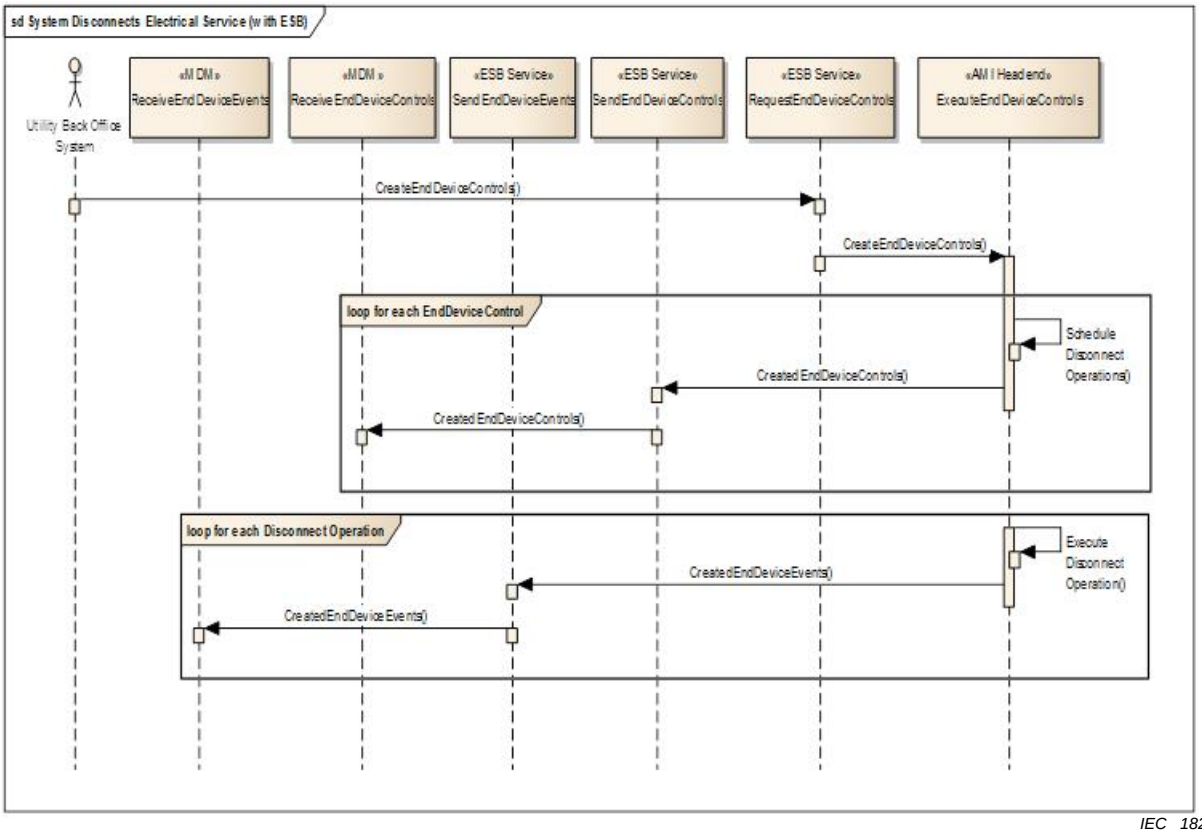
  Suivre <*Service pattern name*>+<*Information Object*> comme ExecuteEndDeviceControls

- Nom de l'opération:

  Suivre <*Verb*>+<*Information Object*> comme CreatedEndDeviceControls

8.3.4.3 Exemple d'intégration des services Web fortement typés

Il est présenté à la Figure 52 un exemple d'utilisation des services Web fortement typés pour mettre en œuvre la fonctionnalité de connexion/déconnexion entre un système MDM et un système AMI utilisant un ESB;



IEC 1820/13

Figure 52 – Exemple d'utilisation d'un service internet

1) Système de gestion des données du compteur (MDM, Meter Data Management system)
   a) Services mis en œuvre
      i) ReceiveEndDeviceControls: traite les messages stéréotype d'événement (notifications) de l'activité EndDeviceControls
      ii) ReceiveEndDeviceEvents: traite les messages stéréotype d'événement (notification) de l'activité EndDeviceEvent.
2) Bus de service d'entreprise (ESB, Enterprise Service Bus)
   a) Services mis en œuvre
      i) RequestEndDeviceControls: traite les demandes pour réaliser certaines activités avec EndDeviceControls. Il les route au(x) point(s) d'extrémité approprié(s) pour exécution.
      ii) ReplyEndDeviceControls: traite les réponses relatives aux messages stéréotype d'événement (notifications) de l'activité EndDeviceControls. Il les publie au(x) point(s) d'extrémité approprié(s).
      iii) SendEndDeviceEvents: traite les messages stéréotype d'événement (notifications) de l'activité EndDeviceEvent. Il les publie au(x) point(s) d'extrémité approprié(s).
3) Système AMI
   a) Services mis en œuvre
      i) ExecuteEndDeviceControls: agit sur (exécute) les jeux de EndDeviceControls.

8.4 JMS

8.4.1 Généralités

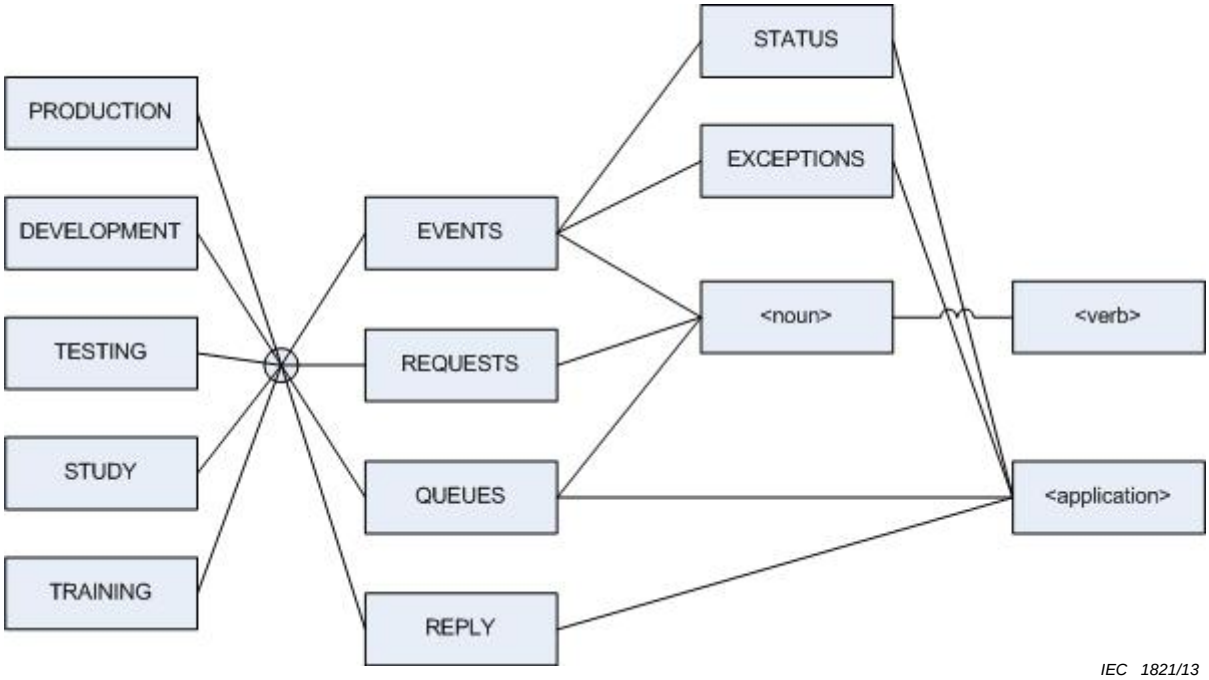
Le Paragraphe 8.4 décrit l'utilisation de JMS. Les messages envoyés au moyen de JMS utilisent des thèmes et/ou des files d'attente. Les différences et similarités entre les thèmes et les files d'attente sont résumées comme suit:

- Les thèmes sont utilisés lorsque la destination d'un message est potentiellement plus d'un processus
- Les files d'attente sont utilisées lorsque la destination d'un message est au maximum un processus
- S'ils sont pris en charge par le fournisseur JMS, les thèmes et files d'attente peuvent être organisés et nommés de manière hiérarchique
- Hormis pour l'utilisation d'un abonnement durable, un processus ne peut recevoir une copie d'un message publié à destination d'un thème que s'il est en cours d'exécution et dispose d'un abonnement actif.
- Un message publié à destination des files d'attente reste dans la file jusqu'à ce qu'il en soit retiré par un processus destinataire (constatant qu'il peut y avoir des options d'expiration et de persistance de file d'attente par la mise en œuvre de JMS spécifique)
- Une File d'attente (Queue) est en effet un cas particulier d'un abonnement durable à un thème où un seul processus consomme un message.

8.4.2 Désignation des thèmes et files d'attente

Lors de la désignation d'un thème ou d'une file d'attente, il convient que le niveau supérieur identifie le "contexte". Exemples de contextes: production, testing, developpement ou training. L'objectif est de s'assurer que les messages de différents contextes sont séparés de manière logique s'ils ne sont pas séparés physiquement. Par exemple, il est crucial qu'aucun message relatif à une activité training ne puisse être injecté dans une activité production. L'utilisation de séparations physique et logique est souhaitable.

La Figure 53 décrit une organisation possible des thèmes et files d'attente. Cette organisation n'est donnée qu'à titre illustratif et non normatif.



Légende

Anglais	Français
PRODUCTION	PRODUCTION
DEVELOPMENT	DEVELOPPEMENT
TESTING	ESSAIS
STUDY	ETUDE
TRAINING	FORMATION
EVENTS	EVENEMENTS
REQUESTS	DEMANDES
QUEUES	FILES D'ATTENTE
REPLY	REPONSE
STATUS	STATUT
EXCEPTIONS	EXCEPTIONS
<noun>	<nom>
<verb>	<verbe>
<application>	<application>

Figure 53 – Exemple d'organisation des thèmes et files d'attente

Cette organisation des thèmes/files d'attente donnerait des noms tels que:

- PRODUCTION.EVENTS.STATUS
- PRODUCTION.EVENTS.BidSet.created
- PRODUCTION.REQUESTS.BidSet.create
- STUDY.EVENTS.Contingency.created

Il est important de noter que les mises en œuvre JMS permettent généralement d'utiliser des caractères de substitution dans les abonnements. L'objectif de l'organisation décrite est de permettre de tirer profit de cette fonction. De plus, il est possible d'étendre les définitions de thèmes afin de fournir des abonnements plus granuleux. Par exemple, un thème sous la

forme <context>.EVENTS.<Noun> peut être augmenté pour inclure un ID d'objet spécifique pour la mise en œuvre d'un projet spécifique.

8.4.3 Champs de messages JMS

Il est également important de noter que certains champs d'en-tête JMS sont relatifs au champ situé dans l'en-tête du message CEI 61968. Le tableau suivant présente les cas où certains champs d'en-tête JMS peuvent être mis en correspondance avec les champs d'en-tête CEI 61968-100 en guise de bonne pratique; mais cette mise en correspondance n'est pas obligatoire.

Champ d'en-tête JMS	Défini par	Champ d'en-tête 61968-100
JMSDestination	méthode d'envoi ou de publication	Sans Objet
JMSDeliveryMode	méthode d'envoi ou de publication	Sans Objet
JMSExpiration	méthode d'envoi ou de publication	Sans Objet
JMSPriority	méthode d'envoi ou de publication	Sans Objet
JMSMessageID	méthode d'envoi ou de publication	MessageID
JMSTimestamp	méthode d'envoi ou de publication	TimeStamp
JMSCorrelationID	Client	CorrelationID
JMSReplyTo	Client	ReplyAddress
JMSType	Client	Sans Objet
JMSRedelivered	Fournisseur JMS	Sans Objet

9 Sécurité

La sécurité est une préoccupation principale pour la plupart des mises en œuvre. Les exigences de sécurité peuvent différer en fonction du scénario d'intégration spécifique. Voici certains de ces différents exemples de scénarii:

- Intégration intra-application de composants dans un environnement contrôlé
- Intégration inter-application dans un environnement contrôlé
- intégration inter-application dans une entreprise
- Intégration inter-entreprises entre des partenaires de confiance au moyen d'une infrastructure de confiance
- Intégration extra entreprise comprenant une intégration d'application vers appareil en entreprise
- Services accessibles publiquement

Il existe de nombreux mécanismes et approches qui peuvent être employés en fonction des exigences.

L'utilisation d'une enveloppe SOAP (utilisable avec les services JMS et les services Web) procure certains avantages, grâce auxquels de nombreux produits tireront profit des en-têtes SOAP.

Dans les cas où des messages utilisent un réseau public, la sécurité est une question importante, bien qu'il y ait d'autres cas dans lesquels la sécurité peut être une question importante. La sécurité peut inclure l'authentification, l'autorisation, le cryptage et la non-répudiation. Les détails de la mise en œuvre de la sécurité ne relèvent pas du domaine d'application de la présente norme.

Il y a deux étapes de base dans la protection des interactions de messagerie. Premièrement, la couche transport est sécurisée. La seconde étape consiste à sécuriser le message lui-même. De manière générale, la couche transport est sécurisée par l'utilisation de SSL (Secure Socket Layer) et de TLS (Transport Layer Security). En plus de créer un canal de communication sécurisé entre un client et un service, les échanges de messages nécessitent que les informations de sécurité soient intégrées au message lui-même. C'est souvent le cas lorsqu'un message doit être traité par plusieurs nœuds intermédiaires avant d'atteindre le service cible ou lorsqu'un message doit être filtré parmi plusieurs services dans le but d'être traité.

Il est important de noter que la sécurité au niveau du message est très utile dans les applications centrées sur les documents XML car différentes sections du document XML peuvent avoir différentes exigences de sécurité ou être destinées à différents utilisateurs.

10 Contrôle de version

Il est important de noter que de nouvelles versions des interfaces peuvent être fournies de temps à autre, principalement en conséquence de:

- L'activation de la mise en œuvre initiale
- Nouvelles exigences
- Mises à niveau apportées à des produits de fournisseurs

Lorsque c'est possible, les interfaces évoluent par l'augmentation, lorsqu'une nouvelle version d'une interface est compatible avec une version précédente d'une interface. Cependant, ce n'est pas toujours possible. De nouvelles versions d'interfaces se manifestent par:

- Des modifications des WSDL
- Des modifications des Schémas XML
- Des modifications des mises en œuvre logicielles

Conformément aux lignes directrices OASIS pour les espaces de noms, il est fortement souhaitable de préserver les espaces de noms, particulièrement lorsque des définitions sont rétrocompatibles. Il convient de ne créer des nouveaux espaces de noms que lorsqu'une définition ne peut pas être rétrocompatible. Il existe deux types de mises à jour en termes de contrôle de version:

- Mise à jour de version majeure:

  Dans ce cas, une mise à jour majeure a été faite dans un XSD, ce qui a détruit sa rétrocompatibilité.

- Mise à jour de version mineure:

  Dans ce cas, la rétrocompatibilité est intacte. Exemple de mise à jour mineure: ajout d'un nouvel élément optionnel.

Une convention de désignation du contrôle de version des charges utiles de message est proposée ici pour utiliser XSD targetNamespace, l'attribut de la version et l'annotation comme suit:

- targetNamespace="http://iec.ch/TC57/yyyy/<Payload Type Name>**"**
- **version="**<Major version>.<Minor version>**".**
- Annotation ajoutée à la description en détail comme “Version 1.0 created in 2009/02”.

Voici deux exemples de mises à jour majeures et mineures de XSD, respectivement.

Dans cet exemple, une version 2009/02 reçoit une mise à jour majeure. Son targetNamespace et sa version peuvent passer de:

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2010/EndDeviceControls"
version="1.0">
   <xs:annotation>
      <xs:documentation>
         Major version 1.0 created in 2010/11
      </xs:documentation>
   </xs:annotation>
```

Vers

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2011/EndDeviceControls"
version="2.0">
   <xs:annotation>
      <xs:documentation>
         Major version 2.0 created in 2011/03
      </xs:documentation>
   </xs:annotation>
```

Toutefois, si une mise à jour est mineure, son targetNamespace et sa version peuvent être modifiés, comme suit, de:

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2010/EndDeviceControls"
version="1.0">
   <xs:annotation>
      <xs:documentation>
         Major version 1.0 created in 2010/11
      </xs:documentation>
   </xs:annotation>
```

Vers l'exemple suivant avec une version mineure:

```
<xs:schema ... targetNamespace="http://iec.ch/TC57/2010/EndDeviceControls"
version="1.1">
   <xs:annotation>
      <xs:documentation>
         Major version 1.0 created in 2010/11
Minor version 1.1 created in 2010/12
      </xs:documentation>
   </xs:annotation>
```

L'attribut de "version" ne s'applique pas à la validation XML par rapport à un XSD, donc son changement de contenu (2ème exemple, changement mineur) ne détruit pas la validation par rapport à la version XSD précédente.

Pour le contrôle des versions du Message.xsd, des règles similaires s'appliquent.

Annexe A
(normative)

Schéma XML pour enveloppe de message commune

Le schéma XML suivant est utilisé pour définir une enveloppe de message commune (CME) pour les messages de demande, de réponse et d'évènement tels qu'ils sont référencés dans la présente norme.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Common Message Specification for IEC 61968 -->
<!--            Change Log                      -->
<!-- 2010/12/15 Added OperationSet to Payload   -->
<!-- 2011/03/09 Corrected FaultMessageType      -->
<!-- 2011/03/09 Baseline for version control    -->
<!-- 2011/03/10 Created type definitions for OperationSet and Operation to improve
compatibility with SoapUI  -->
<!-- 2011/05/06 Removed deprecated verbs, added 'executed'  -->
<!-- 2011/05/06 Changed base namespace to follow WG14 convention of 'iec.ch'  -->
<!-- 2012/02/10 Added relatedObject to Error element  -->
<!-- 2012/02/11 Created a new ObjectType for use in Error element  -->
<!-- 2012/02/11 Removed enumeration for Header.Context  -->
<!-- 2012/02/12 Added note that Error.object.Name elements are deprecated  -->
<!-- 2012/02/12 Added more comments to message elements  -->
<!-- 2012/02/16 Corrected comment for Reply.Error.level  -->
<!-- 2012/02/16 Revised comment for Reply.Error.code  -->
<!-- 2012/02/22 Added ID to Payload for optional use by close/cancel/delete  -->
<!-- 2012/02/22 Extended ID elements to have attributes for idType, idAuthority,
iSmRID -->
<!-- 2012/02/22 Extended ErrorType elements to use ID and relatedID elements, with
deprecation of object -->
<!-- 2012/02/24 Added kind attribute to ID elements in place of iSmRID -->
<!-- 2012/03/19 Corrected ID and relatedID definitions in ErrorType -->
<!-- 2012/03/20 Revised ID elements to use an attribute group -->
<!-- 2012/03/21 Corrected Payload.ID elements -->
<!-- 2012/04/03 Corrected Reply.Error.object.Name -->
<!-- 2012/04/03 Corrected Header.User.Organization made optional -->
<!-- 2012/06/08 Updated IDatts attribute group to include objectType attribute as
string -->
<!-- 2012/10/14 corrections and revisions to annotations for FDIS -->
<xs:schema xmlns="http://iec.ch/TC57/2011/schema/message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://iec.ch/TC57/2011/schema/message"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
<xs:complexType name="RequestType">
<xs:annotation>
<xs:documentation>Request type definition</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:annotation>
<xs:documentation>Request package is typically used to supply parameters for 'get'
requests</xs:documentation>
</xs:annotation>
<xs:element name="StartTime" type="xs:dateTime" minOccurs="0">
<xs:annotation>
<xs:documentation>Start time of interest</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="EndTime" type="xs:dateTime" minOccurs="0">
<xs:annotation>
<xs:documentation>End time of interest</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Option" type="OptionType" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Request type specialization</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ID" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Object ID for request</xs:documentation>
```

```
</xs:annotation>
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attributeGroup ref="IDatts"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>This can be a CIM profile defined as an XSD with a CIM-specific
namespace This may also be used for custom extensions.</xs:documentation>
</xs:annotation>
</xs:any>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ReplyType">
<xs:annotation>
<xs:documentation>Reply type definition</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:annotation>
<xs:documentation>Reply package is used to confirm success or report
errors</xs:documentation>
</xs:annotation>
<xs:element name="Result">
<xs:annotation>
<xs:documentation>Reply code: OK, PARTIAL or FAILED</xs:documentation>
</xs:annotation>
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="OK"/>
<xs:enumeration value="PARTIAL"/>
<xs:enumeration value="FAILED"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Error" type="ErrorType" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Reply details describing one or more errors</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ID" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Resulting transaction ID (usually consequence of
create)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attributeGroup ref="IDatts"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Used for custom extensions</xs:documentation>
</xs:annotation>
</xs:any>
<xs:element name="operationId" type="xs:integer" minOccurs="0">
<xs:annotation>
<xs:documentation>The reply.operationId provides the unique identifier of the
Operation for which this reply.result is relevant. Thus, it is  assumed that this is a
partial reply in direct response to one of the operations contained in an OperationSet
request.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="PayloadType">
<xs:annotation>
<xs:documentation>Payload container</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:choice>
```

```
<xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>For XML payloads, usually CIM profiles defined using an XSD in a
profile-specific namespace. May also be used for custom extensions.</xs:documentation>
</xs:annotation>
</xs:any>
<xs:element name="OperationSet" type="OperationSet" minOccurs="0">
<xs:annotation>
<xs:documentation>Each operation set is a collection of operations that may require
operational-integrity and/or sequence control.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Compressed" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>For compressed and/or binary, uuencoded payloads If compressed, Gzip
compression is used.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ID" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Used to supply object IDs for cancel/close/delete operations in
cases where they are not otherwise specified using a type-specific
payload</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attributeGroup ref="IDatts"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:choice>
<xs:element name="Format" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Hint as to format of payload, e.g. XML, RDF, SVF, BINARY, PDF,
...</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="OperationType">
<xs:annotation>
<xs:documentation>For master data set synchronization XML payloads.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="operationId" type="xs:integer">
<xs:annotation>
<xs:documentation>The payload.operation.operationId provides the unique identifier
(within the OperationSet) of the Operation for the purpose of reference in subsequent
messages (e.g. OperationSet reply).</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="noun" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>The payload.operation.##other also identifies the noun, this element
is optionally supplied to simplify processing.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="verb" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>"create", "delete", "change", etc.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="elementOperation" type="xs:boolean" default="false" minOccurs="0">
<xs:annotation>
<xs:documentation>TRUE if the verb is operating at the element level. In such a case,
the verb is to be applied to the elements populated in the payload.operation.##other
below. If omitted, assume FALSE.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:any namespace="##other" processContents="skip" minOccurs="0">
<xs:annotation>
<xs:documentation>An XML payload which carries a CIM profile defined using an XSD in a
profile-specific namespace. Individual payloads are used collectively to create a
series of related operations. See the "enforce" boolean flags in the header for
instructions on how to process these messages.</xs:documentation>
```

```
</xs:annotation>
</xs:any>
</xs:sequence>
</xs:complexType>
<xs:complexType name="OperationSet">
<xs:annotation>
<xs:documentation>Each operation set is a collection of operations that may require
operational-integrity and/or sequence control.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="enforceMsgSequence" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation>If set to TRUE, the Operation.##other messages must be processed in
the sequence presented. If omitted, assume FALSE.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="enforceTransactionalIntegrity" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation>Set to TRUE when all of the Operation.##other messages must be
processed successfully or else the entire message set must be rolled back. If omitted,
assume FALSE.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Operation" type="OperationType" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ReplayDetectionType">
<xs:annotation>
<xs:documentation>Used to detect and prevent replay attacks</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Nonce" type="xs:string"/>
<xs:element name="Created" type="xs:dateTime"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="UserType">
<xs:annotation>
<xs:documentation>User type definition</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="UserID" type="xs:string">
<xs:annotation>
<xs:documentation>User identifier</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Organization" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>User parent organization identifier</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="HeaderType">
<xs:annotation>
<xs:documentation>Message header type definition</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:annotation>
<xs:documentation>Message header contains control and descriptive information about
the message.</xs:documentation>
</xs:annotation>
<xs:element name="Verb">
<xs:annotation>
<xs:documentation>This enumerated list of verbs that can be used to form message types
in compliance with the IEC 61968 standard.</xs:documentation>
</xs:annotation>
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="cancel"/>
<xs:enumeration value="canceled"/>
<xs:enumeration value="change"/>
<xs:enumeration value="changed"/>
<xs:enumeration value="create"/>
<xs:enumeration value="created"/>
<xs:enumeration value="close"/>
<xs:enumeration value="closed"/>
<xs:enumeration value="delete"/>
```

```
<xs:enumeration value="deleted"/>
<xs:enumeration value="get"/>
<xs:enumeration value="reply"/>
<xs:enumeration value="execute"/>
<xs:enumeration value="executed"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="Noun" type="xs:string">
<xs:annotation>
<xs:documentation>The Noun of the Control Area identifies the main subject of the
message type, typically a real world object defined in the CIM.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Revision" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Revision level of the message type.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ReplayDetection" type="ReplayDetectionType" minOccurs="0">
<xs:annotation>
<xs:documentation>Use to introduce randomness in the message to enhance effectiveness
of encryption</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Context" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Intended context for information usage, e.g. PRODUCTION, TESTING,
TRAINING, ...</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Timestamp" type="xs:dateTime" minOccurs="0">
<xs:annotation>
<xs:documentation>Application level relevant time and date for when this  instance of
the message type was produced. This is not intended to be used by middleware for
message management.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Source" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Source system or application that sends the
message</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="AsyncReplyFlag" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation>Indicates whether or not reply should be
asynchronous</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ReplyAddress" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Address to be used for asynchronous replies, typically a
URL/topic/queue.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="AckRequired" type="xs:boolean" minOccurs="0">
<xs:annotation>
<xs:documentation>Indicates whether or not an acknowledgement is
required</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="User" type="UserType" minOccurs="0">
<xs:annotation>
<xs:documentation>User information of the sender</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="MessageID" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Unique message ID to be used for tracking
messages</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="CorrelationID" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>ID to be used by applications for correlating
replies</xs:documentation>
</xs:annotation>
```

```
</xs:element>
<xs:element name="Comment" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Optional comment</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Property" type="MessageProperty" minOccurs="0"
maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Message properties can be used to identify information needed for
extended routing and filtering capabilities</xs:documentation>
</xs:annotation>
</xs:element>
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Used to allow custom extensions</xs:documentation>
</xs:annotation>
</xs:any>
</xs:sequence>
</xs:complexType>
<xs:element name="Message" type="MessageType">
<xs:annotation>
<xs:documentation>Common IEC 61968 Message Definition</xs:documentation>
</xs:annotation>
</xs:element>
<xs:complexType name="MessageProperty">
<xs:annotation>
<xs:documentation>Message properties can be used for extended routing and
filtering</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Name" type="xs:string"/>
<xs:element name="Value" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:element name="RequestMessage" type="RequestMessageType">
<xs:annotation>
<xs:documentation>Request message structure</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ResponseMessage" type="ResponseMessageType">
<xs:annotation>
<xs:documentation>Response message structure</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="EventMessage" type="EventMessageType">
<xs:annotation>
<xs:documentation>Event message structure. </xs:documentation>
</xs:annotation>
</xs:element>
<xs:complexType name="MessageType">
<xs:annotation>
<xs:documentation>Generic Message Type</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Header" type="HeaderType"/>
<xs:element name="Request" type="RequestType" minOccurs="0"/>
<xs:element name="Reply" type="ReplyType" minOccurs="0"/>
<xs:element name="Payload" type="PayloadType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="RequestMessageType">
<xs:annotation>
<xs:documentation>Request Message Type, which will typically result in a
ResponseMessage to be returned. This is typically used to initiate a transaction or a
query request.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Header" type="HeaderType"/>
<xs:element name="Request" type="RequestType" minOccurs="0"/>
<xs:element name="Payload" type="PayloadType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ResponseMessageType">
<xs:annotation>
<xs:documentation>Response MessageType, typically used to reply to a
RequestMessage</xs:documentation>
</xs:annotation>
```

```
<xs:sequence>
<xs:element name="Header" type="HeaderType"/>
<xs:element name="Reply" type="ReplyType"/>
<xs:element name="Payload" type="PayloadType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="FaultMessageType">
<xs:annotation>
<xs:documentation>Fault Message Type, which is used in cases where the incoming
message (including the header) cannot be parsed</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Reply" type="ReplyType"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="EventMessageType">
<xs:annotation>
<xs:documentation>Event Message Type, which is used to indicate a condition of
potential interest. Note that the Payload may be required in the
future.</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Header" type="HeaderType"/>
<xs:element name="Payload" type="PayloadType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ErrorType">
<xs:annotation>
<xs:documentation>Error Structure</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="code" type="xs:string">
<xs:annotation>
<xs:documentation>Defined error code, as defined by IEC 61968-100, related standards
or local implementation</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="level" minOccurs="0">
<xs:annotation>
<xs:documentation>Severity level, e.g. INFORM, WARNING, FATAL,
CATASTROPHIC</xs:documentation>
</xs:annotation>
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="INFORM"/>
<xs:enumeration value="WARNING"/>
<xs:enumeration value="FATAL"/>
<xs:enumeration value="CATASTROPHIC"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="reason" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Description of the error</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="details" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Free form detailed text description of error</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="xpath" type="xs:QName" minOccurs="0">
<xs:annotation>
<xs:documentation>XPath expression to identify specific XML element</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="stackTrace" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Stack trace as generated by software upon
exception</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Location" type="LocationType" minOccurs="0">
<xs:annotation>
<xs:documentation>Location of exception within software</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ID" minOccurs="0">
```

```
<xs:annotation>
<xs:documentation>ID of object</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attributeGroup ref="IDatts"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="relatedID" minOccurs="0">
<xs:annotation>
<xs:documentation>ID of related object, used in cases where there is an error between
the relationship of two objects</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attributeGroup ref="IDatts"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="object" type="ObjectType" minOccurs="0">
<xs:annotation>
<xs:documentation>Deprecated</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="operationId" type="xs:integer" minOccurs="0">
<xs:annotation>
<xs:documentation>The reply.operationId provides the unique identifier of the
Operation for which this reply.result.error is relevant. Thus, it is  assumed that
this is an error from one of the operations contained in an OperationSet
request.</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="OptionType">
<xs:annotation>
<xs:documentation>Request options</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="value" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="LocationType">
<xs:annotation>
<xs:documentation>Process location where error was encountered</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="node" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Name of the pipeline/branch/route node where error
occurred</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="pipeline" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Name of the pipeline where error occurred (if
applicable)</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="stage" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Name of the stage where error occurred (if
applicable)</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ObjectType">
<xs:annotation>
<xs:documentation>Used to identify an object of interest</xs:documentation>
</xs:annotation>
<xs:sequence>
```

```
<xs:element name="mRID" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>A UUID-based name for the object</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Name" type="Name" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>The Name structure is deprecated. It will be completely removed in
the next edition</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="objectType" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Type of object</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="NameType">
<xs:annotation>
<xs:documentation>From CIM</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string" minOccurs="0"/>
<xs:element name="NameTypeAuthority" type="NameTypeAuthority" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Name">
<xs:annotation>
<xs:documentation>From CIM</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="NameType" type="NameType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="NameTypeAuthority">
<xs:annotation>
<xs:documentation>From CIM</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:element name="FaultMessage" type="FaultMessageType">
<xs:annotation>
<xs:documentation>Fault message structure</xs:documentation>
</xs:annotation>
</xs:element>
<xs:simpleType name="IDKindType">
<xs:annotation>
<xs:documentation>ID Kind Type</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:string">
<xs:enumeration value="name"/>
<xs:enumeration value="uuid"/>
<xs:enumeration value="transaction"/>
<xs:enumeration value="other"/>
</xs:restriction>
</xs:simpleType>
<xs:attributeGroup name="IDatts">
<xs:annotation>
<xs:documentation>ID attribute group</xs:documentation>
</xs:annotation>
<xs:attribute name="idType" type="xs:string"/>
<xs:attribute name="idAuthority" type="xs:string"/>
<xs:attribute name="kind" type="IDKindType"/>
<xs:attribute name="objectType" type="xs:string"/>
</xs:attributeGroup>
</xs:schema>
```

Annexe B
(normative)

Verbes

Le Tableau B.1 fournit des définitions normatives des verbes à utiliser dans les en-têtes de messages, comme cela est défini dans la CEI 61968-1. Elles sont réalisées sous la forme de valeurs énumérées dans Message.xsd.

Tableau B.1 – Définitions normatives des verbes

Verbes	Signification	Structure du message
Create	Le verbe 'create' (créer) sert à éditer une demande auprès du système maître pour créer un nouvel objet. Le système maître peut à son tour éditer le nouvel objet comme un événement à l'aide du verbe 'created' (créé). Le système maître peut aussi utiliser le verbe 'reply' (répondre) pour répondre à une demande 'create', indiquant si la demande a été traitée avec succès ou non.	Le message de demande inclura les structures HeaderType et Payload.
Change	Le verbe 'change' (changer) sert à éditer une demande auprès du système maître pour apporter un changement à un objet en fonction de l'information contenue dans le message. Le système maître peut à son tour éditer l'objet changé comme un événement à l'aide du verbe 'changed' (changé) pour signaler que l'objet a été modifié depuis sa dernière édition. Le système maître peut aussi utiliser le verbe 'reply' (répondre) pour répondre à une demande 'change', indiquant si la demande a été traitée avec succès ou non.	Le message de demande inclura les structures HeaderType et RequestType ainsi que, facultativement, la structure Payload. La structure requestType identifiera potentiellement les ID d'objets spécifiques.
Cancel	Le verbe 'cancel' (annuler) sert à éditer une demande auprès du système maître pour annuler un objet, le plus souvent dans les cas où l'objet représente un document commercial. Le système maître peut à son tour éditer le message annulé comme un événement à l'aide du verbe 'canceled' (annulé) pour signaler que le document a été annulé depuis sa dernière édition. Le système maître peut aussi utiliser le verbe 'reply' (répondre) pour répondre à une demande 'cancel', indiquant si la demande a été traitée avec succès ou non. Le verbe 'cancel' est utilisé lorsque le contenu commercial du document n'est plus valide en raison d'une ou plusieurs erreurs.	Le message de demande inclura les structures HeaderType et RequestType ainsi que, facultativement, la structure Payload. La structure requestType identifiera potentiellement les ID d'objets spécifiques.
Close	Le verbe 'close' (fermer) sert à éditer une demande auprès du système maître pour fermer un objet, le plus souvent dans les cas où l'objet représente un document commercial. Le système maître peut à son tour éditer le message fermé comme un événement à l'aide du verbe 'closed' (fermé) pour signaler que le document a été fermé depuis sa dernière édition. Le système maître peut aussi utiliser le verbe 'reply' (répondre) pour répondre à une demande 'close', indiquant si la demande a été traitée avec succès ou non. Le verbe 'close' (fermer) est utilisé lorsque le document commercial atteint la fin de son cycle de vie en raison de l'achèvement réussi d'un processus commercial.	Le message de demande inclura les structures HeaderType et RequestType ainsi que, facultativement, la structure Payload. La structure requestType identifiera potentiellement les ID d'objets spécifiques.
Delete	Le verbe 'delete' (supprimer) est utilisé pour éditer une demande auprès du système maître pour supprimer un ou plusieurs objets. Le système maître peut à son tour éditer le message fermé comme un événement à l'aide du verbe 'deleted' (supprimé pour signaler que l'objet a été supprimé depuis sa dernière édition. Le système maître peut aussi utiliser le verbe 'reply' (répondre) pour répondre à une demande 'delete', indiquant si la demande a été traitée avec succès ou non. Le verbe 'delete' est utilisé lorsqu'il convient de ne plus conserver l'objet commercial dans les systèmes intégrés soit en raison d'une ou plusieurs erreurs, soit pour des raisons d'archivage. Cependant, le système maître conservera très vraisemblablement un enregistrement historique de l'objet après sa suppression.	Le message de demande inclura les structures HeaderType et RequestType ainsi que, facultativement, la structure Payload. La structure requestType identifiera potentiellement les ID d'objets spécifiques.
Execute	Utilisé lorsque le message transmet une transaction complexe impliquant une variété d'opérations create, delete et/ou change par l'utilisation de l'élément Payload.OperationSet.	Voir Payload.OperationSet dans Message.xsd.

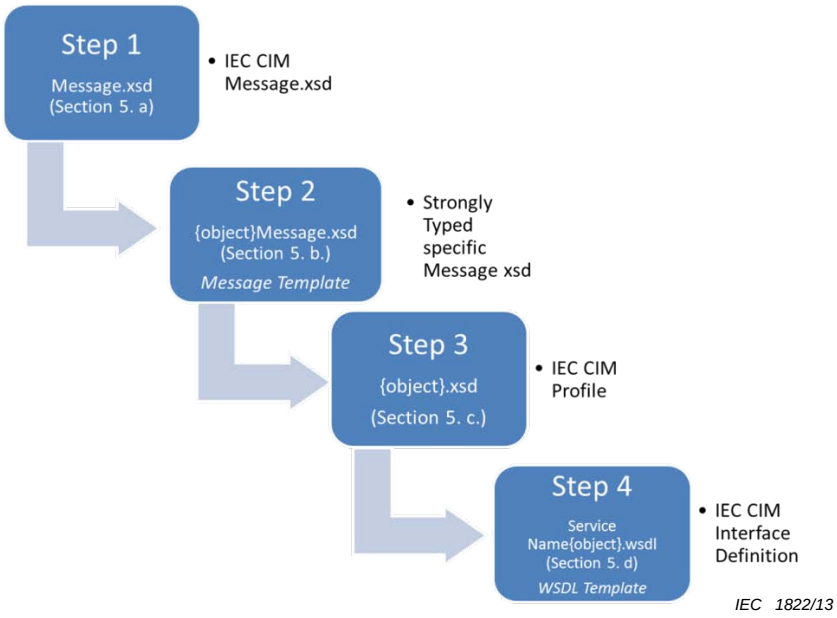
Verbes	Signification	Structure du message
Get	Le verbe 'get' (récupérer) est utilisé pour émettre une demande d'interrogation à un système maître pour retourner un ensemble d'un nombre zéro ou supérieur d'objets qui satisfont à un critère spécifié. Le système maître peut à son tour retourner un nombre zéro ou supérieur d'objets à l'aide du verbe 'reply' (répondre) dans un message de réponse.	Le message de demande inclura les structures HeaderType et RequestType. La structure requestType identifiera potentiellement les paramètres spécifiques pour qualifier la demande, tels que les ID d'objets.
Created	Le verbe 'créé' est utilisé pour éditer un événement qui est une notification de la création d'un objet suite à une demande externe ou à une action interne au système maître de l'objet en question. Ce type de message fait habituellement l'objet d'un abonnement de la part des systèmes intéressés et pourrait être utilisé pour des mises à jour en masse. Il n'est pas indispensable de répondre à ce type de message.	Le message d'événement inclura les structures HeaderType et Payload.
Changed	Le verbe 'changé' est utilisé pour éditer un événement qui est une notification du changement d'un objet suite à une demande externe ou à une action interne au système maître de l'objet en question. Il pourrait s'agir d'un changement générique du contenu de l'objet ou d'un changement de statut spécifique tel que “approved” (approuvé), “issued” (émis) etc. Ce type de message fait habituellement l'objet d'un abonnement de la part des systèmes intéressés et pourrait être utilisé pour des mises à jour en masse. Il n'est pas indispensable de répondre à ce type de message.	Le message d'événement inclura les structures HeaderType et Payload.
Closed	Le verbe 'closed' (fermé) est utilisé pour éditer un événement qui est une notification de la fermeture normale d'un objet suite à une demande externe ou à une action interne au système maître de l'objet en question. Ce type de message fait habituellement l'objet d'un abonnement de la part des systèmes intéressés et pourrait être utilisé pour des mises à jour en masse. Il n'est pas indispensable de répondre à ce type de message.	Le message d'événement inclura les structures HeaderType et Payload.
Canceled	Le verbe 'canceled' (annulé) est utilisé pour éditer un événement qui est une notification de l'annulation d'un objet suite à une demande externe ou à une action interne au système maître de l'objet en question. Ce type de message fait habituellement l'objet d'un abonnement de la part des systèmes intéressés et pourrait être utilisé pour des mises à jour en masse. Il n'est pas indispensable de répondre à ce type de message.	Le message d'événement inclura les structures HeaderType et Payload.
Deleted	Le verbe 'deleted' (supprimé) est utilisé pour éditer un événement qui est une notification de la suppression d'un objet suite à une demande externe ou à une action interne au système maître de l'objet en question. Ce type de message fait habituellement l'objet d'un abonnement de la part des systèmes intéressés et pourrait être utilisé pour des mises à jour en masse. Il n'est pas indispensable de répondre à ce type de message.	Le message d'événement inclura les structures HeaderType et Payload.
Executed	Subvient aux besoins d'un évènement indiquant l'exécution d'une transaction complexe qui utilise l'élément Payload.OperationSet.	Voir Payload.OperationSet dans Message.xsd.
Reply	Il existe deux principales utilisations du verbe 'reply' (répondre). Toutefois, dans les deux cas, il est seulement utilisé en réponse à des messages de demande, que le modèle utilisé soit synchrone ou asynchrone. La première utilisation est d'indiquer le succès, le succès partiel ou l'échec d'une demande transactionnelle au système maître de créer, changer, supprimer, annuler ou fermer un document. La seconde utilisation est en réponse à une demande 'get', lorsque les objets d'intérêt peuvent être retournés dans la réponse.	Utilisé uniquement pour les messages de réponse. Pour les réponses à des demandes transactionnelles, le message contiendra les structures HeaderType et ReplyType. Pour les réponses à des demandes "get", le message contiendra les structures HeaderType et ReplyType et, potentiellement, la structure Payload.

Annexe C
(normative)

Procédure pour les services Web fortement typés

C.1 Généralités

La présente annexe a pour objet de décrire le processus de génération des WSDL et des artefacts associés. La Figure C.1 fournit un aperçu du processus nécessaire pour créer et référencer les artefacts de conception spécifiques.



Légende

Anglais	Français
Step	Etape
IEC CIM Message.xsd	Message.xsd CEI CIM
Strongly Typed specific Message.xsd	Message.xsd fortement typé spécifique
IEC CIM Profile	Profil CEI CIM
IEC CIM Interface Definition	Définition d'interface CEI CIM
{object}Message.xsd	{objet}Message.xsd
{object}.xsd	{objet}.xsd
ServiceName{object}.wsdl	ServiceName{objet}.wsdl

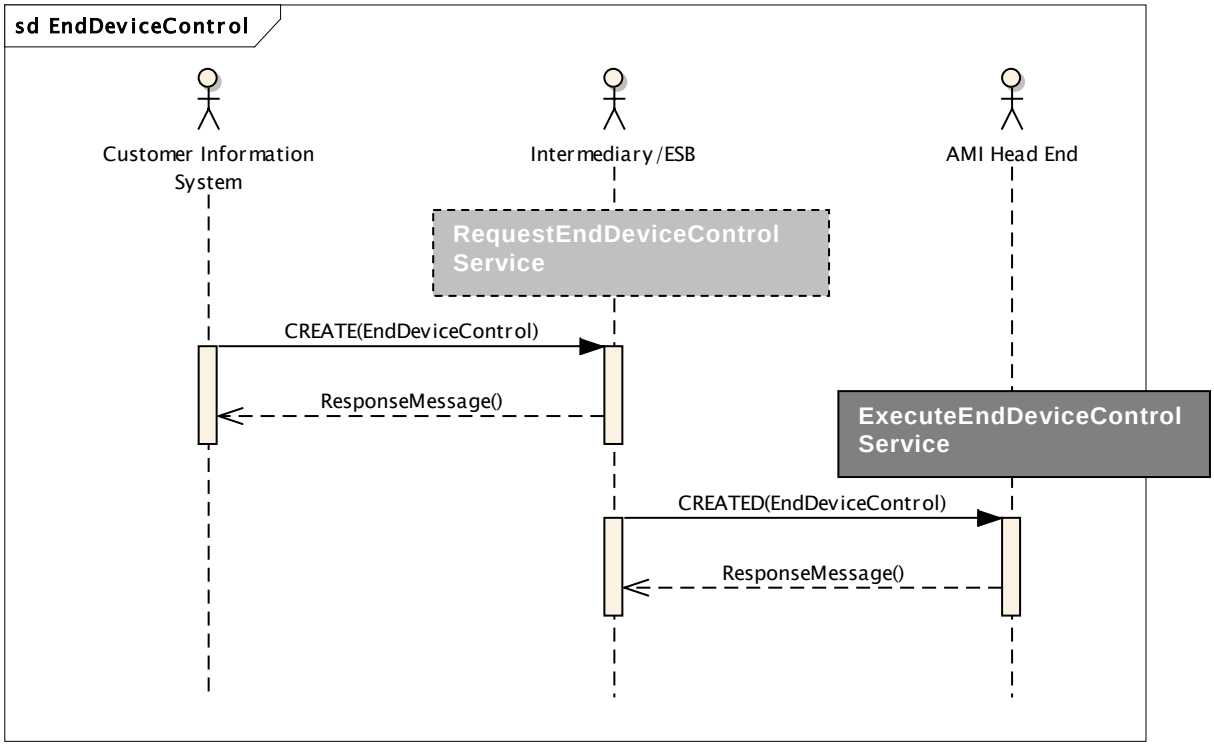
Figure C.1 – Processus de génération WSDL

Avec l'aide de modèles, le processus permet la création d'un WSDL qui définit un ensemble d'opérations pour un type d'objet donné (nom spécifique). Chaque opération représente la combinaison d'un verbe et d'un nom. Ce WSDL référencera ensuite une enveloppe de message spécifique au modèle qui référence à la fois les définitions de structure d'enveloppe de la norme 61968 et une définition de profil pour le nom donné.

C.2 Etapes de définition WSDL

Etape 1) Diagramme de séquence basé sur un cas d'utilisation

Le diagramme de séquence de la Figure C.2 affiche le flux de messages, les prestataires de services et les consommateurs en se basant sur un cas d'utilisation. Il présente des messages en séquence basée sur des exigences d'intégration. Le diagramme suivant montre un flux de messages EndDeviceControl provenant de CIS à destination de HeadEnd en passant par un ESB intermédiaire. CIS, dans ce cas, est demandeur de EndDeviceControl, donc son message à l'ESB contient le verbe "Create" au présent. Le nom du message suit la convention de nom des opérations <Verbe>+<Objet d'information> telle qu'elle est décrite en 8.3.4. Dans ce cas, le verbe est "Create" et l'objet d'information est "EndDeviceControl".



Légende

Anglais	Français
Customer Information System	Système d'informations client
Intermediary/ESB	Intermédiaire/ESB
AMI Head End	AMI final de tête

Figure C.2 – Exemple de diagramme de séquence

Etape 2) Sémantique des services

Deux services sont fournis pour tout le flux de messages sur la base du diagramme de séquence, l'un par ESB et l'autre par HeadEnd. La désignation de ces deux services se base sur leur modèle de service (voir 8.3.4), comme le rôle du service fourni par l'ESB est de "Request" (demander) la commande de dispositif final et le modèle du service fourni par HeadEnd est "Execute" (exécuter) la commande de dispositif final. Chaque service reçoit la même opération indiquée sous forme de messages dans le diagramme de séquence. En conséquence, les deux services reçoivent l'opération "CreatedEndDeviceControl" mais ont un nom de service différent; l'un est appelé "RequestEndDeviceControl" et l'autre "ExecuteEndDeviceControl".

Etape 3) Créer une structure de dossier

Sous le dossier racine préférentiel, créer un dossier xsd pour les modèles xsd et les profils CIM. Les artefacts de WDSL seront un niveau au-dessus du dossier xsd.

La Figure C.3 montre la structure du répertoire WSDL.

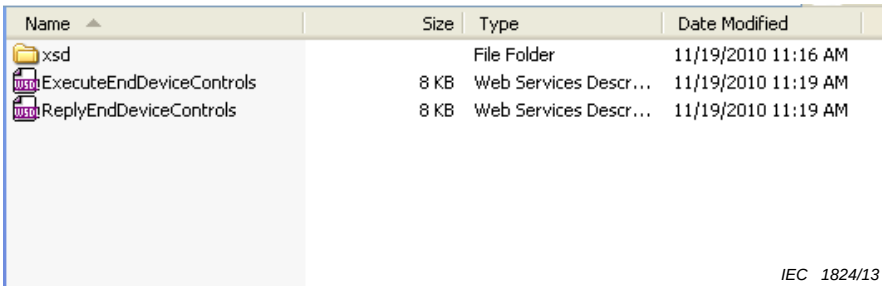


Figure C.3 – Structure d'un répertoire WSDL

Les WSDL sont situés dans le dossier racine. Les fichiers XSD référencés sont situés dans le dossier XSD. Un xsd de profil CIM (EndDeviceControls.xsd), un message.xsd commun et un {objet}Message.xsd se trouvent dans ce dossier. L'emplacement du schéma est spécifié dans la section wsdl:types comme l'exemple d'EndDeviceControlMessage énuméré par la Figure C.4.

```
<wsdl:types>

   <xs:schema targetNamespace="http://iec.ch/TC57/2011/schema/message"
               elementFormDefault="qualified">

      <xs:include schemaLocation="xsd/EndDeviceControlsMessage.xsd"/>

   </xs:schema>

</wsdl:types>
```

IEC 1825/13

Figure C.4 – Définitions de type WSDL

L'enveloppe du message commun, Message.xsd, peut être trouvée dans l'Annexe A.

Etape 4) Définition de charge utile de message & génération de WSDL

La définition de services suit l'Article 0. Le style littéral du document est utilisé dans la liaison SOAP. Comme pour une charge utile importante, MTOM peut être utilisé. Le modèle d'une définition WSDL de style de document conditionné peut être trouvé dans la section Modèle WSDL de cette Annexe.

Cet exemple indique comment générer un WSDL pour le modèle d'intégration Execute. Le même processus peut être utilisé pour tout modèle d'intégration en remplaçant Execute par le modèle de désignation de service voulu.

a) Copier le modèle de message xsd (voir le paragraphe Modèle de messages dans cette Annexe A) et le placer dans le dossier approprié

b) Copier le modèle de message xsd (voir le paragraphe Modèle de messages dans cette Annexe C) et remplacer la variable {Information_Object_Name} par le nom correct.

   – Sauvegarder sous {Information_Object_Name}Message.xsd (EndDeviceControlsMessage.xsd). Ce fichier est enregistré dans le dossier /xsd.

     Il existe deux types de modèles xsd de message d'objet:

     - Send/Receive/Reply/Request/Execute
     - Get/Reply

c) Placer le Profil xsd CEI CIM dans le dossier xsd créé à l'Etape 3.

d) Copier le modèle Execute wsdl (voir le paragraphe Modèle WSDL dans cette Annexe) et remplacer la variable {Pattern_Name} par le modèle correct et remplacer la variable {Information_Object_Name} avec le noun correct

– Sauvegarder sous Execute{Information_Object_Name}.wsdl (ExecuteEndDeviceControls.wsdl). Ce fichier est enregistré dans le dossier racine.

Il existe deux types de modèles xsd de message d'objet:

- Request/Execute
- Send/Receive/Reply

A noter qu'il s'agit d'un exemple de modèle *Execute*, mais les étapes sont identiques pour les autres modèles de services comme Reply.

C.3 Modèles de message

Le WSDL référencera un ensemble de structures de messages spécifique au modèle qui tirera à son tour profit du Message.xsd standard indépendant du modèle tel qu'il est décrit en Annexe A. Les occurrences de {Information_Object_Name} dans le modèle seraient remplacées par un nom de profil spécifique.

Deux modèles de messages sont fournis dans le Paragraphe C.3.

1) Modèle de message XSD pour:

- **Send/Receive/Reply**
- **Request/Execute**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msg="http://iec.ch/TC57/2011/schema/message"
xmlns:obj="http://iec.ch/TC57/2011/{Information_Object_Name}#"
targetNamespace="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
<!-- Base Message Definitions -->
<xs:import namespace="http://iec.ch/TC57/2011/schema/message"
schemaLocation="Message.xsd"/>
<!-- CIM Information Object Definition -->
<xs:import namespace="http://iec.ch/TC57/2011/{Information_Object_Name}#"
schemaLocation="{Information_Object_Name}.xsd"/>
<!-- PayloadType Definition -->
<xs:complexType name="{Information_Object_Name}PayloadType">
<xs:sequence>
<xs:element ref="obj:{Information_Object_Name}"/>
<xs:element name="OperationSet" type="msg:OperationSet" minOccurs="0"/>
<xs:element name="Compressed" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>For compressed and/or binary, uuencoded payloads</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Format" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Hint as to format of payload, e.g. XML, RDF, SVF, BINARY, PDF,
...</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<!-- Message Types -->
<!-- RequestMessageType -->
<xs:complexType name="{Information_Object_Name}RequestMessageType">
<xs:sequence>
<xs:element name="Header" type="msg:HeaderType"/>
<xs:element name="Request" type="msg:RequestType" minOccurs="0"/>
<xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

```
<!-- ResponseMessageType -->
<xs:complexType name="{Information_Object_Name}ResponseMessageType">
<xs:sequence>
<xs:element name="Header" type="msg:HeaderType"/>
<xs:element name="Reply" type="msg:ReplyType"/>
<xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<!-- EventMessageType -->
<xs:complexType name="{Information_Object_Name}EventMessageType">
<xs:sequence>
<xs:element name="Header" type="msg:HeaderType"/>
<xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<!-- FaultMessageType -->
<xs:complexType name="{Information_Object_Name}FaultMessageType">
<xs:sequence>
<xs:element name="Reply" type="msg:ReplyType"/>
</xs:sequence>
</xs:complexType>
<xs:element name="Create{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
<xs:element name="Change{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
<xs:element name="Cancel{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
<xs:element name="Close{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
<xs:element name="Delete{Information_Object_Name}"
type="tns:{Information_Object_Name}RequestMessageType"/>
<xs:element name="Created{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
<xs:element name="Changed{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
<xs:element name="Canceled{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
<xs:element name="Closed{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
<xs:element name="Deleted{Information_Object_Name}"
type="tns:{Information_Object_Name}EventMessageType"/>
<xs:element name="{Information_Object_Name}ResponseMessage"
type="tns:{Information_Object_Name}ResponseMessageType"/>
<xs:element name="{Information_Object_Name}FaultMessage"
type="tns:{Information_Object_Name}FaultMessageType"/>
</xs:schema>
```

2) Modèle de message XSD pour

- **Get et Reply**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msg="http://iec.ch/TC57/2011/schema/message"
xmlns:obj1="http://iec.ch/TC57/2011/{Information_Object_Name}#"
xmlns:obj2="http://iec.ch/TC57/2011/Get{Information_Object_Name}#"
targetNamespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
<!-- Base Message Definitions -->
<xs:import namespace="http://iec.ch/TC57/2011/schema/message"
schemaLocation="Message.xsd"/>
<!-- CIM Information Object Definition -->
<xs:import namespace="http://iec.ch/TC57/2011/{Information_Object_Name}#"
schemaLocation="{Information_Object_Name}.xsd"/>
<!-- Remove this Import if there is no "Get" Profile associated with this Object. -->
<xs:import namespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}#"
schemaLocation="Get{Information_Object_Name}.xsd"/>
<!-- RequestType Definition -->
<xs:complexType name="Get{Information_Object_Name}RequestType">
<xs:sequence>
<xs:element name="StartTime" type="xs:dateTime" minOccurs="0">
<xs:annotation>
<xs:documentation>Start time of interest</xs:documentation>
```

```
</xs:annotation>
</xs:element>
<xs:element name="EndTime" type="xs:dateTime" minOccurs="0">
<xs:annotation>
<xs:documentation>End time of interest</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Option" type="msg:OptionType" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Request type specialization</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="ID" type="xs:string" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>Object ID for request</xs:documentation>
</xs:annotation>
</xs:element>
<!-- Remove this Element if there is no "Get" Profile associated with this Object. -->
<xs:element ref="obj2:Get{Information_Object_Name}"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded">
<xs:annotation>
<xs:documentation>This can be a CIM profile defined as an XSD with a CIM-specific
namespace</xs:documentation>
</xs:annotation>
</xs:any>
</xs:sequence>
</xs:complexType>
<!-- PayloadType Definition -->
<xs:complexType name="{Information_Object_Name}PayloadType">
<xs:sequence>
<xs:element ref="obj1:{Information_Object_Name}" minOccurs="0"/>
<xs:element name="OperationSet" type="msg:OperationSet" minOccurs="0"/>
<xs:element name="Compressed" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>For compressed and/or binary, uuencoded payloads</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="Format" type="xs:string" minOccurs="0">
<xs:annotation>
<xs:documentation>Hint as to format of payload, e.g. XML, RDF, SVF, BINARY, PDF,
...</xs:documentation>
</xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>
<!-- Message Types -->
<!-- RequestMessageType -->
<xs:complexType name="Get{Information_Object_Name}RequestMessageType">
<xs:sequence>
<xs:element name="Header" type="msg:HeaderType"/>
<xs:element name="Request" type="tns:Get{Information_Object_Name}RequestType"/>
<xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<!-- ResponseMessageType -->
<xs:complexType name="{Information_Object_Name}ResponseMessageType">
<xs:sequence>
<xs:element name="Header" type="msg:HeaderType"/>
<xs:element name="Reply" type="msg:ReplyType"/>
<xs:element name="Payload" type="tns:{Information_Object_Name}PayloadType"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<!-- FaultMessageType -->
<xs:complexType name="{Information_Object_Name}FaultMessageType">
<xs:sequence>
<xs:element name="Reply" type="msg:ReplyType"/>
</xs:sequence>
</xs:complexType>
<xs:element name="Get{Information_Object_Name}"
type="tns:Get{Information_Object_Name}RequestMessageType"/>
<xs:element name="{Information_Object_Name}ResponseMessage"
type="tns:{Information_Object_Name}ResponseMessageType"/>
<xs:element name="{Information_Object_Name}FaultMessage"
type="tns:{Information_Object_Name}FaultMessageType"/>
</xs:schema>
```

NOTE Les chaînes suivantes sont à remplacer pour les deux modèles.

{Information_Object_Name}

Remplacé par le profil CIM en cours d'utilisation. Par exemple, EndDeviceControls. On utilise la forme plurielle de l'objet d'informations pour éviter les collisions dans le XSD.

C.4 Modèles WSDL

Cette section fournit des modèles WSDL à modifier afin de créer des artefacts de conception pour des services Web fortement typés.

WSDL pour les demandes ''Get''

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   name="Get{Information_Object_Name}"
         targetNamespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}"
         xmlns:tns="http://iec.ch/TC57/2011/Get{Information_Object_Name}"
         xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
         xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"

xmlns:infoMessage="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message">
         <wsdl:types>

         <xs:schema
targetNamespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}"
                     elementFormDefault="qualified">

<xs:import namespace="http://iec.ch/TC57/2011/Get{Information_Object_Name}Message"
schemaLocation="xsd/Get{Information_Object_Name}Message.xsd"/>

         </xs:schema>

      </wsdl:types>

      <!-- Message Definitions -->
      <wsdl:message name="Get{Information_Object_Name}RequestMessage">
         <wsdl:part name="Get{Information_Object_Name}RequestMessage"
element="infoMessage:Get{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="ResponseMessage">
         <wsdl:part name="ResponseMessage"
element="infoMessage:{Information_Object_Name}ResponseMessage"/>
      </wsdl:message>

      <wsdl:message name="FaultMessage">
         <wsdl:part name="FaultMessage"
element="infoMessage:{Information_Object_Name}FaultMessage"/>
      </wsdl:message>

      <!-- Port Definitions -->
      <wsdl:portType name="Get{Information_Object_Name}_Port">

         <wsdl:operation name="Get{Information_Object_Name}">
            <wsdl:input name="Get{Information_Object_Name}Request"
message="tns:Get{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Get{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Get{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

      </wsdl:portType>
```

```
      <wsdl:binding name="Get{Information_Object_Name}_Binding"
type="tns:Get{Information_Object_Name}_Port">

         <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

         <wsdl:operation name="Get{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/Get{Information_Object_Name}/Get{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Get{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Get{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Get{Information_Object_Name}Fault">
               <soap:fault name="Get{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
      </wsdl:binding>

      <wsdl:service name="Get{Information_Object_Name}">
         <wsdl:port name="Get{Information_Object_Name}_Port"
binding="tns:Get{Information_Object_Name}_Binding">
            <soap:address
location="http://iec.ch/TC57/2011/Get{Information_Object_Name}"/>
         </wsdl:port>
      </wsdl:service>

</wsdl:definitions>
```

WSDL pour Send, Receive, Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
         name="{Send | Receive | Reply}{Information_Object_Name}"
         targetNamespace="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"
         xmlns:tns="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"
         xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
         xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
         xmlns:infoMessage="http://iec.ch/TC57/2011/{Information_Object_Name}Message">

      <wsdl:types>

         <xs:schema targetNamespace="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"
                        elementFormDefault="qualified">

<xs:import namespace="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
schemaLocation="xsd/{Information_Object_Name}Message.xsd"/>

         </xs:schema>

      </wsdl:types>

      <!-- Message Definitions -->

      <wsdl:message name="Created{Information_Object_Name}EventMessage">
         <wsdl:part name="Created{Information_Object_Name}EventMessage"
element="infoMessage:Created{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Changed{Information_Object_Name}EventMessage">
         <wsdl:part name="Changed{Information_Object_Name}EventMessage"
element="infoMessage:Changed{Information_Object_Name}"/>
      </wsdl:message>
```

```
      <wsdl:message name="Closed{Information_Object_Name}EventMessage">
         <wsdl:part name="Closed{Information_Object_Name}EventMessage"
element="infoMessage:Closed{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Canceled{Information_Object_Name}EventMessage">
         <wsdl:part name="Canceled{Information_Object_Name}EventMessage"
element="infoMessage:Canceled{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="Deleted{Information_Object_Name}EventMessage">
         <wsdl:part name="Deleted{Information_Object_Name}EventMessage"
element="infoMessage:Deleted{Information_Object_Name}"/>
      </wsdl:message>

      <wsdl:message name="ResponseMessage">
         <wsdl:part name="ResponseMessage"
element="infoMessage:{Information_Object_Name}ResponseMessage"/>
      </wsdl:message>

      <wsdl:message name="FaultMessage">
         <wsdl:part name="FaultMessage"
element="infoMessage:{Information_Object_Name}FaultMessage"/>
      </wsdl:message>

      <!-- Port Definitions -->
      <wsdl:portType name="{Information_Object_Name}_Port">

         <wsdl:operation name="Created{Information_Object_Name}">
            <wsdl:input name="Created{Information_Object_Name}Event"
message="tns:Created{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Created{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Created{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Changed{Information_Object_Name}">
            <wsdl:input name="Changed{Information_Object_Name}Event"
message="tns:Changed{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Changed{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Changed{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Canceled{Information_Object_Name}">
            <wsdl:input name="Canceled{Information_Object_Name}Event"
message="tns:Canceled{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Canceled{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Canceled{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Closed{Information_Object_Name}">
            <wsdl:input name="Closed{Information_Object_Name}Event"
message="tns:Closed{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Closed{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Closed{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Deleted{Information_Object_Name}">
            <wsdl:input name="Deleted{Information_Object_Name}Event"
message="tns:Deleted{Information_Object_Name}EventMessage"/>
            <wsdl:output name="Deleted{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Deleted{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

      </wsdl:portType>

      <wsdl:binding name="{Information_Object_Name}_Binding"
type="tns:{Information_Object_Name}_Port">
```

```
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

        <wsdl:operation name="Created{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Created{Information_Objec
ct_Name}" style="document"/>
           <wsdl:input name="Created{Information_Object_Name}Event">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Created{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Created{Information_Object_Name}Fault">
              <soap:fault name="Created{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Changed{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Changed{Information_Objec
ct_Name}" style="document"/>
           <wsdl:input name="Changed{Information_Object_Name}Event">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Changed{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Changed{Information_Object_Name}Fault">
              <soap:fault name="Changed{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Canceled{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Canceled{Information_Obj
ect_Name}" style="document"/>
           <wsdl:input name="Canceled{Information_Object_Name}Event">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Canceled{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Canceled{Information_Object_Name}Fault">
              <soap:fault name="Canceled{Information_Object_Name}Fault"
use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Closed{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Closed{Information_Objec
t_Name}" style="document"/>
           <wsdl:input name="Closed{Information_Object_Name}Event">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Closed{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Closed{Information_Object_Name}Fault">
              <soap:fault name="Closed{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="Deleted{Information_Object_Name}">
           <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Deleted{Information_Objec
ct_Name}" style="document"/>
           <wsdl:input name="Deleted{Information_Object_Name}Event">
              <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output name="Deleted{Information_Object_Name}Response">
              <soap:body use="literal"/>
           </wsdl:output>
           <wsdl:fault name="Deleted{Information_Object_Name}Fault">
              <soap:fault name="Deleted{Information_Object_Name}Fault" use="literal"/>
           </wsdl:fault>
        </wsdl:operation>
     </wsdl:binding>

     <wsdl:service name="{Send | Receive | Reply}{Information_Object_Name}">
```

```
        <wsdl:port name="{Information_Object_Name}_Port"
binding="tns:{Information_Object_Name}_Binding">
           <soap:address location="http://iec.ch/TC57/2011/{Send | Receive |
Reply}{Information_Object_Name}"/>
        </wsdl:port>
     </wsdl:service>

</wsdl:definitions>
```

NOTE {Send | Receive | Reply} should be replaced with a proper service patter name such as Receive.

WSDL pour Request, Execute

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
     name="{Request | Execute}{Information_Object_Name}"
        targetNamespace="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"
        xmlns:tns="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:wsi="http://ws-i.org/schemas/conformanceClaim/"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:infoMessage="http://iec.ch/TC57/2011/{Information_Object_Name}Message">

     <wsdl:types>

        <xs:schema targetNamespace="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"
                   elementFormDefault="qualified">

<xs:import namespace="http://iec.ch/TC57/2011/{Information_Object_Name}Message"
schemaLocation="xsd/{Information_Object_Name}Message.xsd"/>

        </xs:schema>

     </wsdl:types>
     <xs:schema
targetNamespace="http://iec.ch/TC57/2011/ExecuteEndDeviceControlsMessage"
                   elementFormDefault="qualified">
<xs:import namespace="http://iec.ch/TC57/2011/EndDeviceControlsMessage"
schemaLocation="xsd/EndDeviceControlsMessage.xsd"/>
<!--<xs:include schemaLocation="xsd/EndDeviceControlsMessage.xsd"/>-->

        </xs:schema>

     <!-- Message Definitions -->

     <wsdl:message name="Create{Information_Object_Name}RequestMessage">
        <wsdl:part name="Create{Information_Object_Name}RequestMessage"
element="infoMessage:Create{Information_Object_Name}"/>
     </wsdl:message>

     <wsdl:message name="Change{Information_Object_Name}RequestMessage">
        <wsdl:part name="Change{Information_Object_Name}RequestMessage"
element="infoMessage:Change{Information_Object_Name}"/>
     </wsdl:message>

     <wsdl:message name="Close{Information_Object_Name}RequestMessage">
        <wsdl:part name="Close{Information_Object_Name}RequestMessage"
element="infoMessage:Close{Information_Object_Name}"/>
     </wsdl:message>

     <wsdl:message name="Cancel{Information_Object_Name}RequestMessage">
        <wsdl:part name="Cancel{Information_Object_Name}RequestMessage"
element="infoMessage:Cancel{Information_Object_Name}"/>
     </wsdl:message>

     <wsdl:message name="Delete{Information_Object_Name}RequestMessage">
        <wsdl:part name="Delete{Information_Object_Name}RequestMessage"
element="infoMessage:Delete{Information_Object_Name}"/>
```

```
      </wsdl:message>

      <wsdl:message name="ResponseMessage">
         <wsdl:part name="ResponseMessage"
element="infoMessage:{Information_Object_Name}ResponseMessage"/>
      </wsdl:message>

      <wsdl:message name="FaultMessage">
         <wsdl:part name="FaultMessage"
element="infoMessage:{Information_Object_Name}FaultMessage"/>
      </wsdl:message>

      <!-- Port Definitions -->
      <wsdl:portType name="{Information_Object_Name}_Port">

         <wsdl:operation name="Create{Information_Object_Name}">
            <wsdl:input name="Create{Information_Object_Name}Request"
message="tns:Create{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Create{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Create{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Change{Information_Object_Name}">
            <wsdl:input name="Change{Information_Object_Name}Request"
message="tns:Change{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Change{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Change{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Cancel{Information_Object_Name}">
            <wsdl:input name="Cancel{Information_Object_Name}Request"
message="tns:Cancel{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Cancel{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Cancel{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Close{Information_Object_Name}">
            <wsdl:input name="Close{Information_Object_Name}Request"
message="tns:Close{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Close{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Close{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

         <wsdl:operation name="Delete{Information_Object_Name}">
            <wsdl:input name="Delete{Information_Object_Name}Request"
message="tns:Delete{Information_Object_Name}RequestMessage"/>
            <wsdl:output name="Delete{Information_Object_Name}Response"
message="tns:ResponseMessage"/>
            <wsdl:fault name="Delete{Information_Object_Name}Fault"
message="tns:FaultMessage"/>
         </wsdl:operation>

      </wsdl:portType>

      <wsdl:binding name="{Information_Object_Name}_Binding"
type="tns:{Information_Object_Name}_Port">

         <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

         <wsdl:operation name="Create{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Create{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Create{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Create{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
```

```
            <wsdl:fault name="Create{Information_Object_Name}Fault">
               <soap:fault name="Create{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Change{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Change{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Change{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Change{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Change{Information_Object_Name}Fault">
               <soap:fault name="Change{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Cancel{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Cancel{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Cancel{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Cancel{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Cancel{Information_Object_Name}Fault">
               <soap:fault name="Cancel{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Close{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Close{Information_Object
_Name}" style="document"/>
            <wsdl:input name="Close{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Close{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Close{Information_Object_Name}Fault">
               <soap:fault name="Close{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
         <wsdl:operation name="Delete{Information_Object_Name}">
            <soap:operation
soapAction="http://iec.ch/TC57/2011/{Information_Object_Name}/Delete{Information_Objec
t_Name}" style="document"/>
            <wsdl:input name="Delete{Information_Object_Name}Request">
               <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="Delete{Information_Object_Name}Response">
               <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="Delete{Information_Object_Name}Fault">
               <soap:fault name="Delete{Information_Object_Name}Fault" use="literal"/>
            </wsdl:fault>
         </wsdl:operation>
      </wsdl:binding>

      <wsdl:service name="{Request | Execute}{Information_Object_Name}">
         <wsdl:port name="{Information_Object_Name}_Port"
binding="tns:{Information_Object_Name}_Binding">
            <soap:address location="http://iec.ch/TC57/2011/{Request |
Execute}{Information_Object_Name}"/>
         </wsdl:port>
      </wsdl:service>

</wsdl:definitions>
```

NOTE 1 Il convient de remplacer {Request | Execute} par un nom de modèle de service correct, tel que Execute.

NOTE 2 La chaîne suivante est à remplacer dans tous les modèles.

{Information_Object_Name}

Remplacé par le profil CIM en cours d'utilisation, par exemple, EndDeviceControls. La forme plurielle de l'objet d'informations est utilisée pour éviter les collisions dans le XSD.

Annexe D
(normative)

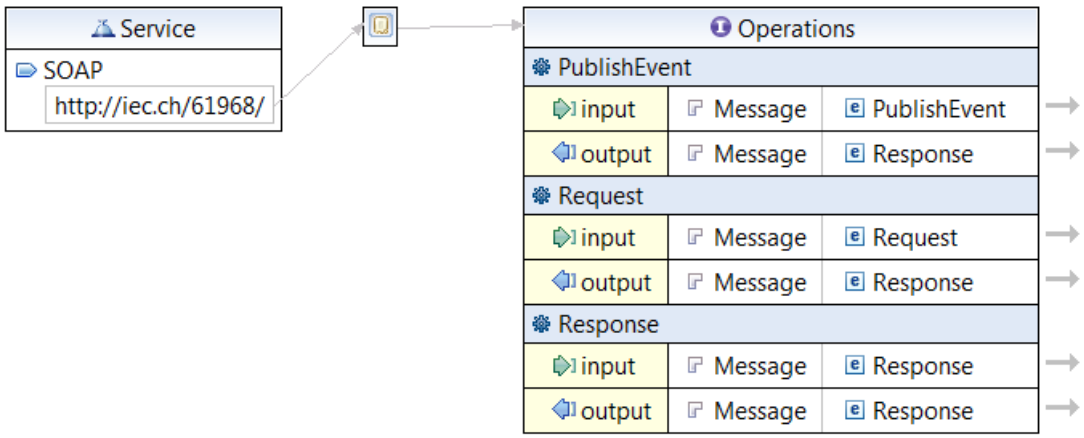
WSDL générique

Le but de cette annexe est de décrire un WSDL générique qui ne soit pas fortement typé à une combinaison verbe et noun spécifique. A la place, ce WSDL offre la capacité de transmettre des messages qui peuvent utiliser n'importe quel verbe/nom et combinaison de nom valide sans apporter de modification à l'enveloppe de message commune définie par Message.xsd. Le WSDL générique effectue trois opérations:

- Request: pour émettre des demandes, auxquelles une réponse peut être renvoyée
- Response: pour émettre des réponses asynchrones
- PublishEvent: pour envoyer des messages d'évènement. On considère qu'un intermédiaire est responsable de la publication de l'évènement pour tous les "écouteurs" potentiellement intéressés.

Cette approche a l'avantage de s'affranchir du besoin de construire des variations de Message.xsd.

La Figure D.1 fournit un aperçu des opérations et messages.



IEC 1826/13

Légende

Anglais	Français
Operations	Opérations
Input	Entrée
Output	Sortie
PublishEvent	Publier événement
Request	Demande
Response	Réponse

Figure D.1 – Structure WSDL générique

Ce qui suit est un XML qui définit le WSDL abstrait pour la mise en œuvre d'un service Web générique CEI 61968-100.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- IEC 61968 WSDL for Generic, Type-Independent Web Services -->
<!-- Uses document wrapped WSDL style -->
```

```
<wsdl:definitions xmlns:ns="http://iec.ch/TC57/2011/abstract"
xmlns:ns2="http://iec.ch/TC57/2011/schema/message"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://iec.ch/TC57/2011/abstract">
   <wsdl:types>
      <xsd:schema targetNamespace="http://iec.ch/TC57/2011/schema/message">
         <xsd:include schemaLocation="xsd/Message.xsd"/>
         <xsd:element name="PublishEvent" type="ns2:EventMessageType"/>
         <xsd:element name="Request" type="ns2:RequestMessageType"/>
         <xsd:element name="Response" type="ns2:ResponseMessageType"/>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="EventMessage">
      <wsdl:part name="Message" element="ns2:EventMessage"/>
   </wsdl:message>
   <wsdl:message name="RequestMessage">
      <wsdl:part name="Message" element="ns2:RequestMessage"/>
   </wsdl:message>
   <wsdl:message name="ResponseMessage">
      <wsdl:part name="Message" element="ns2:ResponseMessage"/>
   </wsdl:message>
   <wsdl:portType name="Operations">
      <wsdl:operation name="PublishEvent">
         <wsdl:input message="ns:EventMessage"/>
         <wsdl:output message="ns:ResponseMessage"/>
      </wsdl:operation>
      <wsdl:operation name="Request">
         <wsdl:input message="ns:RequestMessage"/>
         <wsdl:output message="ns:ResponseMessage"/>
      </wsdl:operation>
      <wsdl:operation name="Response">
         <wsdl:input message="ns:ResponseMessage"/>
         <wsdl:output message="ns:ResponseMessage"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="SOAP" type="ns:Operations">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
<!-- Operation for publication of events -->
      <wsdl:operation name="PublishEvent">
         <soap:operation soapAction="http://iec.ch/61968/PublishEvent"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
<!-- Operation for request/reply interactions -->
      <wsdl:operation name="Request">
         <soap:operation soapAction="http://iec.ch/61968/Request"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
<!-- Operation for asynchronous responses -->
      <wsdl:operation name="Response">
         <soap:operation soapAction="http://iec.ch/61968/Response"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="Service">
      <wsdl:port name="SOAP" binding="ns:SOAP">
         <soap:address location="http://iec.ch/61968/"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Annexe E
(informative)

AMQP

Le protocole AMQP (Advanced Message Queueing Protocol[3]) définit un protocole à "câble" ouvert pour les messageries basées sur les files d'attente. Les produits qui utilisent le protocole AMQP sont de plus en plus fréquents sur le marché, tout comme la disponibilité des offres en source libre. Cela est contrasté par, mais également complémentaire du fait que JMS fournit une API normalisée, mais dont les mises en œuvre ne disposent pas d'un protocole de câble normalisé.

L'utilisation d'AMQP avec la norme CEI 61968-100 est identique à l'utilisation de JMS avec des files d'attente dans les cas où les clients utilisent l'API JMS. Du point de vue du code d'application du client, l'utilisation d'AMQP peut être complètement transparente. Les données d'application transmises dans les messages sont simplement transmises au moyen de l'enveloppe de message commune telle qu'elle est définie par Message.xsd. La même approche générale peut être entreprise avec d'autres API AMQP.

---

3 Plus d'informations sur AMQP sont disponibles à l'addresse suivante: http://amqp.org.

Annexe F
(informative)

Exemple de compression de charge utile

Le but de cette annexe est de fournir un exemple du code requis pour compresser et encoder une charge utile, qui est ensuite filtrée comme contenu de l'élément Payload/Compressed. La compression de charge utile peut être utilisée pour toute technologie de messagerie, y compris JMS, des services Web génériques et des services Web fortement typés.

Il est présenté ci-dessous un exemple de classe Java qui tire avantage des classes communément utilisées pour la compression et l'encodage en base64.

```
package soap.test;

import java.io.*;
import java.util.zip.*;
import org.apache.commons.codec.binary.Base64;

publicclass CompressionClientCompressandEncode{
     privatebyte[] input = null;

     public CompressionClientCompressandEncode(String xmlInput) {
       this.input = xmlInput.getBytes();
     }

     public CompressionClientCompressandEncode(byte[] input) {
       this.input = input;
     }

     // Returns the Compressed and Encoded byte[]

     privatebyte[] compressAndEncode() throws Exception {

       // GZIP the contents..
         ByteArrayOutputStream outstream =
           new ByteArrayOutputStream();
         GZIPOutputStream gzipOutstream =
           new GZIPOutputStream(outstream);
         gzipOutstream.write(input);
         gzipOutstream.close();

         // Encode the compressed byte array from the stream
         byte[] b =
           Base64.encodeBase64(outstream.toByteArray());
         return b;
     }
}
```

Le programme suivant (lorsqu'il est combiné à la classe ci-dessus) montre un exemple d'utilisation dans lequel le XML d'entrée est zippé et encodé au moyen du code ci-dessus, puis décodé et dézippé pour laisser place à l'entrée originale:

```
publicstaticvoid main ( String args[]) {
  String s = "<root>" +
             "<name>raju</name>" +
             "</root>";
```

```
        System.out.println("Original Xml");
        System.out.println(s);

        byte[] b = s.getBytes();

        CompressionClientCompressandEncode cc =
            new CompressionClientCompressandEncode(b);

        try {

              byte[] compressedAndEncoded = cc.compressAndEncode();

                      System.out.println("Compressed And Encoded");
                      System.out.println(
                        new String(compressedAndEncoded));

                  // Validate ..
                  byte[] unencoded =
                        Base64.decodeBase64(compressedAndEncoded);

                  ByteArrayInputStream bi1 =
                        new ByteArrayInputStream(unencoded);
                  GZIPInputStream g1 = new GZIPInputStream(bi1);


                  System.out.println("Unencoded and Uncompressed..");
                  for (int c = g1.read(); c != -1; c = g1.read()) {
                      System.out.write(c);
                  }
                  System.out.flush();

              } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
    }
```

Il est important de noter que si la compression est utilisée, soit les systèmes source et cible doivent tous deux prendre en charge la compression, soit les déséquilibres doivent être traités par des processus intermédiaires au sein de l'ESB. Il convient d'utiliser l'algorithme de compression Gzip, comme le montre cet exemple.

Annexe G
(informative)

XMPP

Le Protocole XMPP (Extensible Messaging and Presence Protocol) définit un protocole au niveau application pour les communications quasi en temps réel au moyen de XML. XMPP est normalisé par les documents IETF RFC 6120, 6121 et 6122. Il existe plusieurs fonctionnalités similaires à celles de JMS; il est donc possible de réaliser une mise en correspondance de la CEI 61968-100 avec XMPP.

A l'aide de XMPP, les clients se connectent à un serveur XMPP tout comme les clients JMS se connecteraient à un serveur JMS. Les messages sont envoyés au moyen de ''séquences'' (stanzas) XML, qui acheminent chacune un élément XML qui constitue logiquement un fragment de document XML représentatif d'une session. Il y a trois types fondamentaux de séquences:

- <message> – utilisé pour l'insertion de messages
- <iq> – Info/Query – (Information/requête), utilisé pour les modèles demande/réponse, lorsqu'une séquence IQ peut se présenter sous l'un des quatre types: get, set, result ou error
- <presence> – (présence) utilisé pour transmettre l'état d'un contact.

L'enveloppe de message conforme CEI 61968-100 peut être placée à l'intérieur d'une séquence XMPP. Le type de séquence qu'il convient d'utiliser est fonction du modèle de messagerie. Il convient que les modèles demande/réponse utilisent le type de séquence <iq> et que les modèles de messagerie publication/abonnement utilisent le type de séquence <message>. Il est présenté ci-dessous un exemple de message d'événement conforme CEI 61968-9 transmis dans une séquence XMPP.

```
<message from='meter76943@myUtility.com' to='headend@myUtility.com'>
        <ns0:Message xmlns:ns0="http://www.iec.ch/TC57/2011/schema/message">
        <ns0:Header>
        <ns0:Verb>created</ns0:Verb>
        <ns0:Noun>EndDeviceEvents</ns0:Noun>
        <ns0:Revision>1</ns0:Revision>
        <ns0:Timestamp>2009-11-04T18:52:50.001-05:00</ns0:Timestamp>
        <ns0:Source>Metering System</ns0:Source>
        </ns0:Header>
        <ns0:Payload>
        <ns1:EndDeviceEvents xmlns:ns1="http://iec.ch/TC57/2011EndDeviceEvents#">
        <ns1:EndDeviceEvent ref='3.26.1.185'>
        <ns1:mRID>76943</ns1:mRID>
        <ns1:createdDateTime>2009-11-04T18:52:50.001-05:00</ns1:createdDateTime>
        <ns1:description>Power off alarm</ns1:description>
        <ns1:severity>1</ns1:severity>
        <ns1:Assets><ns1:mRID>AC761473800C7B0417481114A11348C16111911121B46C016BF012C6812110
        6</ns1:mRID>
        </ns1:Assets>
        </ns1:EndDeviceEvent>
        </ns1:EndDeviceEvents>
        </ns0:Payload>
        </ns0:Message>
</message>
```

Les messages XMPP sont traités au moyen des attributs ''from'' et ''to'' au sein des éléments <message> et <iq>. Lorsque les noms de thème ou file d'attente JMS sont utilisés pour le traitement avec JMS, le traitement avec XMPP est défini par l'IETF RFC 6122.

Bibliographie

IEC 61968-9, *Application integration at electric utilities – System interfaces for distribution management – Part 9: Interfaces for meter reading and control* (disponible en anglais seulement)

IEC 61968-13, *Application integration at electric utilities – System interfaces for distribution management – Part 13: CIM RDF Model exchange format for distribution* (disponible en anglais seulement)

IEC 61970-452, *Energy management system application program interface (EMS-API) – Part 452: CIM Statis Transmission Network Model Profiles* (disponible en anglais seulement)

IEC 61970-453[4], *Energy management system application program interface (EMS-API) – Part 453: Diagram Layout Profile* (disponible en anglais seulement)

IEC 62361-100: *Harmonization of Quality Codes across TC 57 – Part 100: Naming and design rules for CIM profiles to XML schema mapping*[5] (disponible en anglais seulement)

EIP: Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison Wesley,October 2003; ISBN-10: 0321200683, ISBN-13: 978-0321200686 (http://www.eaipatterns.com).

SOAP sur Java Message Service – Recommandation proposée. Disponible à l'adresse *http://www.w3.org/TR/soapjms/*

XSL: Extensible Stylesheet Language (XSL).Disponible à l'adresse www.w3.org/TR/xsl/.

XPATH: XML Path Language (XPATH). Disponible à l'adresse *www.w3.org/TR/xpath/*.

IETF RFC 1738, *Univeral Resource Locators* (disponible en anglais seulement)

IETF RFC 2616, *Hyper-Text Transport Protocol 1.1* (disponible en anglais seulement)

IETF RFC 4122, *A Universally Unique Identifier (UUID) URN Namespace* (disponible en anglais seulement)

Extensible Markup Language (XML): *http://www.w3.org/TR/REC-xml* (disponible en anglais seulement)

XML Schema: *http://www.w3.org/XML/Schema* (disponible en anglais seulement)

Simple Object Access Protocol (SOAP) 1.2: *http://www.w3.org/TR/soap12-part1/* (disponible en anglais seulement)

WS-I Basic Profile Version 1.0: *http://www.oasis.org* (disponible en anglais seulement)

JMS API: JSR-914, *Java Message Service (JMS) API* (disponible en anglais seulement)

---

4 A publier.

5 A publier.

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch