SLOVENSKI STANDARD
oSIST prEN 15531-2:2021

01-september-2021

Javni prevoz - Vmesnik za informiranje v realnem času za potrebe delovanja javnega prevoza - 2. del: Komunikacijska infrastruktura

Public transport - Service interface for real-time information relating to public transport operations - Part 2: Communications infrastructure

Öffentlicher Verkehr - Serviceschnittstelle für Echtzeitinformationen bezogen auf Operationen im öffentlichen Verkehr - Teil 2: Kommunikationsstruktur

Transport public - Interface de service pour les informations en temps réel relatives aux opérations de transport public - Partie 2 : Communications

Ta slovenski standard je istoveten z: **prEN 15531-2**

ICS:

35.240.60	Uporabniške rešitve IT v prometu	IT applications in transport

oSIST prEN 15531-2:2021 **en,fr,de**

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

DRAFT prEN 15531-2

July 2021

ICS 35.240.60

Will supersede EN 15531-2:2015

English Version

Public transport - Service interface for real-time information relating to public transport operations - Part 2: Communications infrastructure

Transport public - Interface de service pour les informations en temps réel relatives aux opérations de transport public - Partie 2 : Infrastructure des communications

Öffentlicher Verkehr - Serviceschnittstelle für Echtzeitinformationen bezogen auf Operationen im öffentlichen Verkehr - Teil 2: Kommunikationsstruktur

This draft European Standard is submitted to CEN members for enquiry. It has been drawn up by the Technical Committee CEN/TC 278.

If this draft becomes a European Standard, CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

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

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

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Warning : This document is not a European Standard. It is distributed for review and comments. It is subject to change without notice and shall not be referred to as a European Standard.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Contents

Page

European foreword

This document (prEN 15531-2:2021) has been prepared by Technical Committee CEN/TC 278 “Intelligent transport systems”, the secretariat of which is held by NEN.

This document is currently submitted to the CEN Enquiry.

This document will supersede CEN/TS 15531-2:2015.

SIRI (CEN/TS 15531-2:2006) has been a CEN Technical Specification since 2007 and a European normative standard since 2013 and has been widely used in Europe and elsewhere and proven its usefulness. This document proposes a revised version of SIRI as a European Standard, and is currently submitted to the Formal Vote. The proposed revisions are minor enhancements arising from experience of the deployment of SIRI in many live systems. This document also clarifies the relationship of SIRI to NeTEx, the CEN Technical Standard for the XML exchange of Public Transport Reference data based on the Transmodel CEN European Standard.

This document presents Part 2 of the European Standard known as “SIRI”. SIRI provides a framework for specifying communications and data exchange protocols for organisations wishing to exchange Real-time Information (RTI) relating to public transport operations.

The SIRI European Standard is presented in three parts:

- context and framework, including background, scope and role, normative references, terms and definitions, symbols and abbreviations, business context and use cases (Part 1),
- the mechanisms to be adopted for data exchange communications links (Part 2),
- data structures for a series of individual application interface modules PT, ET, ST, SM, VM, CT, CM, GM (Part 3).

Two additional parts define additional functional services as CEN Technical Specifications:

- additional data structures for additional application interface module FM (Part 4),
- additional data structures for additional application interface module SX (Part 5).

The XML schema can be downloaded from https://github.com/SIRI-CEN/SIRI, guidance on its use, example XML files, and case studies of national and local deployments is located at http://siri-cen.eu/.

It is recognised that SIRI is not complete as it stands, and from time to time will need to continue to be enhanced to add additional capabilities. It is therefore intended that a SIRI Management Group should continue to exist, at European level, based on the composition of SG7.

Introduction

Public transport services rely increasingly on information systems to ensure reliable, efficient operation and widely accessible, accurate passenger information. These systems are used for a range of specific purposes: setting schedules and timetables; managing vehicle fleets; issuing tickets and receipts; providing real-time information on service running, and so on.

This document specifies a Service Interface for Real-time Information (SIRI) about Public Transport. It is intended to be used to exchange information between servers containing real-time public transport vehicle or journey time data. These include the control centres of transport operators and information systems that utilise real-time vehicle information, for example, to deliver services such as travel information.

Well-defined, open interfaces have a crucial role in improving the economic and technical viability of Public Transport Information Systems of all kinds. Using standardised interfaces, systems can be implemented as discrete pluggable modules that can be chosen from a wide variety of suppliers in a competitive market, rather than as monolithic proprietary systems from a single supplier. Interfaces also allow the systematic automated testing of each functional module, vital for managing the complexity of increasing large and dynamic systems. Furthermore, individual functional modules can be replaced or evolved, without unexpected breakages of obscurely dependent function.

This document will improve a number of features of public transport information and service management:

- Interoperability – the document will facilitate interoperability between information processing systems of the transport operators by: (i) introducing common architectures for message exchange; (ii) introducing a modular set of compatible information services for real-time vehicle information; (iii) using common data models and schemas for the messages exchanged for each service; and (iv) introducing a consistent approach to data management.

- Improved operations management – the document will assist in better vehicle management by (i) allowing the precise tracking of both local and roaming vehicles; (ii) providing data that can be used to improve performance, such as the measurement of schedule adherence; and (iii) allowing the distribution of schedule updates and other messages in real-time.

- Delivery of real-time information to end-users – the document will assist the economic provision of improved data by; (i) enabling the gathering and exchange of real-time data between AVMS systems; (ii) providing standardised, well defined interfaces that can be used to deliver data to a wide variety of distribution channels. Version 2.0 of SIRI includes a new Simple Web Service designed to support the widespread, massively scalable use of mobile devices and web browsers and other applications to display public transport data directly to users.

Technical advantages include the following:

- Reusing a common communication layer for all the various technical services enables cost-effective implementations and makes the document readily extensible in future.

History

Version 1.0 of SIRI was developed in 2004-2005 and submitted to vote, eventually passing through the CEN process to become an approved CEN Technical Specification in 2007. As well as the normative Version 1.0 XSD schema, successive informal working versions of the schema (v 1.1 – 1.4) were released to allow for fixes and to implement some very minor enhancements agreed by the working group. A WSDL version was also developed.

Version 2.0 of SIRI was developed in 2012 to coincide with making the SIRI standard a full CEN norm.

SIRI includes a Simple Web Services “SIRI-LITE” as an additional transport method and a WSDL document literal version and a WSDL2 version;

Version 2.1 of SIRI was developed in 2020/21 to address lessons from the now widespread implementation of SIRI.

The changes in SIRI version 2.1 include:

- remove the direct relationship with TPEG and other standards to enable support as the other standards change;
- support for new modes in line with TRANSMODEL and NeTEx;
- support for the Reason / Effect / Advice structure for disruptions in SIRI SX;
- increased granularity for occupancy data and Vehicle structures;
- improved subscription renewal options and filtering options;
- additional options and flexibility for STOP POINTS and relationships between journeys;
- migration of XSD to Github to improve access and change control processes.

Compatibility with previous versions

All changes in version 2.1 are intended to be fully backwards compatible, that is to say, existing documents that validate against earlier versions of the schema will also validate against the 2.1 schema without alteration (other than to schema version numbers), and version 2.1 documents that do not use new features will validate against earlier versions. Version 2.1 documents that use new features will not be backwards compatible.

1 Scope

SIRI uses a consistent set of general communication protocols to exchange information between client and server. The same pattern of message exchange may be used to implement different specific functional interfaces as sets of concrete message content types.

Two well-known specific patterns of client server interaction are used for data exchange in SIRI: *Request/Response* and *Publish/Subscribe*.

- *Request/Response* allows for the ad hoc exchange of data on demand from the client.

- *Publish/Subscribe* allows for the repeated asynchronous push of notifications and data to distribute events and Situations detected by a Real-time Service.

The use of the *Publish/Subscribe* pattern of interaction follows that described in the Publish-Subscribe Notification for Web Services (WS-PubSub) specification, and as far as possible, SIRI uses the same separation of concerns and common terminology for publish/subscribe concepts and interfaces as used in WS-PubSub. WS-PubSub breaks down the server part of the *Publish/Subscribe* pattern into a number of separate named roles and interfaces (for example, Subscriber, Publisher, Notification Producer, and Notification Consumer): in an actual SIRI implementation, certain of these distinct interfaces may be combined and provided by a single entity. Although SIRI is not currently implemented as a full WS-PubSub web service, the use of a WS-PubSub architecture makes this straightforward to do in future.

Publish/Subscribe will not normally be used to support large numbers of end user devices.

For the delivery of data in responses (to both requests and subscriptions), SIRI supports two common patterns of message exchange, as realised in existent national systems:

- A one step 'Direct Delivery', as per the classic client-server paradigm, and normal WS-PubSub publish subscribe usage; and;

- A two-step 'Fetched Delivery' which elaborates the delivery of messages into a sequence of successive messages pairs to first notify the client, and then to send the data when the client is ready. Fetched Delivery is a stateful pattern in its own right.

- Each delivery pattern allows different trade-offs for implementation efficiency to be made as appropriate for different target environments.

- A SIRI implementation may support either or both delivery methods; in order to make the most efficient use of the available computational and communication resources. The delivery method may either be preconfigured and static for a given implementation, or each request or subscription may indicate the delivery method required by the client dynamically as part of the request policy, and the server may refuse a request if it does not support that method, giving an appropriate error code.

- The Interaction patterns and the Delivery patterns are independent aspects of the SIRI protocol and may be used in any combination in different implementations.

- For a given SIRI Functional Service type (Connection Monitoring, Stop Monitoring etc.), the message payload content is the same regardless of whether information is exchanged with a *Request/Response* or *Publish/Subscribe* pattern, or whether it is returned by Direct or Fetched Delivery.

- The SIRI *Publish/Subscribe* Protocol prescribes particular *mediation* behaviour for reducing the number of notifications and the amount of network traffic arising from subscriptions.

- The mediation groups the various subscriptions from a subscriber into one or more Subscriber Channels, and is able to manage notifications and updates for the aggregate.
- Only partial updates to the data set since the last delivery for the subscription need to be sent.
- The SIRI Communication protocols are designed to fail gracefully. Considerations for resilience and recovery are covered below.

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.

EN 15531-1, *Public transport - Service interface for real-time information relating to public transport operations - Part 1: Context and framework*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in EN 15531-1 apply.

4 Symbols and abbreviations

For the purposes of this document, the symbols and abbreviations given in EN 15531-1 apply.

5 Common communication aspects

5.1 Data Exchange Patterns of Interaction

5.1.1 Introduction

There are two main patterns of interaction for Data Exchange in SIRI: *Request/Response* and *Publish/Subscribe*. The patterns are complementary, that is an implementation may support both, and implementers may choose the most efficient pattern according to the nature of their application.

NOTE Publish/Subscribe can emulate a Request/Response interaction by use of a short subscription. A partial SIRI implementation that supports only Request/Response is useful for connecting many types of Public Transport Information System applications to AVMS and other Producer System data.

5.1.2 Request/Response Pattern

The *Request/Response* interaction allows for the immediate fulfilment of one-off data supply requests made by a Requestor to a Service. Pairs of *Request/Response* patterns are also used for the interactions that make up other patterns, such as *Publish/Subscribe*.

In the *Request/Response* interaction used to get data, the Client sends a request message to a Server that offers the required SIRI Functional Service, and immediately receives a Delivery message in response (Figure 1). A Data Delivery may be made as a one-step *Direct Delivery*, or as a two-step *Fetched Delivery* (see later).

The Requestor shall give a unique reference to each request, which will be returned in the matching response.

The Requestor expresses its specific interests through Topic and Delivery Policy parameters on the specific SIRI Functional Service Requests. If the request cannot be satisfied an error condition is returned diagnosing the reason.

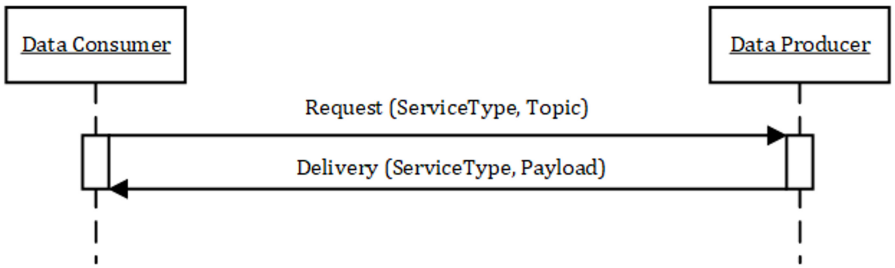

Figure 1 — Request / Response Interaction

Request/response allows for an efficient transmission of data on-demand from the Consumer and is extremely easy to implement using commodity internet software components.

5.1.3 Publish/Subscribe Pattern

The *Publish/Subscribe* interaction (see Figure 2) allows for the asynchronous detection of real-time events by a producer service, whose role is to generate and send notifications to one or more interested consumers.

In the *Publish/Subscribe* interaction, the Subscriber client sends a request message to the Notification Producer of a SIRI Functional Service to create a Subscription, which may or may not be granted. The Subscriber expresses its specific interests through Topic and Subscription Policy parameters, and receives an acknowledgement that this has been created, or an error condition.

Once a Subscription exists, the service, acting as the Notification Producer, uses it to determine when to send a notification to a consumer after a Situation, i.e. event is detected. The incoming event notification to be published is matched against the interests expressed by the Topic and other filter parameters of the Subscription and if satisfied, a notification message is sent to the Consumer. The actual Notification Message Delivery may be made either as a one-step *Direct Delivery* to a Notification Consumer, or as a two-step SIRI *Fetched Delivery*, with separate message pairs first to notify and then to deliver the payload.

In SIRI, the Subscriber and Consumer roles are normally implemented by the same client service, although they are logically separate. Every Consumer must know its Subscriber so that they can interact to handle recovery from service failures.

Subscriptions for different types of SIRI Functional Service are managed separately.

A Subscriber may add different Subscriptions at different times.

A Subscription Request includes an Initial Termination Time indicating the desired duration i.e. lease of the individual Subscription. The subscription will only be granted if this can be met, otherwise an error will be returned.

Subscriptions have a life span as specified by the Subscriber and will be terminated by the Notification Producer service when they reach their expiry time.

Subscribers may terminate their own existing Subscriptions before their predefined expiry time through a Subscription Manager. The Subscription Manager is subordinate to the Notification Producer, and in SIRI implementations, is normally provided by the same entity, although logically distinct. Each Subscription Manager knows its associated Notification Producer, and vice versa. Although the Notification Producer is the factory for creating new subscriptions, it does not manage them once

created; rather this is done by the Subscription Manager. This design (i.e. the Notification Producer finds the Subscription Manager for the Subscriber, rather than the Subscription Manager finds the Notification Producer for the Subscriber) is required to conform to the WS-PubSub architecture. The WS-PubSub architecture allows for additional Subscription management functions to be added through the Subscription Manager for example renewal, pause/resume, or the dynamic tuning of subscription policies, but SIRI does not specify any of these at present. SIRI does however support a Terminate Subscription and a Terminate All Subscriptions function.

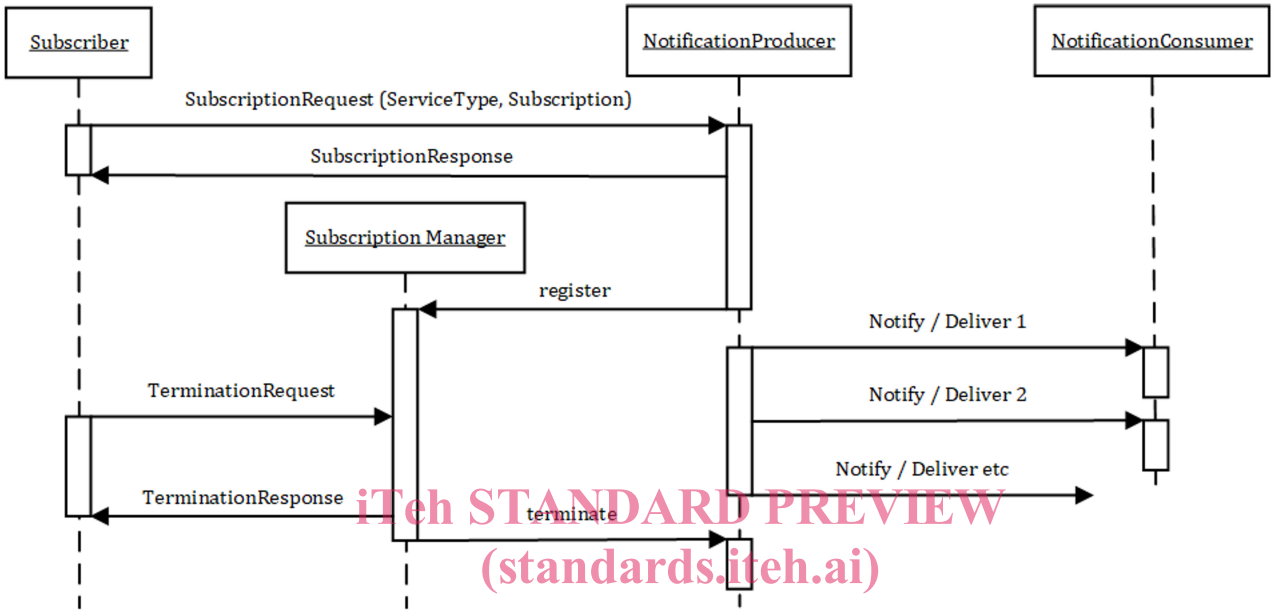

Figure 2 — Simple Publish/Subscribe Interaction

Subscriptions are a stateful resource: they need a unique identifier that can be used by Subscriber, Producer and Consumer to refer to the same subscription on different occasions. They will each hold their own representation of the subscription. In SIRI this identifier is issued by the Subscriber.

Publish/Subscribe allows for an efficient regular event driven exchange of updates to data. It requires a more elaborate implementation, with the holding of state by both participants and the dedication of computing resources to run the notification production.

5.1.4 Publish/Subscribe with Broker Pattern

The WS-PubSub architecture also allows for the logical separation of the concerns of Publishing and Notification Production, and in its fully articulated form, has a separate Publisher role that is a subordinate constituent of the Notification Producer service (see Figure 3). The Publisher produces notifications of any significant situations, i.e. events. For a real-time service, the Publisher monitors the real-time data and if a change has occurred, it produces a notification. The Notification Producer then matches the Notification with the interests and policies expressed by Subscribers and despatches the notification delivery to the Notification Consumer indicated by the Subscription.

It is possible to have more than one Notification Producer sitting between the Publisher and the Consumer, either to carry out successive types of filtering and processing of the notifications, or for scalability. WS-PubSub distinguishes between *direct* notification – where the notification message from the Publisher is delivered unchanged, and *brokered* notification – where the Notification Producer filters and also possibly transforms the message. Both brokered (e.g. for SIRI Stop Monitoring) and unbrokered (e.g. for SIRI General Message) mediation occurs in different SIRI Functional Services.

The separation of concerns between Publisher and Notification producer is transparent to the Subscriber and Consumer, and so in SIRI is merely an implementation choice – which does not currently explicitly mandate any requirements for the interface between the Notification Producer and Publisher. Every Publisher knows its associated Notification Producer(s), and vice versa.

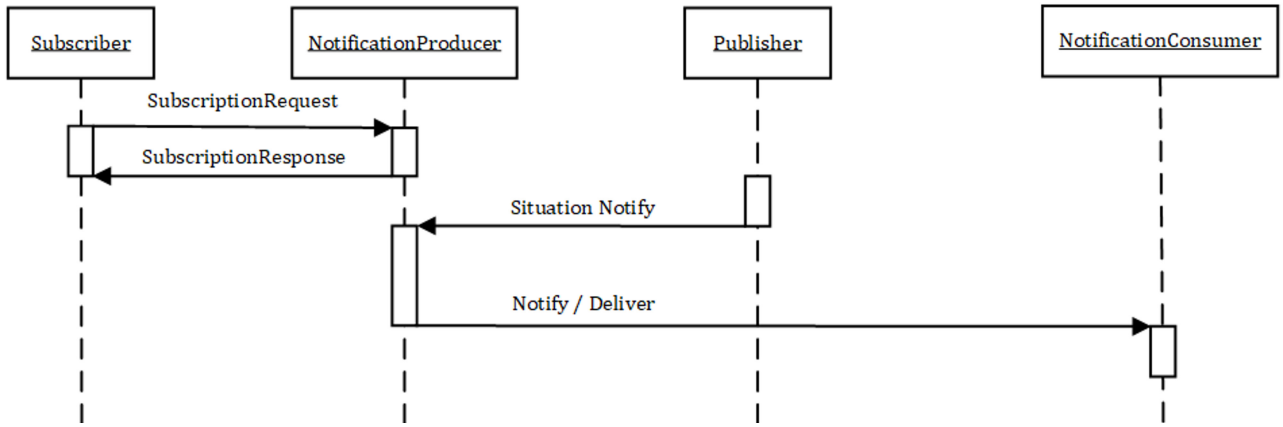

Figure 3 — Brokered Publish/Subscribe Interaction

Further Subscription and Subscriber filtering tasks, in particular the enforcement of SIRI Access controls, are implemented by the Notification Producer, not the Publisher.

5.1.5 Request/Response – Compound Requests

Multiple requests for a single SIRI Functional Service may be included in a single Data *Request/Response* interaction: each request may cover different topics and policies (Figure 4).

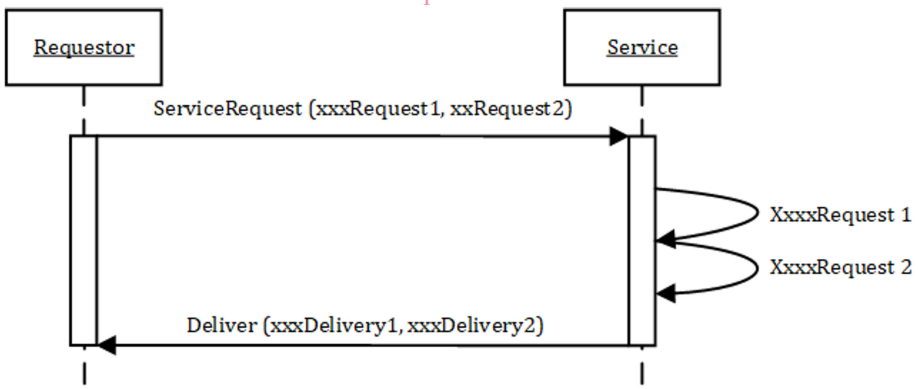

Figure 4 — Request/Response: Compound Requests

5.1.6 Publish/Subscribe – Compound Subscriptions

Multiple subscriptions to a single SIRI Functional Service may be included by a Subscriber in a single Subscription request: each subscription may cover different topics and policies (Figure 5). The handling of notifications and deliveries for compound subscriptions is discussed in the section on Mediation later below.

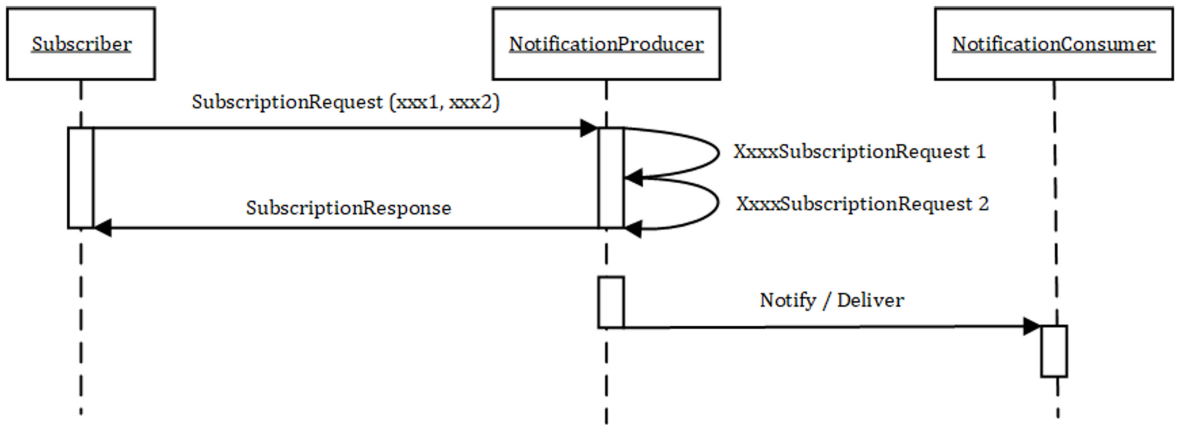

Figure 5 — Publish/Subscribe: Compound Subscriptions

5.2 Delivery Patterns

5.2.1 Introduction

Services return notifications and Situation content to the Consumer using Delivery messages. In real-time applications, it is important to be able to optimise systems to ensure rapid delivery, and SIRI supports two different message pattern variations for making a delivery, that in principle can be used interchangeably: these are; (i) *Direct Delivery*, and; (ii) *Fetched Delivery*.

The choice of delivery patterns may be pre-configured, or if the implementation supports both methods, be specified as a parameter on the request. For systems that support dynamic choice, if the SIRI implementation does not support the requested delivery method for a specific service type, an error message will be returned.

5.2.2 Direct Delivery

In *Direct Delivery,* the payload is sent as the content of a single message to the Consumer Client (Figure 6). For a *Request/Response* this will be the requestor. For a subscription this will be the Notification Consumer as indicated on the Subscription (i.e. the notification and the delivery are the same message.).

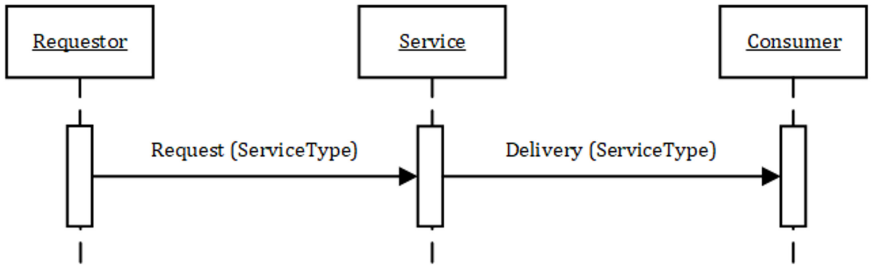

Figure 6 — One Step Direct Delivery

In *Direct Delivery*, the burden of holding and queuing messages is distributed to the client, with some advantages for scaling, as the central server needs neither retain data, nor allocate computation resource to service the additional data supply steps. The interaction is simpler, with fewer messages being exchanged, and a simpler mediation. However, the method does not allow the Consumer to optimise its own activities by separating its processing to detect the existence of an update from its

processing to use the payload data. The full payload is always sent, even if it is not currently of interest to the client. *Direct Delivery* is appropriate for deployment with fast, reliable communications, and with adequate processing capability on the Consumer. It is especially efficient when most updates are relevant to the client and are used immediately.

5.2.3 Fetched Delivery

In *Fetched Delivery*, the delivery is done in two successive steps, separating the notification of an update from the delivery of the data payload (Figure 7). The steps are as follows:

1) The Producer sends a Data Ready Notification message to the Consumer.

2) The Consumer Acknowledges receipt with a Data Ready Response.

3) The Consumer sends a Data Supply Request to the Notification Producer.

4) The Notification Producer responds with a Data Supply Delivery.

The second read is allowed to be destructive, that is, the ability to recreate exactly the same delivery to that point is not guaranteed: the differential update may be deleted once it has been given to the Consumer.

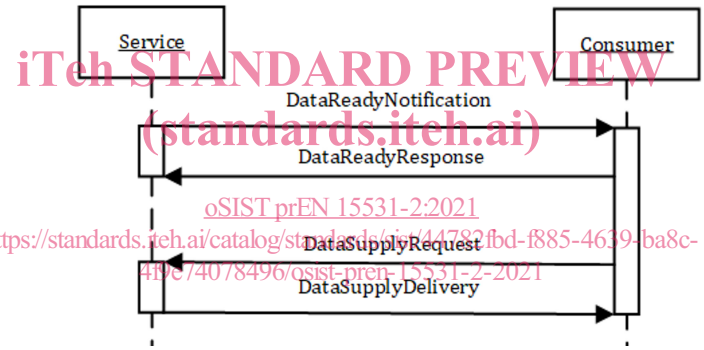

Figure 7 — Fetched Delivery

Fetched Delivery is a stateful pattern of interaction in its own right – requiring the ability of both parties to refer to the data update by a reference that persists for its currency. In this case the reference is issued by the Producer. Whether the identifier needs to be exposed to the Consumer depends on the mediation model (see later): if all updates are aggregated through a single subscriber channel, then there is only one data set to fetch, and an explicit reference is not needed.

Fetched Delivery allows the Consumer to defer the sending of the full payload until it is ready to process it: if in the meantime the Situation has changed further, and a new notification message has arisen, only the latest update need be exchanged. This can give a more efficient use of bandwidth for applications that are bandwidth constrained. In addition, notifications are small messages that can typically be examined using less computational resource than a full delivery message containing a payload, so if the majority of updates are discarded (i.e. never fetched); the processing load on the Consumer may be less. Similarly, the storage requirement on the Consumer to queue small notification messages for *Fetched Delivery* is less than the storage requirement to queue larger payload messages for *Direct Delivery* (but larger on the Notification Producer to retain it until fetched). As a trade-off, there is additional computational and communication overhead required to conduct the extra interactions of fetched data supply messages, and also an additional latency to carry it out (in particular a fourfold communication overhead).