

Public transport — Service interface for real-time information relating to public transport operations —

Part 2: Communications infrastructure

ICS 35.240.60

National foreword

This Draft for Development is the UK implementation of CEN/TS 15531-2:2007.

This publication is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature. It should be applied on this provisional basis, so that information and experience of its practical application can be obtained.

Comments arising from the use of this Draft for Development are requested so that UK experience can be reported to the European organization responsible for its conversion to a European standard. A review of this publication will be initiated not later than 3 years after its publication by the European organization so that a decision can be taken on its status. Notification of the start of the review period will be made in an announcement in the appropriate issue of *Update Standards*.

According to the replies received by the end of the review period, the responsible BSI Committee will decide whether to support the conversion into a European Standard, to extend the life of the Technical Specification or to withdraw it. Comments should be sent to the Secretary of the responsible BSI Technical Committee at British Standards House, 389 Chiswick High Road, London W4 4AL.

The UK participation in its preparation was entrusted to Technical Committee EPL/278, Road transport informatics.

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

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

This Draft for Development was published under the authority of the Standards Policy and Strategy Committee on 31 July 2007

© BSI 2007

ISBN 978 0 580 57213 5

Amendments issued since publication

Amd. No.	Date	Comments

English Version

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

Öffentlicher Verkehr - Dienstleitungsschnittstelle für
zeitnahe Informationen zum Betrieb des öffentlichen
Verkehrs - Teil 2: Kommunikationsinfrastruktur

This Technical Specification (CEN/TS) was approved by CEN on 23 October 2006 for provisional application.

The period of validity of this CEN/TS is limited initially to three years. After two years the members of CEN will be requested to submit their comments, particularly on the question whether the CEN/TS can be converted into a European Standard.

CEN members are required to announce the existence of this CEN/TS in the same way as for an EN and to make the CEN/TS available promptly at national level in an appropriate form. It is permissible to keep conflicting national standards in force (in parallel to the CEN/TS) until the final decision about the possible conversion of the CEN/TS into an EN is reached.

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



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

Management Centre: rue de Stassart, 36 B-1050 Brussels

Contents

Page

Foreword.....	5
Introduction	6
1 Scope	7
2 Normative references	8
3 Terms and definitions	8
4 Symbols and abbreviations	8
5 Common communication aspects	8
5.1 Data Exchange Patterns of Interaction.....	8
5.1.1 General.....	8
5.1.2 Request/Response Pattern	8
5.1.3 Publish/Subscribe Pattern	9
5.1.4 Publish/Subscribe with Broker Pattern	10
5.1.5 Request/Response – Compound Requests	11
5.1.6 Publish/Subscribe – Compound Subscriptions	11
5.2 Delivery Patterns.....	12
5.2.1 General.....	12
5.2.2 Direct Delivery	12
5.2.3 Fetched Delivery	13
5.2.4 Data Horizon for Fetched Delivery	14
5.2.5 Get Current Message.....	15
5.2.6 Multipart Despatch of a Delivery	15
5.2.7 Multipart Despatch of a Fetched Delivery – MoreData.....	15
5.3 Mediation Behaviour	16
5.3.1 General.....	16
5.3.2 Mediation Behaviour – Maintaining Subscription Last Updated State	16
5.3.3 Mediation Behaviour – Subscription Filters	19
5.4 Recovery Considerations for Publish Subscribe	21
5.4.1 General.....	21
5.4.2 Check Status – Polling	22
5.4.3 Heartbeat – Pinging	22
5.4.4 Degrees of Failure.....	22
5.4.5 Detecting a Failure of the Producer	23
5.4.6 Detecting a Failure of the Consumer	24
5.5 Recovery Considerations for Direct Delivery	25
5.6 Request Parameters and Interactions	25
5.7 Error Conditions for Requests	27
5.8 Versioning	29
5.8.1 General.....	29
5.8.2 The Overall SURI Framework Version Level	29
5.8.3 The SURI Functional Service Type Version Level	29
5.9 Access Controls: Security and Authentication	30
5.9.1 General.....	30
5.9.2 System Mechanisms External to SURI Messages	30
5.9.3 Application Access Controls Reflected in SURI Processing.....	30
5.10 Service Discovery	31
5.10.1 General.....	31
5.10.2 Discovery of Servers that Support SURI	31
5.10.3 Discovery of the Capabilities of a SURI Server	31
5.10.4 Discovery of the Coverage of a Given SURI Functional Service.....	31

5.11	Capability Matrix	32
5.11.1	General	32
5.11.2	SIRI General Capabilities	32
6	Request/response	33
6.1	Making a Direct Request	33
6.1.1	General	33
6.1.2	ServiceRequest Message	34
6.1.3	The ServiceRequestContext	35
6.1.4	Common Properties of ServiceRequest Messages	37
6.1.5	ServiceRequest Example	37
6.1.6	Access Controls on a Request	38
6.2	Receiving a Data Delivery	39
6.2.1	General	39
6.2.2	ServiceDelivery Message	40
7	Subscriptions	44
7.1	Setting up Subscriptions	44
7.1.1	General	44
7.1.2	SubscriptionRequest	45
7.1.3	SubscriptionResponse	48
7.2	Subscription Validity	50
7.3	Terminating Subscriptions	50
7.3.1	General	50
7.3.2	The TerminateSubscriptionRequest	50
7.3.3	TerminateSubscriptionResponse	51
8	Delivering data	53
8.1	Direct Delivery	53
8.1.1	Procedure	53
8.1.2	DataReceivedAcknowledgement Message	53
8.1.3	DataReceivedAcknowledgement Example	53
8.2	Fetches Delivery	54
8.2.1	General	54
8.2.2	Signalling Data Availability (DataReadyNotification / DataReadyResponse)	54
8.2.3	Polling Data (DataSupplyRequest/ServiceDelivery)	56
9	Recovery from system failure	57
9.1	General	57
9.2	Recovery after Client Failure	57
9.3	Recovery after Server Failure	57
9.4	Reset after Interruption of Communication	58
9.5	Alive Handling	58
9.5.1	General	58
9.5.2	CheckStatusRequest	58
9.5.3	CheckStatusResponse	59
9.5.4	HeartbeatNotification	61
10	Transport of SIRI messages	62
10.1	Separation of Addressing from Transport Protocol	62
10.2	Logical Endpoint Addresses	62
10.2.1	Endpoint Addresses	62
10.2.2	Endpoint Address Examples	63
10.3	Parallelism and Endpoint Addresses	64
10.4	Encoding of XML messages	65
10.4.1	Principles	65
10.4.2	Encoding of Errors in XML	65
10.4.3	Character Set	65
10.4.4	Schema Packages	65
10.5	Use of SIRI with SOAP	66
10.5.1	General	66
10.5.2	Web Services	66

10.5.3	Use of SOAP	67
10.5.4	SIRI WSDL	68
10.5.5	WSDL Producer Server Operations	69
10.5.6	WSDL Client Operations	70
10.5.7	SIRI WSDL Status	71
11	Capability discovery requests	71
11.1	General.....	71
11.2	Capability Request.....	71
11.3	Service Capability Discovery Request	72
11.4	Service Capability Discovery Response	72
11.5	Service Capability Discovery Response	73
11.5.1	General.....	73
11.5.2	Service Capability Response Example.....	74
11.6	Functional Service Capability Permission Matrix	76
11.6.1	General.....	76
11.6.2	OperatorPermissions	77
11.6.3	LinePermissions	77
11.6.4	ConnectionLinkPermissions	78
11.6.5	StopMonitorPermissions	78
11.6.6	VehicleMonitorPermissions.....	79
11.6.7	InfoChannelPermissions.....	79
12	Shared groups of elements	79
12.1	General.....	79
12.2	FramedVehicleJourneyRef	79
12.3	ServiceInfoGroup.....	80
12.4	VehicleJourneyInfoGroup	80
12.5	JourneyPatternInfoGroup	81
12.6	DisruptionGroup	82
12.7	JourneyProgressGroup	83
12.8	Location	83
12.9	OperationalBlockGroup	84
12.10	OperationalInfoGroup	84
12.11	Error	84
	Bibliography	85

Foreword

This document (CEN/TS 15531-2:2007) has been prepared by Technical Committee CEN/TC 278 “Road transport and traffic telematics”, the secretariat of which is held by NEN.

This presents Part 2 of the European Technical Specification 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.

SIRI 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 (Part 3).

The XML schema can be downloaded from <http://www.siri.org.uk/>, along with available guidance on its use, example XML files, and case studies of national and local deployments.

It is recognised that SIRI is not complete as it stands, and there will be a substantial amount of work required to continue to develop SIRI over the coming years. It is therefore intended that a SIRI Management Group should continue to exist, at European level, based on the composition of SG7.

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to announce this CEN Technical Specification: Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

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 Technical Specification 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 Technical Specification will improve a number of features of public transport information and service management:

- Interoperability – the Technical Specification 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; (ii) 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 Technical Specification 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 Technical Specification will assist the economic provision of improved data by: (i) enabling the gathering and exchange of real-time data between VAMS systems; (ii) providing standardised, well defined interfaces that can be used to deliver data to a wide variety of distribution channels.

Technical advantages include the following:

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

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.

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 be sent.

The SIRI Communication protocols are designed to fail gracefully. Considerations for resilience and recovery are covered below.

2 Normative references

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

CEN/TS 15531-1:2007, *Public transport - Service interface for real-time information relating to public transport operations - Part 1: Context and framework*

ISO/IEC 8859-15, *Information technology - 8-bit single-byte coded graphic character sets - Part 15: Latin alphabet No. 9*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in CEN/TS 15531-1:2007 apply.

4 Symbols and abbreviations

For the purposes of this document, the symbols and abbreviations given in CEN/TS 15531-1:2007 apply.

5 Common communication aspects

5.1 Data Exchange Patterns of Interaction

5.1.1 General

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 implementors may choose the most efficient pattern according to the nature of their application. Note that *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 *Fetches Delivery* (see later).

The Requestor must 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 *Fetches 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 with 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.

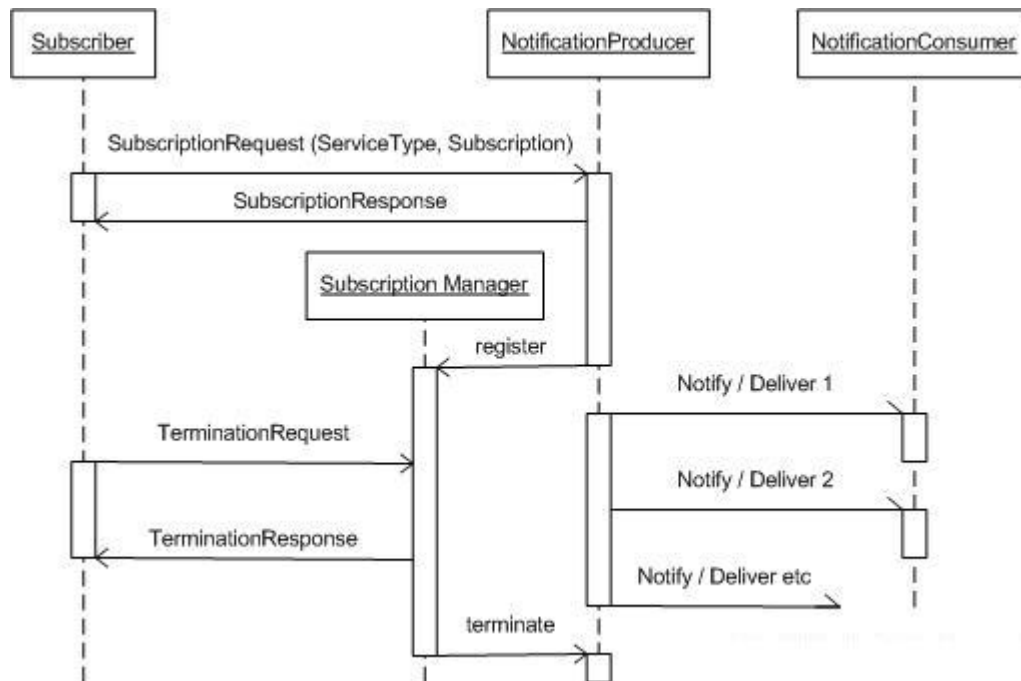


Figure 2 — Simple Publish/Subscribe Interaction

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.

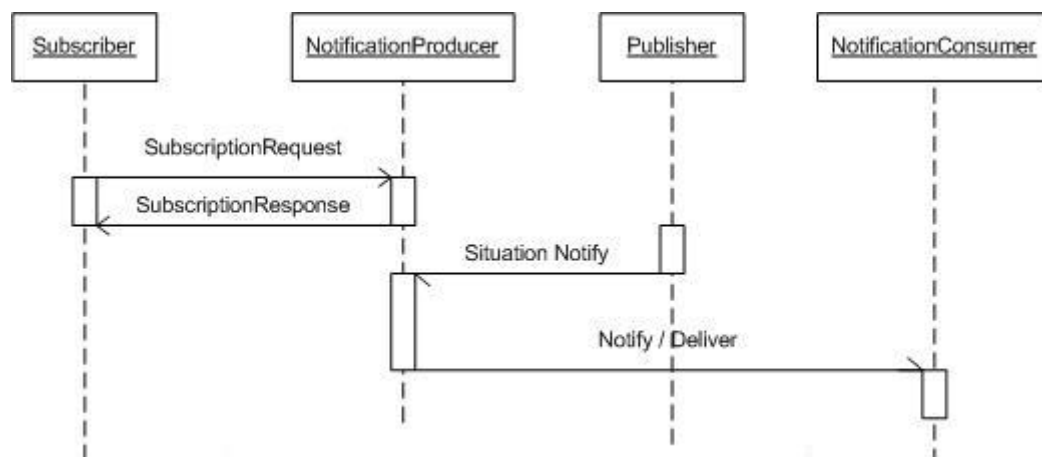


Figure 3 — Brokered Publish/Subscribe Interaction

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

5.1.5 Request/Response – Compound Requests

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

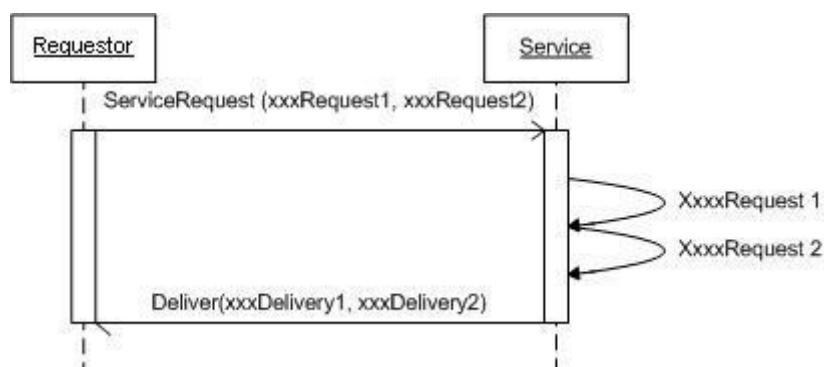


Figure 4 — Request/Response: Compound Requests

5.1.6 Publish/Subscribe – Compound Subscriptions

Multiple subscriptions to a single SIRS 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.

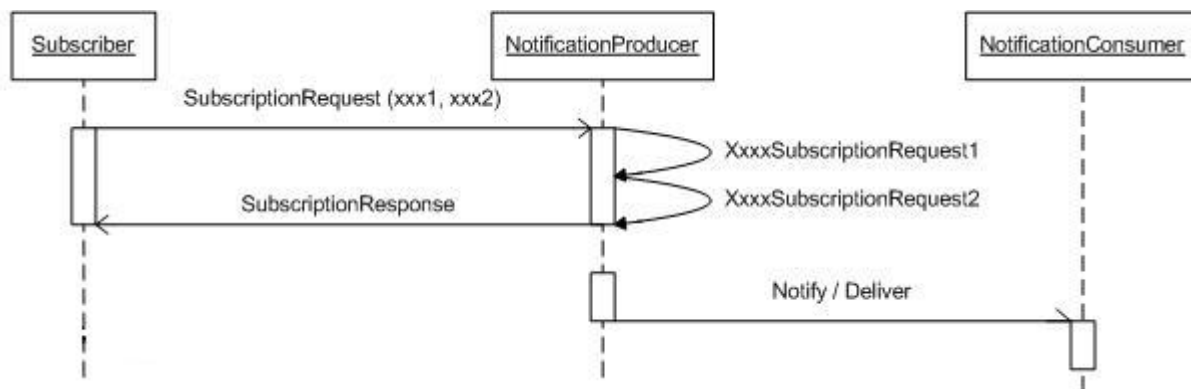


Figure 5 — Publish/Subscribe: Compound Subscriptions

5.2 Delivery Patterns

5.2.1 General

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) *Fetches 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.).

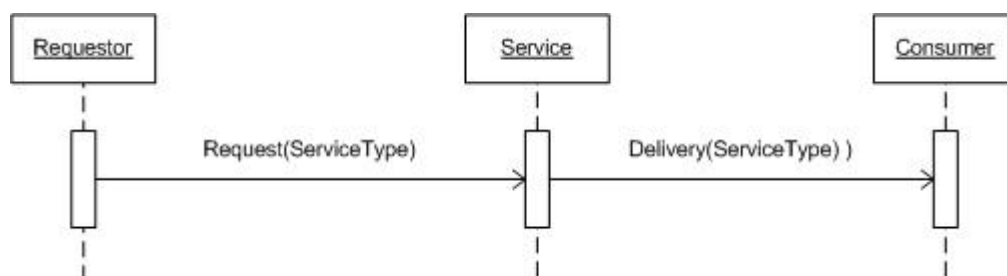


Figure 6 — One Step Direct Delivery

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 a 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.

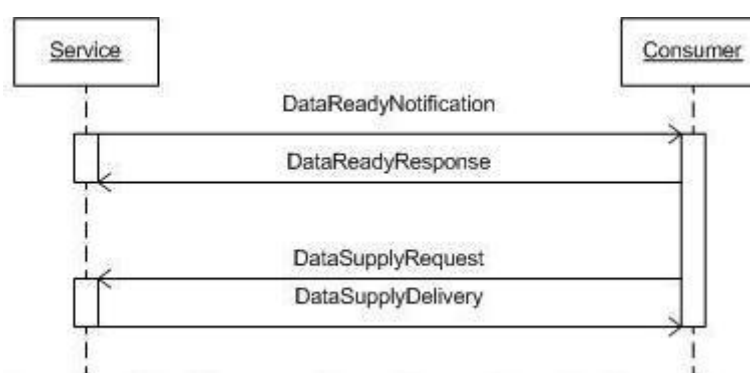


Figure 7 — Fetched Delivery

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).

In practice, *Fetched Delivery* is used predominantly for *Publish/Subscribe*. Figure 8 shows the full sequence of interaction from subscription to delivery.

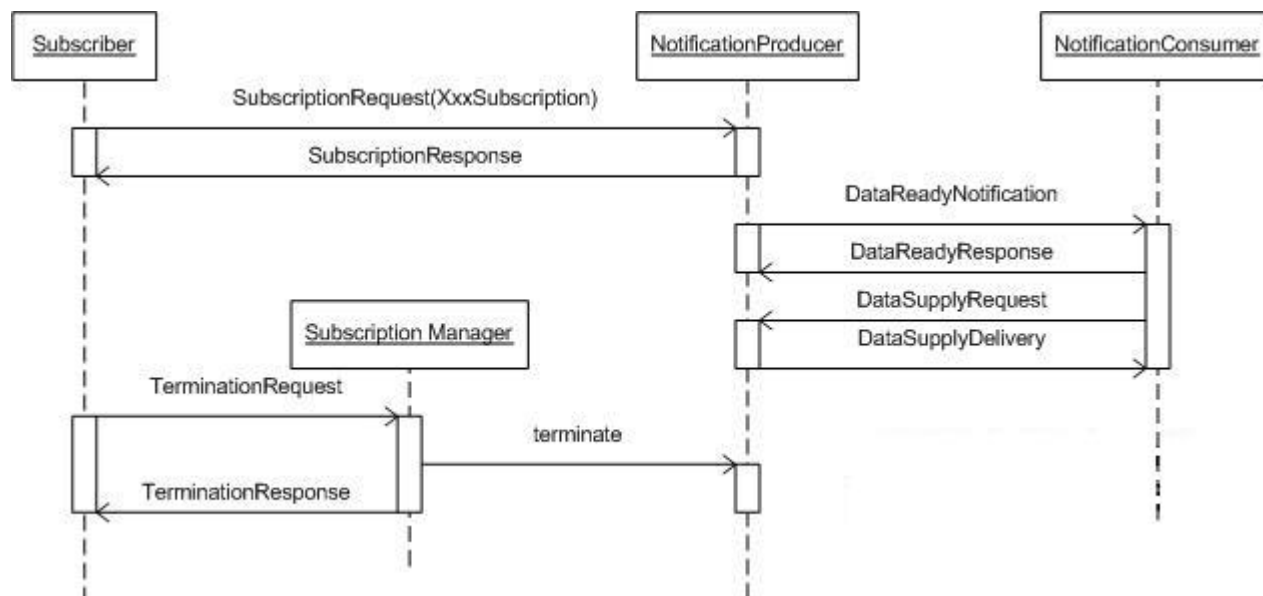


Figure 8 — Fetched Delivery for Publish/Subscribe

For completeness, we note that *Fetched Delivery* can also be used for the delivery of responses in a *Request/Response* interaction (Figure 9).

5.2.4 Data Horizon for Fetched Delivery

For *Fetched Delivery*, implementations may vary as to how long the data notified as ready will still be available to be fetched. At a minimum the most recent update must be available until it is stale, i.e. has reached the end of its currency time, or is superseded by another update. Some implementations may choose to keep previous updates available within a longer data horizon, for example the current day, and support historic access to a log of previous updates.

For further considerations as to the contents of fetched delivery, see discussion of Mediation Behaviour below.

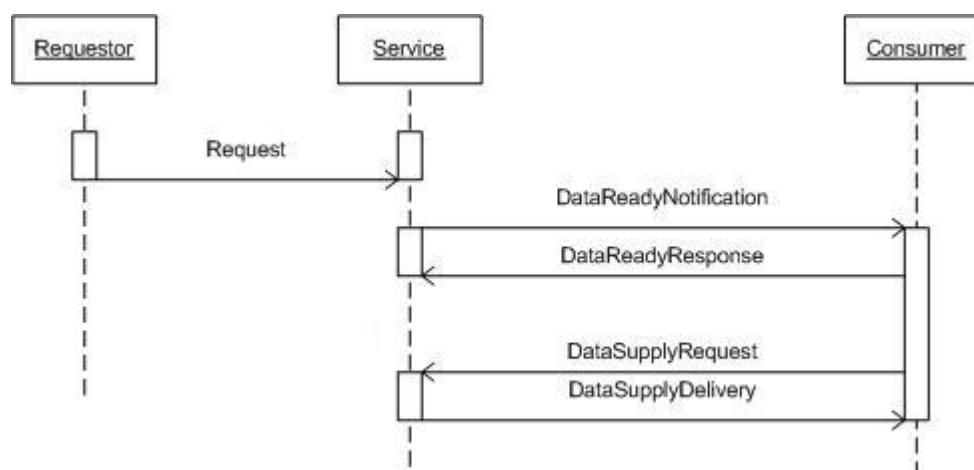


Figure 9 — Fetched Delivery for Request/Response

5.2.5 Get Current Message

The SIRI Data Supply Request may be used by itself without a previous notification to get the current data. This corresponds to the WS-PubSub capability of 'Get Current Message': which allows a Consumer to get the current messages published for a given subscription. In normal WS-PubSub usage this is a non-destructive read: the current data can be reread many times, provided the data is still relevant.

The SIRI Data Supply Request has a parameter which can be used to optimise usage: '*return latest*' i.e. whether to return only the most recent update for the subscription. (Return latest is in fact the default; the opposite, '*return all*' has to be explicitly specified on a request.). 'Return latest' cannot be repeated to obtain the same data, since once delivered, a new latest time will be held for the subscription.

In SIRI one of the mediations carried out by the Notification Producer (See later below) is to aggregate the individual subscriptions of a particular subscriber into a Subscription Channel. In effect, for each subscriber for each service, the Notification Producer keeps a Subscription channel or *Filter*: all subscriptions from the Subscriber are normally assigned to the Subscription Filter. This means that a SIRI Get Current Message request may need to indicate whether to return; (i) the latest data for a specific Subscription (the normal WS-PubSub interpretation); (ii) the latest data for a particular Subscriber Filter (i.e. group of subscriptions belonging to a single subscriber); or (iii) the latest data for a particular Subscriber (i.e. group of all the Subscriber Filters belonging to a particular Subscriber).

Support of '*return latest*' is a required SIRI Feature. Support for Get Current Message with '*return all*' is a required feature of WS-PubSub: a SIRI implementation will need to support it in order to be able to return a full data set to new subscribers who have just joined.

5.2.6 Multipart Despatch of a Delivery

If the amount of data to be delivered is large, some implementations allow the Producer to break the data supply step up into multiple messages, which the Consumer can then assemble into a single update (Figure 10). This has further implications for any recovery processes and is discussed in more detail later.

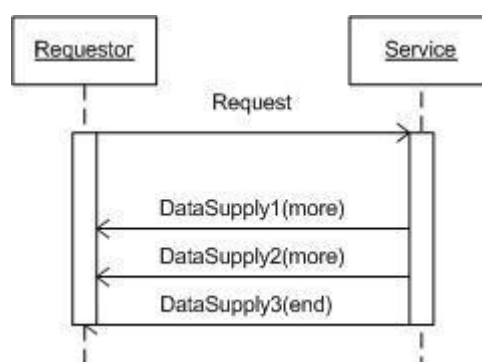


Figure 10 — Multipart Delivery

5.2.7 Multipart Despatch of a Fetched Delivery – MoreData

The data for a single subscription must be completely included within a single Delivery. Data for a *Fetches Delivery* may only be split into separate messages if it is for different subscriptions for the same subscriber.

For multipart despatch, on the Service Delivery message the **MoreData** element indicates whether the content of a **DataSupplyResponse** contains all the updated data, or whether for implementation reasons, the transmission has been split into several sub-messages, requiring retrieval by the consumer with a series of chained data **DataSupplyRequests**; see Figure 11. Each Data Supply Delivery response message in the chain indicates that there is further data by means of a **MoreData** value of "*true*"; in the last data supply response message, the **MoreData** element is set to "*false*".

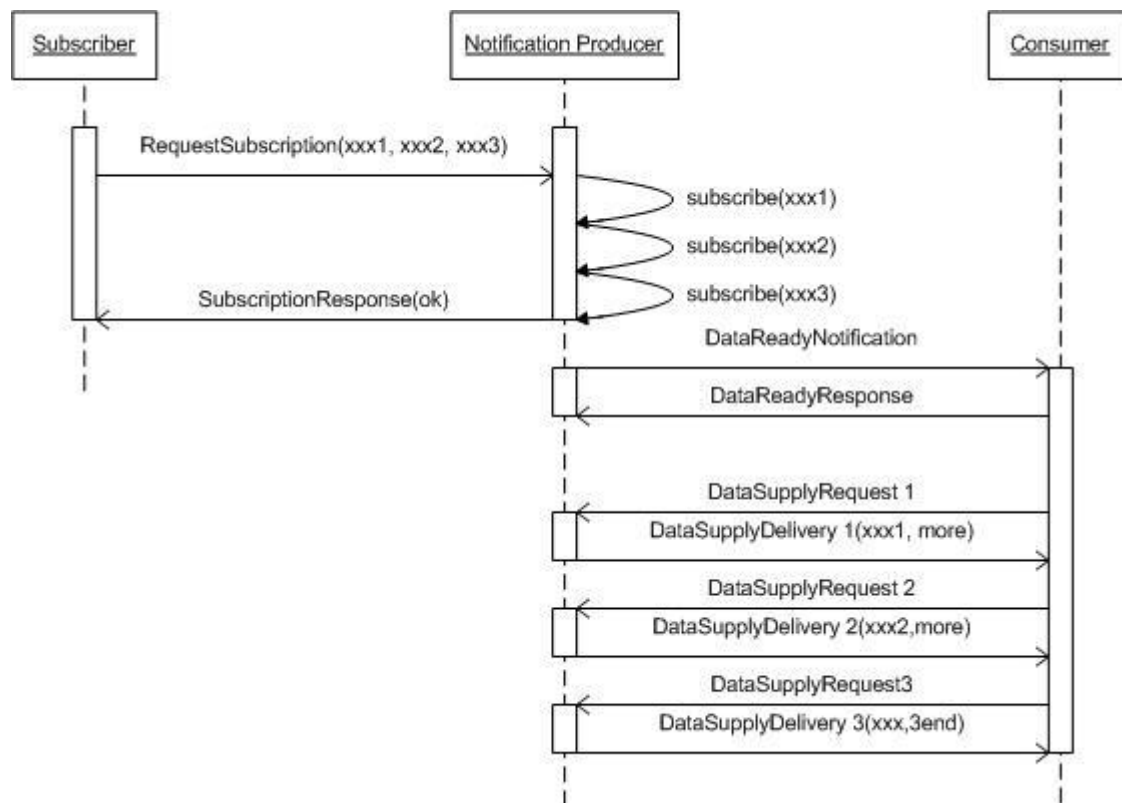


Figure 11 — Fetched Multipart Delivery

With the transmission of the final **ServiceDelivery** message, the Producer service considers the delivery of the data to be complete (i.e. it is permitted to reset the last update flags of the subscription as now being current with known notifications). A renewed **DataSupplyRequest** would then not be answered with the data of the recently polled subscription.

If the Optional SIRI Capability *ConfirmDelivery* is used, then after receiving the final Data Supply Delivery, the Consumer must send a **DataAcknowledgement** message to the Producer to indicate that it has successfully received the data, which permits the Producer to close the delivery.

5.3 Mediation Behaviour

5.3.1 General

In order to reduce the amount of network traffic and the volume of notification and delivery messages that consumers must deal with, SIRI prescribes particular mediation behaviour on the part of the Notification Producer/Publisher. There are several different aspects to this particular mediation to which a SIRI implementation must conform; in particular; (i) Maintaining Subscription Last Updated State, and; (ii) Aggregating Subscriptions into a Subscriber Channel. Both of these are quite simple concepts, but when combined, especially with fetched delivery, lead to some complexity.

5.3.2 Mediation Behaviour – Maintaining Subscription Last Updated State

In the simplest incarnation of the WS-PubSub paradigm, notification messages are despatched to all interested subscribers immediately when they occur. When a Consumer joins a service it may be sent an initial delivery with the set of current data. Thereafter, it is sent only additional notifications and deliveries for new events. The full set of current data for a given topic expression may be retrieved by a Consumer at any subsequent time using the 'Get Current Message' function. In SIRI this basic mediation is further elaborated by (i) the requirement to support the *Fetched Delivery* pattern, and (ii) an ability to set a threshold value for

sensitivity to change in data values for subscriptions in certain SIRI Functional Service types (notably Stop Monitoring, Vehicle Monitoring and Connection Monitoring).

With the SIRI *Fetches Delivery* pattern, payload data is not sent immediately along with the notification. This means that the *Notification Producer must retain state for each subscription to hold the time that the most recent successful delivery was made to the individual Consumer*, so that when the Consumer comes to fetch the data with a data supply message, only the updates 'recent' to that individual Consumer are sent (i.e. those since the last fetch), and not the whole current data set.

The ability to establish *the point of last update for each individual subscription is also required to support the filtering of notification messages by a change threshold* regardless of whether *Direct* or *Fetches Delivery* is used. If a change threshold is set, then a notification for a data event representing a change to an earlier event will only be sent if the change in some quantitative value (for example, the predicted arrival time) exceeds the specified threshold of difference to that in the last delivery.

The use of Last Update mediation for a *Direct Delivery* is shown in Figure 12 as a simple message sequence example. (See Figure 13 for a *Fetches Delivery* sequence). On subscription, the Producer computes the current data based on notifications from the Publisher (for example 1.1, 1.2) according to the subscription topic and sends it to the Consumer; the time (T1) of supply is recorded against the individual subscription. Subsequent notifications from the Publisher (2.1, 2.2, 2.3) are filtered until the sensitivity threshold is exceeded, at which point all the updates since the last update time T1 (2.1, 2.2, 2.3) are aggregated and sent and the new time of supply recorded (T2).

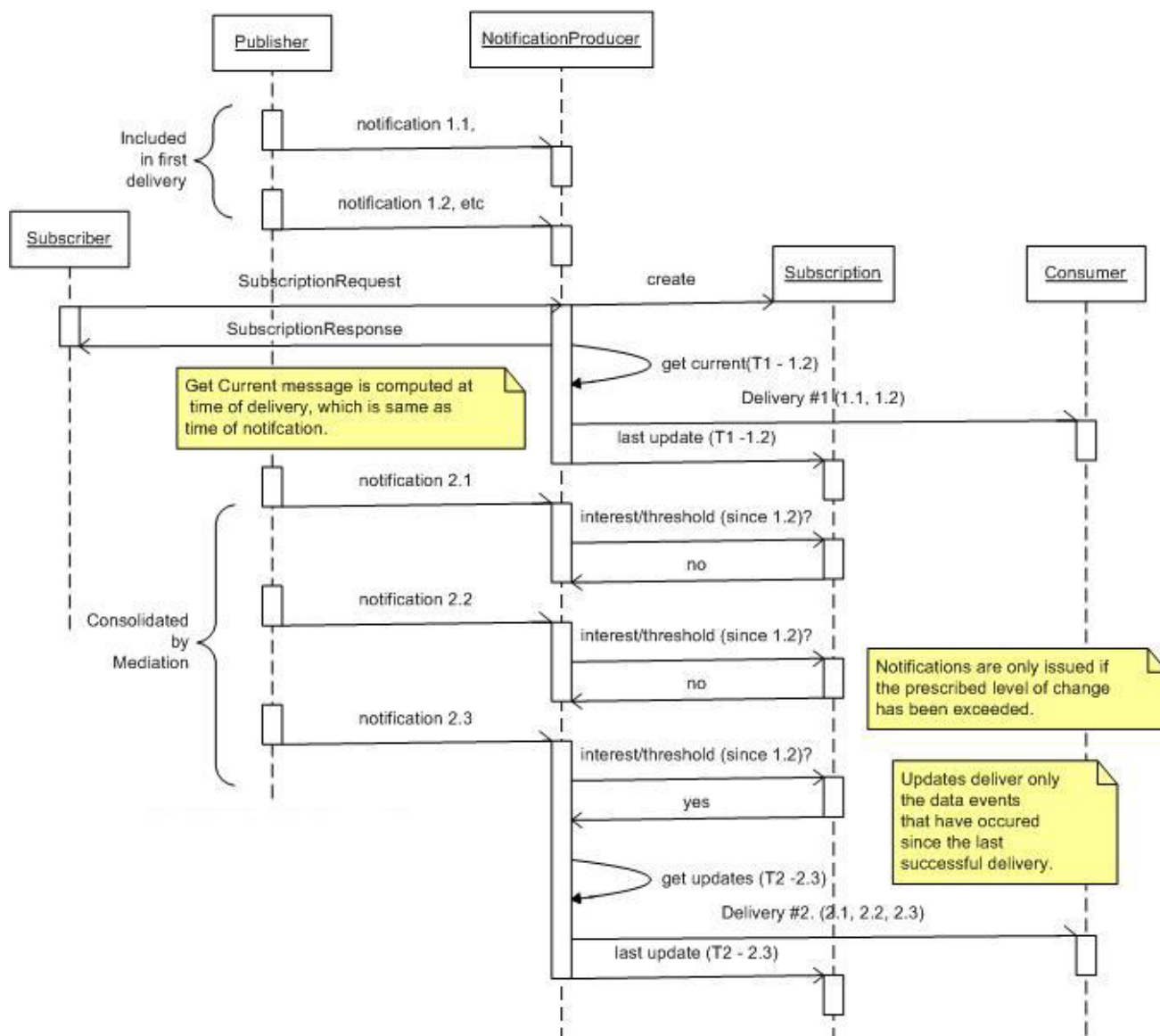


Figure 12 — Mediation: Update Tracking and sensitivity threshold for Direct Delivery

In a *Fetches Delivery* interaction, there may be a delay between the despatch of a Notification that a new update exists to the Consumer and the despatch of the Data Delivery message – which occurs only in response to an explicit Data Supply request from the Consumer. In the meantime, further situations may occur in the interim, giving rise to further notifications from the Publisher to the Notification Producer (but not to the Consumer – the Notification Producer also retains state that a notification has already been issued). The data delivery, when finally made, should represent the most current position: the data supply message should include all known subsequent data events that have taken place since the time of last data supply.

This is shown in Figure 13 for a *Fetches Delivery* sequence. On subscription, the Producer checks if there is any current data (1.1, 1.2) according to the subscription topic and if so, sends a data ready notification (#1) to the Consumer. When the Consumer requests the data supply, the Producer recomputes the current data (1.1, 1.2) and sends it to the Consumer. The time (T1) of supply is recorded against the individual subscription. Subsequent notifications from the Publisher (2.1, 2.2) are filtered until the sensitivity threshold is exceeded, at which point (T2) a new notification (#2) is sent to the Consumer. When the Consumer requests the data supply at a later time (T3), the Producer recomputes the data since the last update which, as well as the notifications that triggered the Data Ready Notification (2.1, 2.2), in the example also includes an additional item (2.3) which has arrived in the meantime. The data is despatched to the Consumer and the new time (T3) of supply is recorded against the individual subscription.

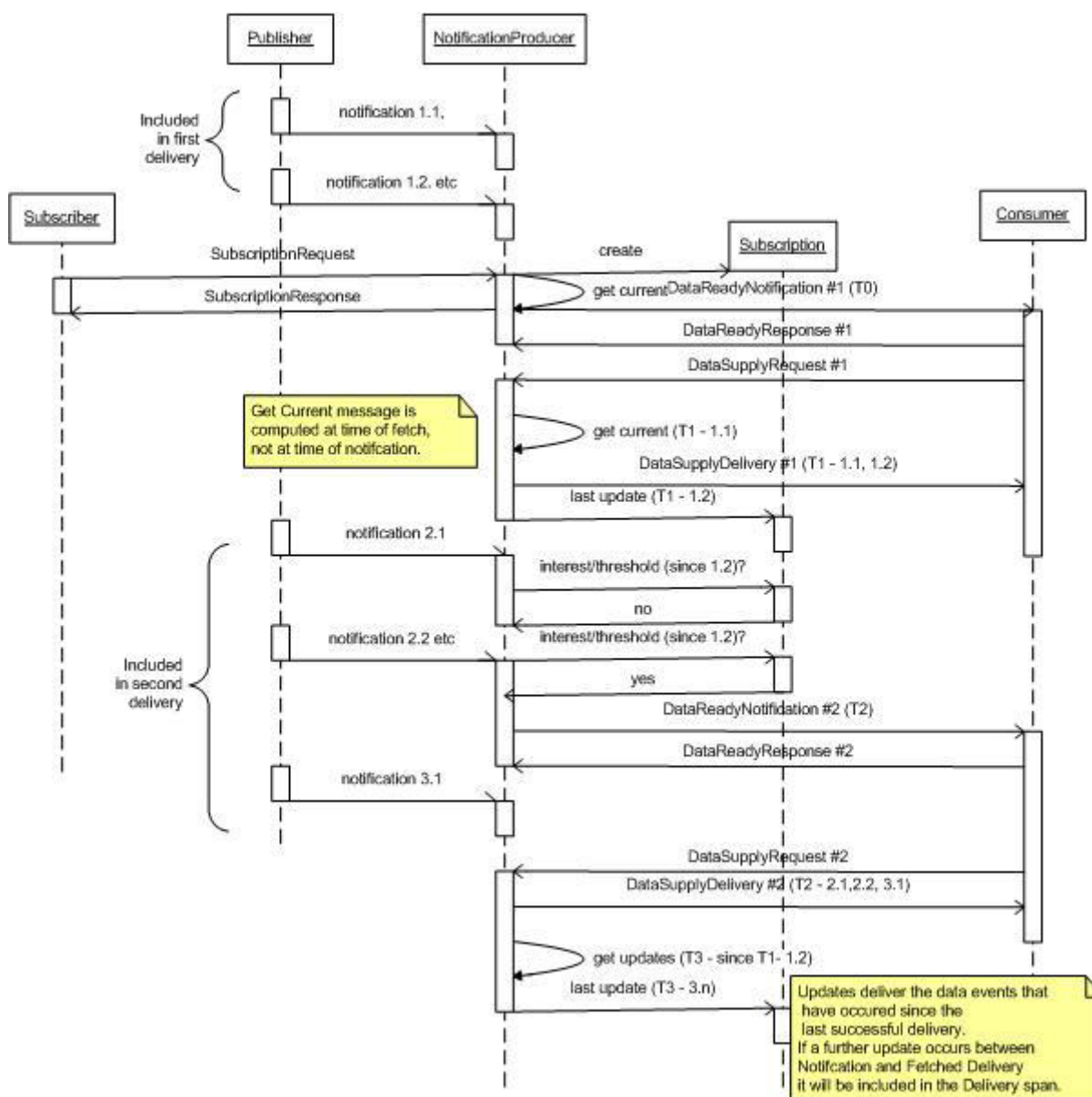


Figure 13 — Mediation: Handling Fetched Delivery Latencies

5.3.3 Mediation Behaviour – Subscription Filters

Another optimisation that SIRI can make in order to reduce the network traffic is the aggregation of subscriptions for a given Subscriber into a single filter group for notification and delivery.

When a Subscription is created for a Subscriber to a given SIRI functional service type, it is automatically assigned to a Subscription Filter. Subsequently, a *single notification is sent for all subscriptions of a Filter*. For example, if a SIRI Vehicle Monitoring Service Subscriber has subscriptions to two separate lines, and Situations arise for both of them (from data notifications from the Publisher) within the data refresh or processing cycle of the Notification Producer, only a single Data Ready Notification will be sent to the service Consumer, and the single Data Supply message will return the payload for both data ready notifications as content within a single Service Delivery. Thus for each Subscription Filter, the system retains 'Already Notified' state, which gets reset every time last update is updated.

A subscriber may add different Subscriptions at different times to a Subscription Filter.

Some implementations support only a single Subscription Filter per Subscriber; others may support multiple filters. If multiple filters are supported and a Subscriber wishes to use a separate Subscription Filter, a Filter Identifier should be indicated on the subscription request. If no filter is specified on a request, the first filter created for the Subscriber will be used by default. The use of multiple filters allows a Subscriber interested in many different subscriptions from the same Notification Producer to keep transaction boundaries (and the computational overhead needed for the filtering, transmission and processing of delivery packages) down to a manageable size. Otherwise there is the possibility that a small urgent item of information will be swamped within a larger packet of data for additional content that is triggered for delivery at the same time.

Subscription Filtering is shown in Figure 14 for a *Direct Delivery* sequence (the *Fetches Delivery* sequence would be logically similar).

- The initial subscription request contains two separate individual service subscription requests (001, 002).
- On subscription, the Producer checks if there is any current data (1.1) according to both subscriptions' topic expressions, and if any data is found, the initial delivery (#1) will contain data for both subscriptions.
- If an additional Subscription (003) is created for the Subscriber, the Producer will add it to the Subscriber's Subscription Filter.
- The Producer only make a notification/delivery (#2) if there is any data for the new subscription: in Figure 14 none is shown as having occurred. However if there was any data, i.e. if any data is sent at all, it will include updates for any subscription of the Consumer, even if the other updates have not reached the threshold. Again, the time (T2) of supply will be recorded against each individual subscription.
- Subsequent notifications from the Publisher (2.1) are filtered until the sensitivity threshold is exceeded, at which point (T3) a new data notification and delivery (#3) is sent to the Consumer. The data supply includes updates since the last supply time T2 affecting any subscription in the Subscription Filter.

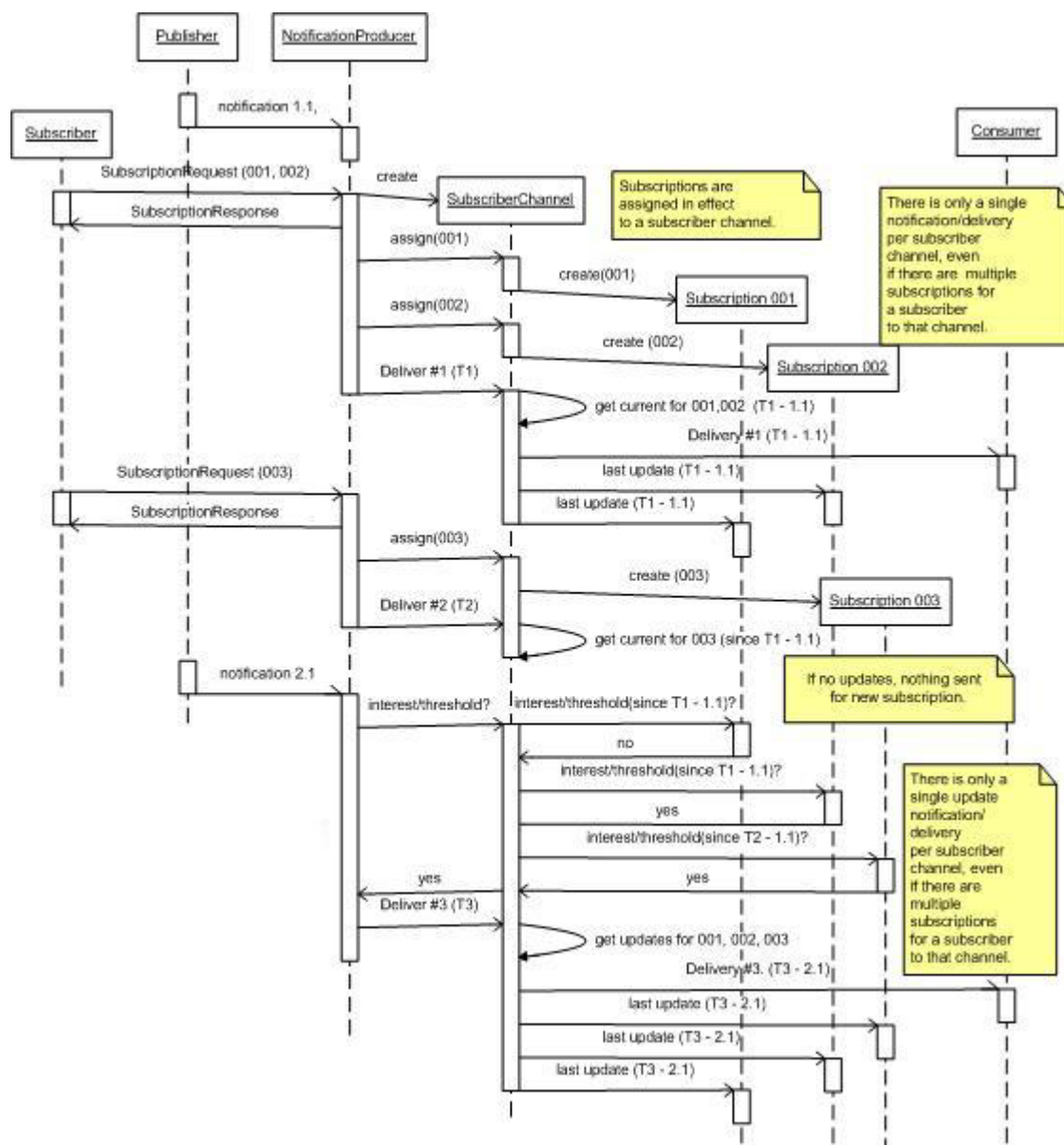


Figure 14 — Mediation: Subscription Filter

5.4 Recovery Considerations for Publish Subscribe

5.4.1 General

Publish/Subscribe is a stateful pattern of interaction, and consideration must be given to failure of either the Notification Producer, or Consumer systems, or to the communications connection between them.

Once created, subscriptions are held by the Notification Producer. In the event of a system failure by the Notification Producer service, SIRI does not require the Notification Producer to recover the subscriptions. Instead it is the responsibility of the Subscriber to recreate new subscriptions to replace the existing ones lost in the failure. Each Consumer knows the Subscriber which submitted its subscriptions (usually these will be the same implementation entity).

It is then the Consumer's responsibility to monitor whether the Notification Producer Service and the connection to it are still active, and to inform the Subscriber if the Subscription needs to be renewed. SIRI

supports two ways of monitoring the Notification Producer (i) Status Polling (a required SIRI capability), and (ii) Heartbeat (an optional SIRI capability). This allows implementors to choose the most efficient approach given the traffic and cycle characteristics of their deployment.

5.4.2 Check Status – Polling

The Consumer may send periodic Check Status requests to the Notification Producer to check that Notification Producer is still active. A Check Status message is a required feature of all SIRI implementations.

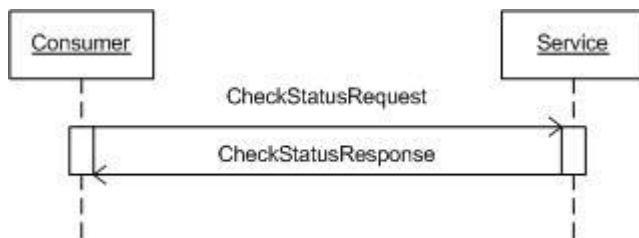


Figure 15 — Check Status

5.4.3 Heartbeat – Pinging

Using a Heartbeat message, the Notification Producer sends a regular notification message to the Consumer at a predefined interval to show that it is still active, even if no update messages have been sent. This removes the need for the Consumer to poll, as the Consumer can detect a potential failure by the failure of a Heartbeat to arrive within the prescribed interval. The heartbeat interval may either be preconfigured for the whole system, or be specified as a parameter in the subscription request specified for individual subscription on the Subscription Policy.

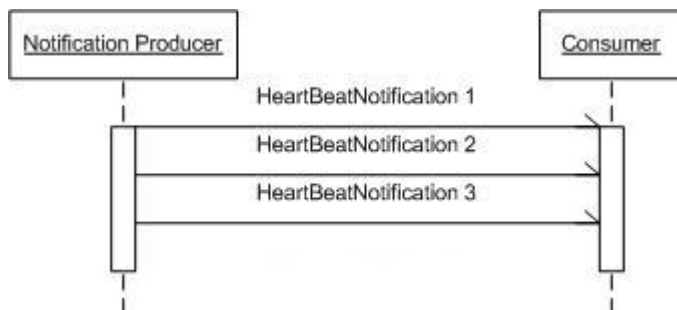


Figure 16 — Heartbeat Message

5.4.4 Degrees of Failure

There are in effect different degrees of failure of the Producer, or in access to the producer: (i) *Total Failure* in which case the subscriptions are lost and (ii) *Partial failure*, in which case the subscriptions are still in existence, but the data flow is interrupted for a period. Total failure must always be marked by a new Service Start time.

To recover from partial failure the Producer must send all current data for a subscription since the last update for that subscription

5.4.5 Detecting a Failure of the Producer

5.4.5.1 Detecting a Failure Using Check Status

A Consumer can at any time explicitly poll to detect a Service breakdown by the Notification Producer, by sending a *CheckStatusRequest* to the Service. A *CheckStatusResponse* should be returned. A Common Check Status message is used for all services (Figure 15).

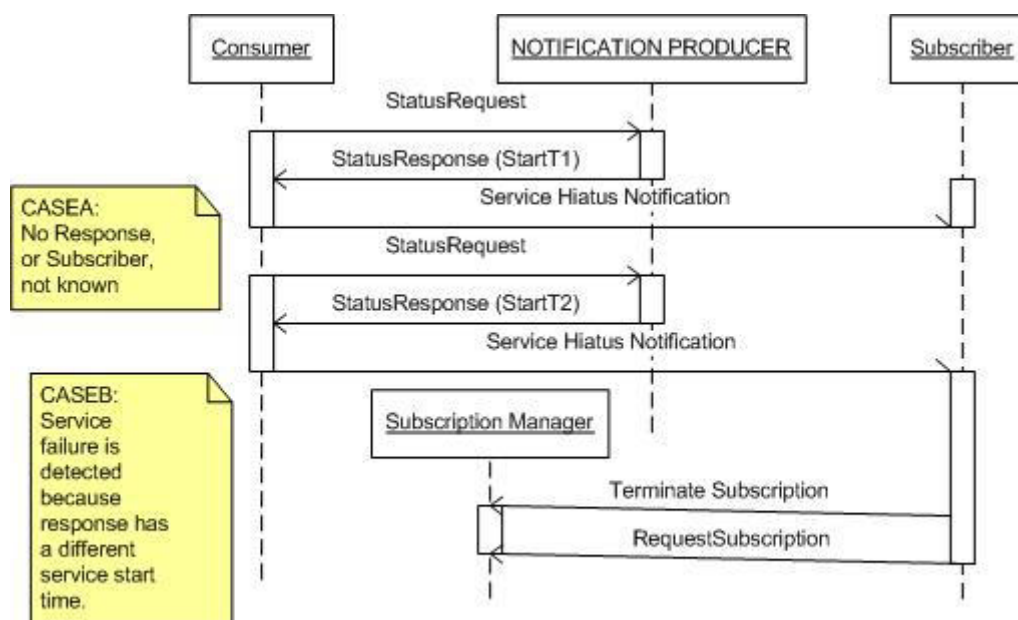


Figure 17 — Check Status with Recovery & Re-subscribe

If there has been only a brief interruption of service, this may be detected by comparing the start time for the service returned on the check status message with the start time found on previous messages, (either a *SubscriptionResponse*, or a *CheckStatusResponse*). A hiatus suggests that there may be missed data, and/or subscriptions, and so the Consumer asks the Subscriber to recreate its subscriptions.

5.4.5.2 Detecting a Failure Using Heartbeat

If a Heartbeat capability is available, a Consumer may detect a Service failure by monitoring for the absence of *HeartbeatNotification* messages within the expected interval, rather than by polling explicitly for service availability and integrity with a *CheckStatusRequest*. If a failure is detected, the Consumer asks the Subscriber to recreate the subscription. Again there are two levels of failure: (i) *Total Absence of Service* with loss of subscriptions, and (ii) *Brief Interruption of Service*, with subscriptions retained but loss of real-time data.

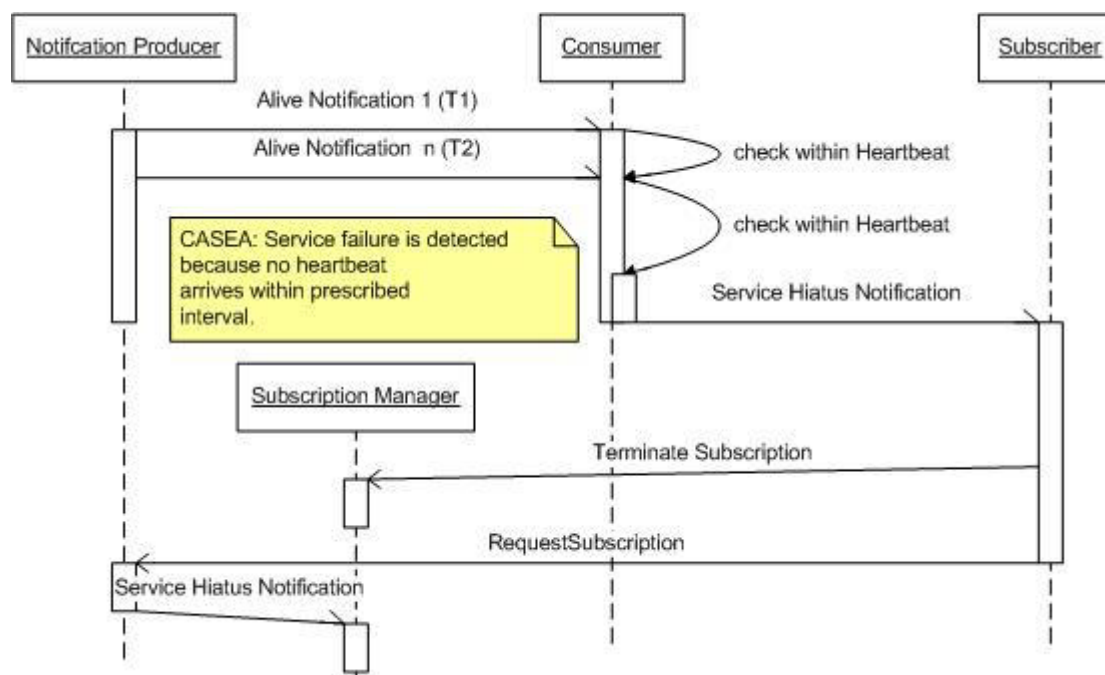


Figure 18 — Heartbeat Monitoring – Loss of Service

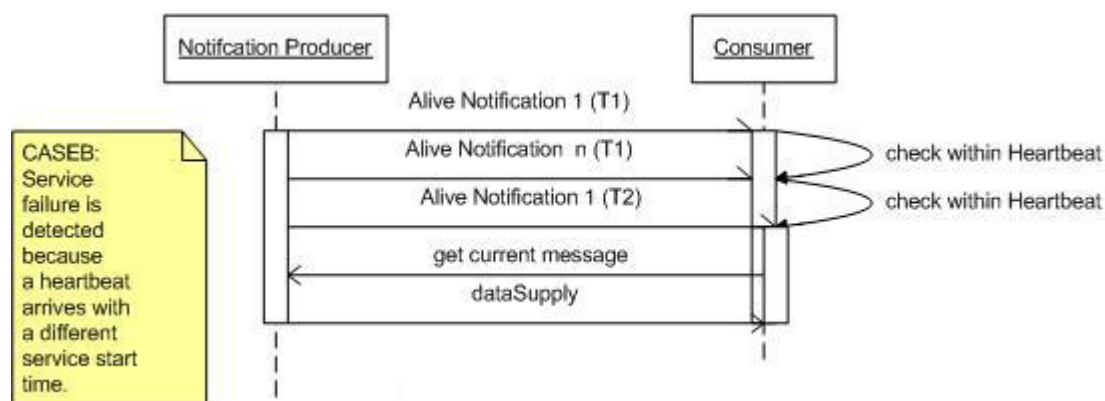


Figure 19 — Heartbeat Monitoring – Interruption of Service

5.4.6 Detecting a Failure of the Consumer

It is the Consumer entity's responsibility to restart itself after a failure of its system. It must decide if it may have missed data and take any appropriate recovery action – either to get the current message, or to completely recreate its subscriptions.

In the event of System failure by the Notification Consumer, the Notification Producer will continue to send Notification messages to the Notification Consumer until the subscription has expired. In order to optimise efficiency, a well behaved Consumer will terminate its subscription if it is no longer needed, or if a prolonged outage is planned.

An implementation may have a policy that states the Notification Producer will drop a subscription if it does not receive an acknowledgement for Deliveries within a certain period of tolerance.

5.5 Recovery Considerations for Direct Delivery

For Direct Delivery, two levels of resilience are possible: (i) *Simple Despatch* and (ii) *Acknowledged Despatch*.

For *Simple Despatch*, data is sent without an acknowledgement of receipt. For a *Request/Response* data supply interaction, the Requestor knows that it is waiting for a response and can determine whether it has been satisfied. However for asynchronous, i.e. *Publish/Subscribe* interaction, the Notification Producer can send a data message to the Consumer at any arbitrary point in time; so if for some reason the message fails to arrive, the transmission failure will not be detected by either party and so no recovery action will be attempted. In many cases this is sufficient, as another update shortly after will update the messages as effectively as a recovery action.

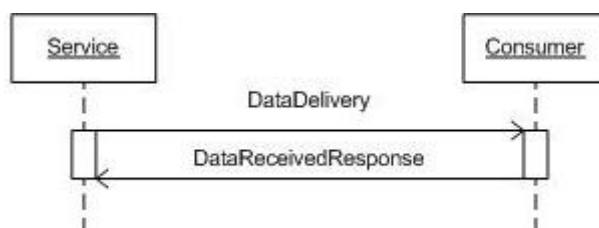


Figure 20 — Robust Direct Delivery

With *Acknowledged Despatch*, reliable transmission is signalled through an additional 'Acknowledge Data Received' response from the Consumer to the Producer, confirming successful receipt of data. If the Notification Producer fails to receive a response from the Consumer, it would then keep resending at predefined intervals, exactly as for the initial notification step of *Full Fetched Delivery*.

5.6 Request Parameters and Interactions

Both *Request/Response* and *Publish/Subscribe* interactions involve the sending of requests to a Service running on a server. In both cases, the requests must specify information about the content that is to be returned, including any variable aspects of real-time behaviour, as well as additional policies for carrying out the request, such as a throttle on the volume of data to return.

SIRI uses a standard set of parameters for the various terms of the requests, organised according to WS Pub-Sub concepts such as *Endpoint Reference*, *Topic*, *Request Policy*, *Subscription Policy* and *Payload*.

For a given type of SIRI Functional Service interface (for example Connection Timetable, Vehicle Monitoring, or General Message), the analogous groups of parameters are used on both *Request/Response* and *Publish/Subscribe* requests, for example to specify the content to be returned, (e.g. the data reference system, the topics of interest, the temporal window, and the level of detail), and the processing policy.

Additional parameters are specified on a subscription to control those aspects of processing which are specific to the asynchronous servicing of subscriptions, i.e. that apply to the Publisher and Notification Producer processes; for example, sensitivity threshold.

Some parameter types must be included on individual requests; others may be configured for all message exchanges.

The SIRI parameter set includes some terms which are common to all requests, regardless of service interface (for example; end point references, delivery policies), and others which vary according to the individual Service type. Table 1 shows the general groups of parameters, and whether they are common or whether they vary by message type.