Author: Malin Nyfeldt, Martin Wikman
Title: FS- Service Request Manager

Version: B
Date: 2006-12-06

FS – Service Request Manager

Content

History

Version	Date	Adjustments
A	2006-10-09	First revision. (MMAWI)
B	2006-12-06	Changed lock, shutdown and unlock to close and open. (MMAWI)

1 Introduction

This document specifies the function of the Service Request Manager component. The Service Request Manager (SRM) provides an interface that can be used by clients in order access services supplied by other M3 components and vice versa.

1.1 Glossary

1.1.1 ServiceId

An M3 component may support several services via the same port. The ServiceId identifies the service to access.

1.1.2 TransactionId

TransactionId is used to identify on going asynchronous requests.

1.1.3 ValidityTime

The ValidityTime is the maximum time a service request is valid. When the validity time has expired, the service request is canceled.

1.1.4 Parameter

A Parameter is a service specific parameter. A parameter is identified using the parameter name. The parameter name is case sensitive.

1.1.5 ResponseRequired

The ResponseRequired specifies if a response is required for the service request.

1.1.6 StatusCode

The status code is included in a service response and indicates if the requested service was successful or not. The status codes may be service unique.

1.1.7 StatusText

The status text is included in a service response and is a textual description of the result of the requested service.

2 Function Requirements (Commercial)

No commercial requirements have been identified.

3 Function Specification (Design Related)

3.1 Introduction

Using the Service Request Manager, clients can access services in external M3 components.

The purpose of the Service Request Manager is only to relay service requests and corresponding responses between components that provides and/or consumes services. The Service Request Manager simply relays the requests and responses using the XMP protocol (see [1]). The Service Request Manager has no service specific knowledge.

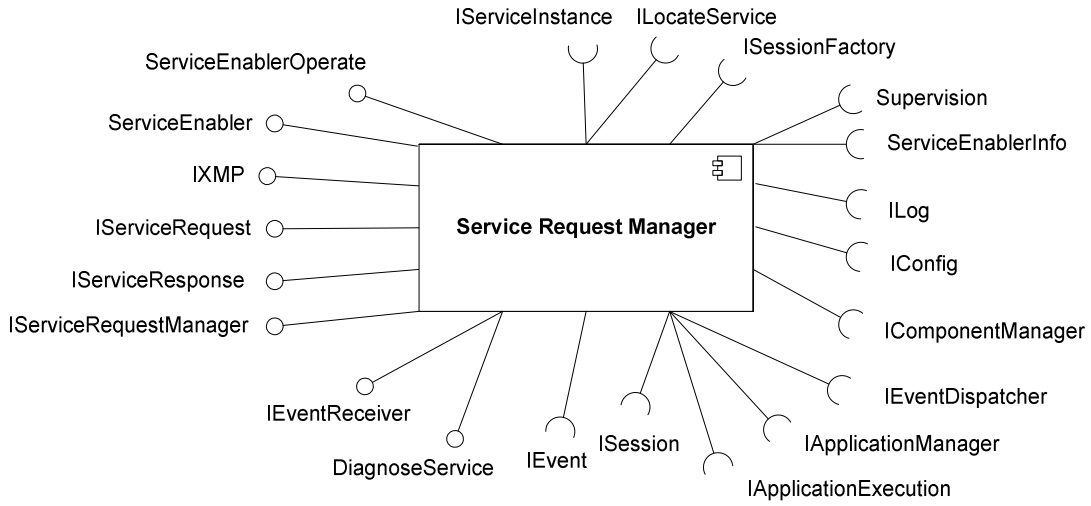

Figure 1 Service Request Manager component

3.2 Exported Interfaces

3.2.1 IServiceRequest

This interface is used to manage a service request.

3.2.1.1 Methods

IServiceRequest offers the following methods:

- *String getServiceId()*
- *int getValidityTime()*

- *Boolean getResponseRequired()*
  Indicates if a response is required for the request.
- *Object getParameter(String parameterName)*
  Returns the value of the specified service request parameter. The parameter name is case sensitive.
- *String[] getParameterNames()*
  Returns a list of the names of the parameters currently set for the request.
- *void setServiceId(String serviceId)*
- *void setValidityTime(int validityTime)*
- *void setResponseRequired(Boolean responseRequired)*
  Indicates if a response is required for the request. If not set, the default value is true, i.e. a response is required.
- *void setParameter(String parameterName, Object parameterValue)*
  Sets the specified parameter to the given value. The parameter name is case sensitive.

3.2.2 IServiceResponse

This interface is used to manage a service response.

3.2.2.1 Methods

IServiceResponse offers the following methods:

- *int getStatusCode()*
- *String getStatusText()*
- *Object getParameter(String parameterName)*
  Returns the value of the specified service parameter. The parameter name is case sensitive.
- *String[] getParameterNames()*
  Returns a list of the names of the parameters currently set for the response.
- *void setStatusCode(int statusCode)*
- *void setStatusText(String statusText)*
- *void setParameter(String parameterName, Object parameterValue)*
  Sets the specified parameter to the given value. The parameter name is case sensitive.

3.2.3 IServiceRequestManager

This interface is used to access external M3 services.

3.2.3.1 Methods

IServiceRequestManager offers the following methods:

- *IServiceResponse sendRequest(IServiceRequest request)*
  Sends the request to an available host (see 3.4.2.1) supporting the requested service. Returns the service response.

- *IServiceResponse sendRequest(IServiceRequest request, String hostName)*
  Sends the request to the specified host. Returns the service response.
- *IServiceResponse sendRequest(IServiceRequest request, String hostname, int portNumber)*
  Sends the request to the specified host using the specified port number. Returns the service response.
- *int sendRequestAsync(IServiceRequest request)*
  Sends the request to an available host (see 3.4.2.1) supporting the requested service. The request is sent asynchronously which means that this method returns immediately with an ID of the ongoing transaction. The transactionID can then be used to retrieve the service response later.
- *int sendRequestAsync(IServiceRequest request, String hostName)*
  Sends the request to the specified host. Returns the service response. The request is sent asynchronously which means that this method returns immediately with an ID of the ongoing transaction. The transactionID can then be used to retrieve the service response later.
- *int sendRequestAsync(IServiceRequest request, String hostname, int portNumber)*
  Sends the request to the specified host using the specified port number. Returns the service response. The request is sent asynchronously which means that this method returns immediately with an ID of the ongoing transaction. The transactionID can then be used to retrieve the service response later.
- *IServiceResponse receiveResponse (int transactionID)*
  Used to retrieve the response for a previous asynchronous request transaction. If the transaction has not completed yet, this method waits until it has before the response is returned.
- *Boolean isTransactionCompleted(int transactionID)*
  Returns true if the previous asynchronous transaction has completed and has a response waiting to be retrieved.
- *void sendResponse(String sessionId, IServiceResponse response)*
  Sends the response to the component that requested the service.

3.2.4 IXMP

This interface is used to accept service requests from external M3 components sent using the XMP protocol specified in [1].

3.2.4.1 Methods

IXMP supports the XMP protocol, i.e. Service Requests and Service Responses. For a complete specification of the XMP protocol, see [1].

3.2.5 IEventReceiver

This interface is used to receive events generated by other components. For details about the interface, see [2].

3.2.6 ServiceEnabler

This interface is used for controlling the service enabler aspect of SRM, e.g. initialize it when it should be used. For details about the interface, see [2].

3.2.7 ServiceEnablerOperate

This interface is for operational administration of a service enabler, for example open, close and load regulation. It is for example used to control the amount of incoming XMP requests using a threshold indicator. For details about the interface, see [4].

3.2.8 DiagnoseService

This interface is for diagnosing the service enabled by SRM. The interface is completely separate from the other SRM functionality in order to not affect normal SRM behavior. Thus, the implementation of this interface is distributed separate from all other SRM functionality. For details about the interface, see [4].

3.3 Imported Interfaces

The Service Request Manager uses the following external interfaces:

- *ILog* for logging.
- *IConfig* for configuration.
- *ILocateService* for M3 service lookup.
- *IServiceInstance* for accessing properties of looked up M3 services instances.
- *Supervision* and *ServiceEnablerInfo* for collecting statistics information.
- *IEventDispatcher* to generate and receive events, see 3.4.3.
- *IComponentManager* to retrieve an instance of another component.
- *IApplicationExecution* to start service applications.
- *IApplicationManager* to locate the service application to start.
- *ISession* to manage active service sessions.
- *ISessionFactory* to create new service sessions.
- *IEvent* to receive events from other components, see 3.4.5.1.

Note that not all interfaces are used during the scenarios and sequences in this document. Configuration, logging, event dispatching, O&M, ILocateService and IServiceInstance are understood to be used whenever needed and not shown in the scenarios.

3.4 Functions

3.4.1 Receive Service Requests

The Service Request Manager parses XMP service requests and relays the content to the component providing the service. It is the component providing the service

that is responsible for interpreting the service specific information, initiate the service and generate a response.

However, the Service Request Manager generates the response in the following situations:

- The XMP request could not be parsed due to syntax error in request (XMP Status Code 500)
- The XMP request could not be parsed due to syntax error in parameters or arguments (XMP Status Code 501)
- The Service Request Manager cannot accept further requests due to load regulation (XMP Status Code 502)
- The XMP validity time has expired (XMP Status Code 408)
- The service session has terminated without sending a response back to the Service Request Manager (XMP Status Code 421)
- The requested service does not exist or is not available (XMP Status Code 421)
- The Service Request Manager has been closed (XMP Status Code 421)

When a request is parsed correctly, the Service Request Managers starts the application responsible for the requested service and relays the request to that application. The Service Request Manager starts the application using the interfaces IApplicationManager and IApplicationExecution (see [2]).

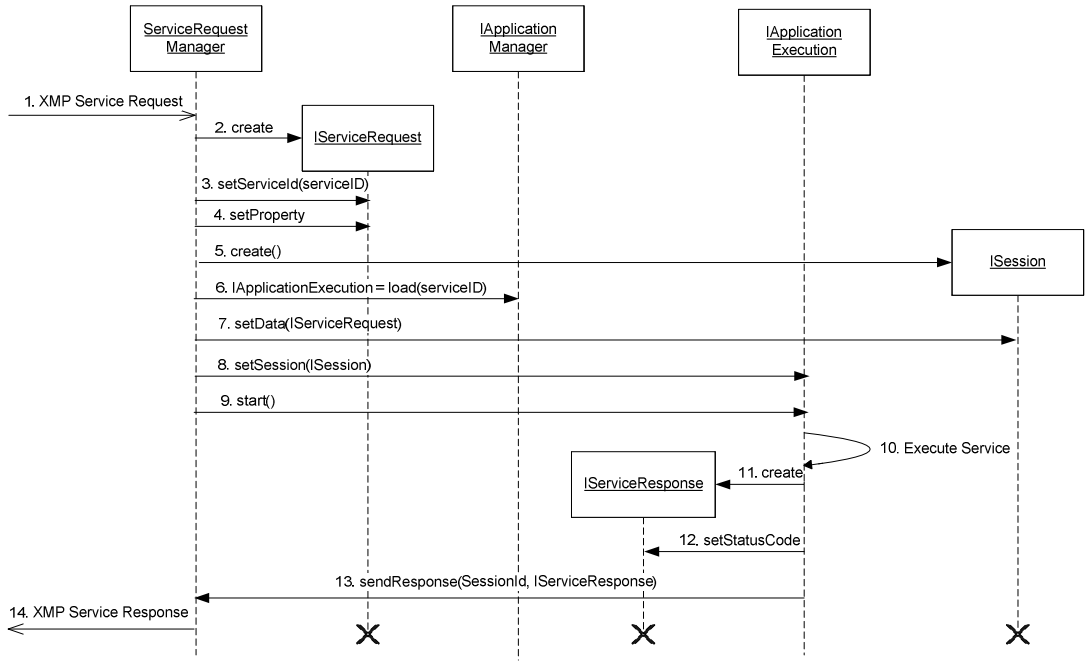

Figure 2 Receiving a Service Request

Figure 2 illustrates the sequence when receiving a Service Request:

1. A client requests a service over XMP.
2. A Service Request is created.
3. The service ID is set based on information in the received XMP request.
4. Other properties are set based on information in the received XMP request.
5. The SRM creates a new session using the ISessionFactory (not shown in diagram).
6. An application instance of the requested service is loaded.
7. Necessary session data is set, i.e. the Service Request is added to the session data and …
8. … the session is added to the application.
9. The application instance is started.
10. The application instance executes the service.
11. A Service Response is created by the application instance.
12. The status code is set for the Service Response. Other properties of the Service Response can be set at this point as well.
13. The application instance sends the response to the Service Request Manager.
14. An XMP response is sent back to the client.

3.4.1.1 Validity Timeout

In case the service requests validity timer expires, the Service Request Manager will send a response indicating timeout error. When this occurs, the Service Request Manager will call the terminate method on the application. Whether or not the service will continue its execution when this happens is up to the service. For some services, e.g. Outdial Notification, it doesn't make sense to interrupt an on going call.

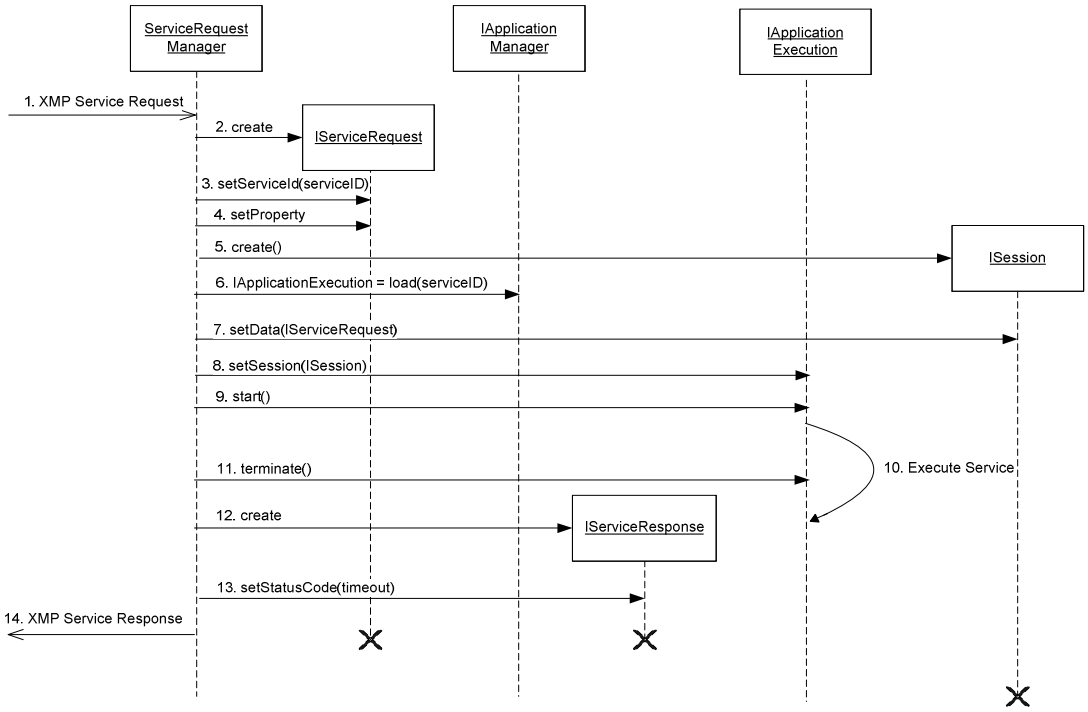

Figure 3 Timeout when executing service

1. A client requests a service over XMP.

2. A Service Request is created.

3. The service ID is set based on information in the received XMP request.

4. Other properties are set based on information in the received XMP request.

5. A new session is created using the ISessionFactory (not shown in diagram).

6. An application instance of the requested service is loaded.

7. Necessary session data is set, i.e. the Service Request is added to the session data and …

8. … the session is added to the application.

9. The application instance is started.

10. The application instance executes the service.

11. While the application instance executes the service, the validity timer expires. The SRM calls the terminate method on the application.

12. The Service Request Manager creates a Service Response.

13. The status code is set indicating that the service request timed out.

14. The Service Response is sent back to the client over XMP.

3.4.1.2 Detection of Deleted Sessions

The Service Request Manager is responsible for sending an error response over XMP if the requested service session has terminated without sending a response. In order to do this, the Service Request Manager registers itself in the event dispatcher as receiver of the event indicating that a session has terminated. This scenario is illustrated in Figure 4.

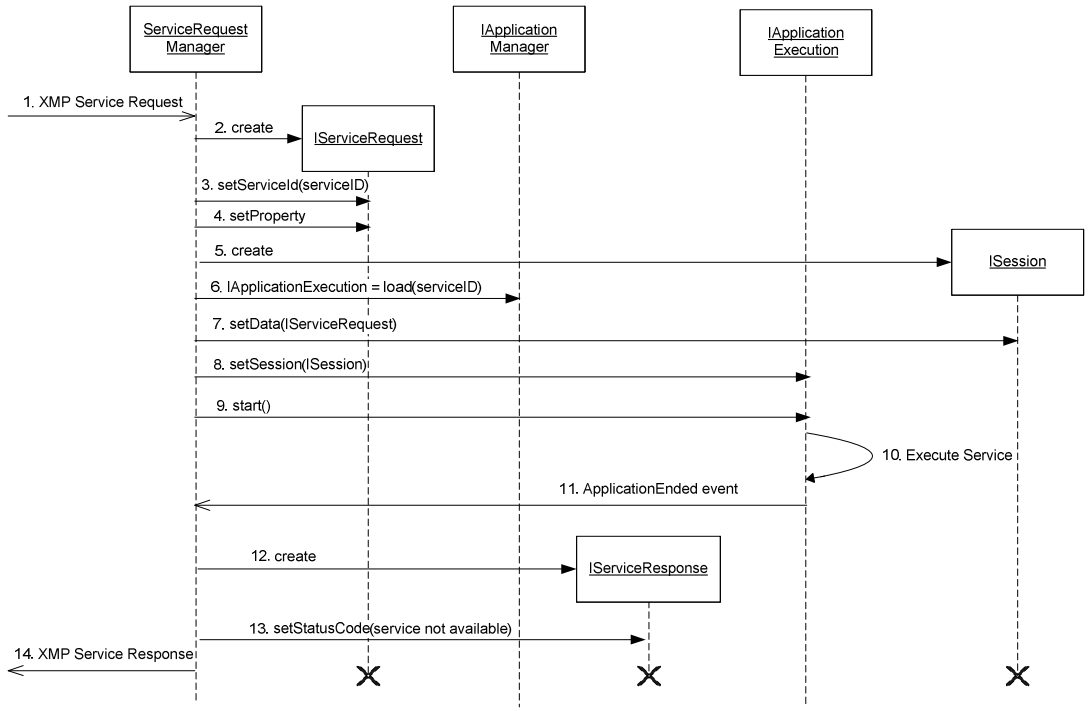

Figure 4 Session terminated before sending response

1. A client requests a service over XMP.

2. A Service Request is created.

3. The service ID is set based on information in the received XMP request.

4. Other properties are set based on information in the received XMP request.

5. A new session is created using the ISessionFactory (not shown in diagram).

6. An application instance of the requested service is loaded.

7. Necessary session data is set, i.e. the Service Request is added to the session data and …

8. … the session is added to the application.

9. The application instance is started.

10. The application instance executes the service.

11. The application instance terminates before a service response has been sent. This is indicated to the Service Request Manager with the ApplicationEnded event.

12. The Service Request Manager creates a Service Response.

13. The status code is set indicating that the service is unavailable.

14. The Service Response is sent back to the client over XMP.

3.4.1.3 Empty Requests

The Service Request Manager supports empty requests as described in [1].

3.4.1.4 Load Regulation

It is possible to stop the Service Request Manager from accepting requests using the load regulation interface. This can be used when the system is heavily loaded.

3.4.2 Send Service Requests

The Service Request Manager is also used when a Service Request shall be sent to an external M3 component.

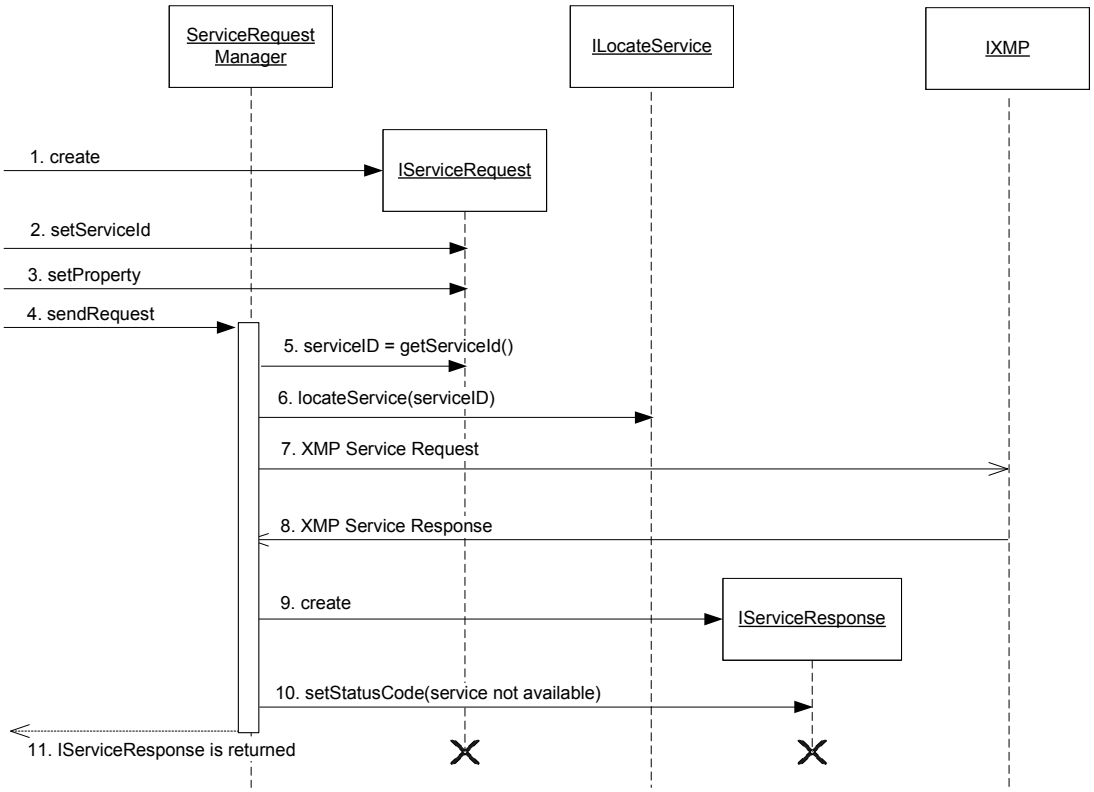

Figure 5 Sending a Service Request

Figure 5 illustrates this scenario:

1. A client creates a Service Request.

2. The service ID is set indicating which service to request.

3. Other properties are also set by the client.

4. The client sends the request to the Service Request Manager.

5. The Service Request Manager retrieves the service ID from the Service Request.
6. An instance supporting that ID is then looked up using the ILocateService (since a host name was not specified in the call to sendRequest).
7. The XMP Service Request is sent to the located instance supporting the service.
8. A while later the XMP Service Response is received.
9. The Service Request Manager creates a Service Response.
10. The status code is set for the Service Response. Other properties of the Service Response can be set at this point as well.
11. The Service Response is sent back to the client.

3.4.2.1 Host Lookup

If a host was specified in the call to sendRequest, that host is used for the XMP communication. If a host was not specified, the preferred host shall be chosen. This is done using the ILocateService interface, see [3]. The Service Request Manager uses the service ID received in the request when using the method locateService.

The Service Request Manager uses IComponentManager (see [2]) to retrieve an instance of the ILocateService interface.

3.4.2.2 Internal Handling of Some XMP Attributes

The XMP attributes TransactionId and ClientId (see [1]) are generated automatically by the Service Request Manager.

3.4.2.3 Retry at Failure

The Service Request Manager will retry sending a request in certain error situations. A retry is done if a response was not received after a configurable amount of milliseconds. The amount of retries is configurable.

Note that retries are only done if the host was omitted in the request (indicating that any available host is possible to use).

In some error situations, if the service instance (used for XMP communication) was retrieved using the ILocateService interface, the error should be reported back to ILocateService. Table 1 illustrates which error situations this applies for.

Table 1 lists the error situations when a retry is applicable and what is done to select a new host to retry with.

Table 1 Error situations and actions

Situation	**Action**
Communication error	Report the error to Component Register (using *reportServiceError*) and select a new host (using *locateService*) if there are more retries. See [3] for description of methods.

Status code 421 and 502 (see [1])	Report the error to Component Register (using *reportServiceError*) and select a new host (using *locateService*) if there are more retries. See [3] for description of methods.
Status code 450 (see [1])	Get another host from the Component Register (using *getAnotherService*) if there are more retries. See [3] for description of methods.

3.4.2.4 Asynchronous/Synchronous Service Requests

A client can send a request using both asynchronous and synchronous versions of *sendRequest*. Figure 5 illustrates when a synchronous version of *sendRequest* is used.

When an asynchronous version of *sendRequest* is used, it returns a transaction id instead of a service response. The Service Request Manager then executes the request and retrieves the response. The method *isTransactionCompleted* can be used to find out the transaction status of the asynchronous request.

When the transaction has completed or when the client cannot do anything but wait for the service response, the version of *sendRequest* that takes a transaction id as input is used to retrieve the service response. This method is not synchronous as does not return until the service response is ready.

3.4.3 Statistics

The Service Request Manager collects statistics per service and reports the result using the ServiceEnablerInfo interface, see [4].

The following statistics is collected:

- Number of successfully sent requests
- Number of times sending a request failed due to timeout
- Status per service request server. The status is considered down if no connection to the server can be established. Otherwise it is considered up.

3.4.4 Session Data Variables

The following session data variables will be set by the Service Request Manager when starting an application instance:

- Service_Request
  This variable contains information about the currently active service request. The type of this variable is IServiceRequest.

3.4.5 Events

3.4.5.1 Consumed

The Service Request Manager registers itself (using IEventDispatcher, see [2]) as a receiver of the following events:

- Configuration has changed (indicates that configuration should be re-read)

- ApplicationEnded (indicates that an application instance has terminated)

3.4.5.2 Produced

The Service Request Manager does not generate any events.

3.4.6 Thread Safe

The Service Request Manager is thread safe.

4 External Operation Conditions

4.1 Configuration

The following are configurable in the Service Request Manager:

- Timeout of XMP service request
- Amount of retries for an XMP service request
- Globally unique instance name (used to generate XMP ClientId, see [1])

5 Capabilities

This paragraph is intentionally left blank.

6 References

[1] IWD – Extensible Messaging Protocol
2/155 19-1/HDB 101 02 Uen

[2] FS – Execution Engine
3/FS MAS0001 Uen

[3] FS – External Component Register
9/FS MAS0001 Uen

[4] FS - Operate and Maintain
17/FS MAS0001 Uen

7 Terminology

XMP	Extensible Messaging Protocol
SRM	Service Request Manager