FD – Service Request Manager

Content

History

Version	Date	Adjustments
A	2006-10-04	First revision. (MMAWI)
B	2006-12-07	Added section O&M Administration, added Appendix. Also some minor updates. (MMAWI)

1 Introduction

This document describes the Service Request Manager (SRM) see [1]

2 Function Structure

2.1 Overview

SRM depends on the following components/modules:

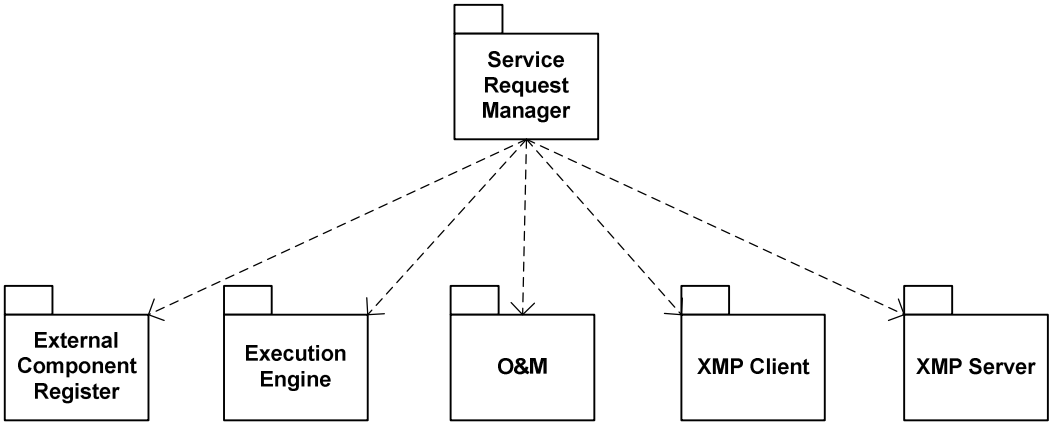

Figure 1 SRM component dependencies

The design of the XMP client and server is based upon the Foundation Java XMP components.

The SRM classes are related according to Figure 2.

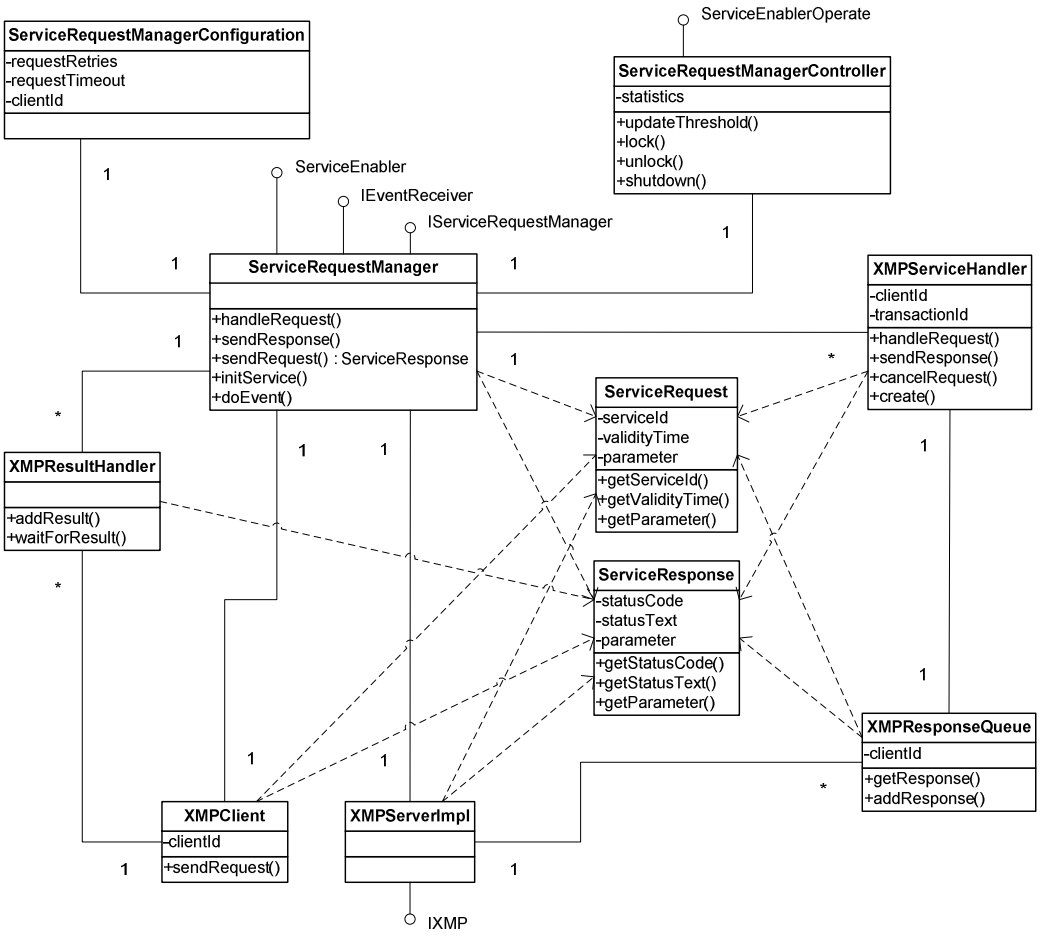

Figure 2 SRM class diagram

The Diagnose Service class is shown in Figure 3.

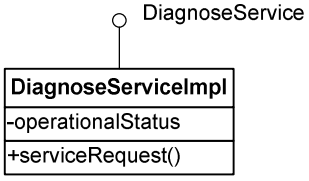

Figure 3 Diagnose Service class diagram

2.2 ServiceRequestManager

ServiceRequestManager is the singleton instance created by Spring Framework. It has an XMPClient instance and an XMPServerImpl instance, which utilizes the XMP implementation from Foundation.

2.3 ServiceRequest

A ServiceRequest holds the required information for a XMP service request. That is the id of the service, the id of the requesting client and some parameters dependent of the type of request.

2.4 ServiceResponse

A ServiceResponse holds the information from a XMP service response. That is the status code, the status text and possibly some parameters dependent of the type of service.

2.5 XMPClient

The XMPClient utilizes the XMP implementation from Foundation. The ServiceRequestManager has an instance of the XMPClient to send requests to an external component in the system, for example MCC.

2.6 XMPServerImpl

The XMPServerImpl utilizes the XMP implementation from Foundation. In basically consists of a HTTP server and an XMP parser. The ServiceRequestManager initializes the server when requested and the server starts listening for XMP messages over HTTP. When a message has been received and parsed, it is sent to ServiceRequestManager for further processing.

2.7 XMPServiceHandler

For every service request the ServiceRequestManager receives from the XMPServerImpl an XMPServiceHandler is created. This handler is responsible for the communication with the application for the current session.

2.8 XMPResponseQueue

The XMPResponseQueue is created by the XMPServerImpl when a service request is received. When a service response is ready it is put in the queue and sent back to the requesting client by the XMPServerImpl.

2.9 XMPResultHandler

The XMPResultHandler is used to manage a response received by the XMPClient. The SRM creates one result handler for each new request, and the XMPClient puts the responses in the result handler for the corresponding request.

2.10 ServiceRequestManagerController

The ServiceRequestManagerController is a singleton instance which is responsible for the O&M parts of the SRM. ServiceRequestManagerController implements the interface ServiceEnablerOperate described in [2] .

ServiceRequestMangerController collects statistics, handles load regulation and handles administrative states for the SRM. The administrative states are handled by a state machine described in section 3.4.1.

2.11 ServiceRequestManagerConfiguration

The ServiceRequestManagerConfiguration class holds the configuration for SRM.

2.12 DiagnoseServiceImpl

To determine the operational state of the SRM, a diagnose service class is used. The DiagnoseServiceImpl is completely separated from SRM with its own Spring configuration.

3 Function Behavior

3.1 Initialization

The SRM can have several provided services, but only one XMP listening point.

At startup, the SRM expects the initService method from the ServiceEnabler interface to be called for each of the provided services. On the first initService call, SRM will initialize the XMP server with the hostname and port given. If initService is called again for another provided service, with another hostname and port, the hostname and port are ignored. All provided services use the same XMP server.

After initialization the administrative state *Closed* is engaged according to section 3.4.

3.2 Receive Service Request

3.2.1 Handle XMP request

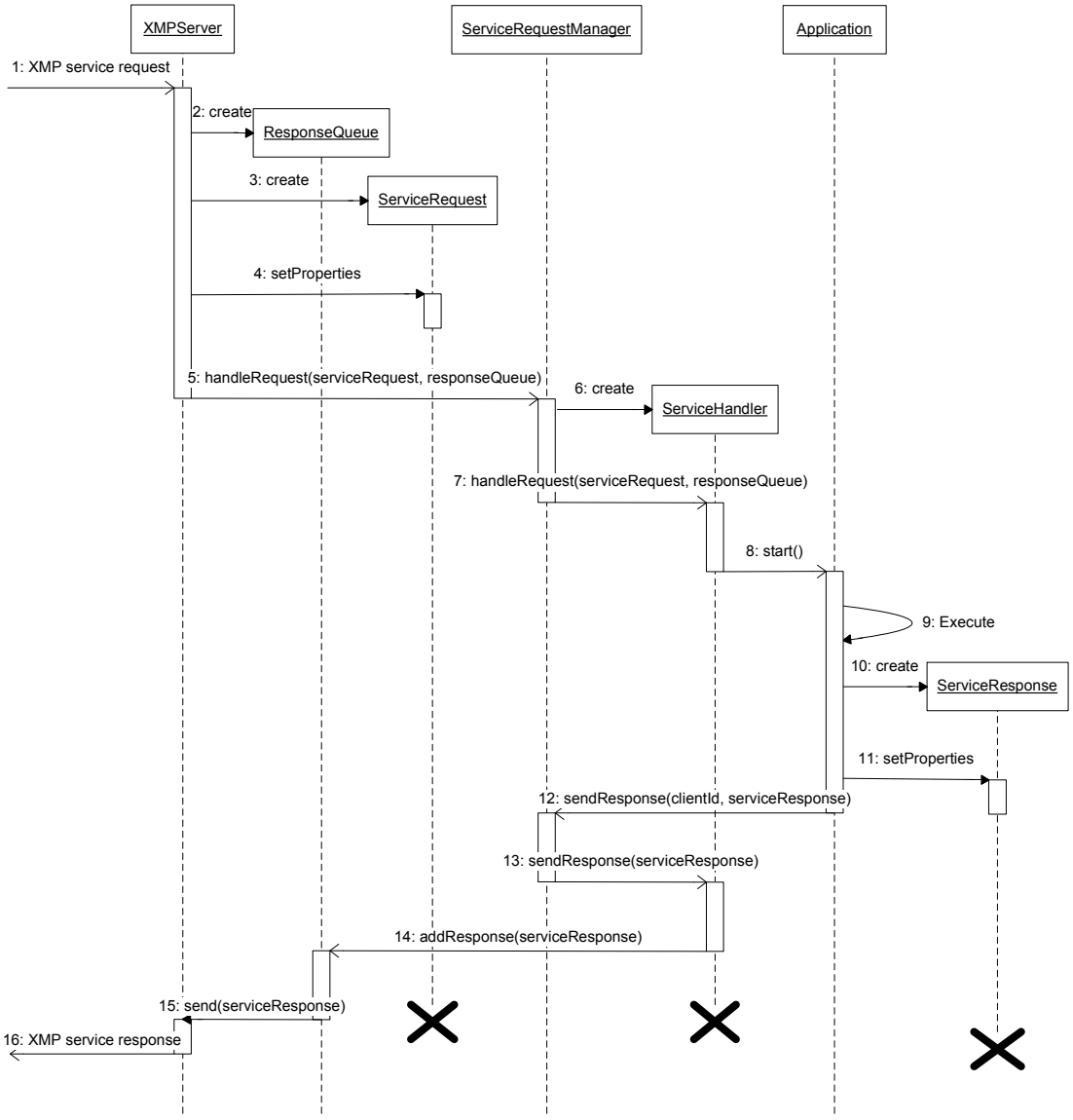

Figure 4 Incoming XMP request

1. The XMP server listens for incoming messages. When a message is received, the XML content is parsed.
2. A response queue for the requesting client is created
3. A ServiceRequest is created and…

4. …the message content is stored in it.
5. The XMP server calls the handleRequest method on the ServiceRequestManager and passes the ServiceRequest and the response queue to it.
6. SRM creates a service handler for this session. The service handler is associated with a session id.
7. The handleRequest method is called on the service handler and the ServiceRequest and response queue is passed to it.
8. The service handler loads the application…
9. …which processes the request.
10. The application creates a ServiceResponse and…
11. …sets the status code and other response information.
12. The application calls the sendResponse method on the SRM…
13. …which uses the provided sessionId to find the service handler used for the session and calls sendResponse on it.
14. The service handler adds the response to the client's response queue.
15. The response is sent to the XMP server…
16. …which returns an XMP response to the client.

3.2.2 Timeout Handling

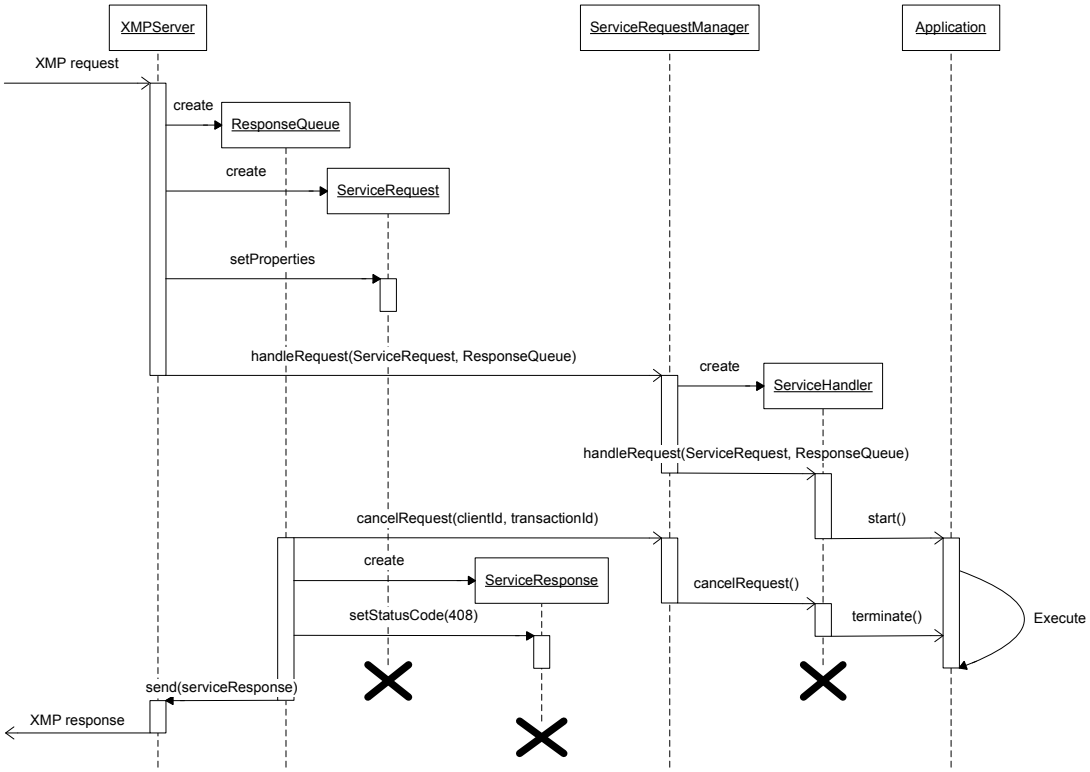

Figure 5 Request Timeout

The XMPResponseQueue is responsible for keeping track of request timeouts. When the queue detects that a transaction's validity timeout is expired, is calls the cancelRequest method on SRM and passes the client id and transaction id to it.

The SRM finds the service handler associated with the client id and transaction id and calls the cancelRequest method on it.

The service handler calls the terminate method on the application.

Concurrently, the response queue creates a ServiceResponse with status code 408 and passes it to the XMPServer which sends the XMP response to the requesting client.

3.3 Send Service Request

3.3.1 Synchronous Send Service Request

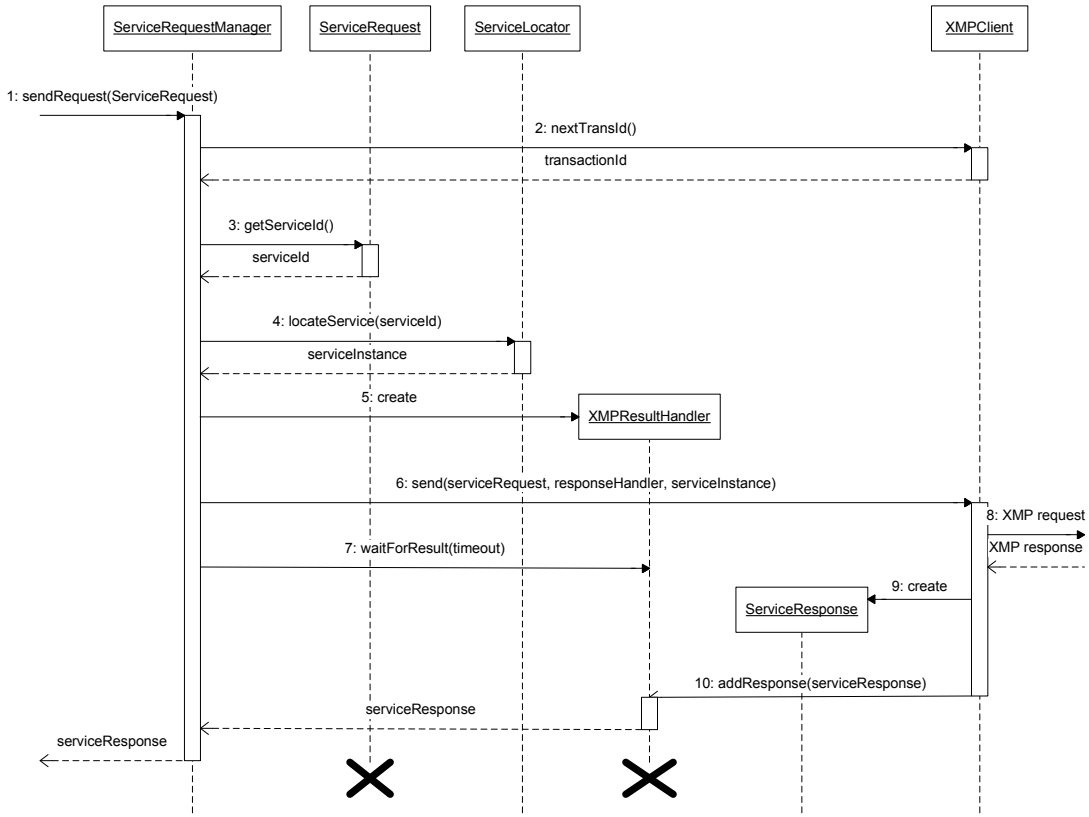

Figure 6 Sending Service Request

1. The sendRequest method is called on the ServiceRequestManager.
2. SRM allocates a transaction id on the XMP client.
3. SRM gets the service id from the received ServiceRequest and…
4. …uses External Component Register to get a host offering the service.
5. A result handler for the request is created.
6. SRM calls the send method on the XMP client and passes the response handler and the host together with the ServiceRequest.
7. SRM calls the waitForResult method to receive the result. A timeout parameter is passed so SRM will only wait until the timeout has expired.
8. The XMP client creates an XMP message of the ServiceRequest and sends it to the host.
9. When the response is received from the host, the XMP client parses the XMP message and creates a ServiceResponse with the content.

10. The ServiceResponse is passed to the response handler which then returns it to the waiting SRM. The SRM returns the response to the caller of sendRequest.

3.3.1.1 Error Handling

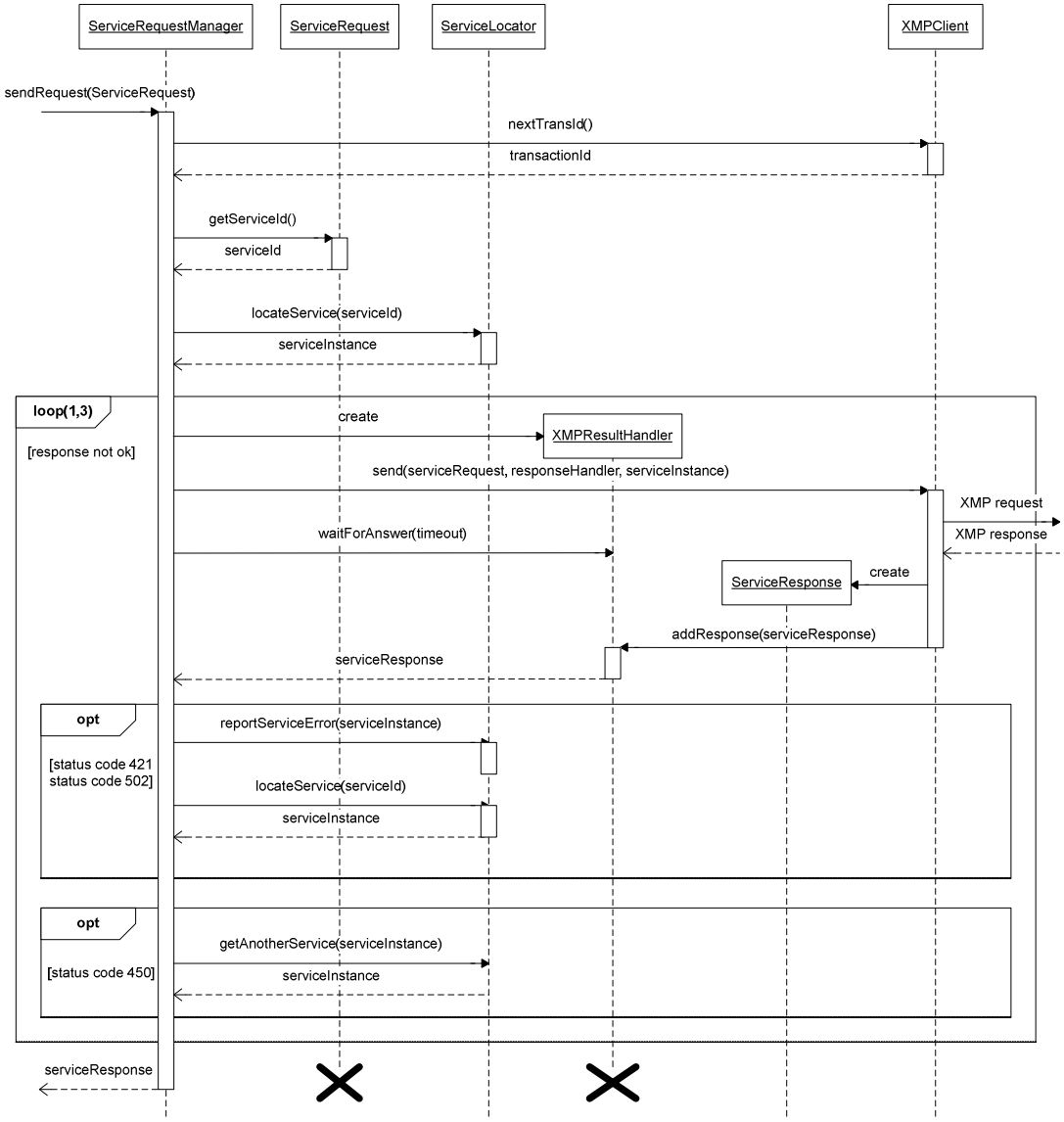

Figure 7 Retry on error response codes

The above sequence diagram shows the behavior when an error response code is received from the remote service host.

On status codes 421 and 502, SRM will report the host to the External Component Register using the reportServiceError method. A new host is then located and the request is sent again.

If status code 450 is received, SRM will just request another host from External Component Register using the getAnotherService method, and retry to send the request.

When all retries are made (configured by the attribute requestretries), SRM will return the last ServiceResponse, e.g status code 421.

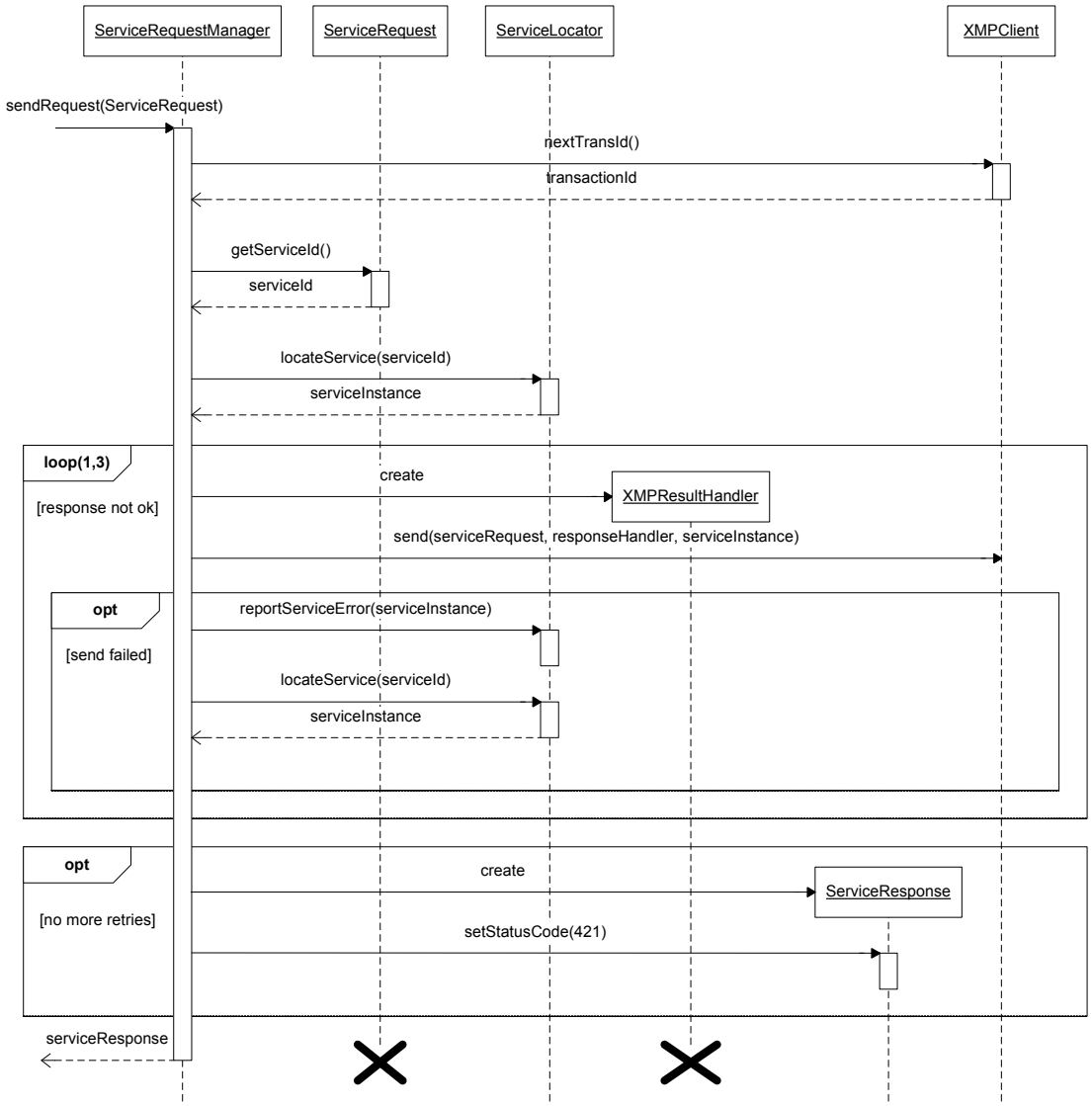

Figure 8 Retry on communication error

The sequence diagram in Figure 8 shows the behavior when communication error occurs.

If the send method fails on the XMP client, SRM will report the service host to External Component Register using the reportServiceError method. A new host is

then located with the locateService method, and the send method is called again on the XMP client.

If all retries fail, an error ServiceResponse is created and returned to the component calling the sendRequest method on SRM. See section 6.1 in Appendix for the error codes in the generated ServiceResponses.

3.3.2 Asynchronous Send Service Request

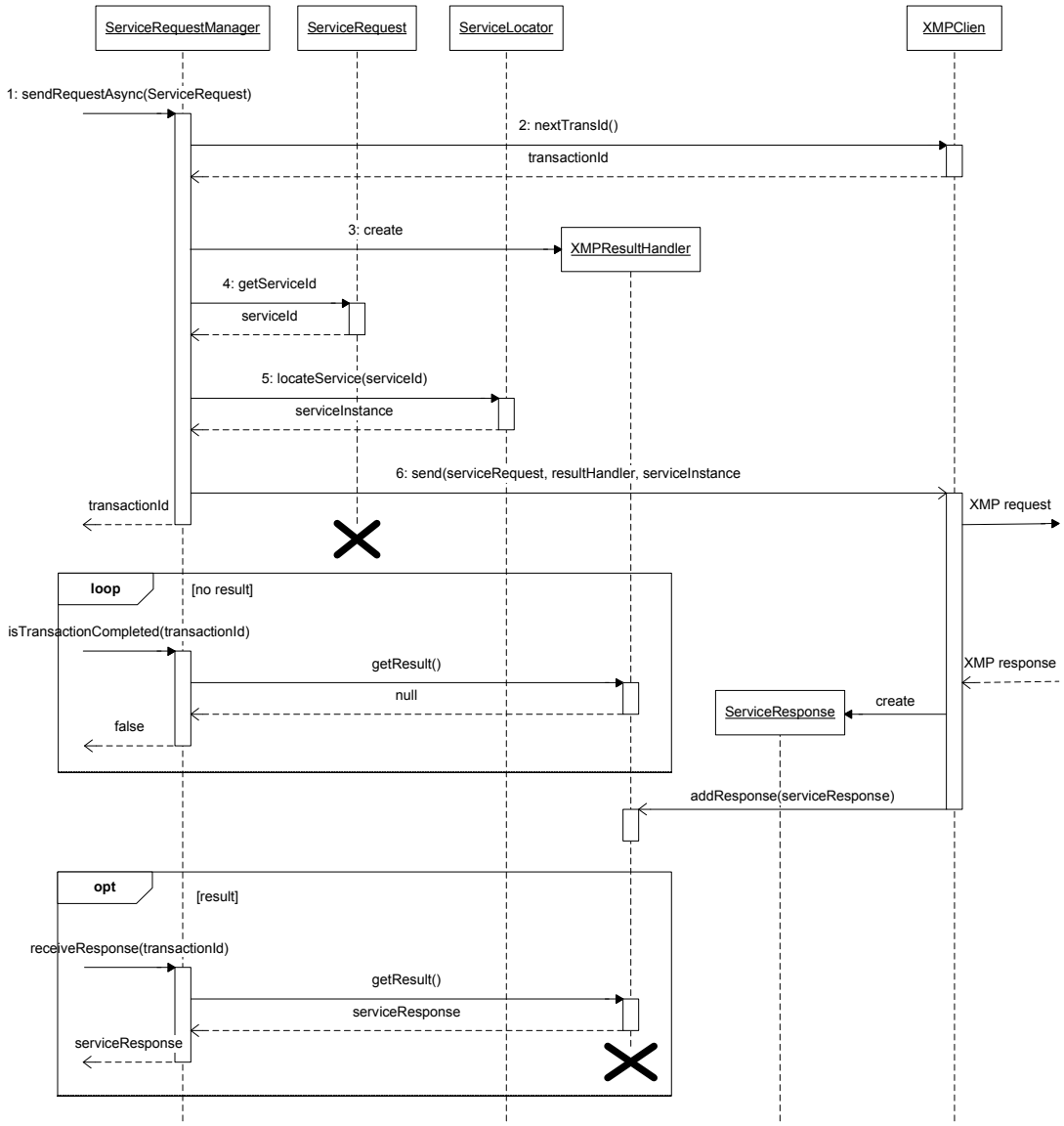

Figure 9 Asynchronous Send Service Request

1. The sendRequestAsync method is called on the SRM.
2. SRM allocates a transaction id on the XMP client.

3. A result handler for the request is created. The result handler is associated with the transaction id.
4. SRM gets the service id from the received ServiceRequest and…
5. …uses External Component Register to get a host offering the service.
6. SRM calls the send method on the XMP client and passes the response handler and the host together with the ServiceRequest and…
7. …returns the transaction id to the component that called sendRequestAsync.

   Now the requesting component starts polling for a result by calling isTransactionCompleted with the transaction id as argument.

   SRM checks for a result in the result handler associated with the transaction id. If no result is received the method returns false.

   When the XMP client receives the response from the host, it parses the XMP message and creates a ServiceResponse with the content. The response is added to the response handler.

   isTransactionCompleted will now return true and the requesting component calls receiveResponse. The ServiceResponse is returned.

3.3.2.1 Error handling

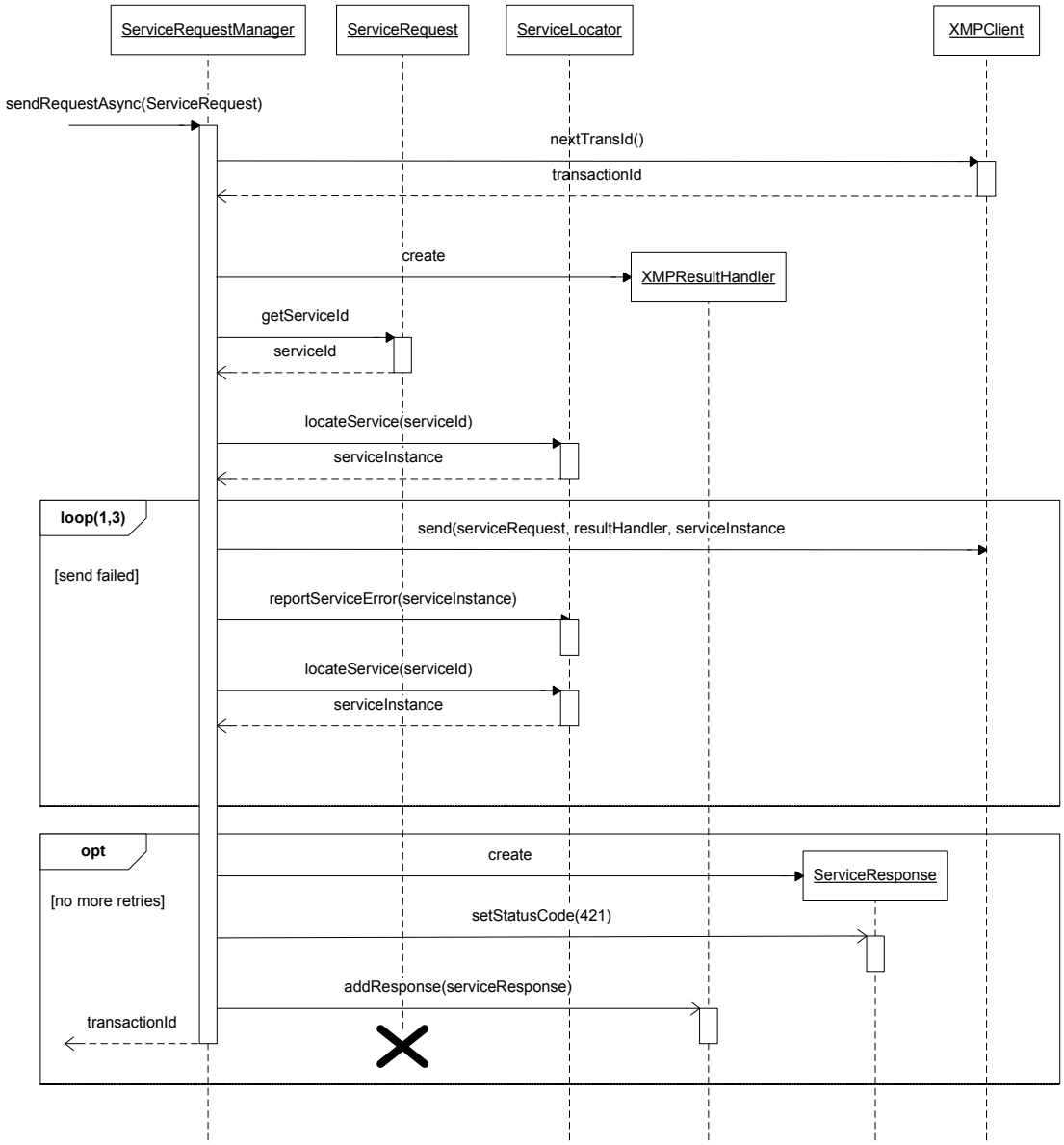

Figure 10 Retry on communication error

The sequence diagram in Figure 10 shows the behavior when communication error occurs on asynchronous send request.

If the send method fails on the XMP client, SRM will report the service host to External Component Register using the reportServiceError method. A new host is then located with the locateService method, and the send method is called again on the XMP client.

If all retries fail, an error ServiceResponse is created and returned to the component calling the sendRequest method on SRM. See section 6.1 in Appendix for the error codes in the generated ServiceResponses.

3.4 O&M Administration

ServiceRequestManagerController implements the ServiceEnablerOperate interface () for O&M administration. A state machine is used to keep track of the administrative state.

3.4.1 Administrative State Machine

This section describes the state machine for the administrative state of SRM.

3.4.1.1 States

The states are shown in Table 1.

Table 1 States in the administrative state machine

State	Description
Closed	In the *Closed* state a previous forced close or unforced close request from O&M has completed and all sessions have terminated. No new XMP requests are accepted by SRM.
ClosingUnforced	In the *ClosingUnforced* state SRM has been requested to unforce close by O&M. No new XMP requests are accepted by SRM, but currently active sessions are allowed to keep running until they are finished.
ClosingForced	In the *ClosingForced* state SRM has been requested to force close by O&M. No new XMP requests are accepted by SRM, and all active sessions are instructed to terminate. A 450 XMP response is sent to the active sessions.
Opened	In the *Opened* state SRM are able to accept traffic from an O&M point of view. However, the load situation may still affect the ability to accept new XMP requests as described in section 3.6.

3.4.1.2 FSA

The complete administrative state machine is illustrated in Figure 11.

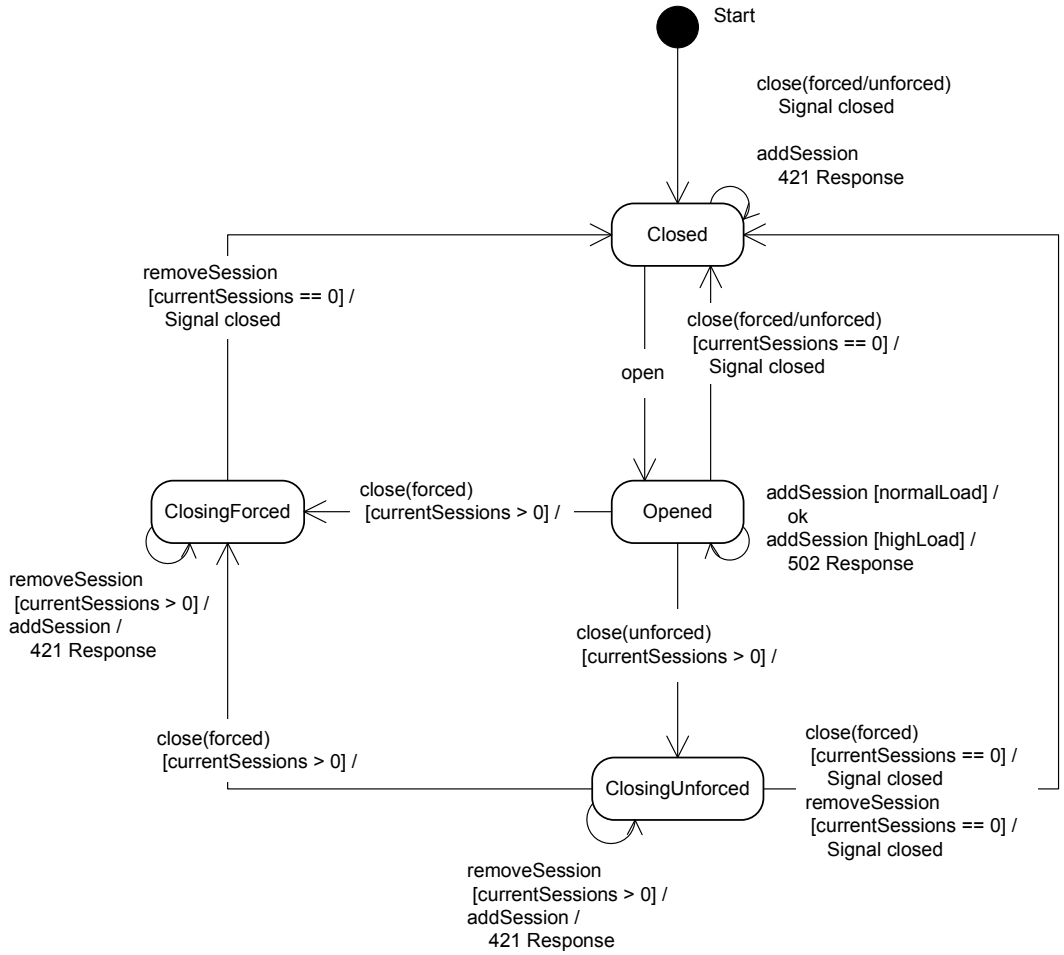

Figure 11 Administrative state machine

3.5 Statistics

ServiceRequestManagerController reports statistics using the ServiceEnablerInfo interface (See [2]). The statistics counter reports successful and failed inbound and outbound service requests.

3.6 Load Regulation

Implementing the ServiceEnablerOperate interface, the ServiceRequestManagerController is responsible for handling load regulation for the SRM. O&M sets the initial threshold, i.e. the maximum number of concurrent XMP requests that may be received, using the updateThreshold method. Since no registration and unregistration of the service is performed when load changes, high watermark and low watermark is ignored.

When the threshold is reached, the SRM will return the status code 502 "Resource limit exceeded" for all incoming XMP requests. The failed request is reported to O&M statistics.

As soon as the load drops below the threshold, the SRM starts accepting requests again.

3.7 Diagnose Service

The operational status of the service provided by SRM is regularly checked using the DiagnoseServiceImpl class.

The diagnose service sends an XMP request with the service id DiagnoseService to the SRM, and expects a service response returned. If a 200 "Service successfully completed" response is returned, the service is considered up. Otherwise, if no response is returned, due to e.g. timeout or communications error, the service is considered down.

4 References

[1] FS – Service Request Manager
10/FS MAS0001 Uen

[2] FS – Operate and Maintain Manager
17/FS MAS0001 Uen

5 Terminology

FSA	Finite State Automaton
SRM	Service Request Manager
XMP	Extensible Messaging Protocol

6 Appendix

6.1 Generated XMP response codes

In some cases when an error has occurred, SRM will generate response codes for incoming and outbound XMP requests. These response codes are listed in Table 2 and Table 3.

Table 2 Response codes for incoming requests

Status Code	Situation
408	*Request Timeout* • The application has not processed the request within the specified validity time.
421	*Service Not Available* • The administrative state is not *Opened*. • An invalid Service ID has been requested.
450	*Request Failed* • The request is terminated because O&M has requested a forced close.
502	*Resource Limit Exceeded* • The maximum number of concurrent sessions has been reached.

Table 3 Response codes for outbound requests

Status Code	Situation
421	*Service Not Available* • Failed to locate a host providing the requested service. • The request timeout has expired (and there are no more retries left). • All request retries have failed.