ETSI ES 202 388-3 V1.1.1 (2005-03)

ETSI Standard

Open Service Access (OSA); Application Programming Interface (API); Test Suite Structure and Test Purposes (TSS&TP); Part 3: Framework (Parlay 4)

Reference

DES/TISPAN-06004-03-OSA

Keywords

API, OSA, TSS&TP

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN).

The present document is part 3 of a multi-part deliverable. Full details of the entire series can be found in part 1 [6].

To evaluate conformance of a particular implementation, it is necessary to have a set of test purposes to evaluate the dynamic behaviour of the Implementation Under Test (IUT). The specification containing those test purposes is called a Test Suite Structure and Test Purposes (TSS&TP) specification.

1 Scope

The present document provides the Test Suite Structure and Test Purposes (TSS&TP) specification for the Framework of the Application Programming Interface for Open Service Access (OSA) defined in ES 202 915-3 [1] in compliance with the relevant requirements, and in accordance with the relevant guidance given in ISO/IEC 9646-2 [4] and ETS 300 406 [5].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

[1] ETSI ES 202 915-3: "Open Service Access (OSA); Application Programming Interface (API); Part 3: Framework (Parlay 4)".

[2] ETSI ES 202 363: "Open Service Access (OSA); Application Programming Interface (API); Implementation Conformance Statement (ICS) proforma specification; (Parlay 4)".

[3] ISO/IEC 9646-1: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts".

[4] ISO/IEC 9646-2: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 2: Abstract Test Suite specification".

[5] ETSI ETS 300 406: "Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology".

[6] ETSI ES 202 388-1: "Open Service Access (OSA); Application Programming Interface (API); Test Suite Structure and Test Purposes (TSS&TP); Part 1: Overview (Parlay 4)".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 202 915-3 [1], ISO/IEC 9646-1 [3], ISO/IEC 9646-2 [4] and the following apply:

abstract test case: Refer to ISO/IEC 9646-1 [3].

Abstract Test Method (ATM): Refer to ISO/IEC 9646-1 [3].

Abstract Test Suite (ATS): Refer to ISO/IEC 9646-1 [3].

Implementation Under Test (IUT): Refer to ISO/IEC 9646-1 [3].

Lower Tester (LT): Refer to ISO/IEC 9646-1 [3].

Implementation Conformance Statement (ICS): Refer to ISO/IEC 9646-1 [3].

ICS proforma: Refer to ISO/IEC 9646-1 [3].

Implementation eXtra Information for Testing (IXIT): Refer to ISO/IEC 9646-1 [3].

IXIT proforma: Refer to ISO/IEC 9646-1 [3].

Test Purpose (TP): Refer to ISO/IEC 9646-1 [3].

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AS	Access Session
ATM	Abstract Test Method
ATS	Abstract Test Suite
FTA	Framework To Application
FTE	Framework To Enterprise
FTS	Framework To Service
FW	FrameWork
ICS	Implementation Conformance Statement
IUT	Implementation Under Test
IXIT	Implementation eXtra Information for Testing
LT	Lower Tester
SCF	Switching Control Function
SUT	System Under Test
TP	Test Purpose
TSM	Trust and Security Management
TSS	Test Suite Structure

4 Test Suite Structure (TSS) for Framework

Framework (FW)

- Framework Access Session (AS)
  - Trust and Security Management (TSM)
- Framework To Application (FTA)
  - Service discovery
  - Service agreement management
  - Integrity management
  - Event notification
- Framework To Service (FTS)
  - Service registration
  - Service instance lifecycle management
  - Service discovery
  - Integrity management
  - Event notification
- Framework To Enterprise operator (FTE)
  - Service subscription

5 Test Purposes (TP) for Framework

For each test requirement a TP is defined.

5.1 TP naming convention

TPs are numbered, starting at 01, within each group. Groups are organized according to the TSS. Additional references are added to identify the actual test suite (see table 1).

Table 1: TP identifier naming convention scheme

Identifier: <suite_id>_<group>_<nnn>		
<suite_id>	= IUT name:	"FW" for **F**rame**W**ork SCF
<group>	= group number:	two character field representing the group reference according to TSS
<nn>	= sequential number:	(01-99)

5.2 Source of TP definition

The TPs are based on ES 202 915-3 [1].

5.3 Test strategy

As the base standard ES 202 915-3 [1] contains no explicit requirements for testing, the TPs were generated as a result of an analysis of the base standard and the ICS specification ES 202 363 [2].

The TPs are only based on conformance requirements related to the externally observable behaviour of the IUT and are limited to conceivable situations to which a real implementation is likely to be faced (see ETS 300 406 [5]).

5.4 TPs for the Framework

All ICS items referred to in this clause are as specified in ES 202 363 [2] unless indicated otherwise by another numbered reference.

All parameters specified in method calls are valid unless specified.

The procedures to trigger the SCF to call methods in the application are dependant on the underlying network architecture and are out of the scope of the present document. Those method calls are preceded by the words "Triggered action".

5.4.1 Framework Access Session API

5.4.1.1 Trust and Security Management (TSM)

Methods/Test Nr	01	02	03	04	05	06	07	08	09
initiateAuthentication	X	X	X	X	X	X	X	X	X
requestAccess	X	X	X	X	X	X	X	X	X
selectEncryptionMethod		X	X	X	X	X	X	X	
Authenticate		X	X	X	X	X	X	X	
abortAuthentication						X			
AuthenticationSucceeded	X	X	X	X	X		X	X	
obtainInterface		X	X	X				X	X
obtainInterfaceWithCallback					X				
endAccess							X		
listInterfaces		X							
releaseInterface								X	

Test FW_AS_TSM_01

Summary: Initial Access for Trusted Parties, no authentication is needed, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.1.1.1.

Precondition: Authentication not required by IUT.

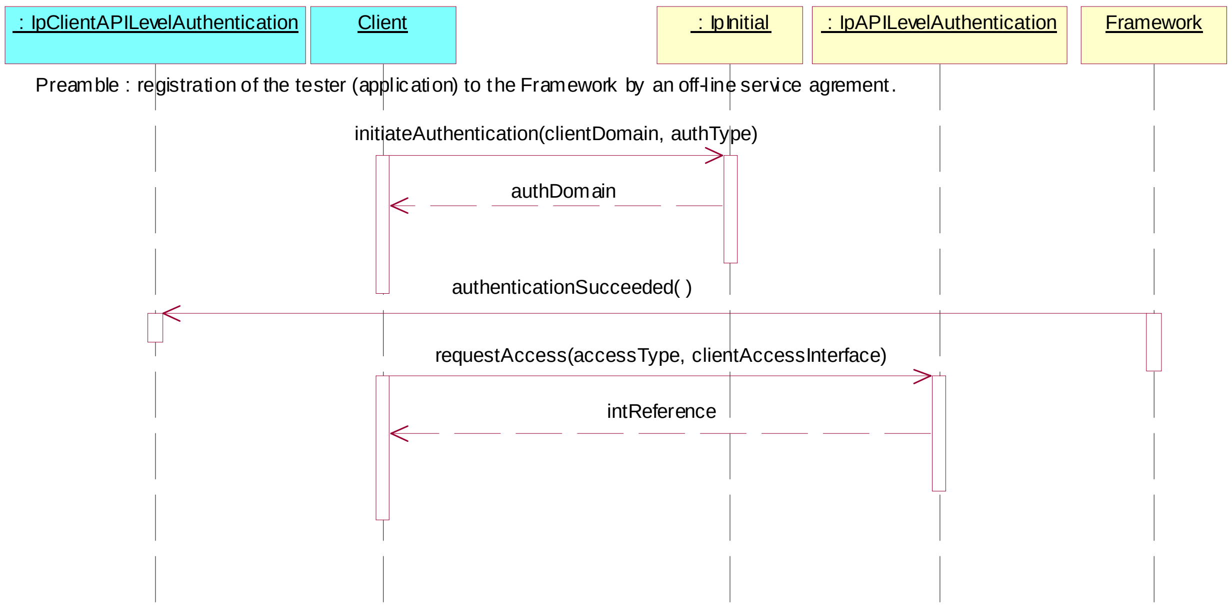
Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication()** on **IpInitial** interface.
   Parameters: clientDomain, authType
   Check: valid value of TpAuthDomain is returned

2. Triggered action: cause IUT to call **authenticationSucceeded()** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

3. Method call **requestAccess()** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
Framework
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, authType)
authDomain
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference

Test FW_AS_TSM_02

Summary: API level authentication, FW authenticates the client only, all methods, successful, use of **listInterface** method to get the name of supported interfaces.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: API level authentication required by IUT, listInterfaces supported.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of **TpEncryptionCapability** is returned

3. Triggered action: cause IUT to call authenticate method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

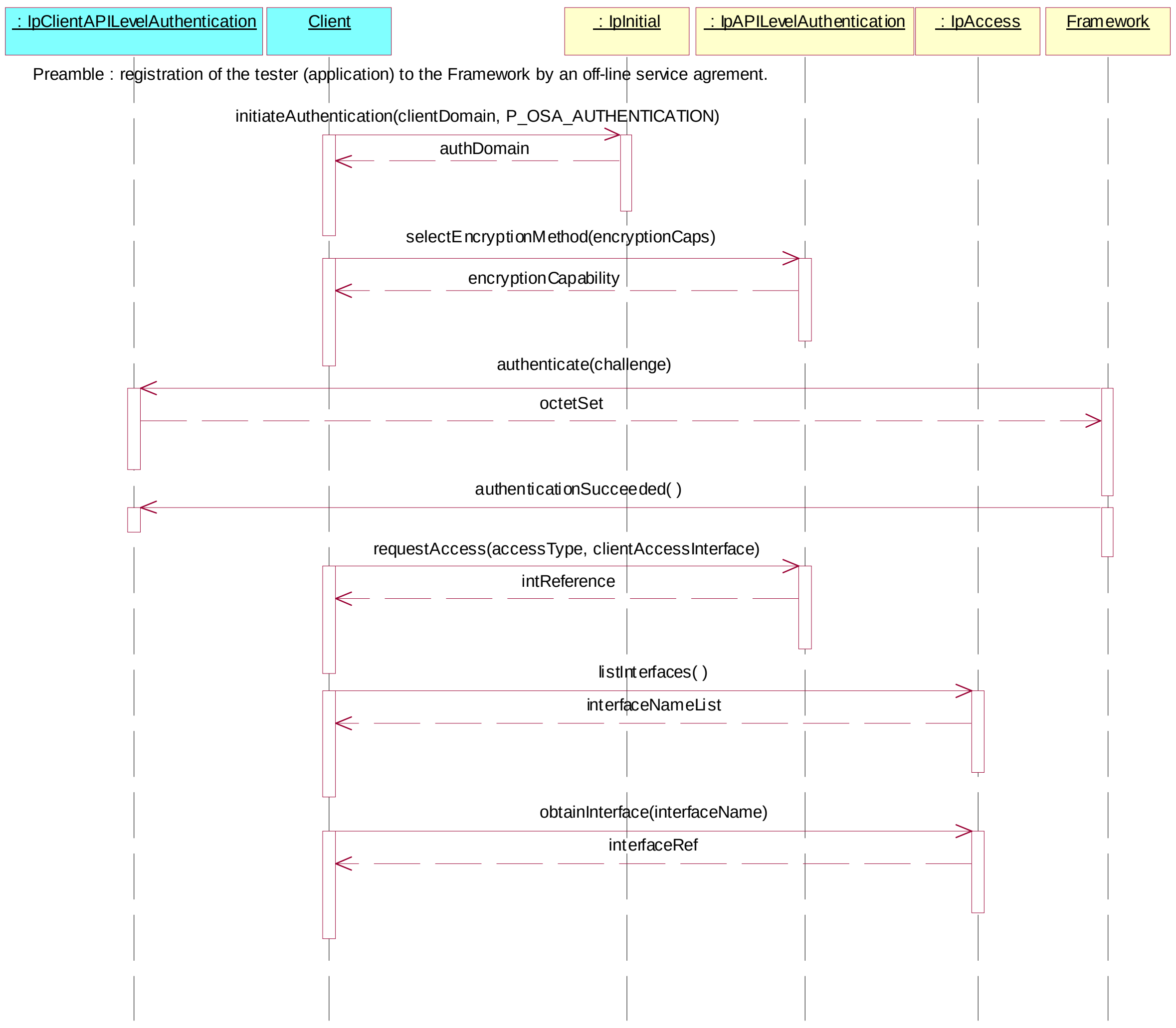
NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: **accessType**, **clientAccessInterface**
   Check: valid value of **TpInterfaceRef** is returned

6. Method call **listInterfaces** on **IpAccess** interface.
   Parameters: none
   Check: valid value of **TpInterfaceNameList** is returned

7. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: **interfaceName** (suggest use of P_DISCOVERY)
   Check: valid value of **IpInterfaceRef** is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
Preamble : registration of the tester (application) to the Framework by an off-line service agreement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
listInterfaces( )
interfaceNameList
obtainInterface(interfaceName)
interfaceRef

Test FW_AS_TSM_03

Summary: API level authentication, FW and client authenticate mutually, all methods, successful.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: **clientDomain**, authType=P_OSA_AUTHENTICATION
   Check: valid value of **TpAuthDomain** is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: **encryptionCaps**
   Check: valid value of **TpEncryptionCapability** is returned

3. Triggered action: cause IUT to call **authenticate** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE 1: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **authenticate** on **IpAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of **TpOctetSet** is returned

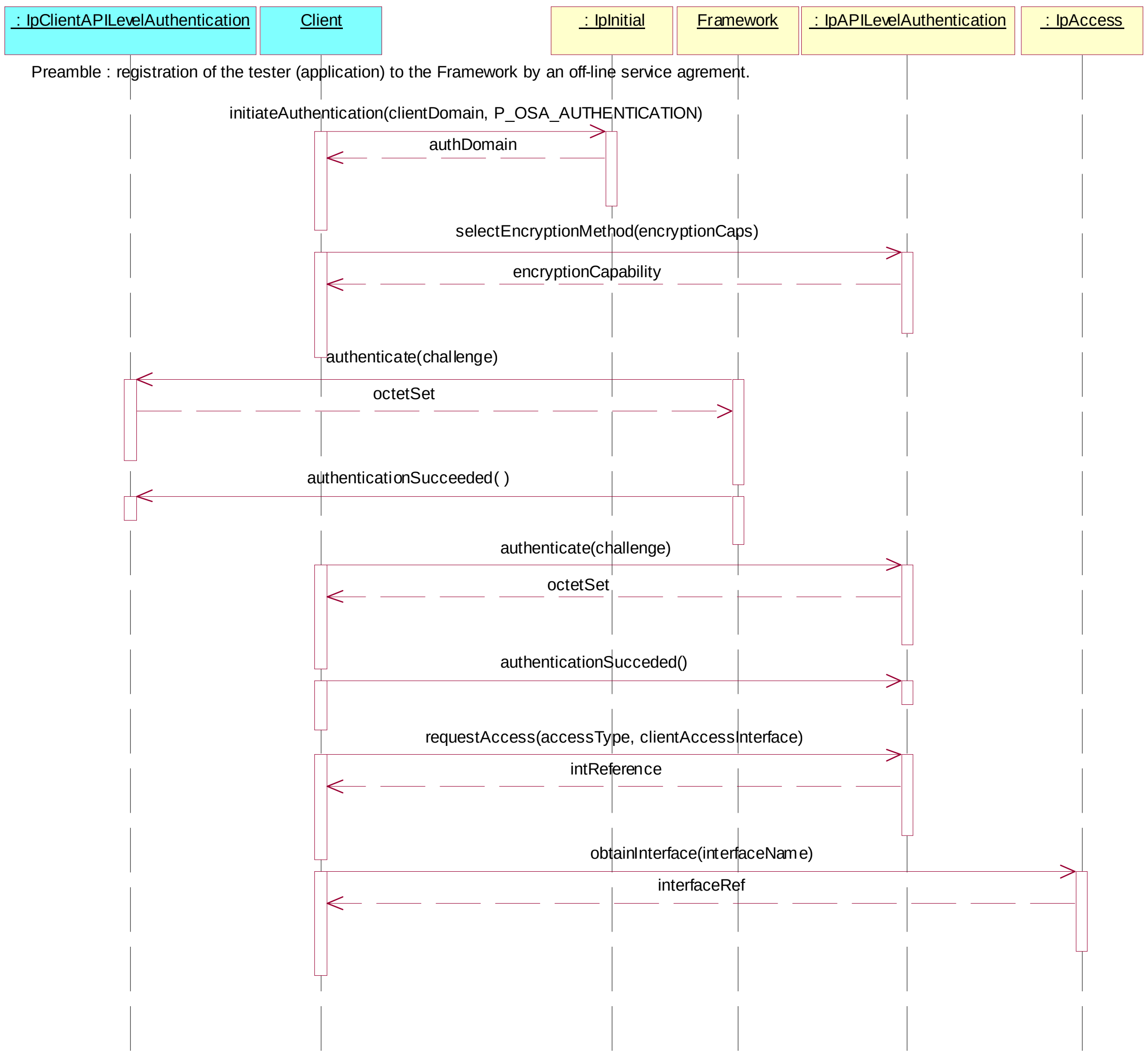
NOTE 2: This method may be repeated with different challenges as required by the tester.

6. Method call **authenticationSucceeded** on **IpAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

NOTE 3: The method calls 5. and 6. may interleave between the method calls 3. and 4.

7. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: **accessType**, **clientAccessInterface**
   Check: valid value of **TpInterfaceRef** is returned

8. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: **interfaceName** (suggest use of P_DISCOVERY)
   Check: valid value of **IpInterfaceRef** is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
authenticate(challenge)
octetSet
authenticationSucceded()
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(interfaceName)
interfaceRef

Test FW_AS_TSM_04

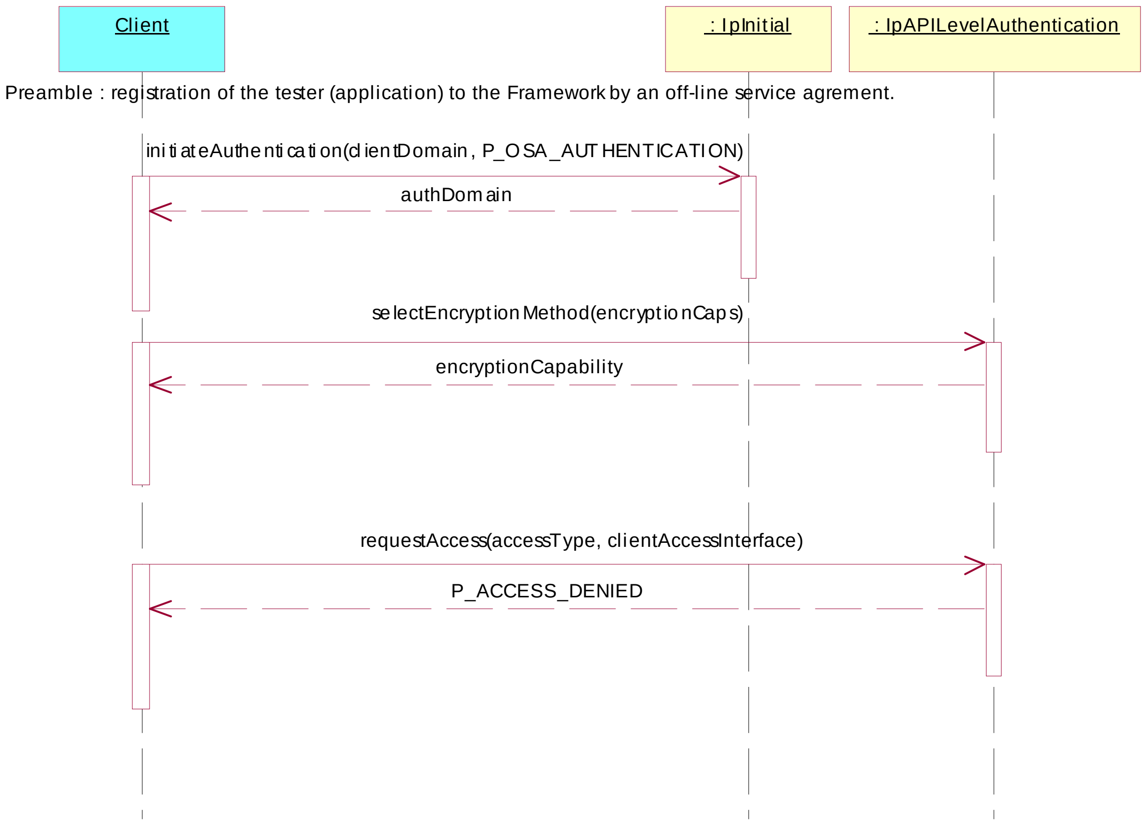
Summary:	API level authentication, FW authenticates the client only, **unsuccessful call of requestAccess** method (preceding **authenticationSucceeded** method call).
Reference:	ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.
Precondition:	Authentication required by IUT.
Preamble:	Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

3. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: P_ACCESS_DENIED is returned


Client
: IpInitial
: IpAPILevelAuthentication
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
requestAccess(accessType, clientAccessInterface)
P_ACCESS_DENIED

Test FW_AS_TSM_05

Summary: API level authentication, FW authenticates the client only, use of **obtainInterfaceWithCallback**, successful.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

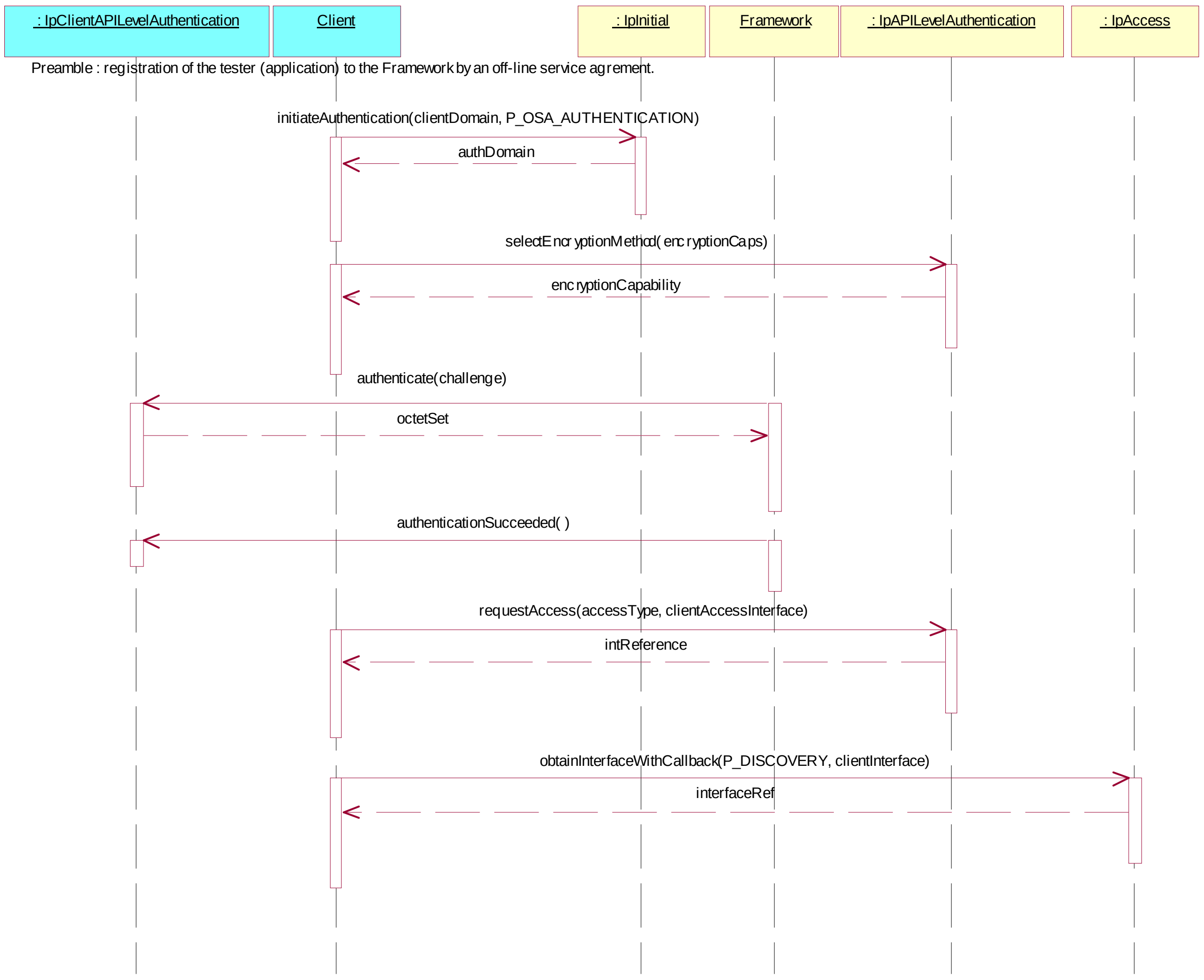
3. Triggered action: cause IUT to call **authenticate** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface

6. Method call **obtainInterfaceWithCallback** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_FAULT_MANAGER), clientInterface
   Check: valid value of IpInterfaceRef is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterfaceWithCallback(P_DISCOVERY, clientInterface)
interfaceRef

Test FW_AS_TSM_06

Summary: API level authentication, FW authenticates the client only and receives **abortAuthentication**, unsuccessful.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

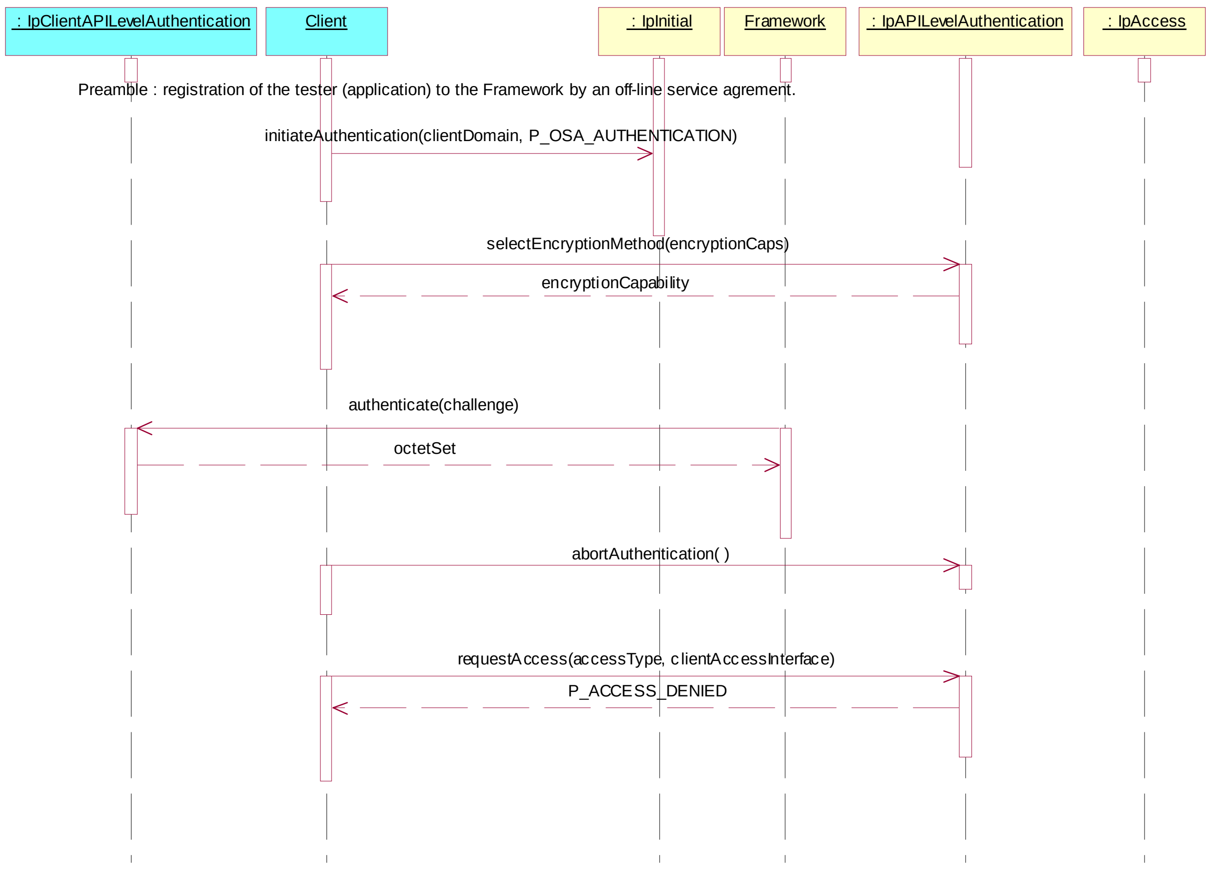
1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

3. Triggered action: cause IUT to call **authenticate** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

4. Method call **abortAuthentication()** on **IpAPILevelAuthentication** interface.
   Parameters: none
   Check: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: P_ACCESS_DENIED value is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
abortAuthentication( )
requestAccess(accessType, clientAccessInterface)
P_ACCESS_DENIED

Test FW_AS_TSM_07

Summary: API level authentication, FW authenticates the client only, successful, checks **endAccess** method.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

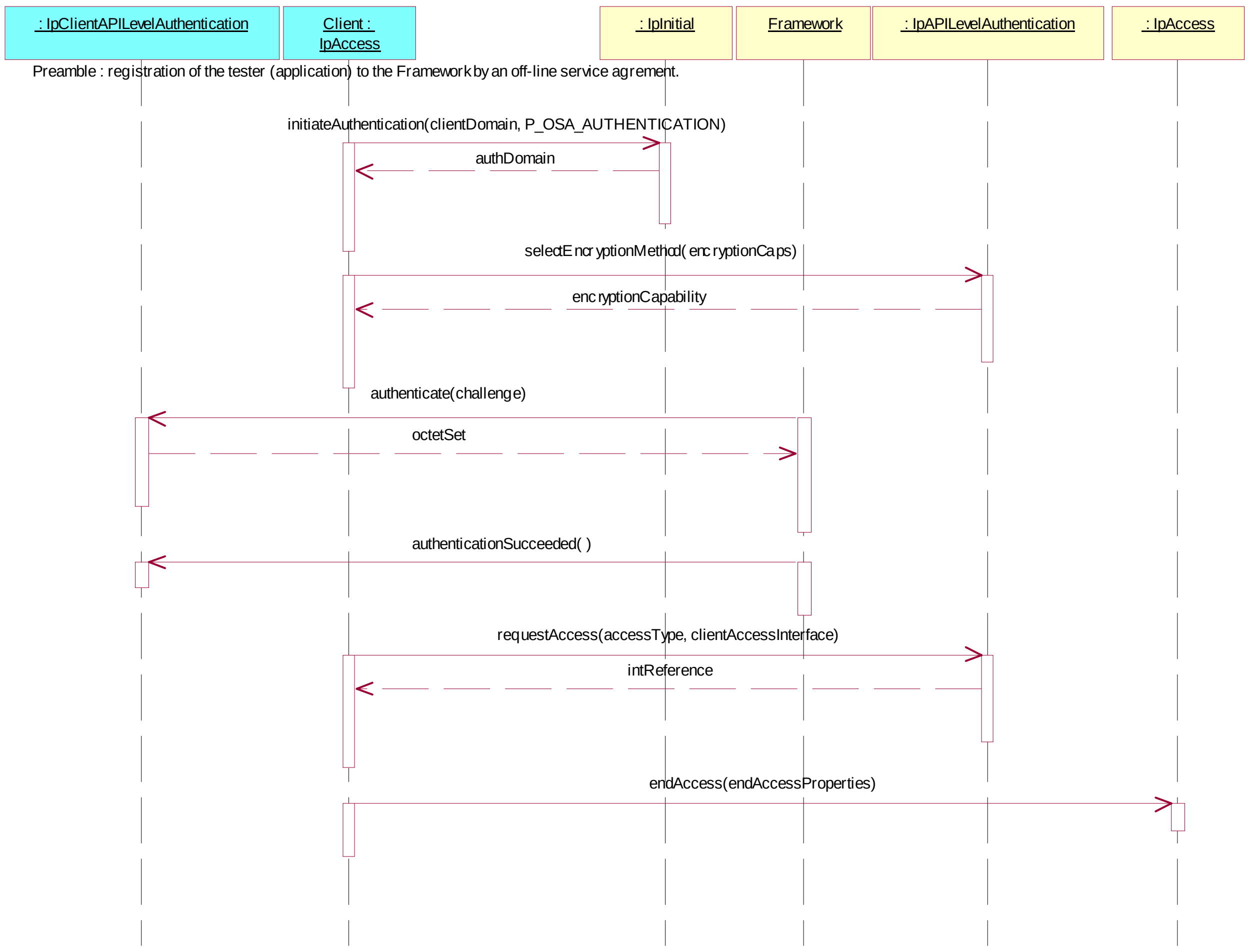
3. Triggered action: cause IUT to call **authenticate** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **endAccess** on **IpAccess** interface.
   Parameters: endAccessProperties
   Check: no exception is returned.


: IpClientAPILevelAuthentication
Client : IpAccess
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
endAccess(endAccessProperties)

Test FW_AS_TSM_08

Summary: API level authentication, FW authenticates the client only, all methods, successful, use of **releaseInterface** method.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

2. Method call **selectEncryptionMethod** on **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps
   Check: valid value of TpEncryptionCapability is returned

3. Triggered action: cause IUT to call **authenticate** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

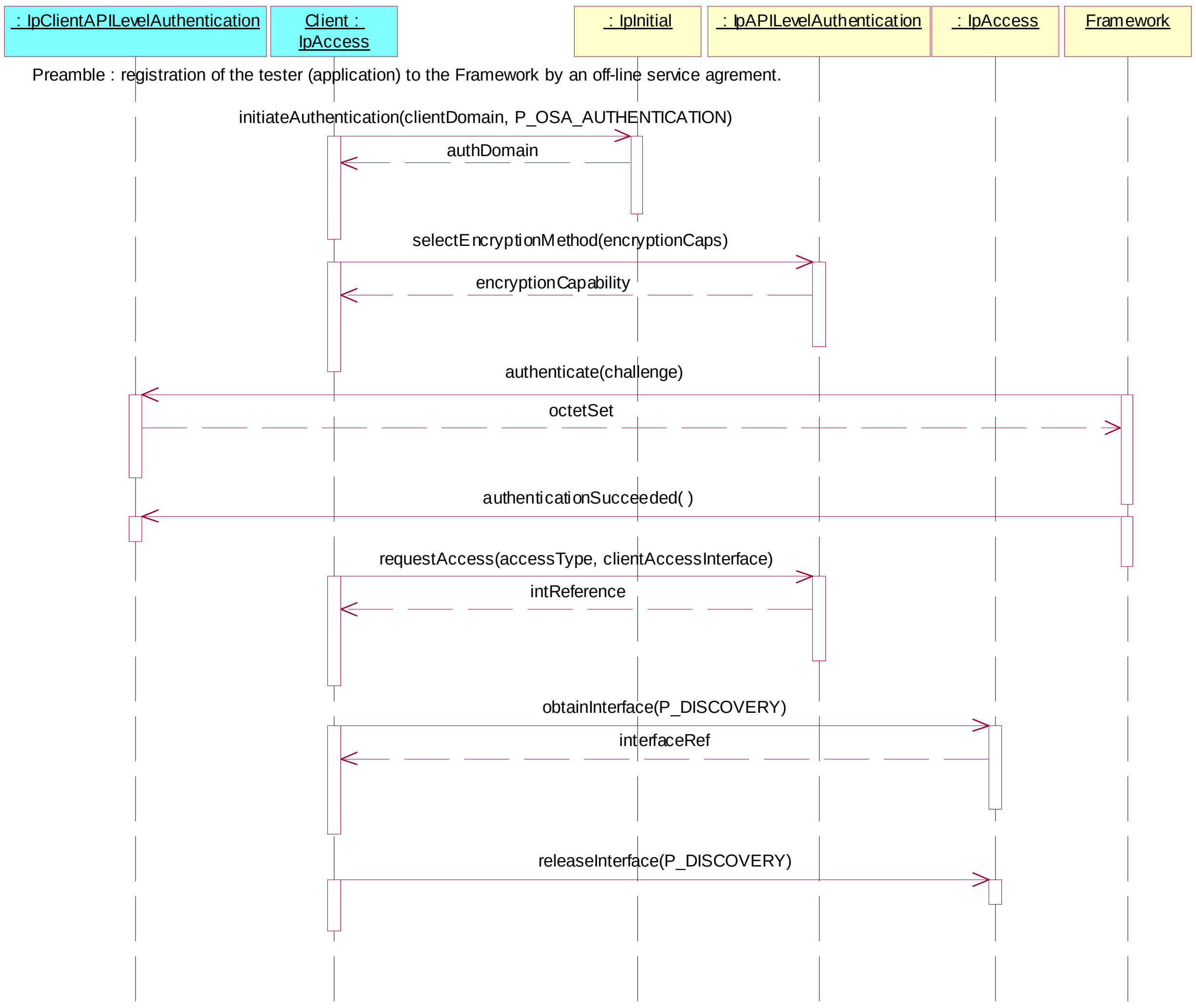
NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_DISCOVERY)
   Check: valid value of IpInterfaceRef is returned

7. Method call **releaseInterface** on **IpAccess** interface.
   Parameters: interfaceName (same value as method call nr 6)
   Check: none


: IpClientAPILevelAuthentication
Client : IpAccess
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(P_DISCOVERY)
interfaceRef
releaseInterface(P_DISCOVERY)

Test FW_AS_TSM_09

Summary: Authentication, using Underlying Distribution Technology Mechanism, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.1.1.3.

Precondition: Underlying authentication supported.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

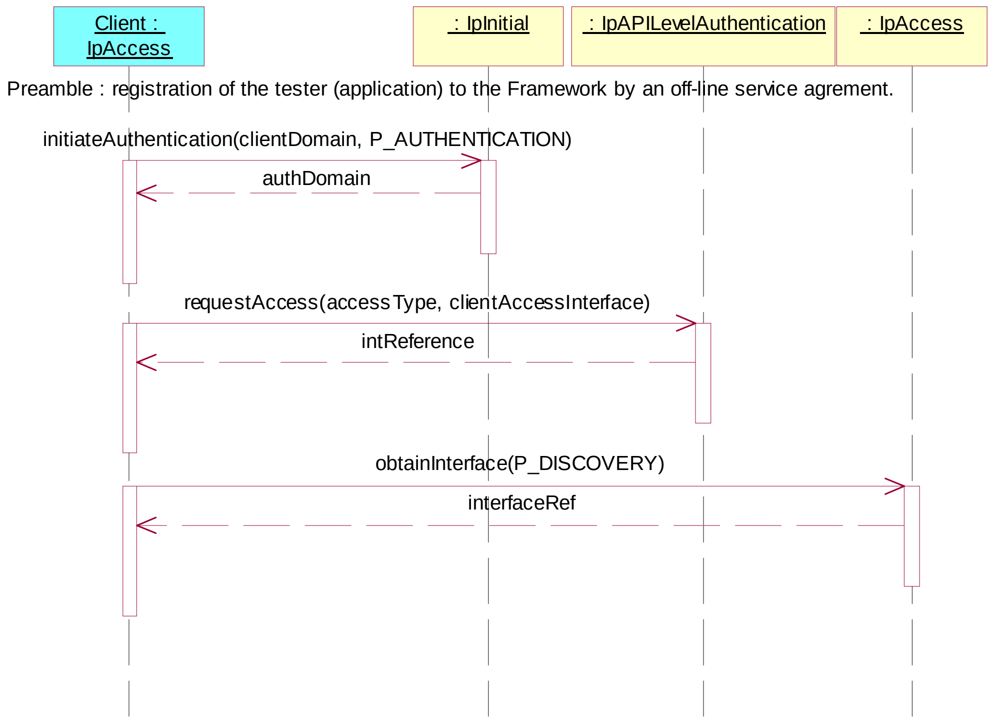
Test Sequence:

1. Perform underlying authentication between tester and IUT.

2. Method call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_ AUTHENTICATION
   Check: valid value of TpAuthDomain is returned

3. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

4. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_DISCOVERY)
   Check: valid value of IpInterfaceRef is returned


Client : IpAccess
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthentication(clientDomain, P_AUTHENTICATION)
authDomain
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(P_DISCOVERY)
interfaceRef

Test FW_AS_TSM_10

Summary: Initial Access for Trusted Parties, no authentication is needed, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.1.1.1.

Precondition: Authentication not required by IUT.

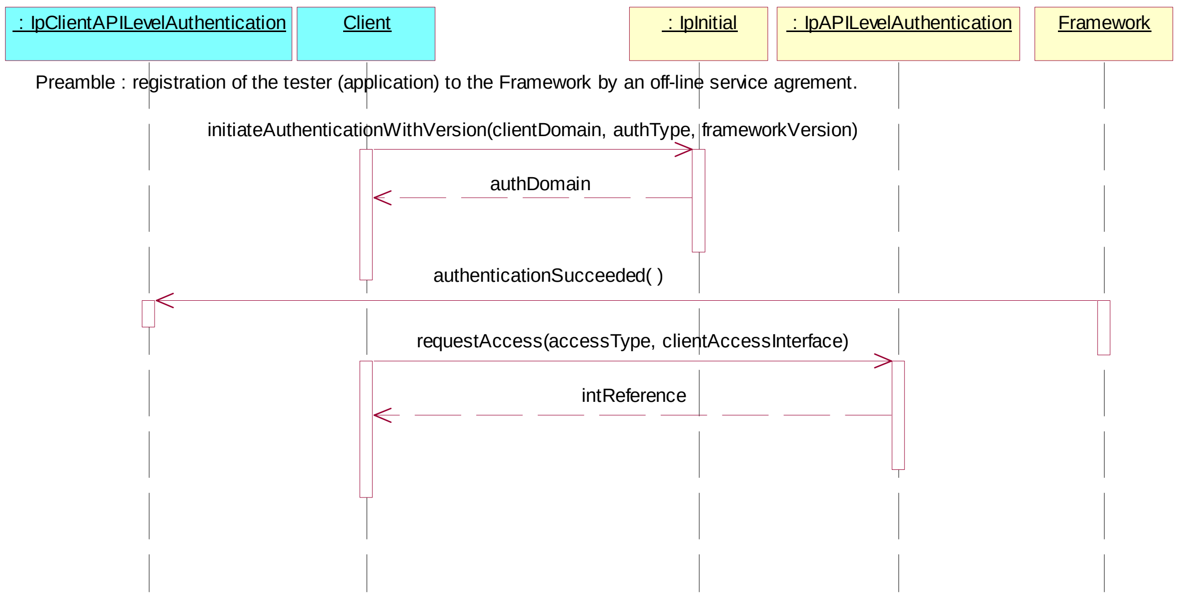
Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion()** on **IpInitial** interface.
   Parameters: clientDomain, authType, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Triggered action: cause IUT to call **authenticationSucceeded()** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

3. Method call **requestAccess()** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned



: IpClientAPILevelAuthentication
Client
: IpInitial
: IpAPILevelAuthentication
Framework
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference

Test FW_AS_TSM_11

Summary:	API level authentication, FW authenticates the client only, all methods, successful, use of **listInterface** method to get the name of supported interfaces.
Reference:	ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.
Precondition:	API level authentication required by IUT, listInterfaces supported.
Preamble:	Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

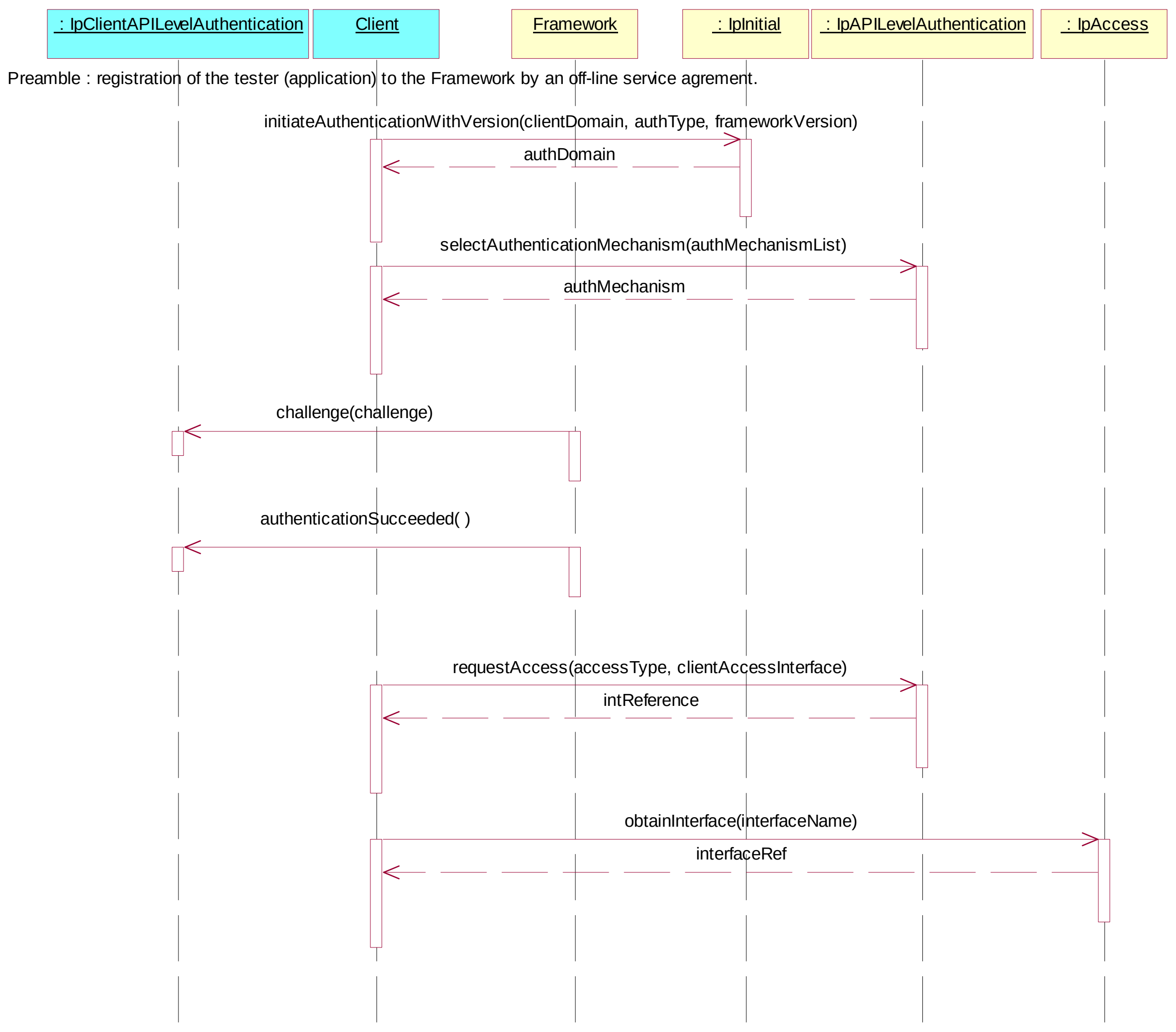
3. Triggered action: cause IUT to call **challenge** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: **accessType**, **clientAccessInterface**
   Check: valid value of **TpInterfaceRef** is returned

6. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: **interfaceName** (suggest use of P_DISCOVERY)
   Check: valid value of **IpInterfaceRef** is returned


: IpClientAPILevelAuthentication
Client
Framework
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(interfaceName)
interfaceRef

Test FW_AS_TSM_12

Summary: API level authentication, FW authenticates the client only, all methods, successful, use of **listInterface** method to get the name of supported interfaces.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: API level authentication required by IUT, listInterfaces supported.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

3. Triggered action: cause IUT to call **challenge** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE 1: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **challenge** method on **IpAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned.

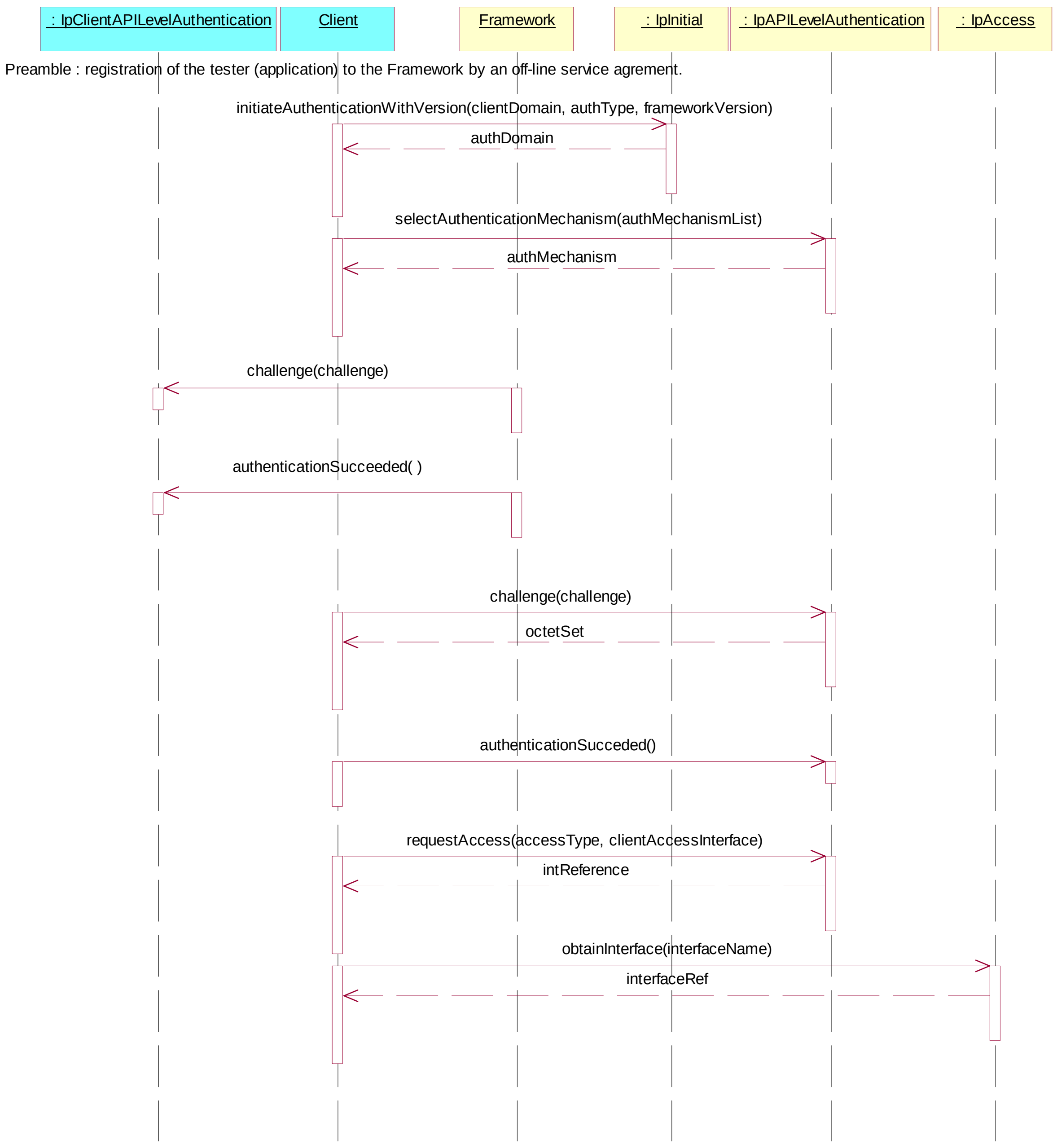
NOTE 2: This method may be repeated with different challenges as required by the tester.

6. Method call **authenticationSucceeded** method on **IpAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned.

NOTE 3: The method calls 5. and 6. may interleave between the method calls 3. and 4.

7. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: **accessType**, **clientAccessInterface**
   Check: valid value of **TpInterfaceRef** is returned

8. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: **interfaceName** (suggest use of P_DISCOVERY)
   Check: valid value of **IpInterfaceRef** is returned


: IpClientAPILevelAuthentication
Client
Framework
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
authenticationSucceeded( )
challenge(challenge)
octetSet
authenticationSucceded()
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(interfaceName)
interfaceRef

Test FW_AS_TSM_13

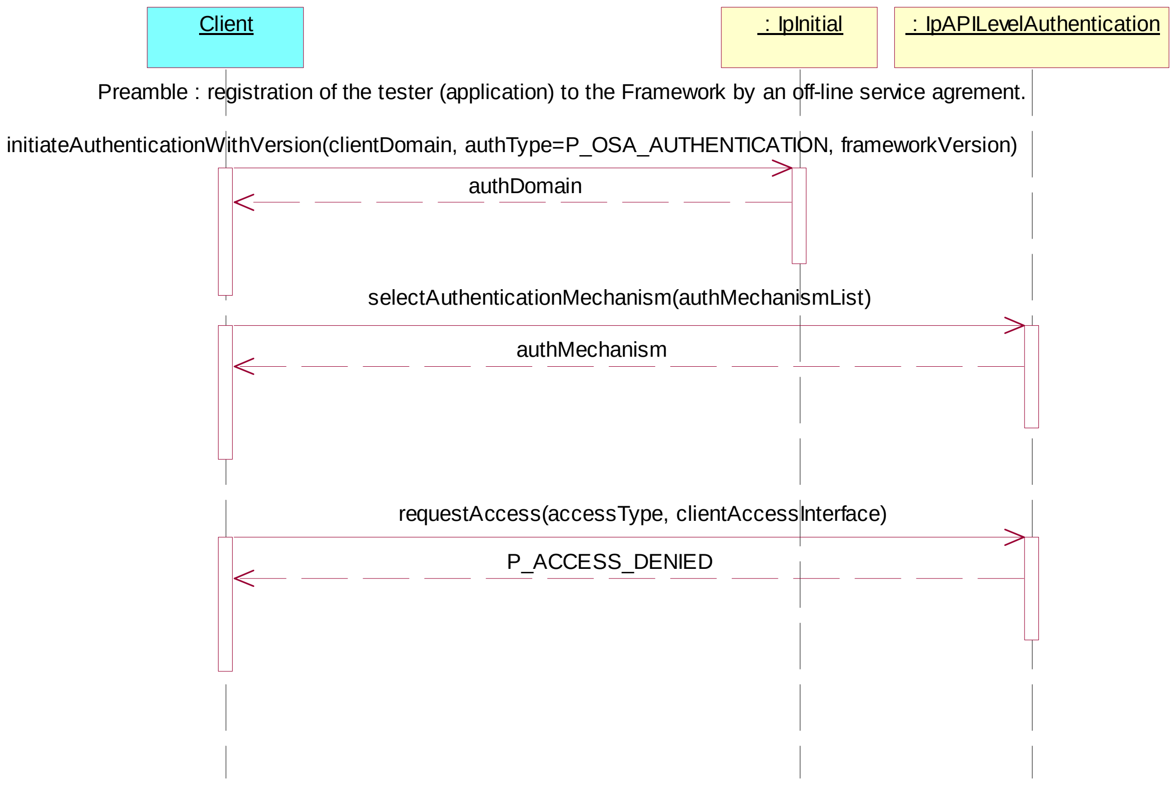
Summary:	API level authentication, FW authenticates the client only, **unsuccessful call of requestAccess** method (preceding **authenticationSucceeded** method call).
Reference:	ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.
Precondition:	Authentication required by IUT.
Preamble:	Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

3. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: P_ACCESS_DENIED is returned


Client
: IpInitial
: IpAPILevelAuthentication
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
requestAccess(accessType, clientAccessInterface)
P_ACCESS_DENIED

Test FW_AS_TSM_14

Summary: API level authentication, FW authenticates the client only, use of **obtainInterfaceWithCallback**, successful.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

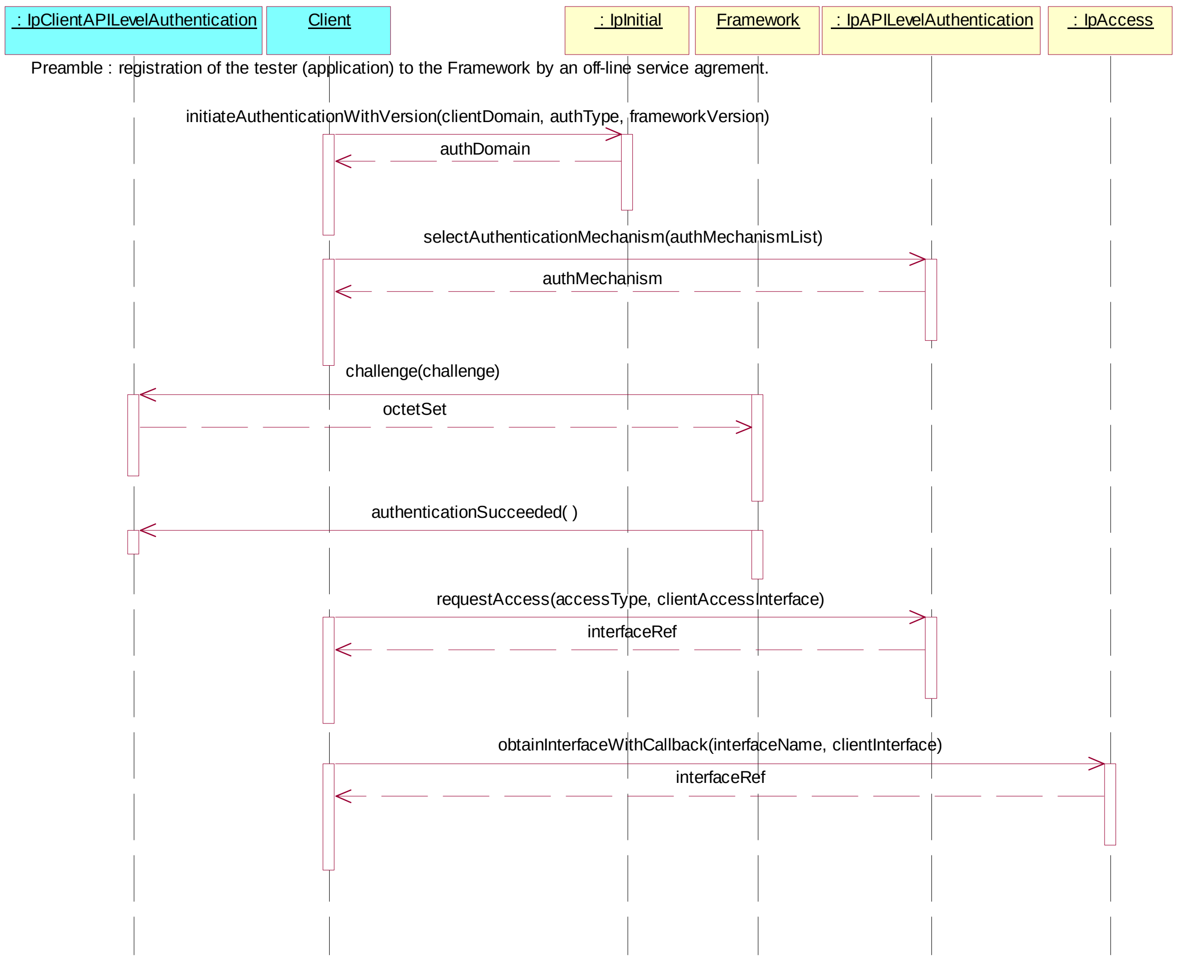
3. Triggered action: cause IUT to call **challenge** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface

6. Method call **obtainInterfaceWithCallback** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_FAULT_MANAGER), clientInterface
   Check: valid value of IpInterfaceRef is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
interfaceRef
obtainInterfaceWithCallback(interfaceName, clientInterface)
interfaceRef

Test FW_AS_TSM_15

Summary: API level authentication, FW authenticates the client only and receives **abortAuthentication**, unsuccessful.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

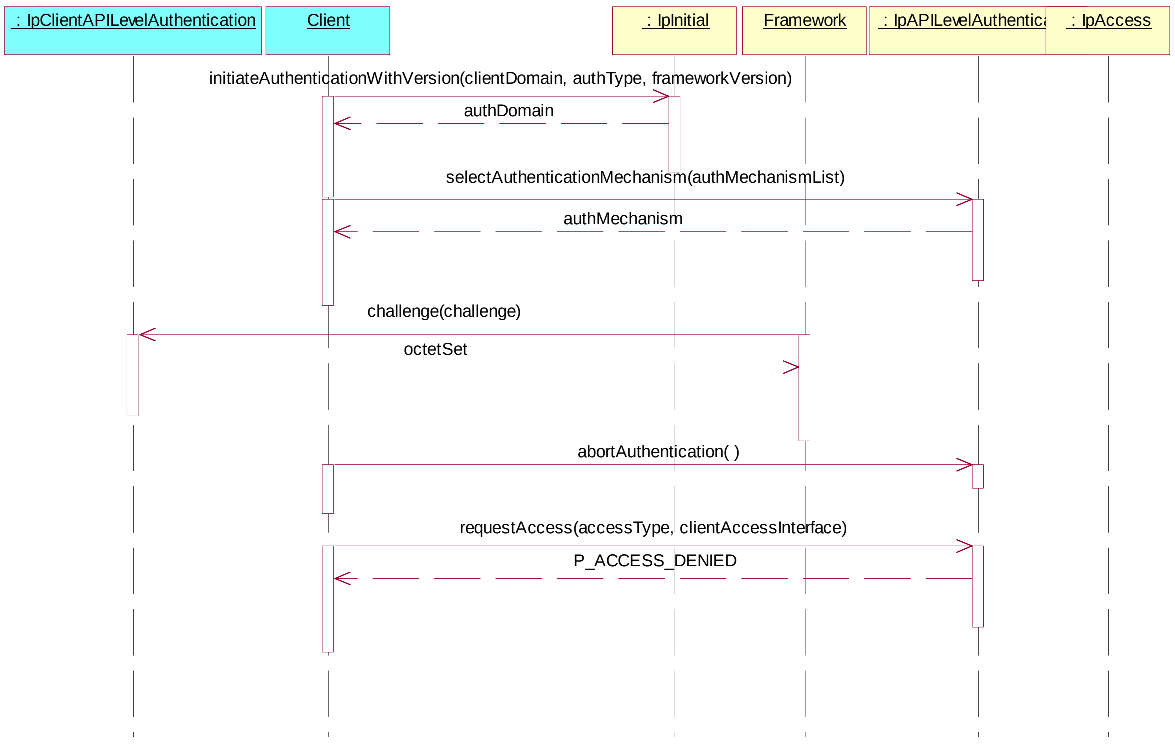
1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

3. Triggered action: cause IUT to call **challenge** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

4. Method call **abortAuthentication()** on **IpAPILevelAuthentication** interface.
   Parameters: none
   Check: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: P_ACCESS_DENIED value is returned


: IpClientAPILevelAuthentication
Client
: IpInitial
Framework
: IpAPILevelAuthentica
: IpAccess
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
octetSet
abortAuthentication( )
requestAccess(accessType, clientAccessInterface)
P_ACCESS_DENIED

Test FW_AS_TSM_16

Summary: API level authentication, FW authenticates the client only, successful, checks **terminateAccess** method.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

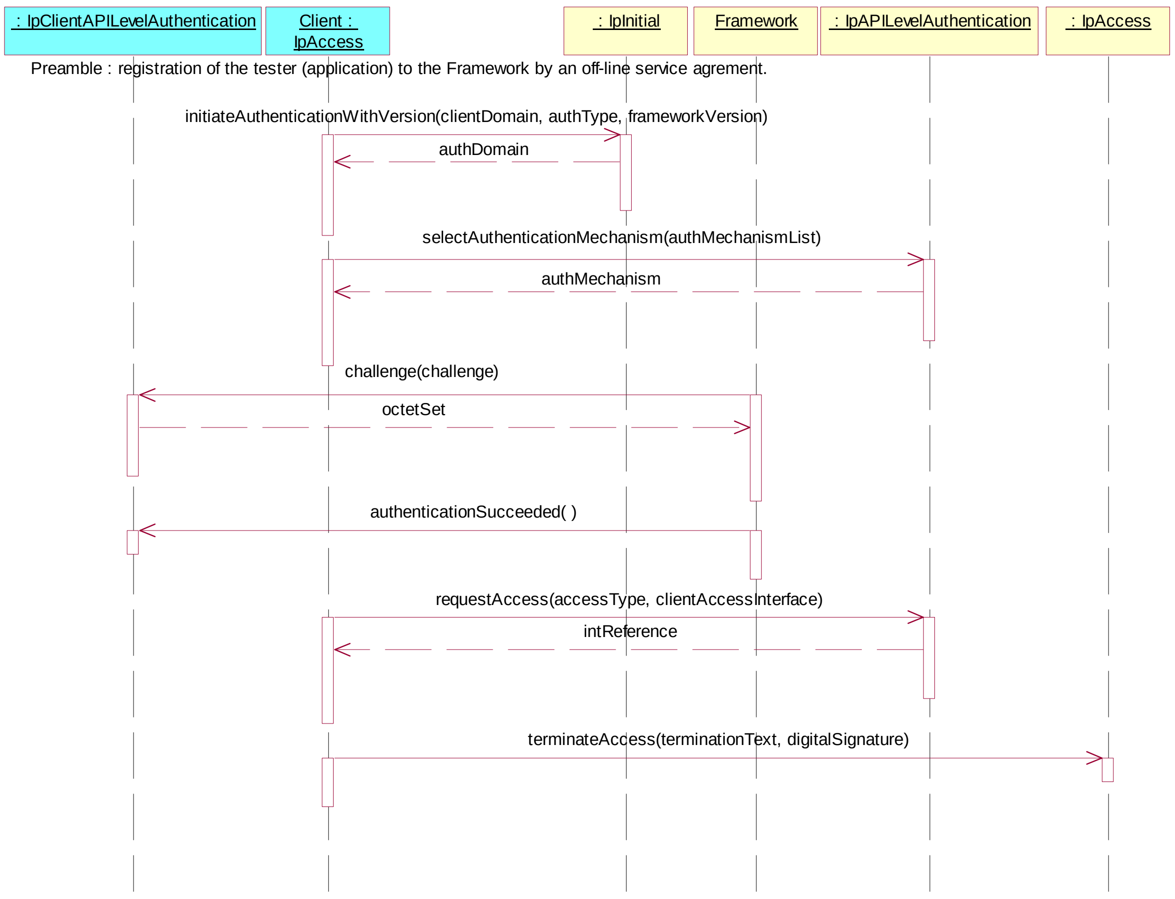
3. Triggered action: cause IUT to call **challenge** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **terminateAccess** on **IpAccess** interface.
   Parameters: terminationText, digitalSignature
   Check: no exception is returned.


: IpClientAPILevelAuthentication
Client : IpAccess
: IpInitial
Framework
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
terminateAccess(terminationText, digitalSignature)

Test FW_AS_TSM_17

Summary: API level authentication, FW authenticates the client only, all methods, successful, use of **relinquishInterface** method.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Precondition: Authentication required by IUT.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

2. Method call **selectAuthenticationMechanism** on **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList
   Check: valid value of **TpAuthMechanism** is returned

3. Triggered action: cause IUT to call **challenge** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: challenge

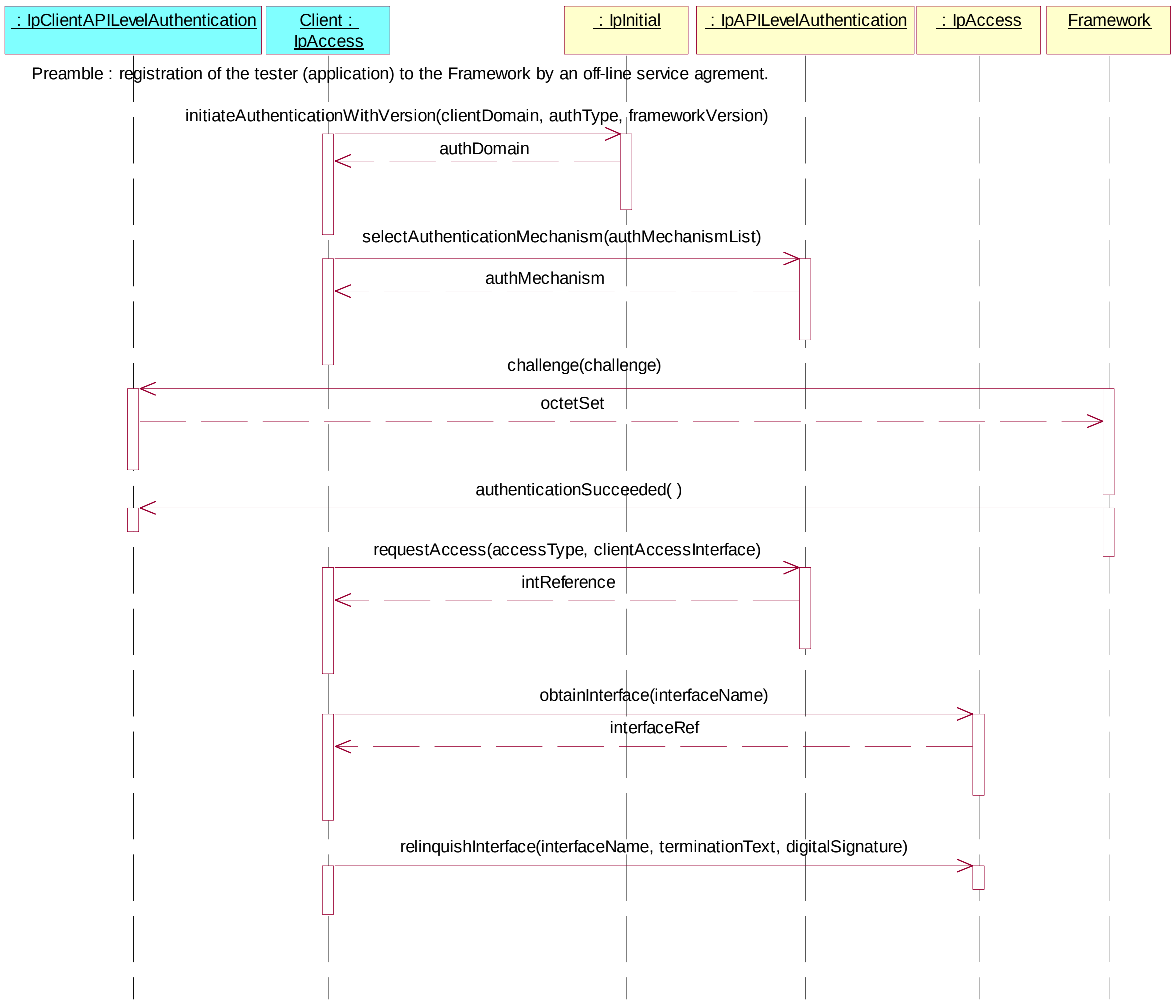
NOTE: This method may be repeated with different challenges as required by the IUT.

4. Triggered action: cause IUT to call **authenticationSucceeded** method on the tester's (Application) **IpClientAPILevelAuthentication** interface.
   Parameters: none

5. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

6. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_DISCOVERY)
   Check: valid value of IpInterfaceRef is returned

7. Method call **relinquishInterface** on **IpAccess** interface.
   Parameters: interfaceName (same value as method call nr 6), terminationText, digitalSignature
   Check: none


: IpClientAPILevelAuthentication
Client : IpAccess
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Framework
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
octetSet
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(interfaceName)
interfaceRef
relinquishInterface(interfaceName, terminationText, digitalSignature)

Test FW_AS_TSM_18

Summary:	Authentication, using Underlying Distribution Technology Mechanism, all methods, successful.
Reference:	ES 202 915-3 [1], clause 6.1.1.3.
Precondition:	Underlying authentication supported.
Preamble:	Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

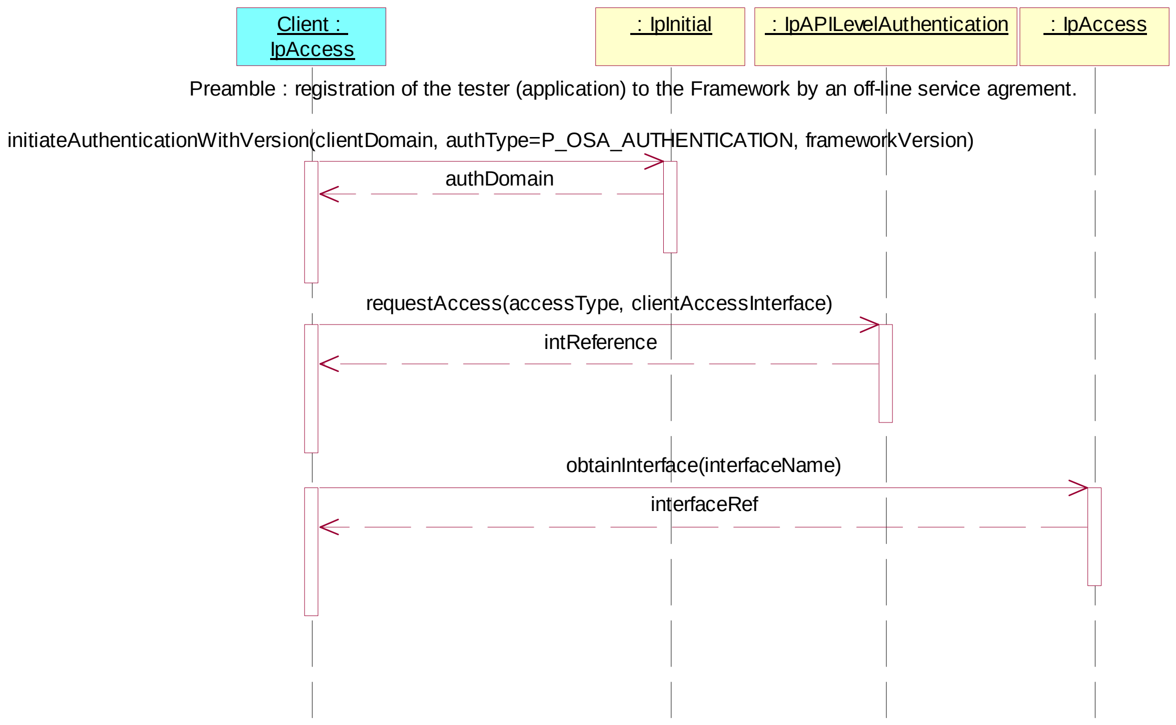
Test Sequence:

1. Perform underlying authentication between tester and IUT.

1. Method call **initiateAuthenticationWithVersion** on **IpInitial interface**.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion
   Check: valid value of TpAuthDomain is returned

3. Method call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
   Check: valid value of TpInterfaceRef is returned

4. Method call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName (suggest use of P_DISCOVERY)
   Check: valid value of IpInterfaceRef is returned


Client : IpAccess
: IpInitial
: IpAPILevelAuthentication
: IpAccess
Preamble : registration of the tester (application) to the Framework by an off-line service agrement.
initiateAuthenticationWithVersion(clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion)
authDomain
requestAccess(accessType, clientAccessInterface)
intReference
obtainInterface(interfaceName)
interfaceRef

5.4.2 Framework to Application API

5.4.2.1 Service Discovery (SD)

Test FW_FA_SD_01

Summary: **IpServiceDiscovery** all methods, successful.

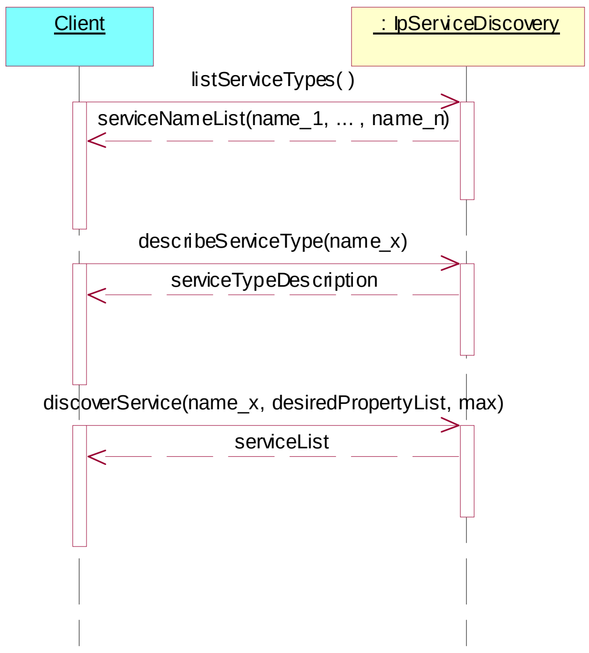
Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName from the list returned in 1.
   Check: valid value of TpServiceTypeDescription is returned

3. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., valid desiredPropertyList, valid max
   Check: valid value of TpServiceList is returned


Client
: IpServiceDiscovery
listServiceTypes( )
serviceNameList(name_1, ... , name_n)
describeServiceType(name_x)
serviceTypeDescription
discoverService(name_x, desiredPropertyList, max)
serviceList

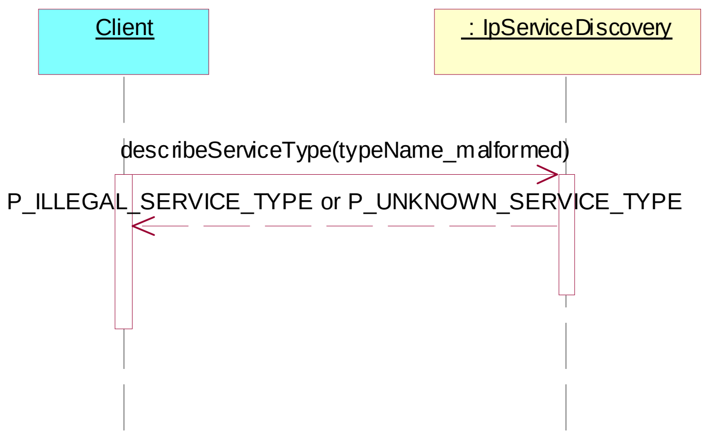
Test FW_FA_SD_02

Summary: **IpServiceDiscovery** describeServiceType, P_ILLEGAL_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

1. Method call **describeServiceType()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE is returned.


Client
: IpServiceDiscovery
describeServiceType(typeName_malformed)
P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE

Test FW_FA_SD_03

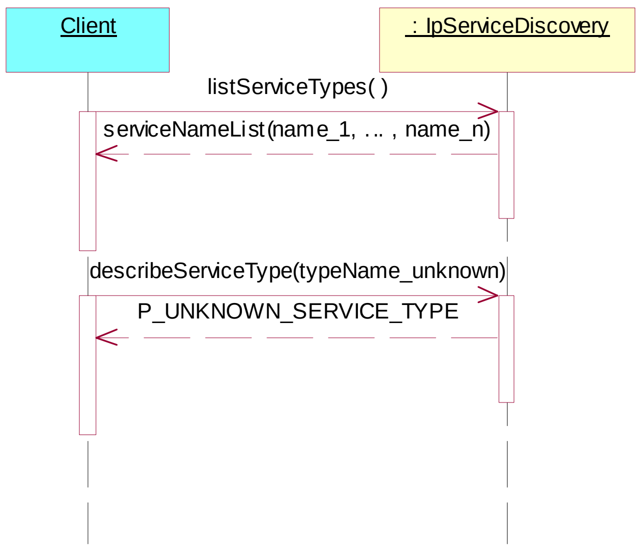
Summary: **IpServiceDiscovery** describeServiceType, P_UNKNOWN_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE is returned.


Client
: IpServiceDiscovery
listServiceTypes( )
serviceNameList(name_1, ... , name_n)
describeServiceType(typeName_unknown)
P_UNKNOWN_SERVICE_TYPE

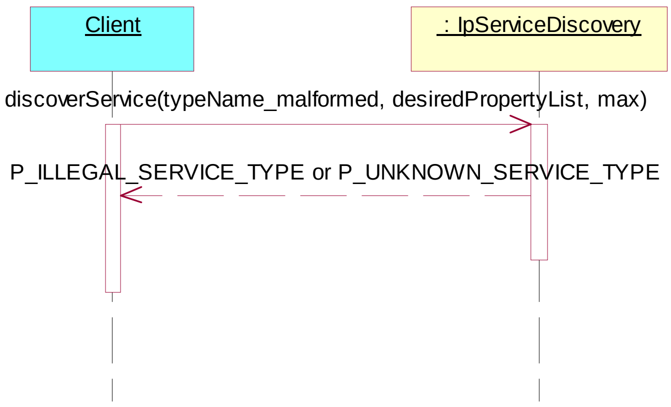
Test FW_FA_SD_04

Summary: **IpServiceDiscovery** discoverService, P_ILLEGAL_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

1. Method call **discoverService()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE is returned.


Client
: IpServiceDiscovery
discoverService(typeName_malformed, desiredPropertyList, max)
P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE

Test FW_FA_SD_05

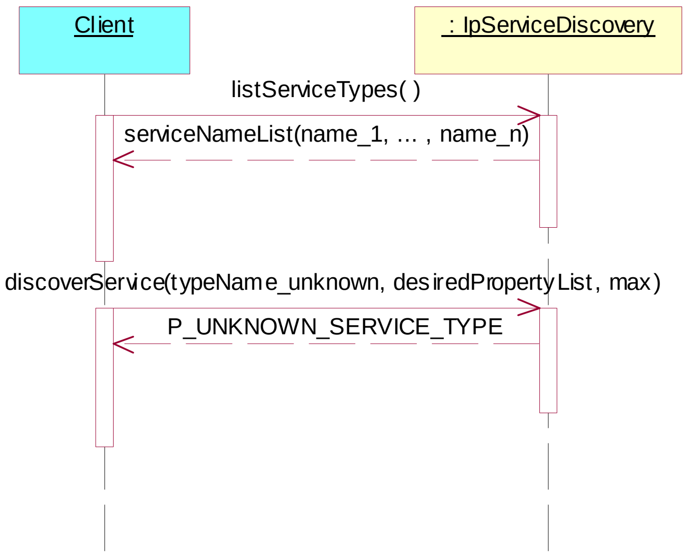
Summary: **IpServiceDiscovery** discoverService, P_UNKNOWN_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE is returned.


Client
: IpServiceDiscovery
listServiceTypes( )
serviceNameList(name_1, ... , name_n)
discoverService(typeName_unknown, desiredPropertyList, max)
P_UNKNOWN_SERVICE_TYPE

Test FW_FA_SD_06

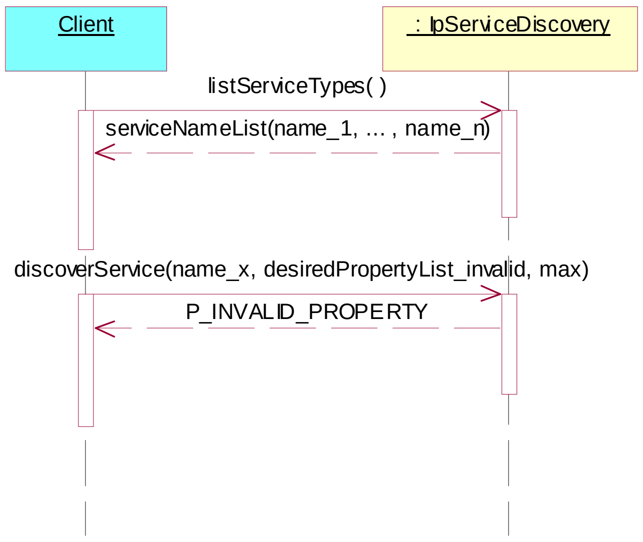
Summary: **IpServiceDiscovery** discoverService, P_INVALID_PROPERTY.

Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., invalid desiredPropertyList, valid max
   Check: P_INVALID_PROPERTY is returned.


Client
: IpServiceDiscovery
listServiceTypes( )
serviceNameList(name_1, ... , name_n)
discoverService(name_x, desiredPropertyList_invalid, max)
P_INVALID_PROPERTY

Test FW_FA_SD_07

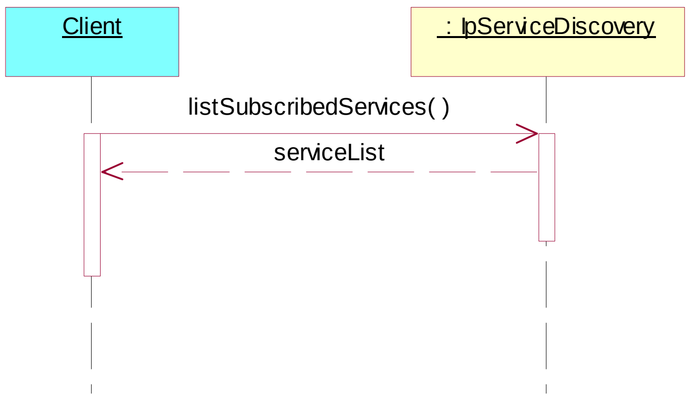
Summary: **IpServiceDiscovery** listSubscribedService.

Reference: ES 202 915-3 [1], clause 7.3.1.

Precondition: listSubscribedServices supported.

Test Sequence:

1. Method call **listSubscribedServices()**
   Parameters: none
   Check: valid value of TpServiceList is returned



5.4.2.2 Service Agreement Management (SA)

Test FW_FA_SA_01

Summary: **IpServiceAgreementManagement**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

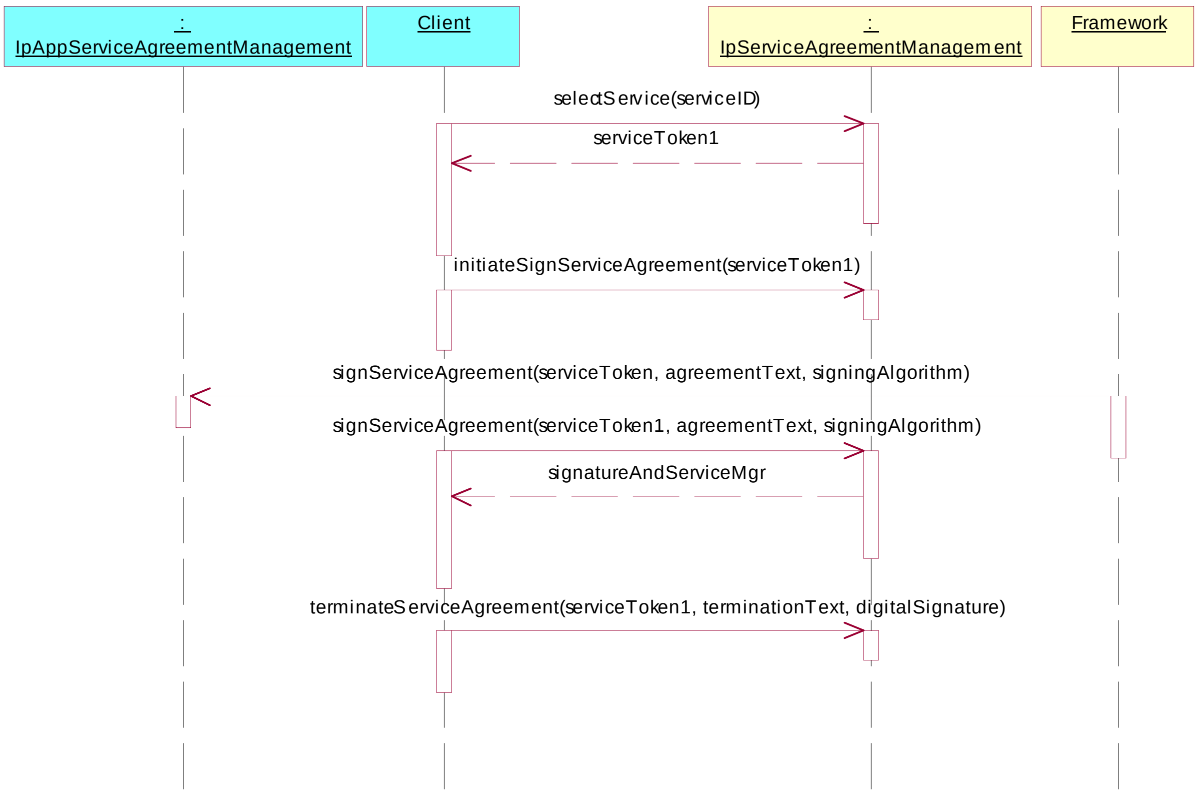
1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpSignatureAndServiceMgr is returned

5. Method call **terminateServiceAgreement()**
   Parameters: serviceToken returned in 1., terminationText, digitalSignature
   Check: no exception is returned


: IpAppServiceAgreementManagement
Client
: IpServiceAgreementManagement
Framework
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken1)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signServiceAgreement(serviceToken1, agreementText, signingAlgorithm)
signatureAndServiceMgr
terminateServiceAgreement(serviceToken1, terminationText, digitalSignature)

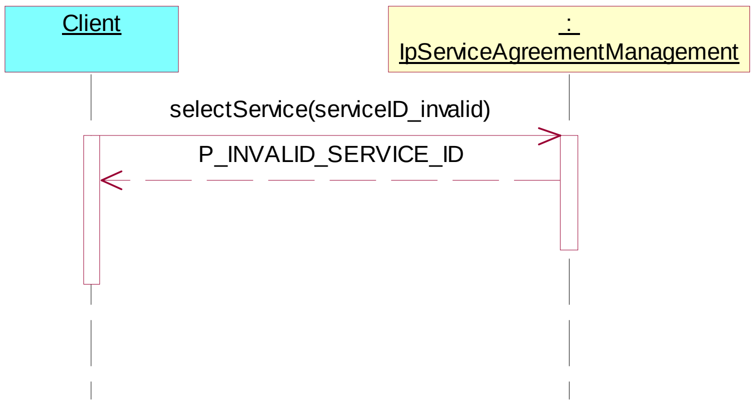
Test FW_FA_SA_02

Summary: **IpServiceAgreementManagement**, selectService, P_INVALID_SERVICE_ID.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

1. Method call **selectService()**
   Parameters: invalid serviceID
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpServiceAgreementManagement
selectService(serviceID_invalid)
P_INVALID_SERVICE_ID

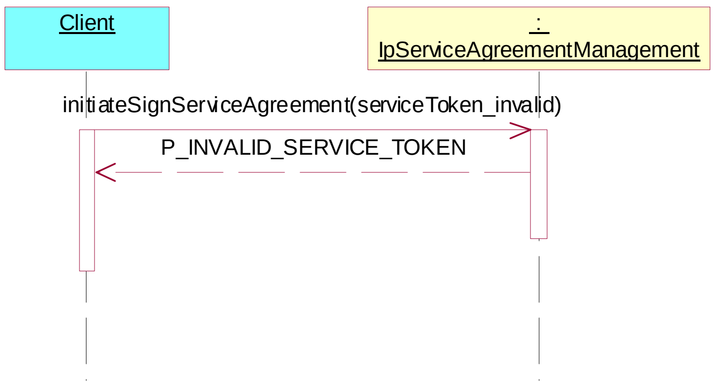
Test FW_FA_SA_03

Summary: **IpServiceAgreementManagement**, initiateSignServiceAgreement, P_INVALID_SERVICE_TOKEN.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

1. Method call **initiateSignServiceAgreement()**
   Parameters: invalid serviceToken
   Check: P_INVALID_SERVICE_TOKEN is returned.


Client
: IpServiceAgreementManagement
initiateSignServiceAgreement(serviceToken_invalid)
P_INVALID_SERVICE_TOKEN

Test FW_FA_SA_04

Summary: **IpServiceAgreementManagement**, signServiceAgreement, P_INVALID_SERVICE_TOKEN.

Reference: ES 202 915-3 [1], clause 7.3.2.

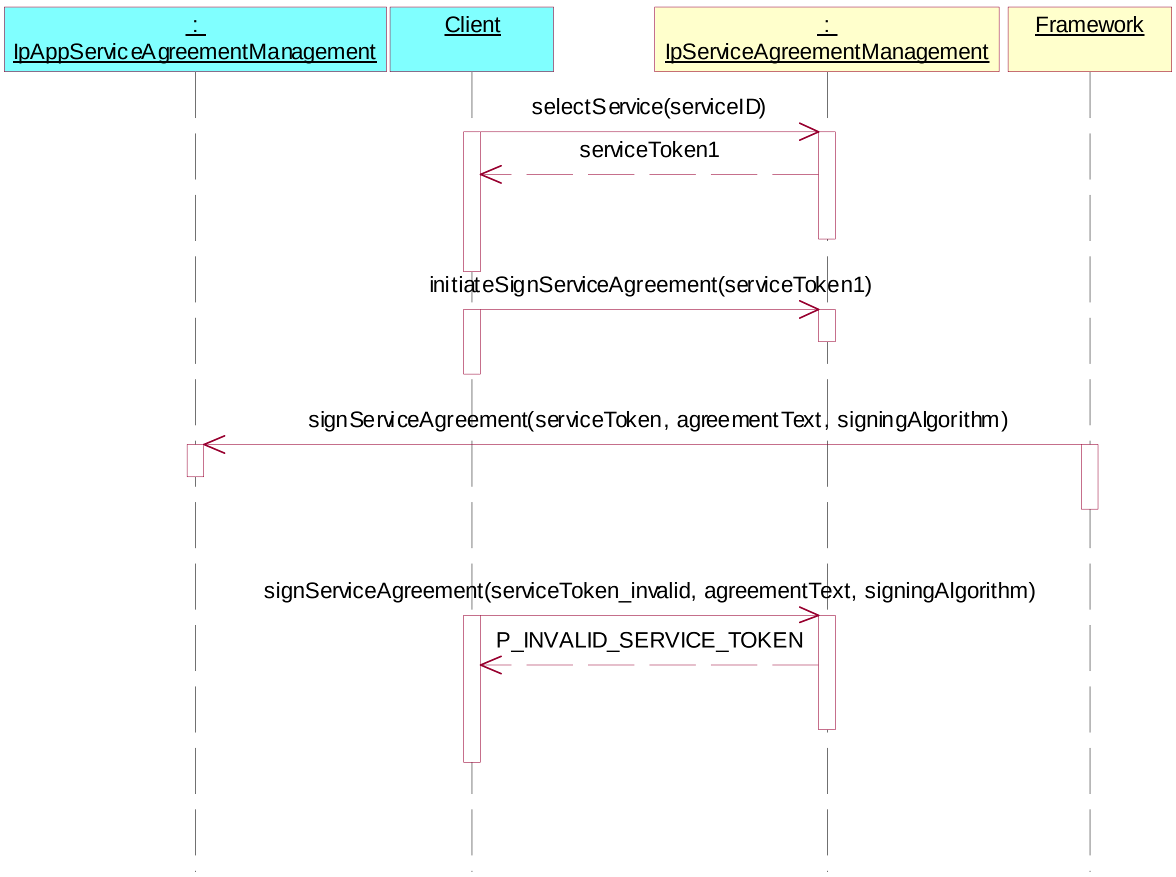
Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in step 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: invalid serviceToken, valid agreementText, valid signingAlgorithm
   Check: P_INVALID_SERVICE_TOKEN is returned.


: IpAppServiceAgreementManagement
Client
: IpServiceAgreementManagement
Framework
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken1)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signServiceAgreement(serviceToken_invalid, agreementText, signingAlgorithm)
P_INVALID_SERVICE_TOKEN

Test FW_FA_SA_05

Summary: **IpServiceAgreementManagement**, signServiceAgreement, P_INVALID_AGREEMENT_TEXT.

Reference: ES 202 915-3 [1], clause 7.3.2.

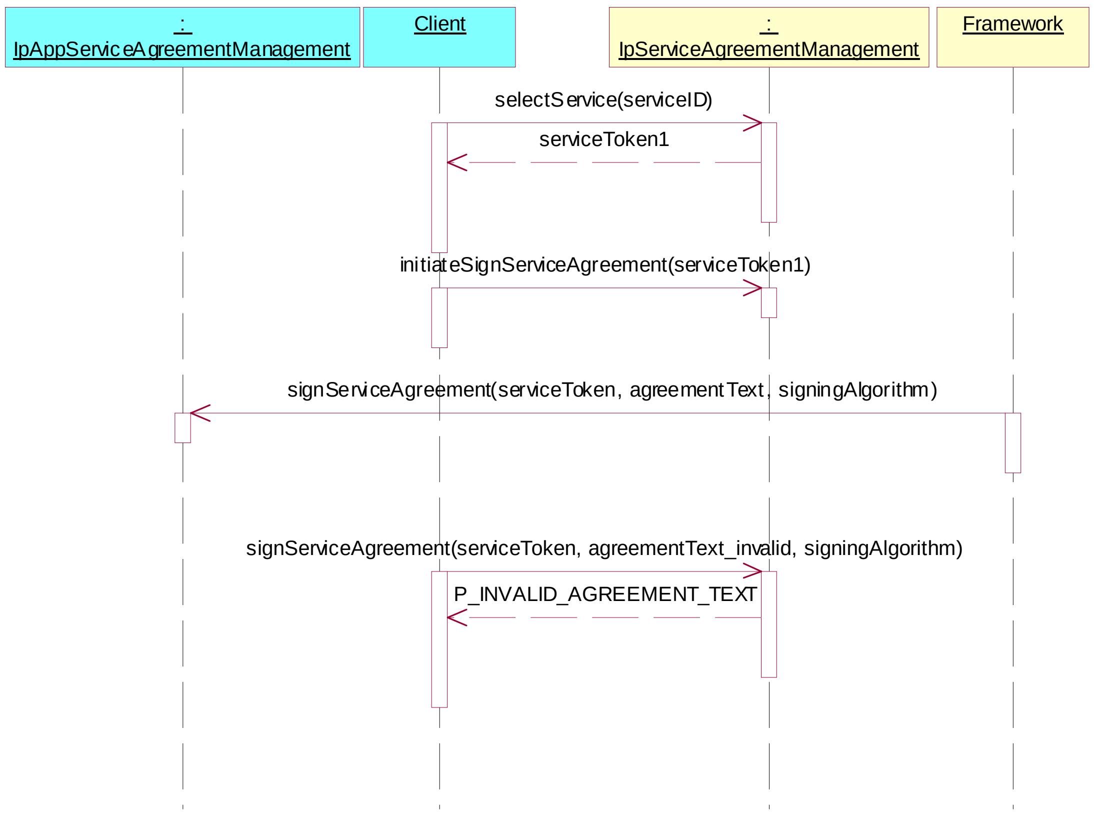
Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: TpServiceToken returned in step 1.
   Check: No exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken

4. Method call **signServiceAgreement()**
   Parameters: serviceToken, invalid agreementText, signingAlgorithm
   Check: P_INVALID_AGREEMENT_TEXT is returned.


: IpAppServiceAgreementManagement
Client
: IpServiceAgreementManagement
Framework
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken1)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signServiceAgreement(serviceToken, agreementText_invalid, signingAlgorithm)
P_INVALID_AGREEMENT_TEXT

Test FW_FA_SA_06

Summary: **IpServiceAgreementManagement**, signServiceAgreement, P_INVALID_SIGNING_ALGORITHM.

Reference: ES 202 915-3 [1], clause 7.3.2.

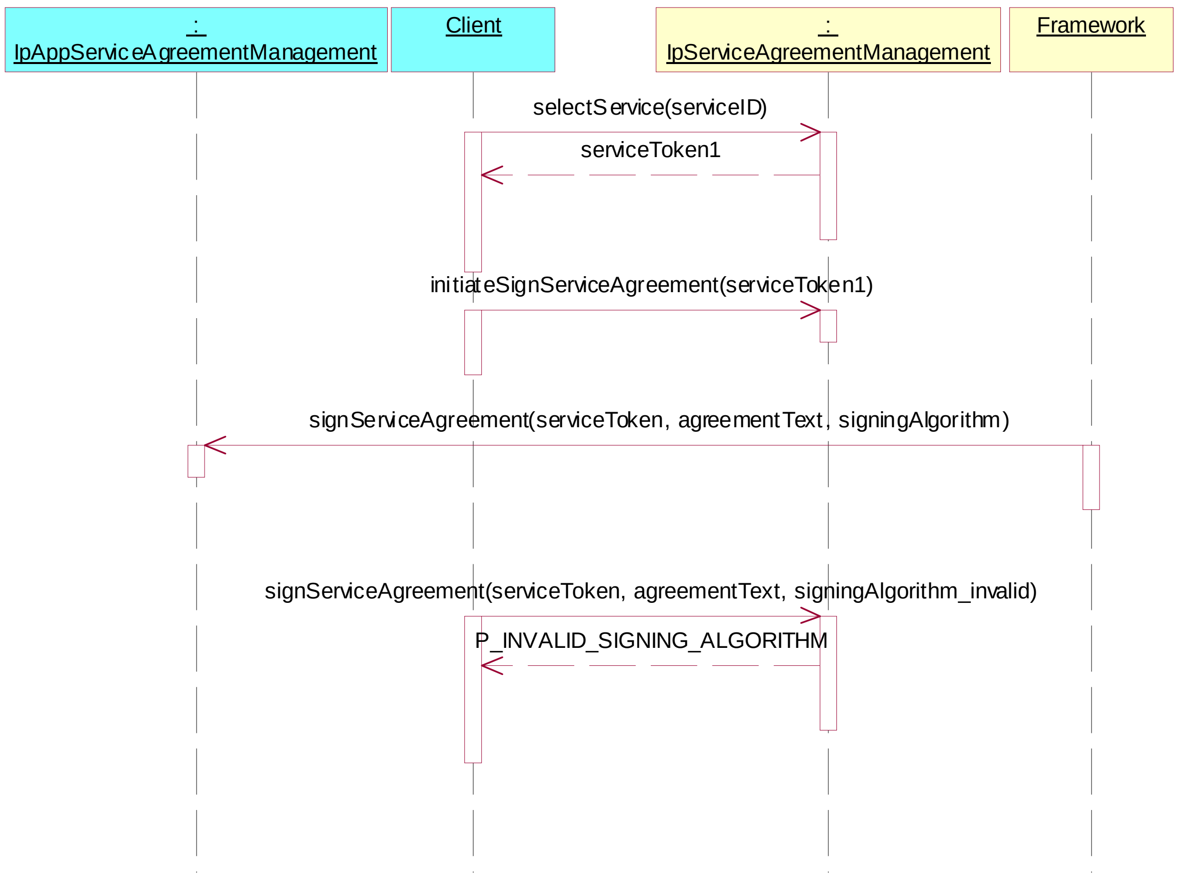
Test Sequence:

1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in step 1.
   Check: No exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken

4. Method call **signServiceAgreement()**
   Parameters: serviceToken, agreementText, invalid signingAlgorithm
   Check: P_INVALID_SIGNING_ALGORITHM is returned.


: IpAppServiceAgreementManagement
Client
: IpServiceAgreementManagement
Framework
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken1)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm_invalid)
P_INVALID_SIGNING_ALGORITHM

Test FW_FA_SA_07

Summary: **IpServiceAgreementManagement**, terminateServiceAgreement, P_INVALID_SERVICE_TOKEN.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

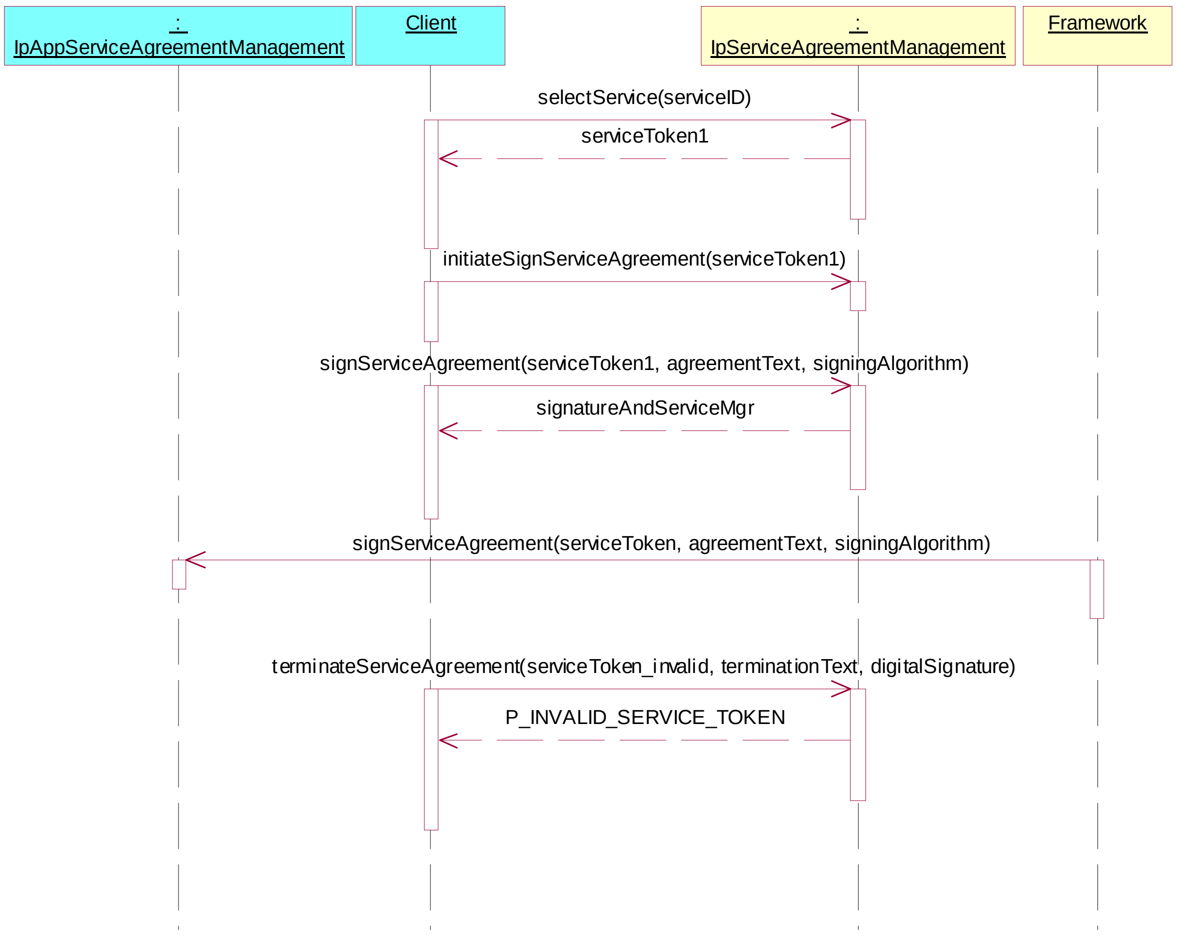
1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpSignatureAndServiceMgr is returned

5. Method call **terminateServiceAgreement()**
   Parameters: invalid serviceToken, terminationText, digitalSignature
   Check: P_INVALID_SERVICE_TOKEN is returned.


: IpAppServiceAgreementManagement
Client
: IpServiceAgreementManagement
Framework
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken1)
signServiceAgreement(serviceToken1, agreementText, signingAlgorithm)
signatureAndServiceMgr
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
terminateServiceAgreement(serviceToken_invalid, terminationText, digitalSignature)
P_INVALID_SERVICE_TOKEN

Test FW_FA_SA_08

Summary: **IpServiceAgreementManagement**, terminateServiceAgreement, P_INVALID_SIGNATURE.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

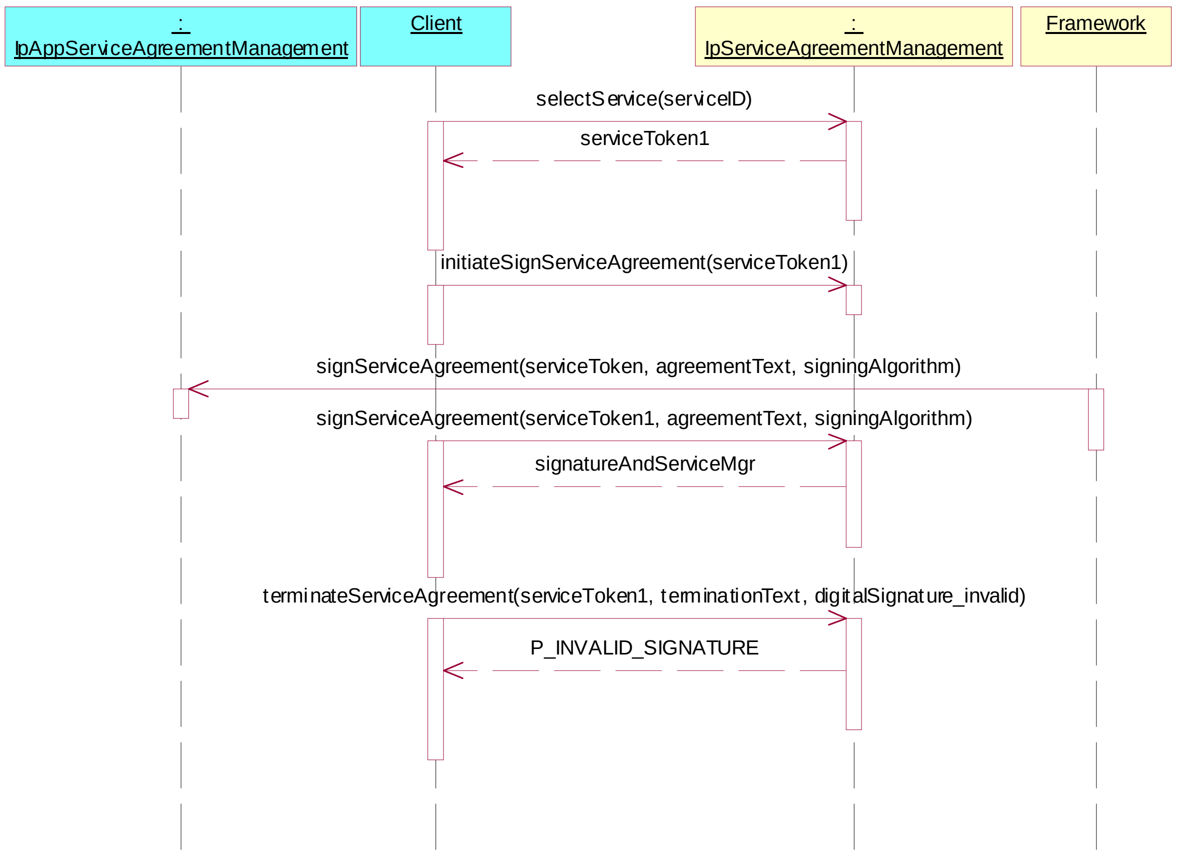
1. Method call **selectService()**
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Method call **initiateSignServiceAgreement()**
   Parameters: serviceToken returned in 1.
   Check: no exception is returned

3. Triggered action: cause IUT to call **signServiceAgreement()** method on the tester's (Application) **IpAppServiceAgreementManagement** interface.
   Parameters: serviceToken.

4. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpSignatureAndServiceMgr is returned

5. Method call **terminateServiceAgreement()**
   Parameters: serviceToken returned in 1., terminationText, invalid digitalSignature
   Check: P_INVALID_SIGNATURE is returned.


: IpAppServiceAgreementManagement
Client
: IpServiceAgreementManagement
Framework
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken1)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signServiceAgreement(serviceToken1, agreementText, signingAlgorithm)
signatureAndServiceMgr
terminateServiceAgreement(serviceToken1, terminationText, digitalSignature_invalid)
P_INVALID_SIGNATURE

5.4.2.3 Integrity Management (IM)

Test FW_FA_IM_01

Summary: **IpHeartBeatMgmt**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpHeartBeatMgmt supported.

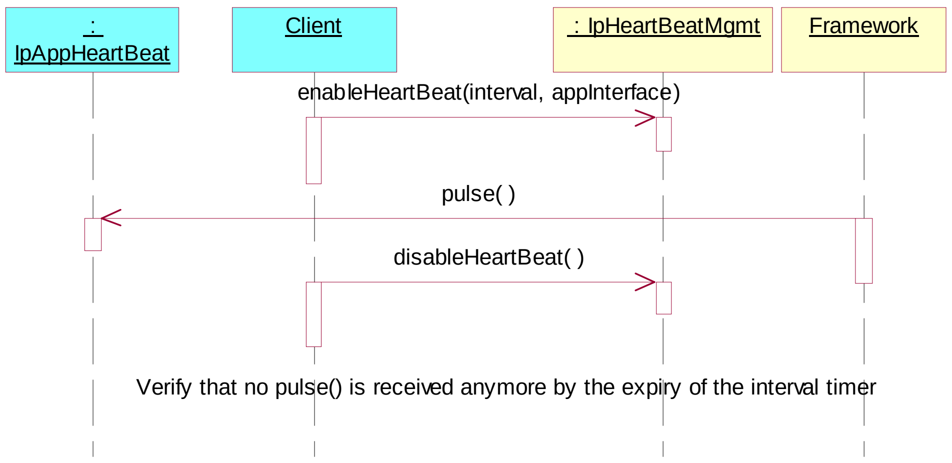
Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

1. Method call **enableHeartBeat()**
   Parameters: interval, appInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to regularly call pulse() method on the tester's (Application) **IpAppHeartBeat** interface.
   Parameters: none
   Check: check that the pulse() method is invoked at the requested interval.

3. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.


: IpAppHeartBeat
Client
: IpHeartBeatMgmt
Framework
enableHeartBeat(interval, appInterface)
pulse( )
disableHeartBeat( )
Verify that no pulse() is received anymore by the expiry of the interval timer

Test FW_FA_IM_02

Summary: **IpHeartBeatMgmt**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpHeartBeatMgmt, changeInterval supported.

Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

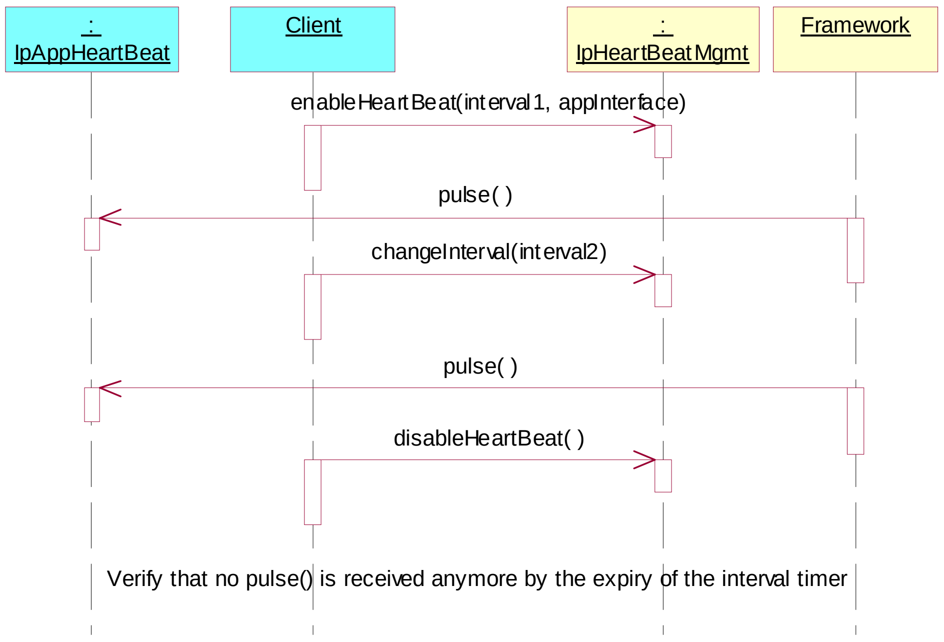
1. Method call **enableHeartBeat()**
   Parameters: interval, appInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to call pulse() method regularly on the tester's (Application) **IpAppHeartBeat** interface.
   Parameters: none
   Check: check that the pulse() method is invoked at the requested interval.

3. Method call **changeInterval()**
   Parameters: interval
   Check: no exception is returned.

4. Triggered action: cause IUT to call pulse() method regularly on the tester's (Application) **IpAppHeartBeat** interface.
   Parameters: none
   Check: the pulse() method is invoked at the new requested interval.

5. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.


: IpAppHeartBeat
Client
: IpHeartBeatMgmt
Framework
enableHeartBeat(interval1, appInterface)
pulse( )
changeInterval(interval2)
pulse( )
disableHeartBeat( )
Verify that no pulse() is received anymore by the expiry of the interval timer

Test FW_FA_IM_03

Summary: **IpHeartBeat**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

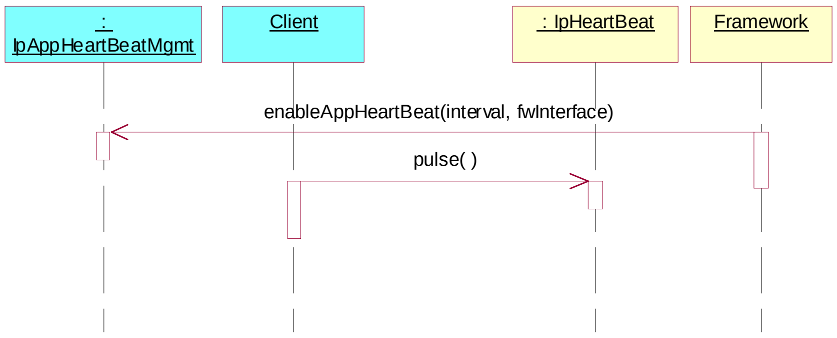
Precondition: IpHeartBeat is supported.

Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

1. Triggered action: cause IUT to call **enableAppHeartBeat()** method on the tester's (Application) **IpAppHeartBeatMgmt** interface.
   Parameters: interval, fwInterface

2. Method call **pulse()**
   Parameters: none
   Check: no exception



Test FW_FA_IM_04

Summary: **IpFaultManager** activityTestReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

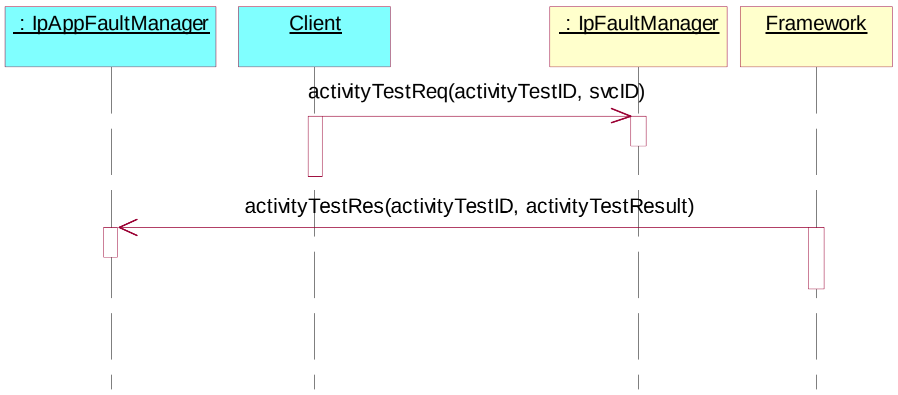
Precondition: IpFaultManager, activityTestReq supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: activityTestID, svcID
   Check: no exception is returned

2. Triggered action: cause IUT to call **activityTestRes ()** method on the tester's (Application) IpAppFaultManager interface.
   Parameters: activityTestID, activityTestResult


: IpAppFaultManager
Client
: IpFaultManager
Framework
activityTestReq(activityTestID, svcID)
activityTestRes(activityTestID, activityTestResult)

Test FW_FA_IM_05

Summary: **IpFaultManager** activityTestReq on Framework, successful.

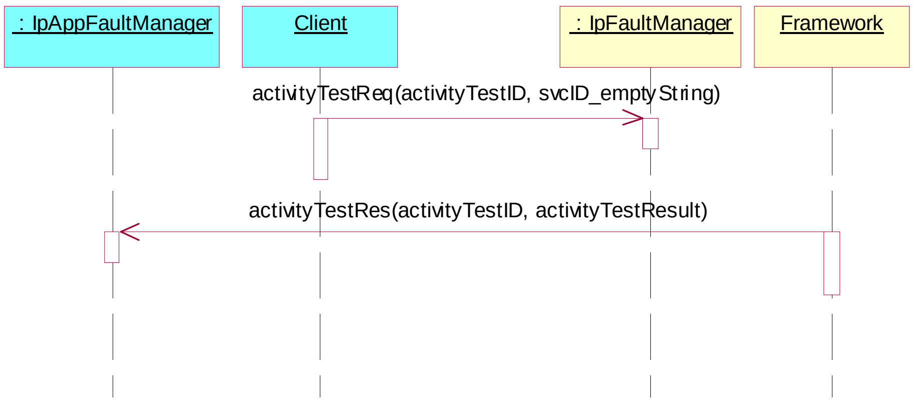
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, activityTestReq supported.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: **activityTestID**, svcID with emptystring value
   Check: no exception is returned

2. Triggered action: cause IUT to call **activityTestRes()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID, activityTestResult


: IpAppFaultManager
Client
: IpFaultManager
Framework
activityTestReq(activityTestID, svcID_emptyString)
activityTestRes(activityTestID, activityTestResult)

Test FW_FA_IM_06

Summary: **IpFaultManager** activityTestReq , P_INVALID_SERVICE_ID exception.

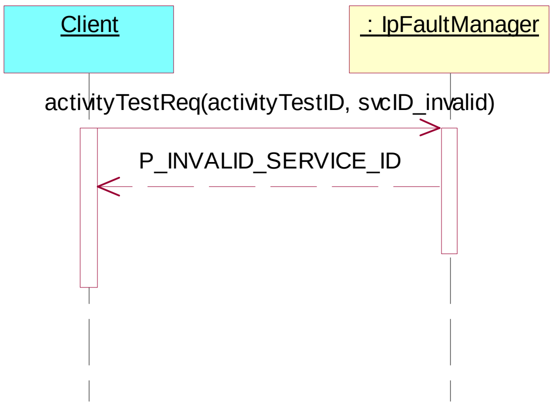
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, activityTestReq supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: activityTestID, invalid svcID
   Check: P_INVALID_SERVICE_ID is returned.



Test FW_FA_IM_07

Summary: **IpFaultManager** generateFaultStatisticsRecordReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

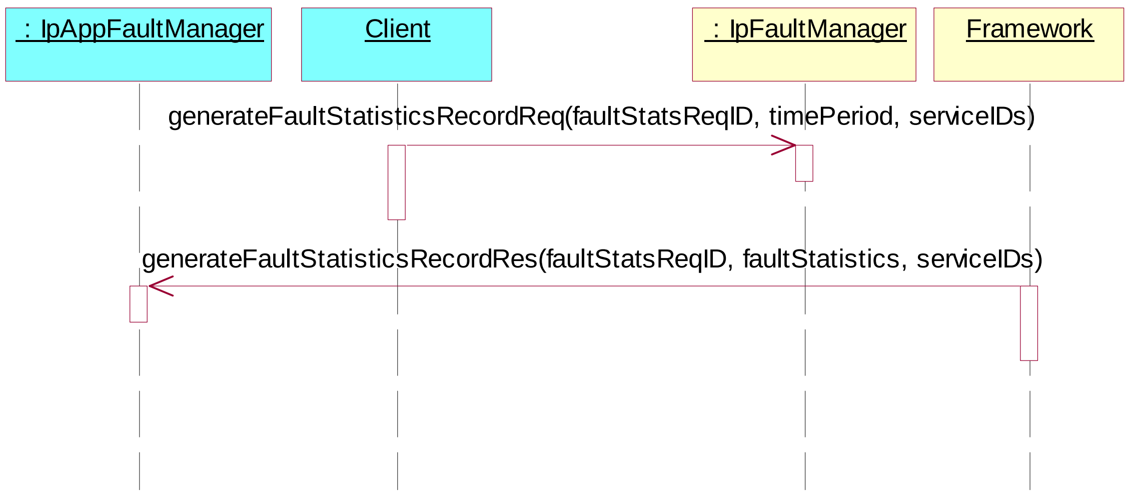
Precondition: IpFaultManager, **generateFaultStatisticsRecordReq ()** supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod, serviceIDs
   Check: no exception is returned

2. Triggered action: cause IUT to call **generateFaultStatisticsRecordRes ()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: faultStatsReqID, faultStatistics, ServiceIDs


: IpAppFaultManager
Client
: IpFaultManager
Framework
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, serviceIDs)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics, serviceIDs)

Test FW_FA_IM_08

Summary: **IpFaultManager** generateFaultStatisticsRecordReqon Framework, successful.

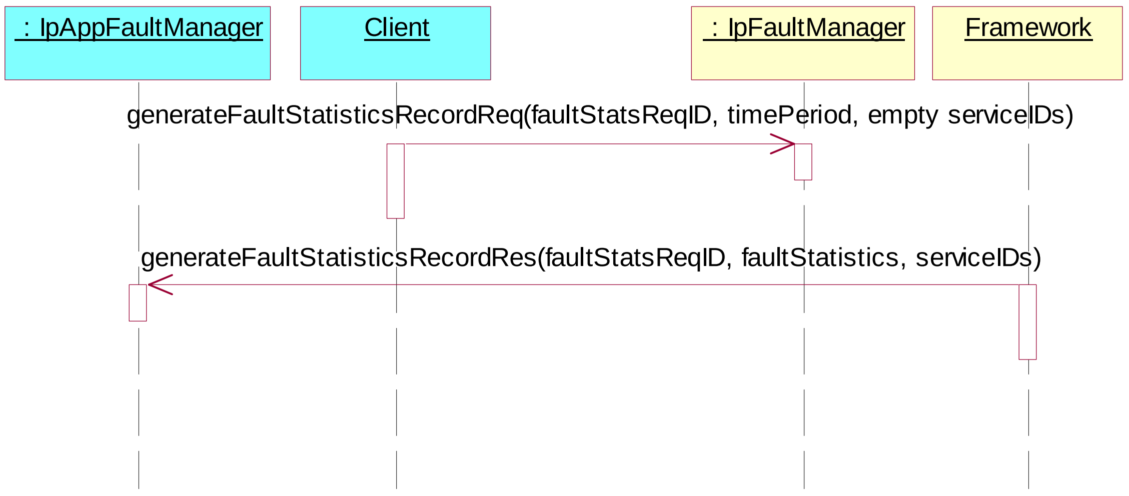
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **generateFaultStatisticsRecordReq ()** supported.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod, serviceIDs with emptystring value
   Check: no exception is returned

2. Triggered action: cause IUT to call **generateFaultStatisticsRecordRes ()** method on the tester's (Application) IpAppFaultManager interface.
   Parameters: faultStatsReqID, faultStatistics, ServiceIDs


: IpAppFaultManager
Client
: IpFaultManager
Framework
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, empty serviceIDs)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics, serviceIDs)

Test FW_FA_IM_09

Summary: **IpFaultManager** generateFaultStatisticsRecordReq, P_INVALID_SERVICE_ID exception.

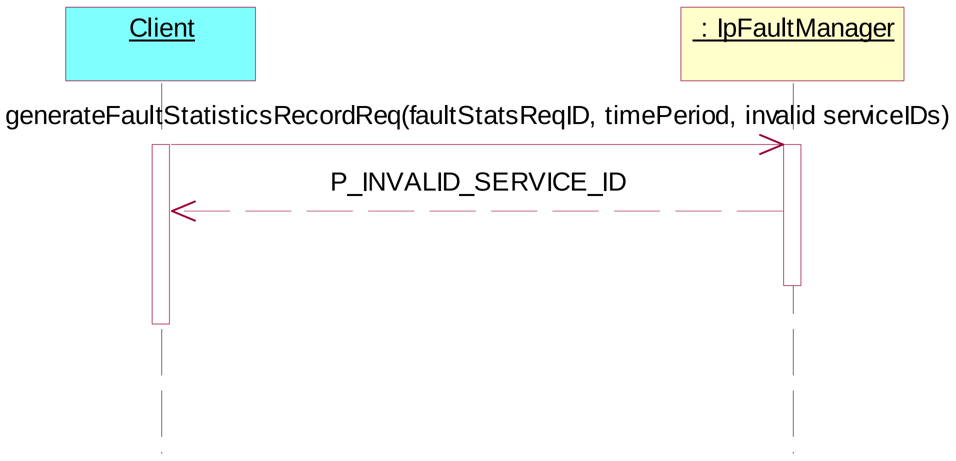
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **generateFaultStatisticsRecordReq ()** supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod, invalid serviceIDs
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, invalid serviceIDs)
P_INVALID_SERVICE_ID

Test FW_FA_IM_10

Summary: **IpFaultManager** svcUnavailableInd, successful.

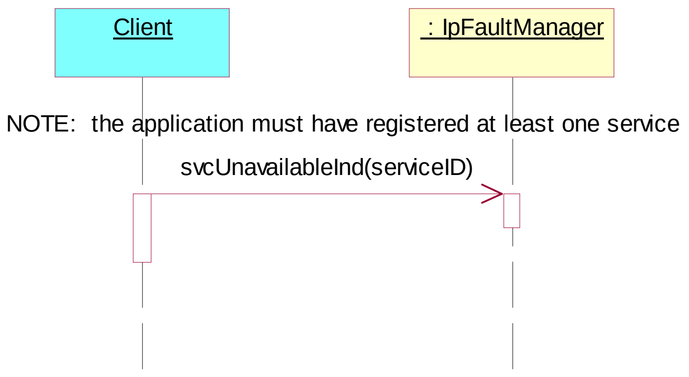
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **svcUnavailableInd()** supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: serviceID
   Check: no exception is returned


Client
: IpFaultManager
NOTE: the application must have registered at least one service
svcUnavailableInd(serviceID)

Test FW_FA_IM_11

Summary: **IpFaultManager** svcUnavailableInd, P_INVALID_SERVICE_ID exception.

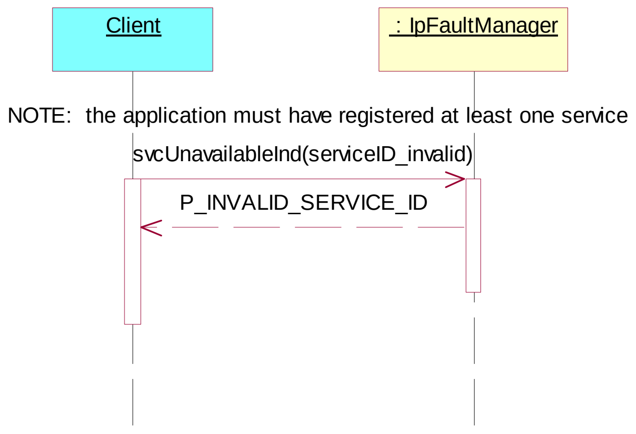
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **svcUnavailableInd()** supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: invalid serviceID
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpFaultManager
NOTE: the application must have registered at least one service
svcUnavailableInd(serviceID_invalid)
P_INVALID_SERVICE_ID

Test FW_FA_IM_12

Summary: **IpFaultManager** appActivityTestRes, successful.

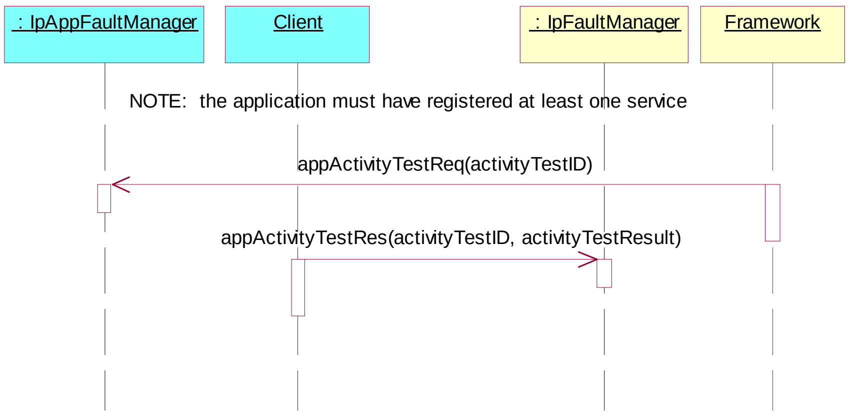
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **appActivityTestRes()** supported.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestRes()**
   Parameters: activityTestID, activityTestResult
   Check: no exception is returned



: IpAppFaultManager
Client
: IpFaultManager
Framework
NOTE: the application must have registered at least one service
appActivityTestReq(activityTestID)
appActivityTestRes(activityTestID, activityTestResult)

Test FW_FA_IM_13

Summary: **IpFaultManager** appActivityTestRes, P_INVALID_ACTIVITY_TEST_ID Exception.

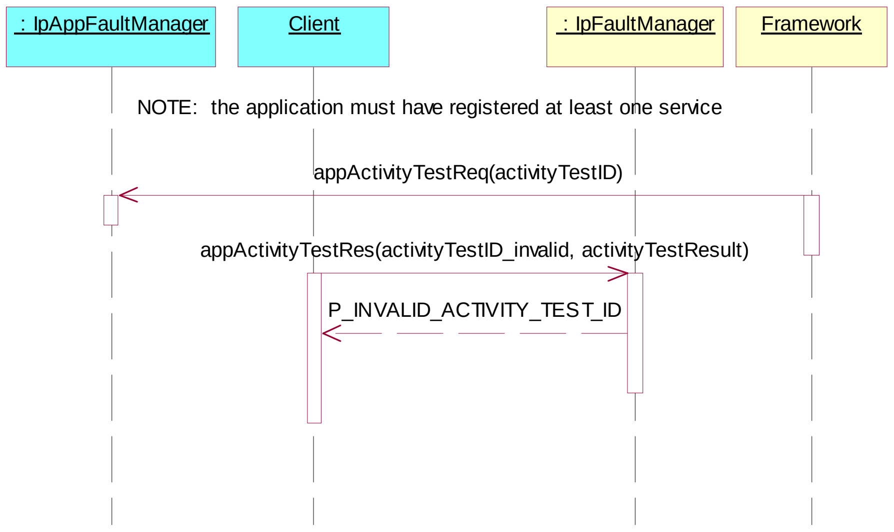
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **appActivityTestRes()** supported.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestRes()**
   Parameters: invalid activityTestID, activityTestResult
   Check: P_INVALID_ACTIVITY_TEST_ID is returned.


: IpAppFaultManager
Client
: IpFaultManager
Framework
NOTE: the application must have registered at least one service
appActivityTestReq(activityTestID)
appActivityTestRes(activityTestID_invalid, activityTestResult)
P_INVALID_ACTIVITY_TEST_ID

Test FW_FA_IM_14

Summary: **IpFaultManager** appActivityTestErr, successful.

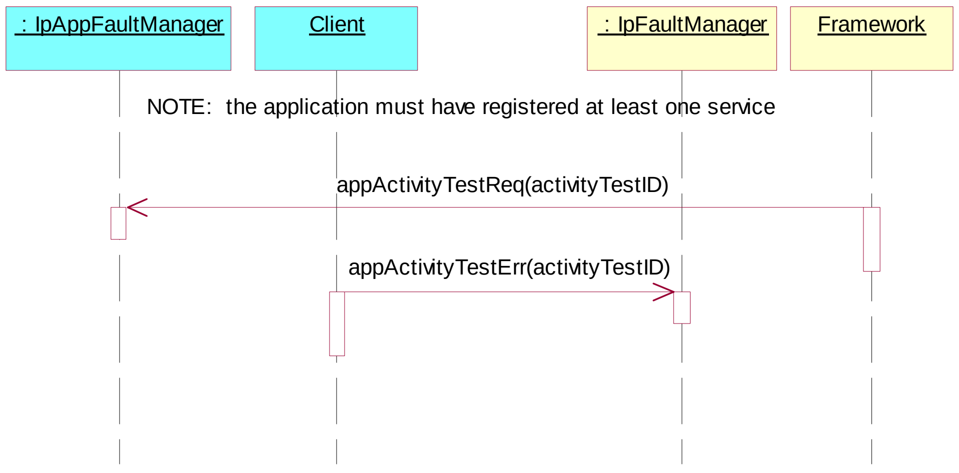
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **appActivityTestRes()** supported.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestErr()**
   Parameters: activityTestID
   Check: no exception is returned


: IpAppFaultManager
Client
: IpFaultManager
Framework
NOTE: the application must have registered at least one service
appActivityTestReq(activityTestID)
appActivityTestErr(activityTestID)

Test FW_FA_IM_15

Summary: **IpFaultManager** appActivityTestErr, P_INVALID_ACTIVITY_TEST_ID exception.

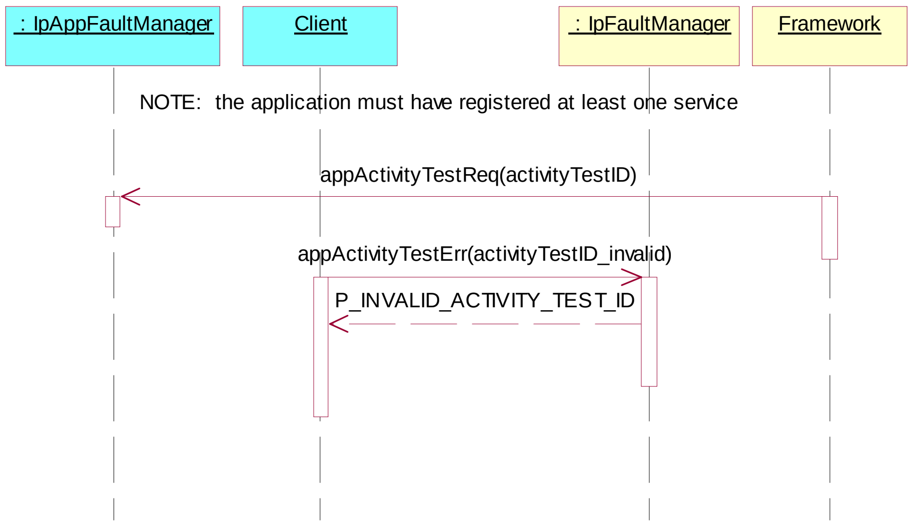
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **appActivityTestRes()** supported.

Test Sequence:

1. Triggered action: cause IUT to call **appActivityTestReq()** method on the tester's (Application) **IpAppFaultManager** interface.
   Parameters: activityTestID

2. Method call **appActivityTestErr()**
   Parameters: invalid activityTestID
   Check: P_INVALID_ACTIVITY_TEST_ID is returned.


: IpAppFaultManager
Client
: IpFaultManager
Framework
NOTE: the application must have registered at least one service
appActivityTestReq(activityTestID)
appActivityTestErr(activityTestID_invalid)
P_INVALID_ACTIVITY_TEST_ID

Test FW_FA_IM_16

Summary: **IpFaultManager** appAvailStatusInd, successful.

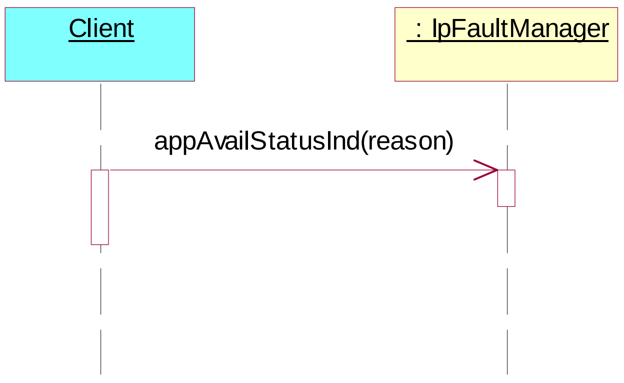
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpFaultManager, **appAvailStatusInd ()** supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **appAvailStatusInd ()**
   Parameters: reason
   Check: no exception is returned



Test FW_FA_IM_17

Summary: **IpLoadManager** createLoadLevelNotification and destroyLoadLevelNotification methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpLoadManager, createLoadLevelNotification and destroyLoadLevelNotification supported.

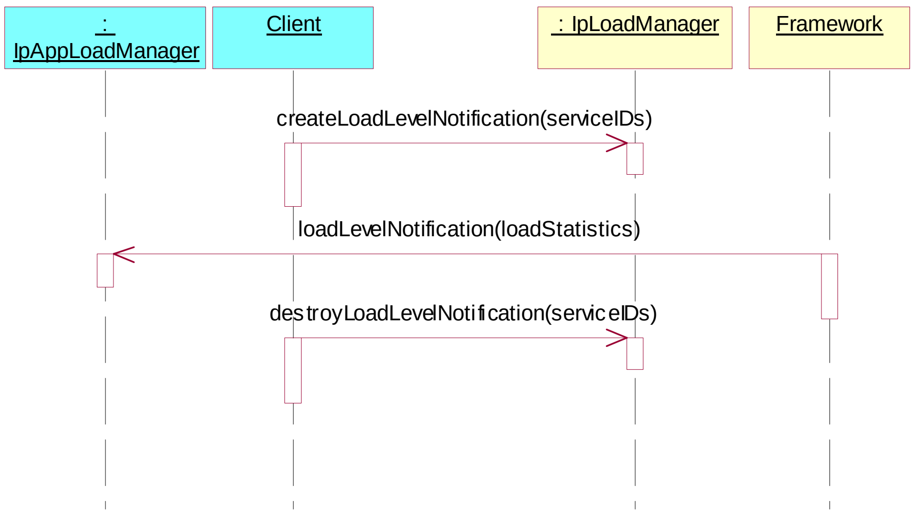
Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

2. Triggered action: cause IUT to call **loadLevelNotification()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatistics

3. Method call **destroyLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned


: IpAppLoadManager
Client
: IpLoadManager
Framework
createLoadLevelNotification(serviceIDs)
loadLevelNotification(loadStatistics)
destroyLoadLevelNotification(serviceIDs)

Test FW_FA_IM_18

Summary: **IpLoadManager** All methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpLoadManager, notifications with suspendNotification and resumeNotification supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

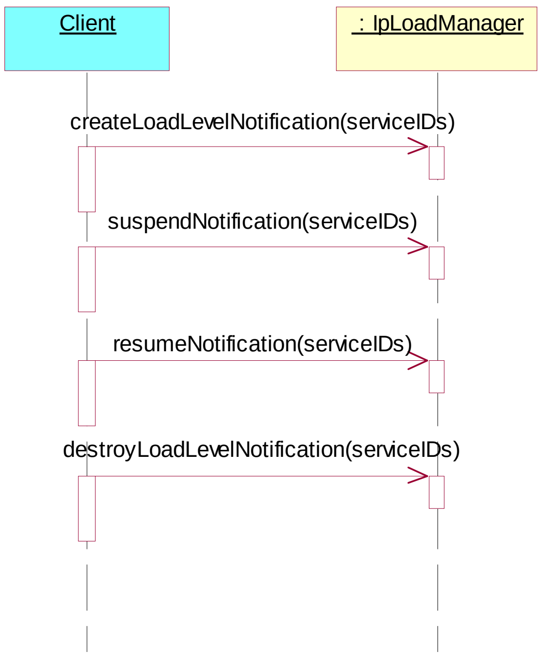
Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

2. Method call **suspendNotification()**
   Parameters: serviceIDs
   Check: no exception is returned, no load level notifications received until resumeNotification() is called.

3. Method call **resumeNotification()**
   Parameters: serviceIDs
   Check: no exception is returned

4. Method call **destroyLoadLevelNotification()**
   Parameters: serviceIDs
   Check: no exception is returned


Client
: IpLoadManager
createLoadLevelNotification(serviceIDs)
suspendNotification(serviceIDs)
resumeNotification(serviceIDs)
destroyLoadLevelNotification(serviceIDs)

Test FW_FA_IM_19

Summary:	**IpLoadManager** All methods on Framework, successful.
Reference:	ES 202 915-3 [1], clause 7.3.3.
Precondition:	IpLoadManager, notifications with suspendNotification and resumeNotification supported.
Preamble:	The application (Tester) must have subscribed to at least one service instance.

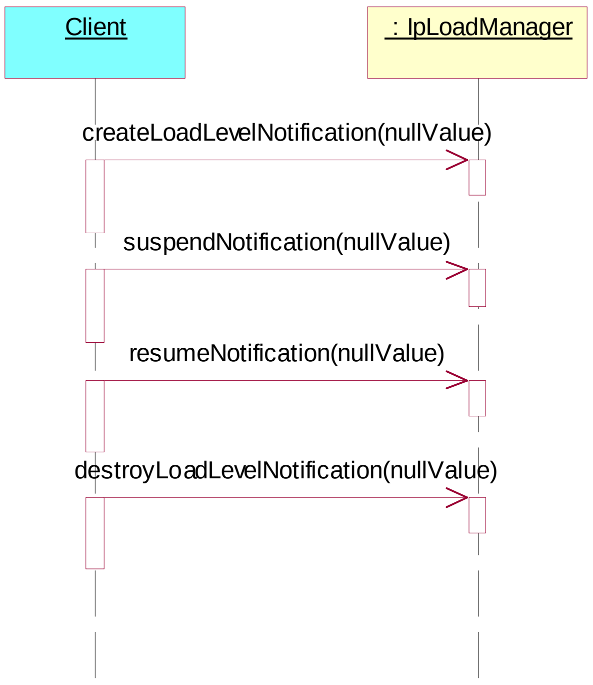
Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **suspendNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned, no load level notifications received until resumeNotification() is called.

3. Method call **resumeNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

4. Method call **destroyLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned


Client
: IpLoadManager
createLoadLevelNotification(nullValue)
suspendNotification(nullValue)
resumeNotification(nullValue)
destroyLoadLevelNotification(nullValue)

Test FW_FA_IM_20

Summary: **IpLoadManager** createLoadLevelNotification, P_INVALID_SERVICE_ID.

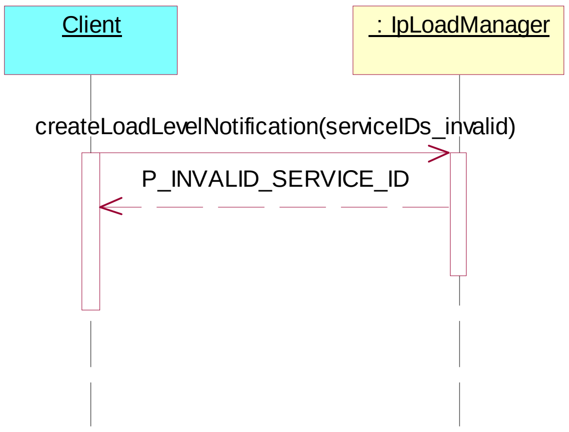
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpLoadManager, createLoadLevelNotification supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID is returned.



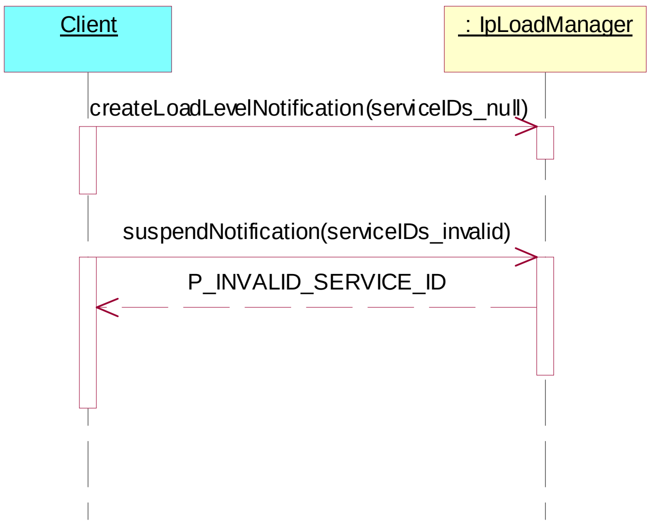
Test FW_FA_IM_21

Summary:	**IpLoadManager** suspendNotification, P_INVALID_SERVICE_ID.
Reference:	ES 202 915-3 [1], clause 7.3.3.
Precondition:	IpLoadManager, notifications with suspendNotification supported.
Preamble:	The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **suspendNotification ()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpLoadManager
createLoadLevelNotification(serviceIDs_null)
suspendNotification(serviceIDs_invalid)
P_INVALID_SERVICE_ID

Test FW_FA_IM_22

Summary: **IpLoadManager** resumeNotification, P_INVALID_SERVICE_ID.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpLoadManager, notifications with suspendNotification and resumeNotification supported.

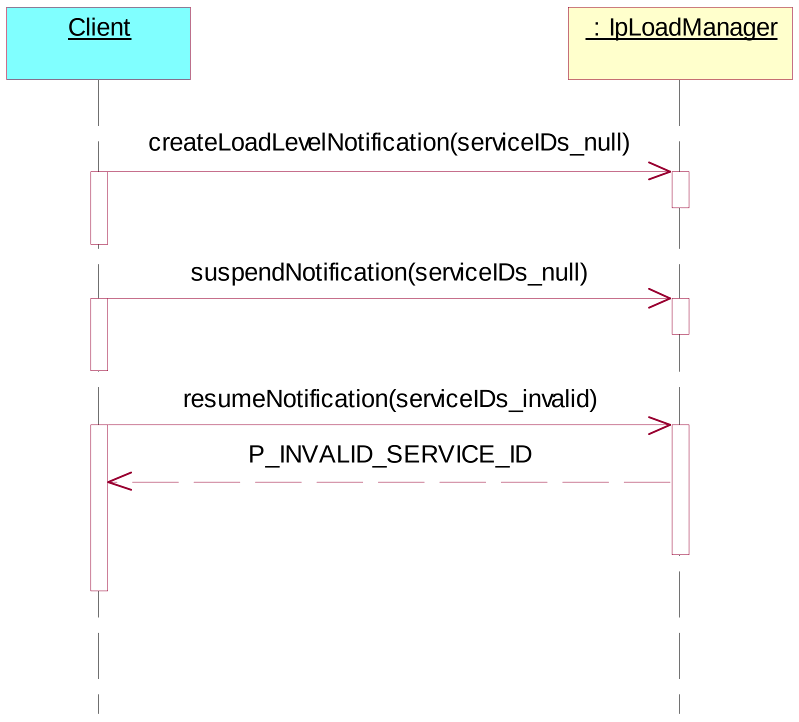
Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **suspendNotification ()**
   Parameters: serviceIDs is null
   Check: no exception is returned

3. Method call **resumeNotification ()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpLoadManager
createLoadLevelNotification(serviceIDs_null)
suspendNotification(serviceIDs_null)
resumeNotification(serviceIDs_invalid)
P_INVALID_SERVICE_ID

Test FW_FA_IM_23

Summary: **IpLoadManager** destroyLoadLevelNotification, P_INVALID_SERVICE_ID.

Reference: ES 202 915-3 [1], clause 7.3.3.

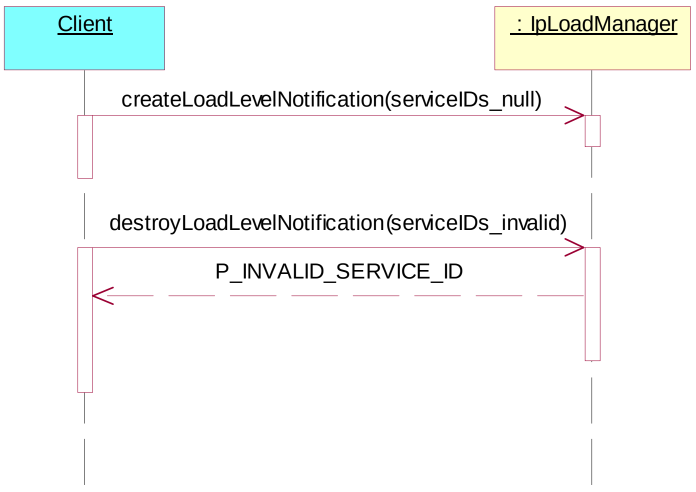
Precondition: IpLoadManager, notifications supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: serviceIDs is null
   Check: no exception is returned

2. Method call **destroyLoadLevelNotification ()**
   Parameters: invalid serviceIDs
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpLoadManager
createLoadLevelNotification(serviceIDs_null)
destroyLoadLevelNotification(serviceIDs_invalid)
P_INVALID_SERVICE_ID

Test FW_FA_IM_24

Summary: **IpLoadManager** reportLoad, successful.

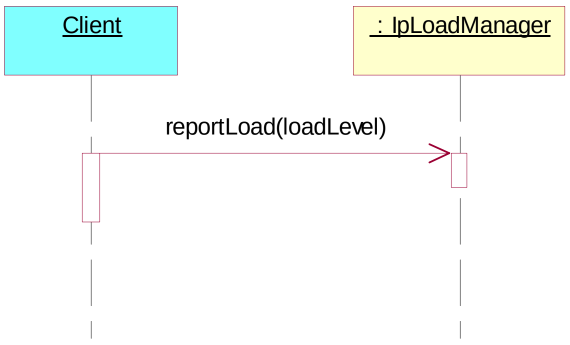
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpLoadManager, reportLoad supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **reportLoad ()**
   Parameters: loadLevel
   Check: no exception is returned



Test FW_FA_IM_25

Summary: **IpLoadManager** queryLoadStatsReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

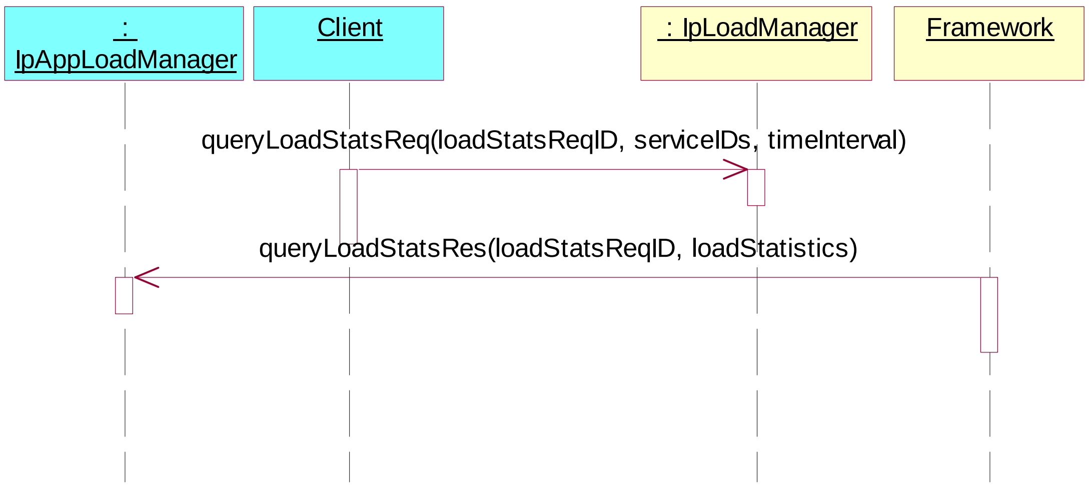
Precondition: IpLoadManager, queryLoadStatsReq supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **queryLoadStatsReq()**
   Parameters: loadStatsReqID, serviceIDs, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryLoadStatsRes ()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatsReqID, loadStatistics


: IpAppLoadManager
Client
: IpLoadManager
Framework
queryLoadStatsReq(loadStatsReqID, serviceIDs, timeInterval)
queryLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FA_IM_26

Summary: **IpLoadManager** queryLoadStatsReq on Framework, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

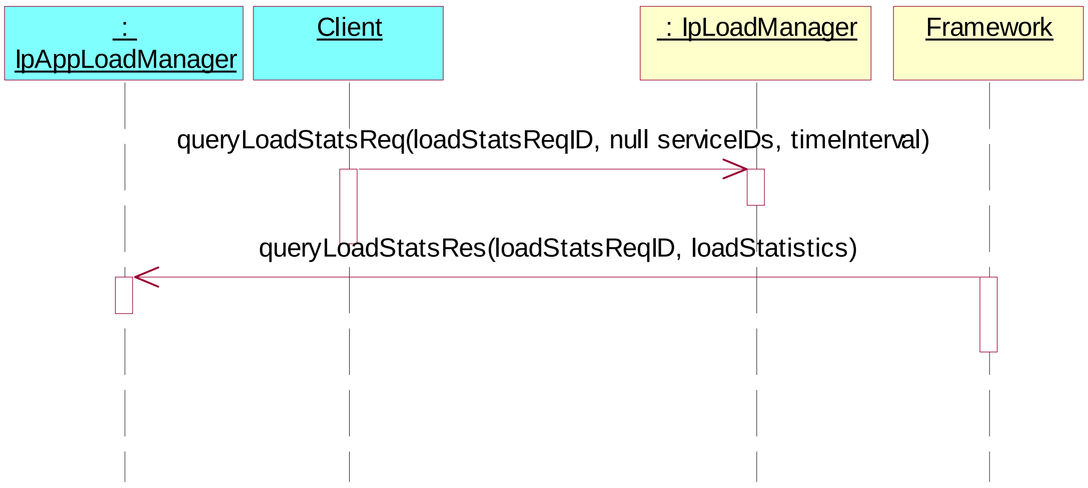
Precondition: IpLoadManager, queryLoadStatsReq supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **queryLoadStatsReq()**
   Parameters: loadStatsReqID, serviceIDs as null, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryLoadStatsRes ()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatsReqID, loadStatistics


: IpAppLoadManager
Client
: IpLoadManager
Framework
queryLoadStatsReq(loadStatsReqID, null serviceIDs, timeInterval)
queryLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FA_IM_27

Summary: **IpLoadManager** queryLoadStatsReq, P_INVALID_SERVICE_ID.

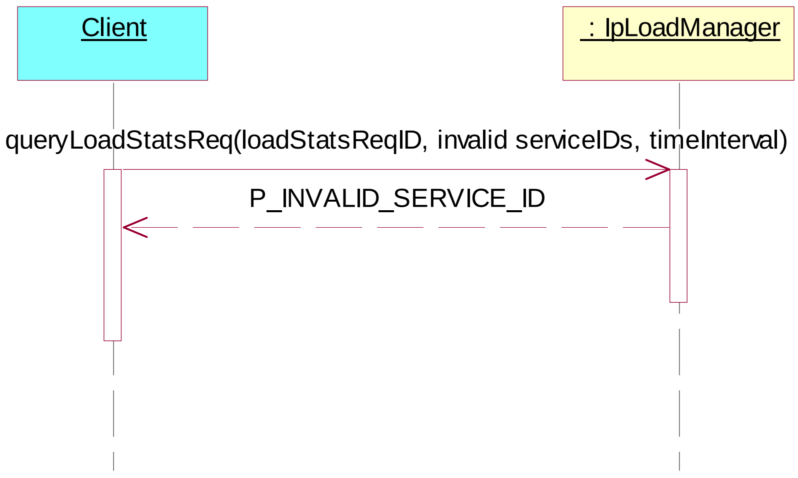
Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpLoadManager, queryLoadStatsReq supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Method call **queryLoadStatsReq()**
   Parameters: loadStatsReqID, invalid serviceIDs, timeInterval
   Check: P_INVALID_SERVICE_ID is returned.


Client
: IpLoadManager
queryLoadStatsReq(loadStatsReqID, invalid serviceIDs, timeInterval)
P_INVALID_SERVICE_ID

Test FW_FA_IM_28

Summary: **IpLoadManager** queryAppLoaStatsdRes, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

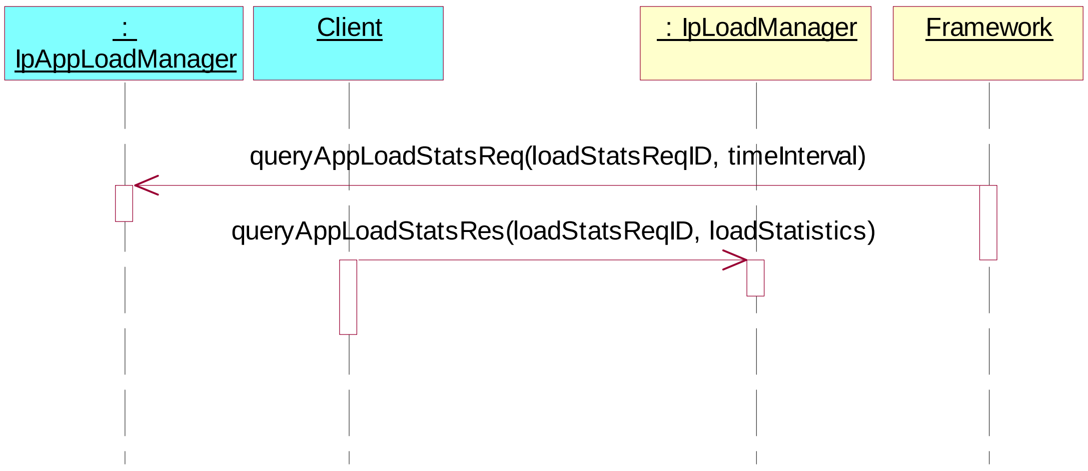
Precondition: IpLoadManager, queryAppLoadStatsRes supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Triggered action: cause IUT to call **queryAppLoadStatsReq ()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatsReqID, timeInterval

2. Method call **queryAppLoadStatsRes ()**
   Parameters: loadStatsReqID, loadStatistics
   Check: no exception is returned


: IpAppLoadManager
Client
: IpLoadManager
Framework
queryAppLoadStatsReq(loadStatsReqID, timeInterval)
queryAppLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FA_IM_29

Summary: **IpLoadManager** queryAppLoadStatsErr, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

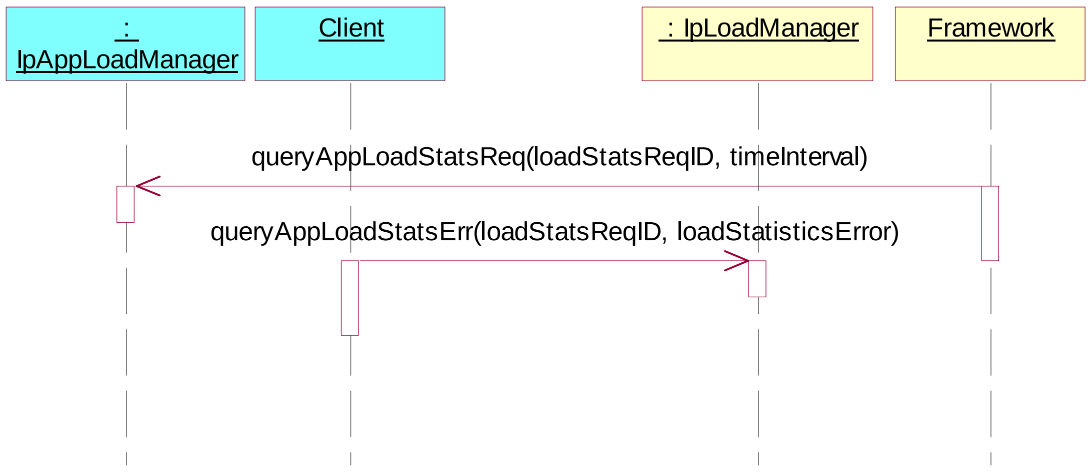
Precondition: IpLoadManager, queryAppLoadStatsRes supported.

Preamble: The application (Tester) must have subscribed to at least one service instance.

Test Sequence:

1. Triggered action: cause IUT to call **queryAppLoadStatsReq()** method on the tester's (Application) **IpAppLoadManager** interface.
   Parameters: loadStatsReqID, timeInterval

2. Method call **queryAppLoadStatsErr()**
   Parameters: loadStatsReqID, loadStatisticsError
   Check: no exception is returned


: IpAppLoadManager
Client
: IpLoadManager
Framework
queryAppLoadStatsReq(loadStatsReqID, timeInterval)
queryAppLoadStatsErr(loadStatsReqID, loadStatisticsError)

Test FW_FA_IM_30

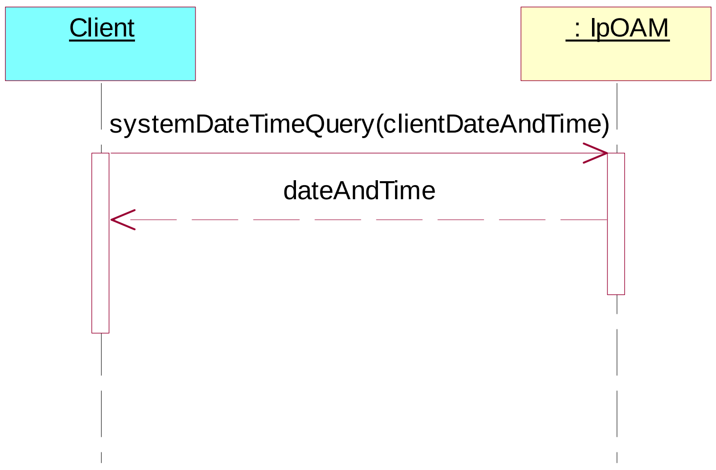
Summary: **IpOAM**, systemDateTimeQuery, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpOAM supported.

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: clientDateAndTime
   Check: valid value of TpDateAndTime is returned



Test FW_FA_IM_31

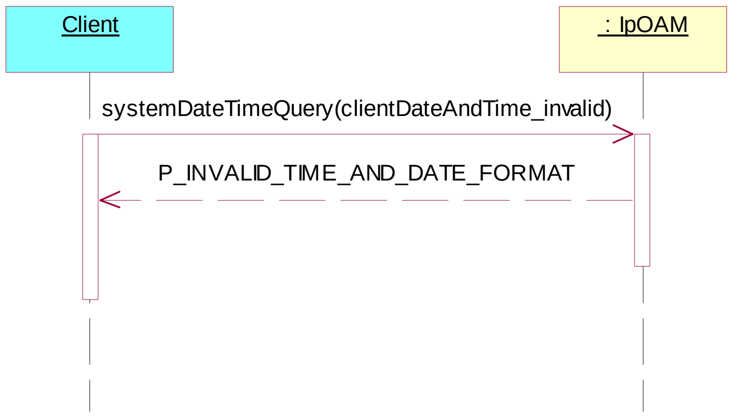
Summary: **IpOAM**, systemDateTimeQuery, P_INVALID_TIME_AND-DATE_FORMAT exception.

Reference: ES 202 915-3 [1], clause 7.3.3.

Precondition: IpOAM supported.

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: invalid clientDateAndTime
   Check: P_INVALID_TIME_AND-DATE_FORMAT is returned.


Client
: IpOAM
systemDateTimeQuery(clientDateAndTime_invalid)
P_INVALID_TIME_AND_DATE_FORMAT

5.4.2.4 Event Notification (EN)

Test FW_FA_EN_01

Summary: **IpEventNotification**, create and destroy methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.4.

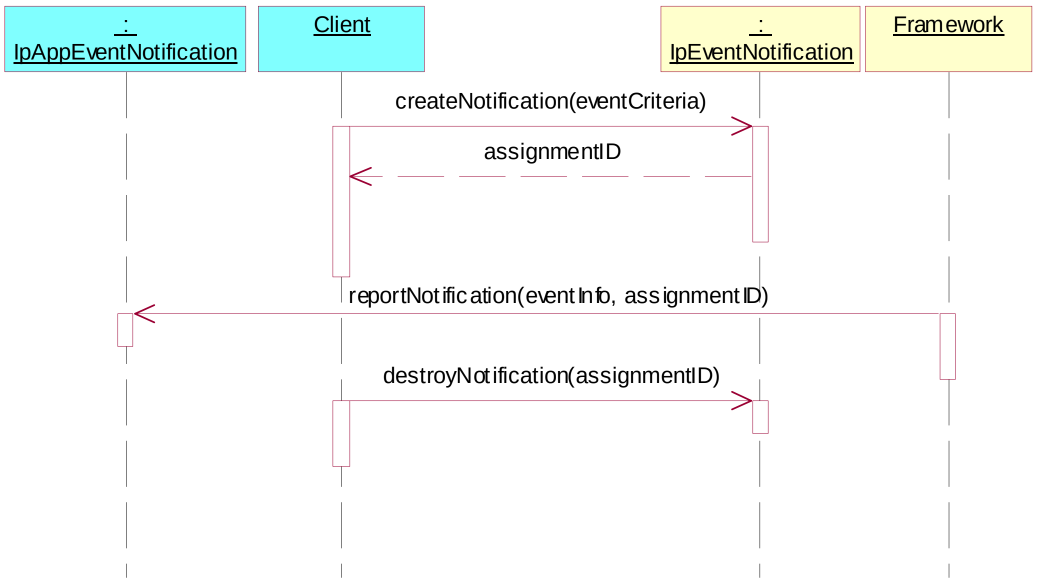
Precondition: IpEventNotification supported.

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Application) **IpAppEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned


: IpAppEventNotification
Client
: IpEventNotification
Framework
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
destroyNotification(assignmentID)

Test FW_FA_EN_02

Summary: **IpEventNotification**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.4.

Precondition: IpEventNotification supported.

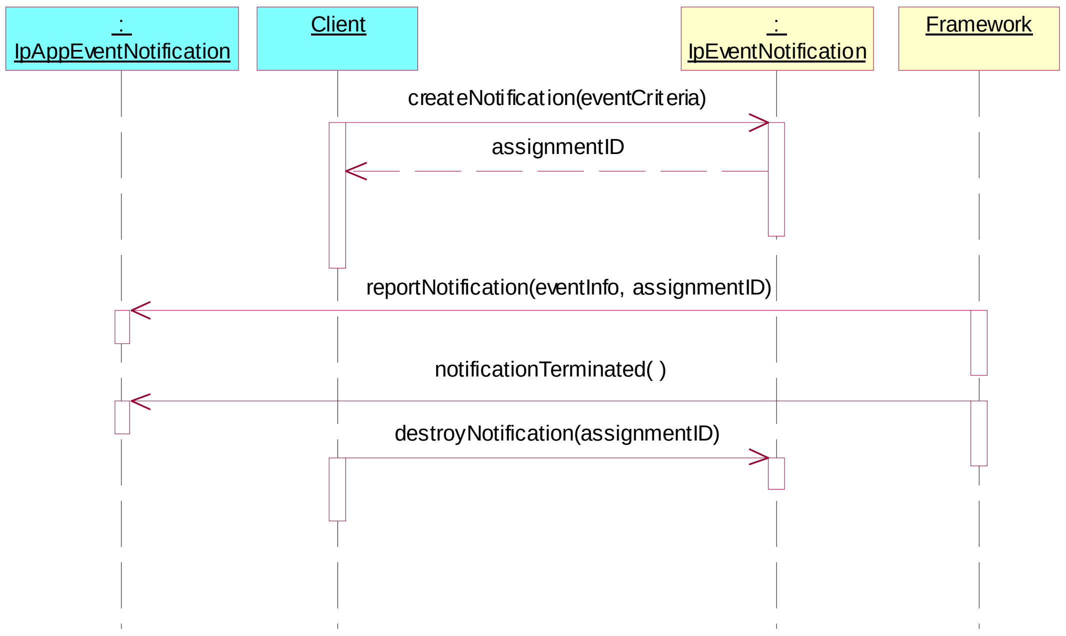
Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Application) **IpAppEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Triggered action: cause IUT to call **notificationTerminated()** method on the tester's (Application) **IpSvcEventNotification** interface.
   Parameters: none

4. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned


: IpAppEventNotification
Client
: IpEventNotification
Framework
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
notificationTerminated( )
destroyNotification(assignmentID)

Test FW_FA_EN_03

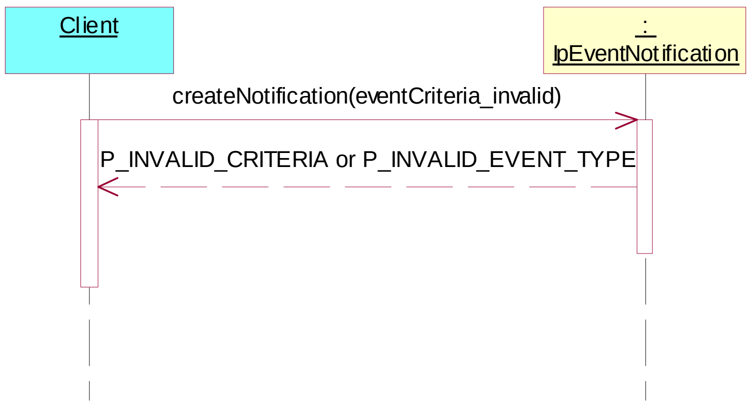
Summary: **IpEventNotification**, createNotification, P_INVALID_CRITERIA.

Reference: ES 202 915-3 [1], clause 7.3.4.

Precondition: IpEventNotification supported.

Test Sequence:

1. Method call **createNotification()**
   Parameters: invalid eventCriteria
   Check: P_INVALID_ CRITERIA or P_INVALID_EVENT_TYPE is returned.


Client
: IpEventNotification
createNotification(eventCriteria_invalid)
P_INVALID_CRITERIA or P_INVALID_EVENT_TYPE

Test FW_FA_EN_04

Summary: **IpEventNotification**, destroyNotification, P_INVALID_ ASSIGNMENT_ID.

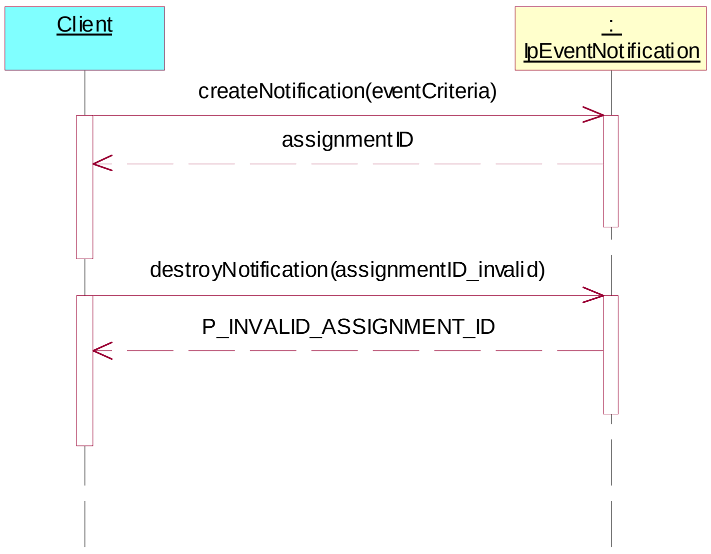
Reference: ES 202 915-3 [1], clause 7.3.4.

Precondition: IpEventNotification supported.

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Method call **destroyNotification()**
   Parameters: invalid assignmentID
   Check: P_INVALID_ ASSIGNMENT_ID is returned.


Client
: IpEventNotification
createNotification(eventCriteria)
assignmentID
destroyNotification(assignmentID_invalid)
P_INVALID_ASSIGNMENT_ID

5.4.3 Framework to Enterprise OperatorAPI

5.4.3.1 Service Subscription (SS)

Test FW_FO_SS_01

Summary: **IpClientAppManagement**, all methods, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **addSAGMembers()**
   Parameters: sagID, clientAppIDs
   Check: no exception is returned

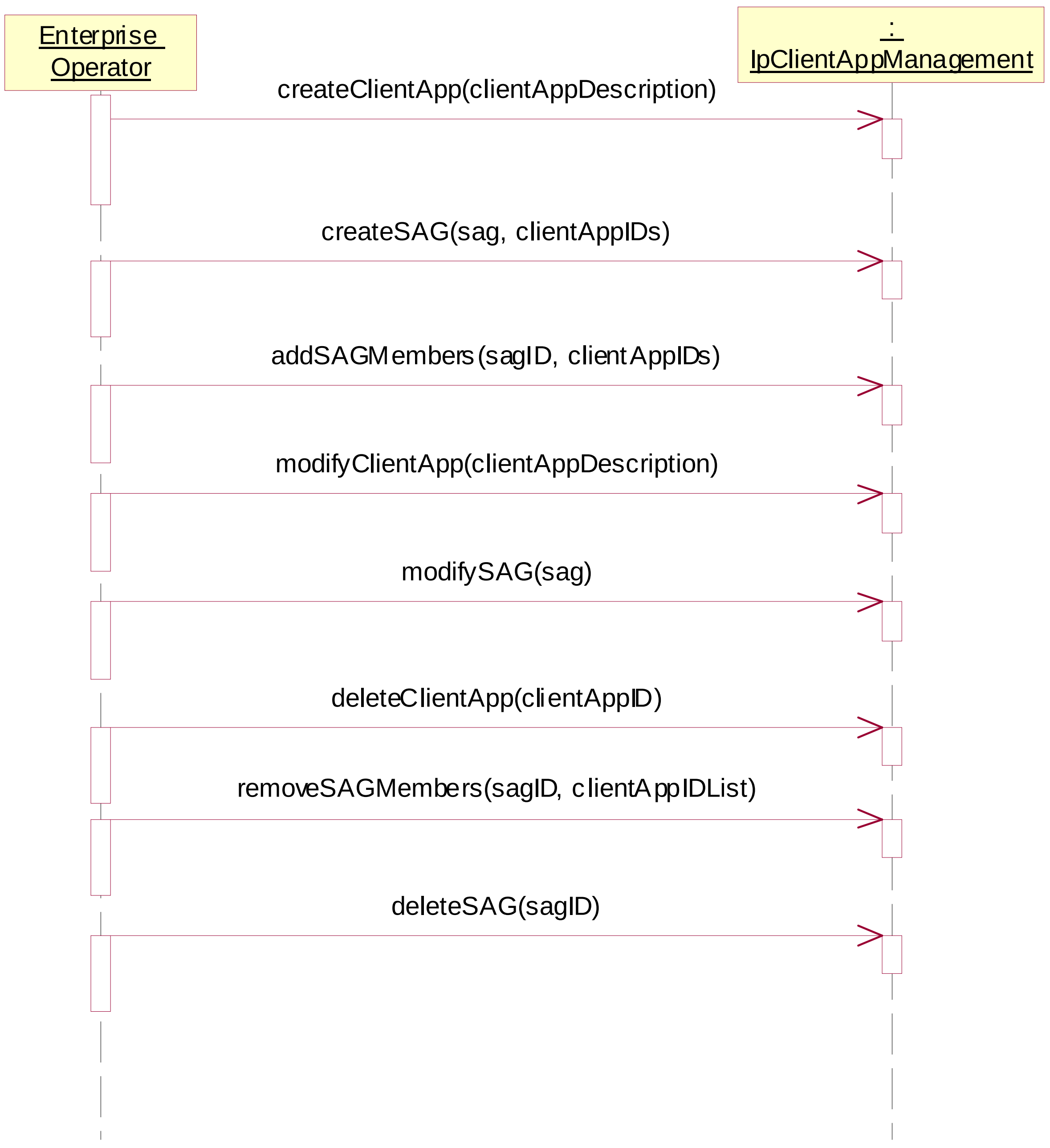
4. Method call **modifyClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

5. Method call **modifySAG()**
   Parameters: sag
   Check: no exception is returned

6. Method call **deleteClientApp()**
   Parameters: clientAppID
   Check: no exception is returned

7. Method call **removeSAGMembers()**
   Parameters: sagID, clientAppIDList
   Check: no exception is returned

8. Method call **deleteSAG()**
   Parameters: sagID
   Check: no exception is returned


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(sagID, clientAppIDs)
modifyClientApp(clientAppDescription)
modifySAG(sag)
deleteClientApp(clientAppID)
removeSAGMembers(sagID, clientAppIDList)
deleteSAG(sagID)

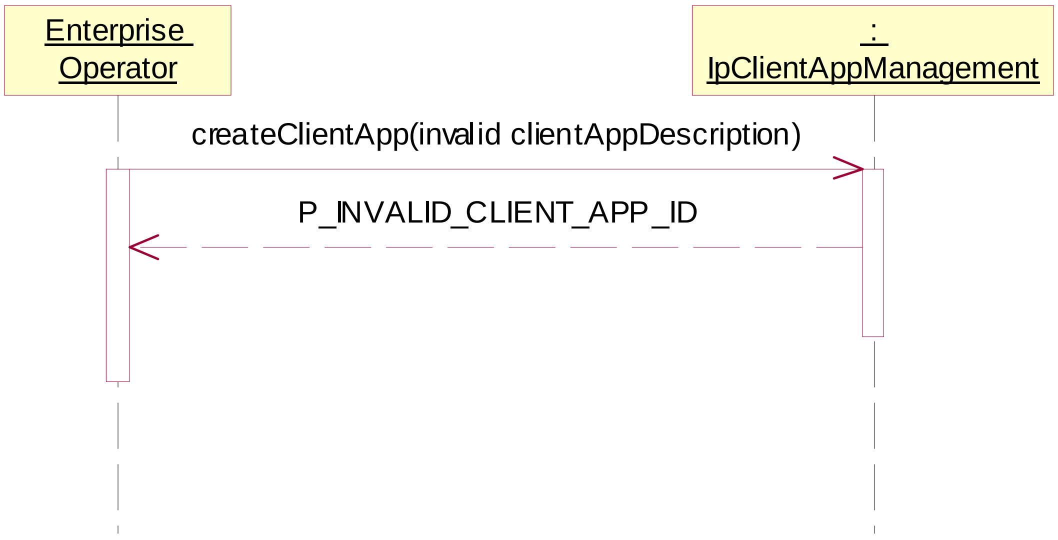
Test FW_FO_SS_02

Summary: **IpClientAppManagement**, createClientApp, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: invalid clientAppDescription
   Check: P_INVALID_CLIENT_APP_ID is returned.



Test FW_FO_SS_03

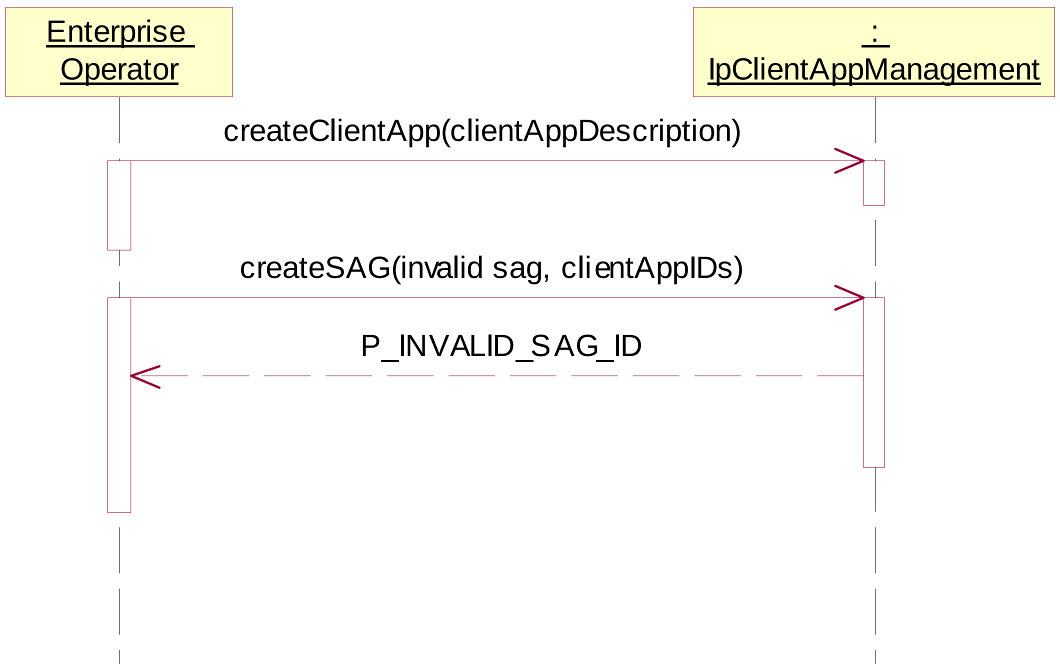
Summary: **IpClientAppManagement**, createSAG, P_INVALID_SAG_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: invalid sag, clientAppIDs
   Check: P_INVALID_SAG_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(invalid sag, clientAppIDs)
P_INVALID_SAG_ID

Test FW_FO_SS_04

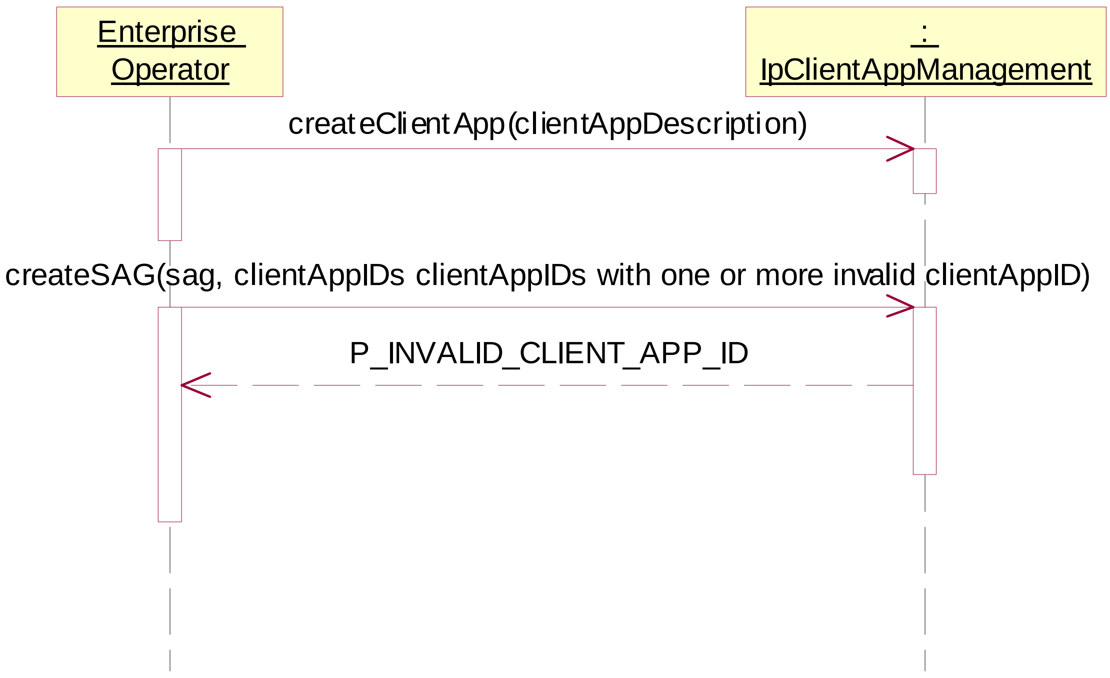
Summary: **IpClientAppManagement**, createSAG, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs with one or more invalid clientAppID
   Check: P_INVALID_CLIENT_APP_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs clientAppIDs with one or more invalid clientAppID)
P_INVALID_CLIENT_APP_ID

Test FW_FO_SS_05

Summary: **IpClientAppManagement**, addSAGMembers, P_INVALID_SAG_ID.

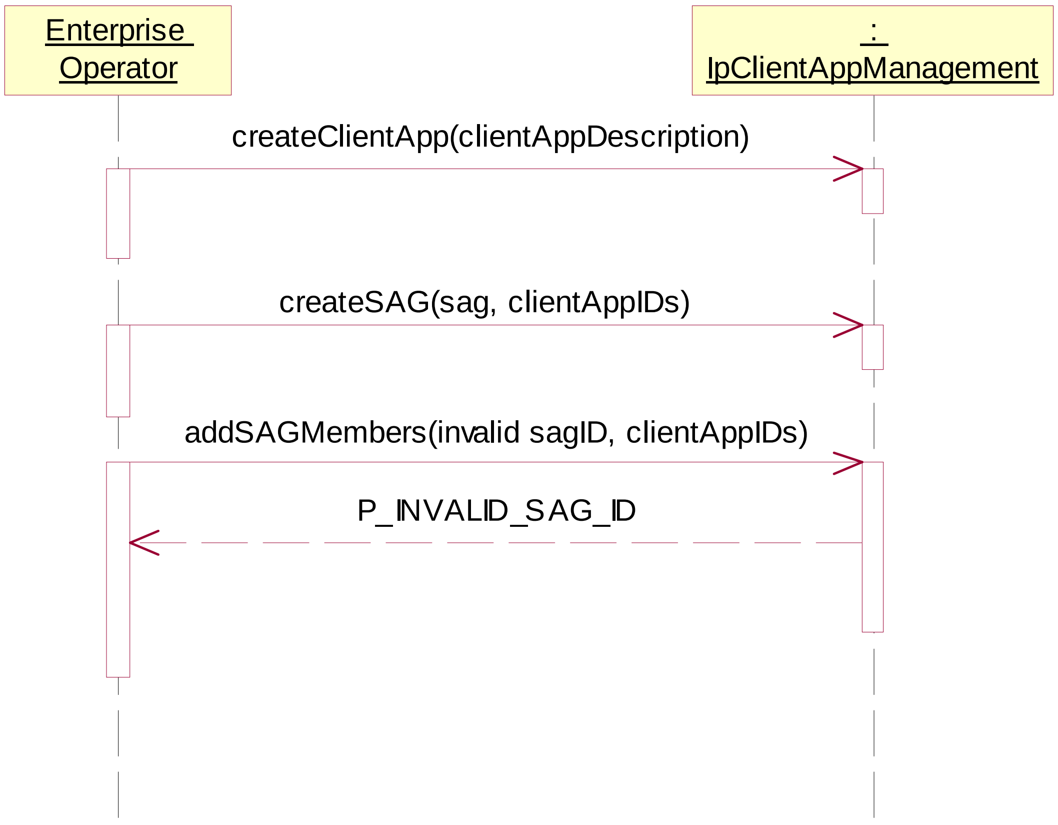
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **addSAGMembers ()**
   Parameters: invalid sagID, clientAppIDs
   Check: P_INVALID_SAG_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(invalid sagID, clientAppIDs)
P_INVALID_SAG_ID

Test FW_FO_SS_06

Summary: **IpClientAppManagement**, addSAGMembers, P_INVALID_CLIENT_APP_ID.

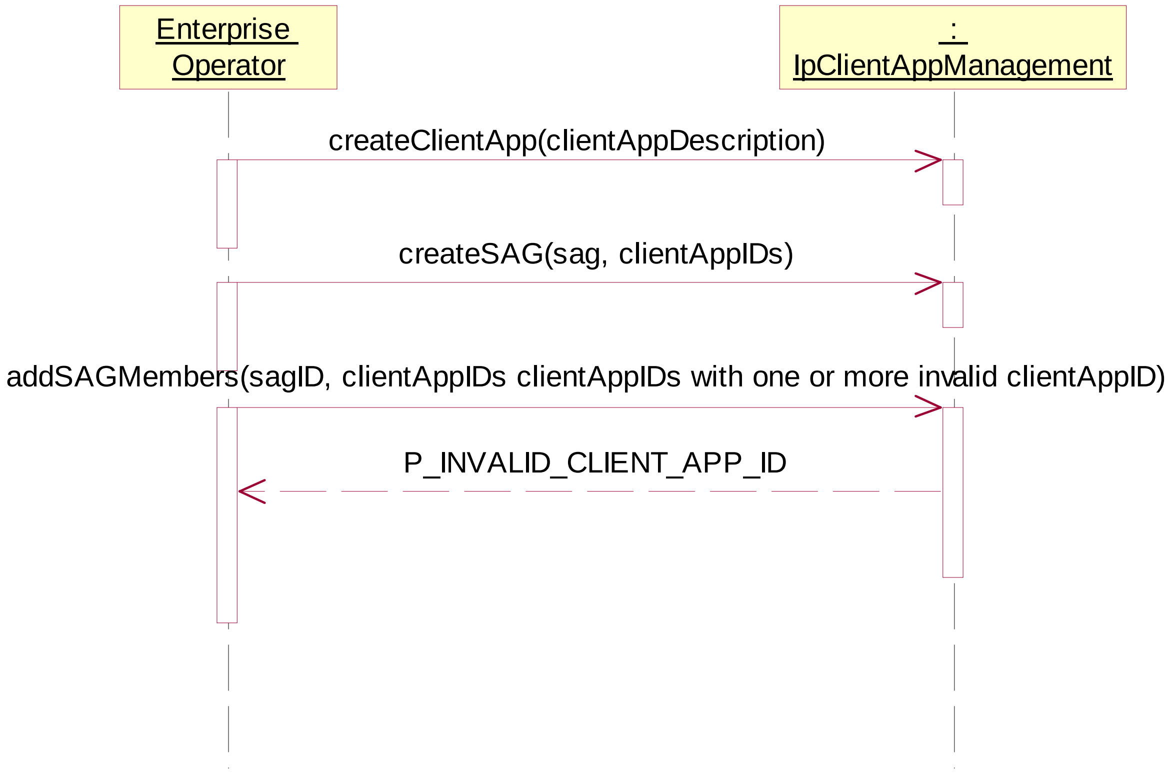
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **addSAGMembers ()**
   Parameters: sagID, clientAppIDs with one or more invalid clientAppID
   Check: P_INVALID_CLIENT_APP_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(sagID, clientAppIDs clientAppIDs with one or more invalid clientAppID)
P_INVALID_CLIENT_APP_ID

Test FW_FO_SS_07

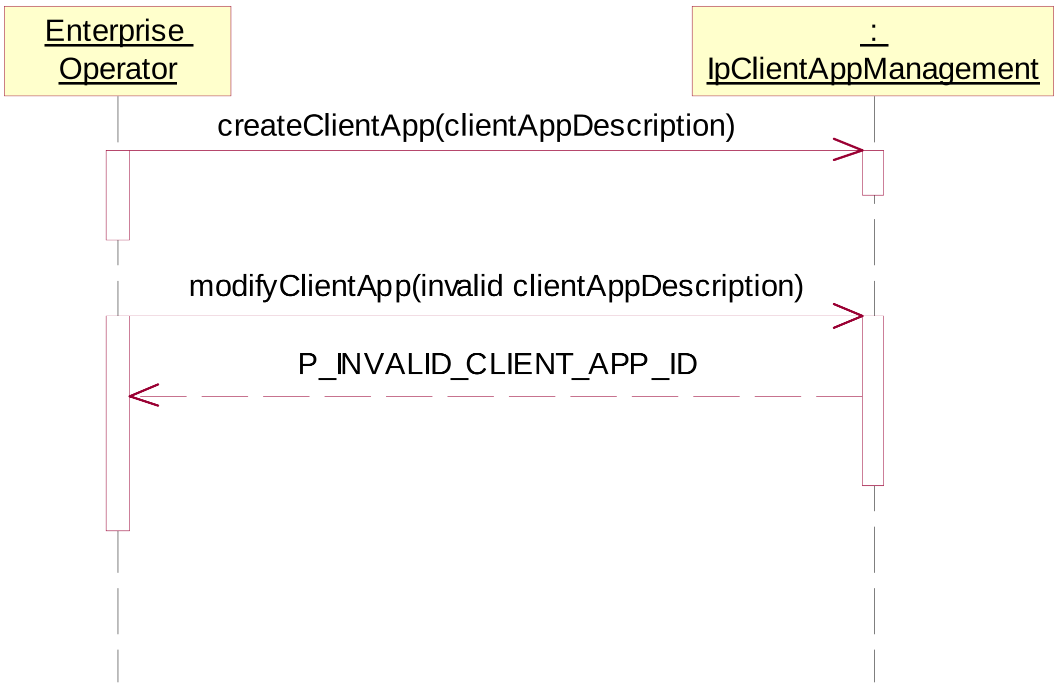
Summary: **IpClientAppManagement**, modifyClientApp, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **modifyClientApp ()**
   Parameters: invalid clientAppDescription
   Check: P_INVALID_CLIENT_APP_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
modifyClientApp(invalid clientAppDescription)
P_INVALID_CLIENT_APP_ID

Test FW_FO_SS_08

Summary: **IpClientAppManagement**, modifySAG, P_INVALID_SAG_ID.

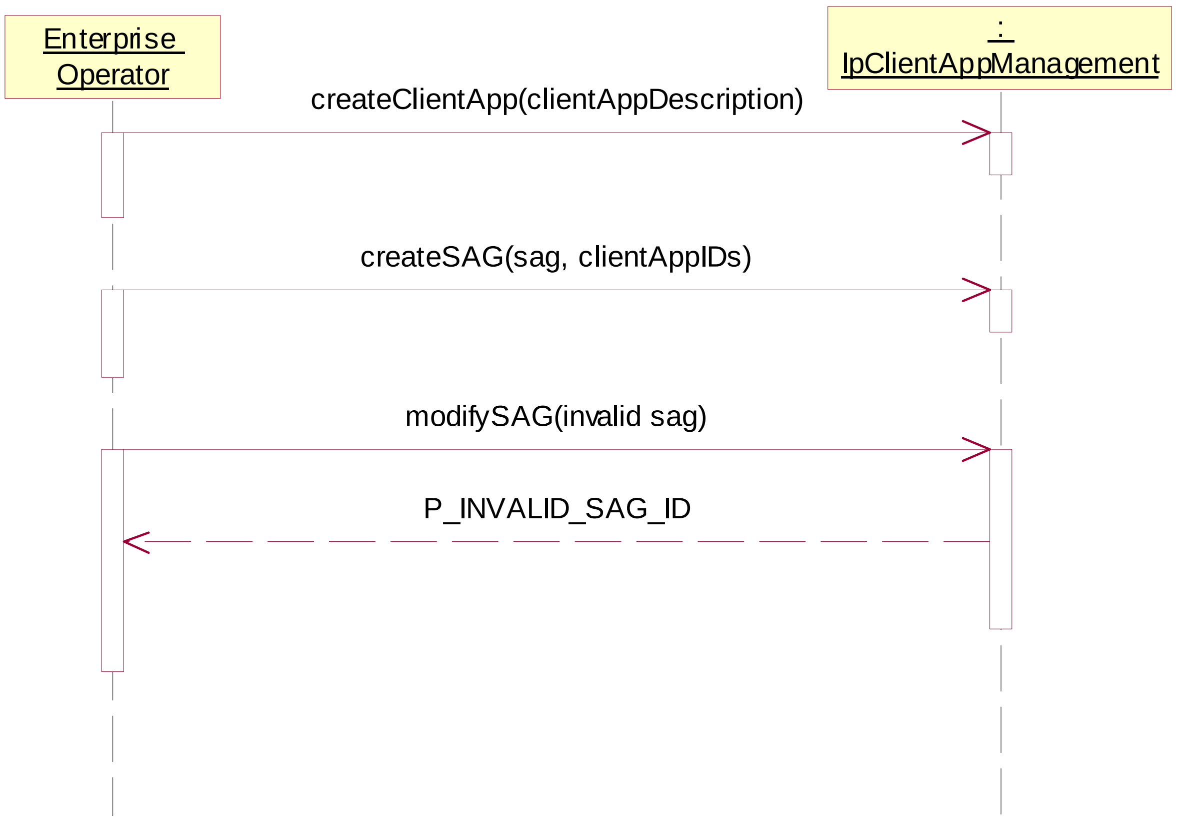
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **modifySAG()**
   Parameters: invalid sag
   Check: P_INVALID_SAG_ID is returned.



Test FW_FO_SS_09

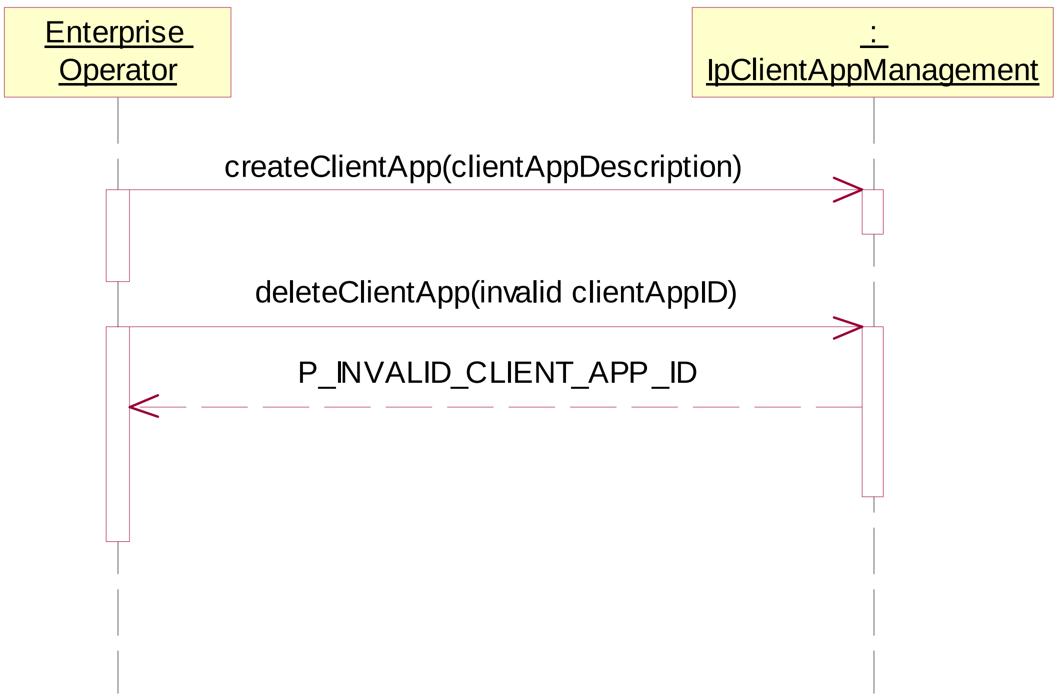
Summary: **IpClientAppManagement**, deleteClientApp, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **deleteClientApp ()**
   Parameters: invalid clientAppID
   Check: P_INVALID_CLIENT_APP_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
deleteClientApp(invalid clientAppID)
P_INVALID_CLIENT_APP_ID

Test FW_FO_SS_10

Summary: **IpClientAppManagement**, removeSAGMembers, P_INVALID_SAG_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

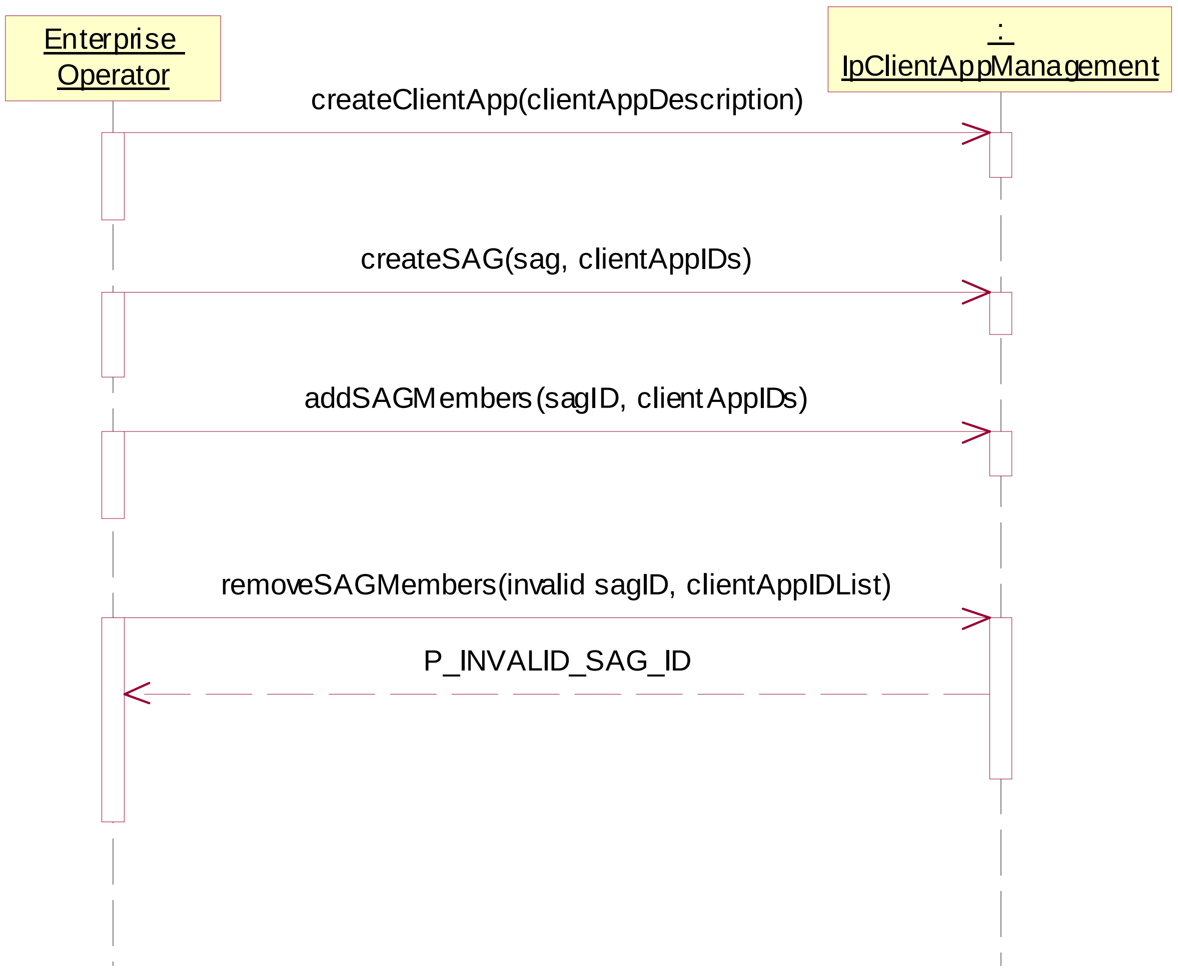
Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **addSAGMembers()**
   Parameters: sagID, clientAppIDs
   Check: no exception is returned

4. Method call **removeSAGMembers ()**
   Parameters: invalid sagID, clientAppIDList
   Check: P_INVALID_SAG_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(sagID, clientAppIDs)
removeSAGMembers(invalid sagID, clientAppIDList)
P_INVALID_SAG_ID

Test FW_FO_SS_11

Summary: **IpClientAppManagement**, removeSAGMembers, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

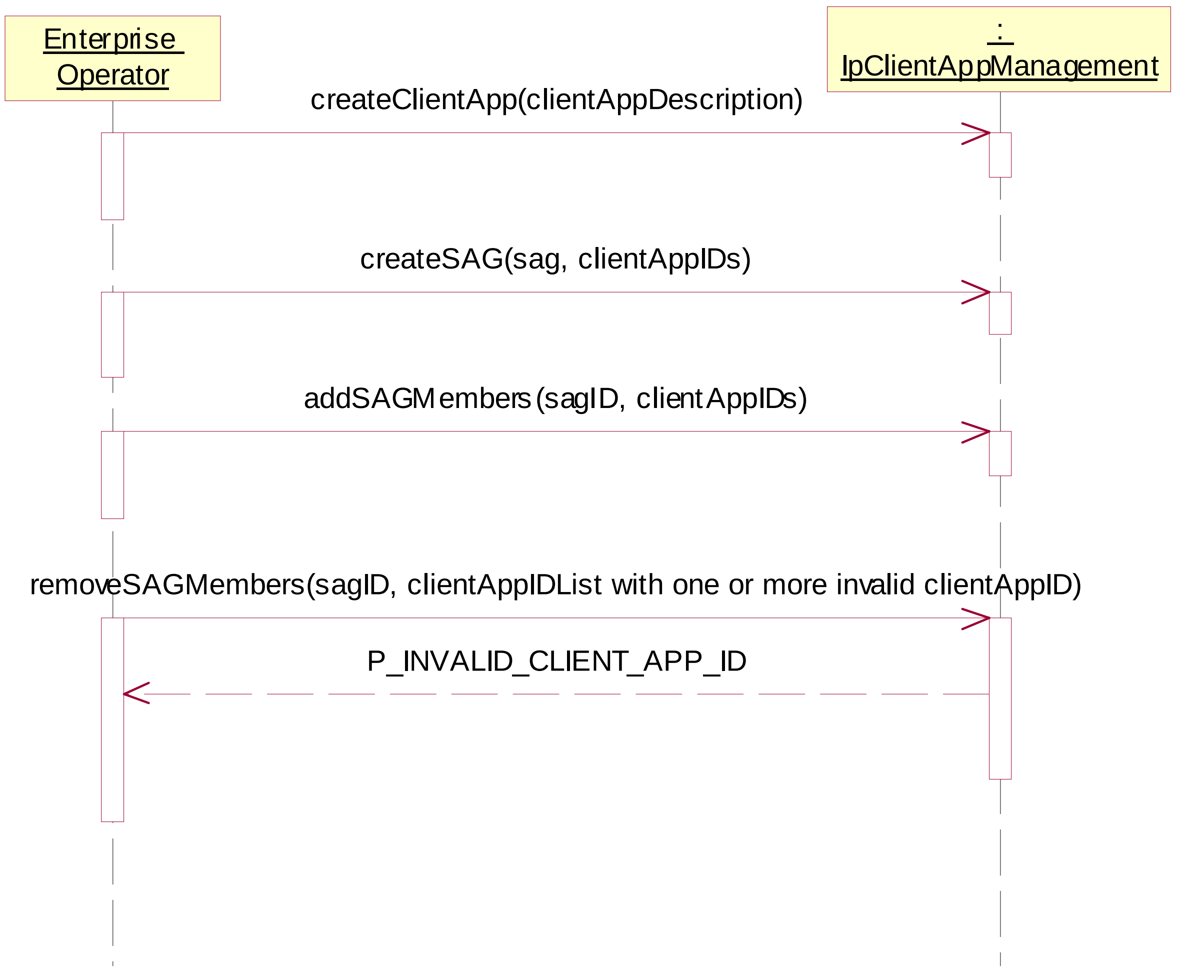
Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **addSAGMembers()**
   Parameters: sagID, clientAppIDs
   Check: no exception is returned

4. Method call **removeSAGMembers ()**
   Parameters: sagID, clientAppIDList with one or more invalid clientAppID
   Check: P_INVALID_CLIENT_APP_ID is returned.


Enterprise Operator
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(sagID, clientAppIDs)
removeSAGMembers(sagID, clientAppIDList with one or more invalid clientAppID)
P_INVALID_CLIENT_APP_ID

Test FW_FO_SS_12

Summary: **IpClientAppManagement**, deleteSAG, P_INVALID_SAG_ID.

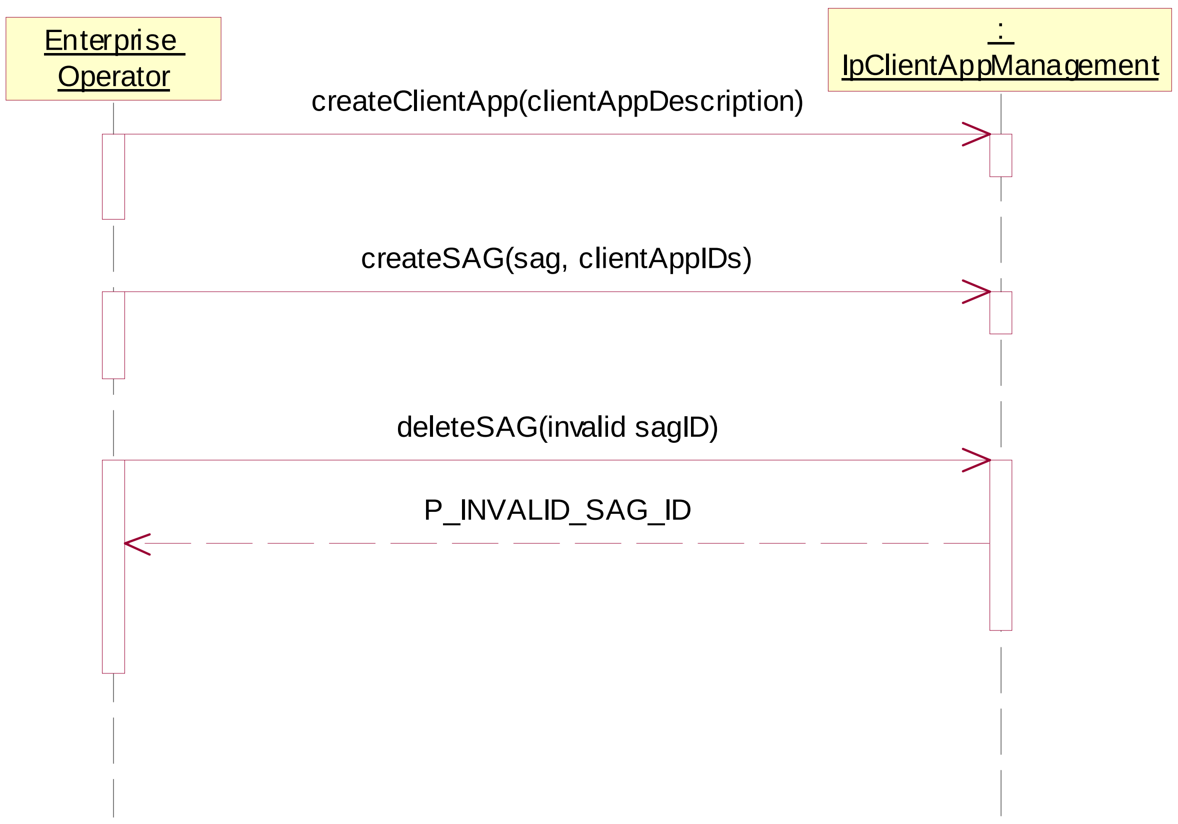
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **deleteSAG ()**
   Parameters: invalid sagID
   Check: P_INVALID_SAG_ID is returned.



Test FW_FO_SS_41

Summary: **IpClientAppManagement**, requestConflictInfo, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **createClientApp()**
   Parameters: clientAppDescription with parameters incompatible with other Client App in same SAG.
   Check: no exception is returned

3. Method call **createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

4. Method call **addSAGMembers()**
   Parameters: sagID, clientAppIDs including ID for ClientApp describe in 1.
   Check: no exception is returned

5. Method call **addSAGMembers()**
   Parameters: sagID, clientAppIDs including ID for ClientApp describe in 1. and in 2.
   Check: P_INVALID_ADDITION_TO_SAG is returned

6. Method call **requestConflictInfo ()**
   Parameters: None
   Check: valid TpAddSagMembersConflictList is returned

Test FW_FO_SS_13

Summary: **IpClientAppInfoQuery**, all methods, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **listClientApps()**
   Parameters: None
   Check: valid TpClientAppIDList is returned including ClientAppID from 1.

3. Method call **describeClientApp()**
   Parameters: clientAppID from 1.
   Check: valid TpClientAppDescription is returned

4. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

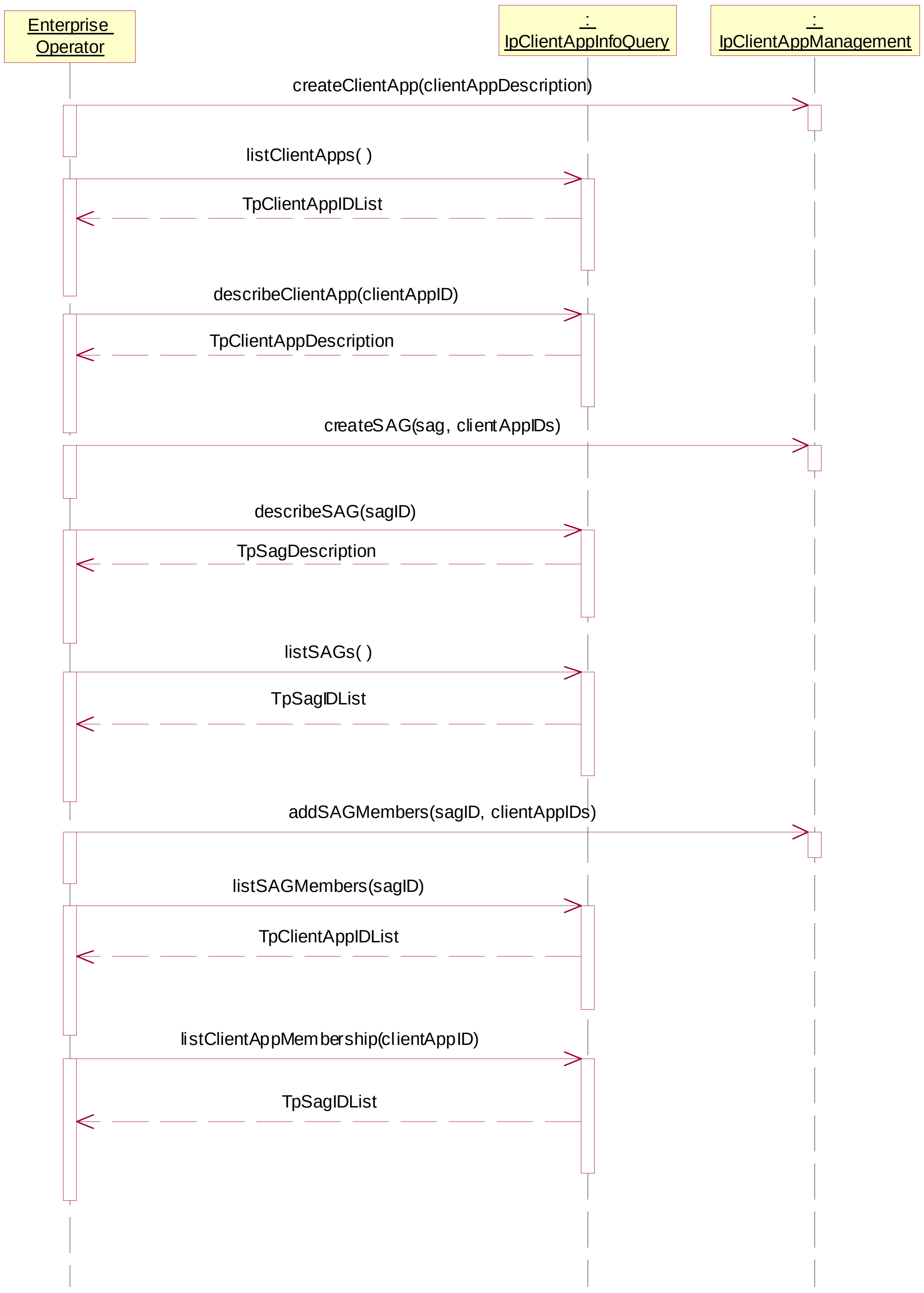
5. Method call **describeSAG()**
   Parameters: sagID given in 4.
   Check: TpSagDescription is returned

6. Method call **listSAGs()**
   Parameters: None
   Check: valid TpSagIDList is returned including sag given in 4.

7. Method call **IpClientAppManagement.addSAGMembers()**
   Parameters: sagID given in 4, clientAppIDs
   Check: no exception is returned

8. Method call **listSAGMembers()**
   Parameters: sagID given in 4
   Check: valid TpClientAppIDList is returned included clientAppIDs given in 7.

9. Method call **listClientAppMembership()**
   Parameters: clientAppID given in 1.
   Check: valid TpSagIDList


Enterprise Operator
: IpClientAppInfoQuery
: IpClientAppManagement
createClientApp(clientAppDescription)
listClientApps( )
TpClientAppIDList
describeClientApp(clientAppID)
TpClientAppDescription
createSAG(sag, clientAppIDs)
describeSAG(sagID)
TpSagDescription
listSAGs( )
TpSagIDList
addSAGMembers(sagID, clientAppIDs)
listSAGMembers(sagID)
TpClientAppIDList
listClientAppMembership(clientAppID)
TpSagIDList

Test FW_FO_SS_14

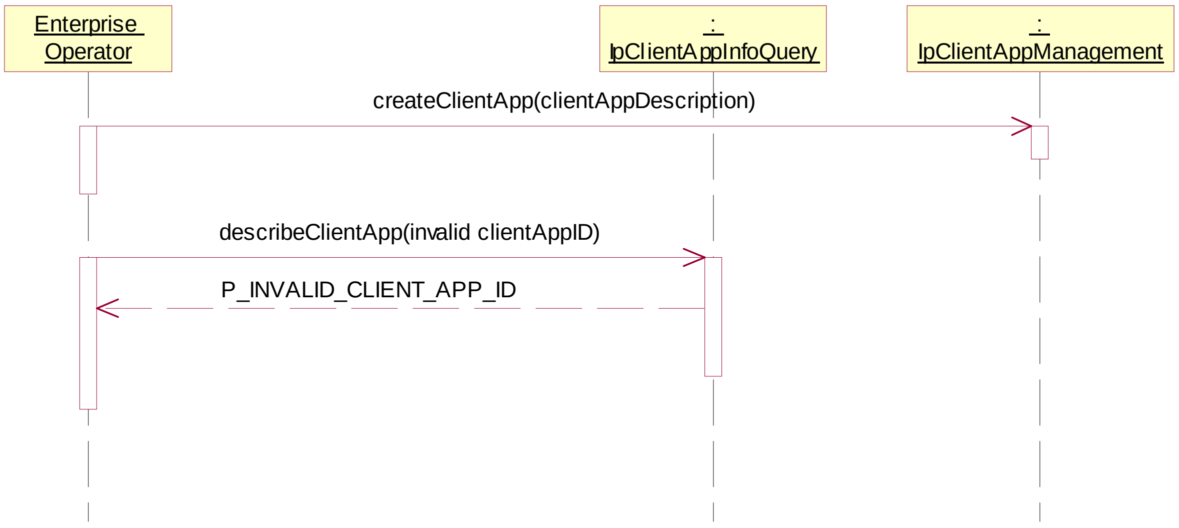
Summary: **IpClientAppInfoQuery**, describeClientApp, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **describeClientApp ()**
   Parameters: invalid clientAppID
   Check: P_INVALID_CLIENT_APP_ID is returned.



Test FW_FO_SS_15

Summary: **IpClientAppInfoQuery**, describeSAG, P_INVALID_SAG_ID.

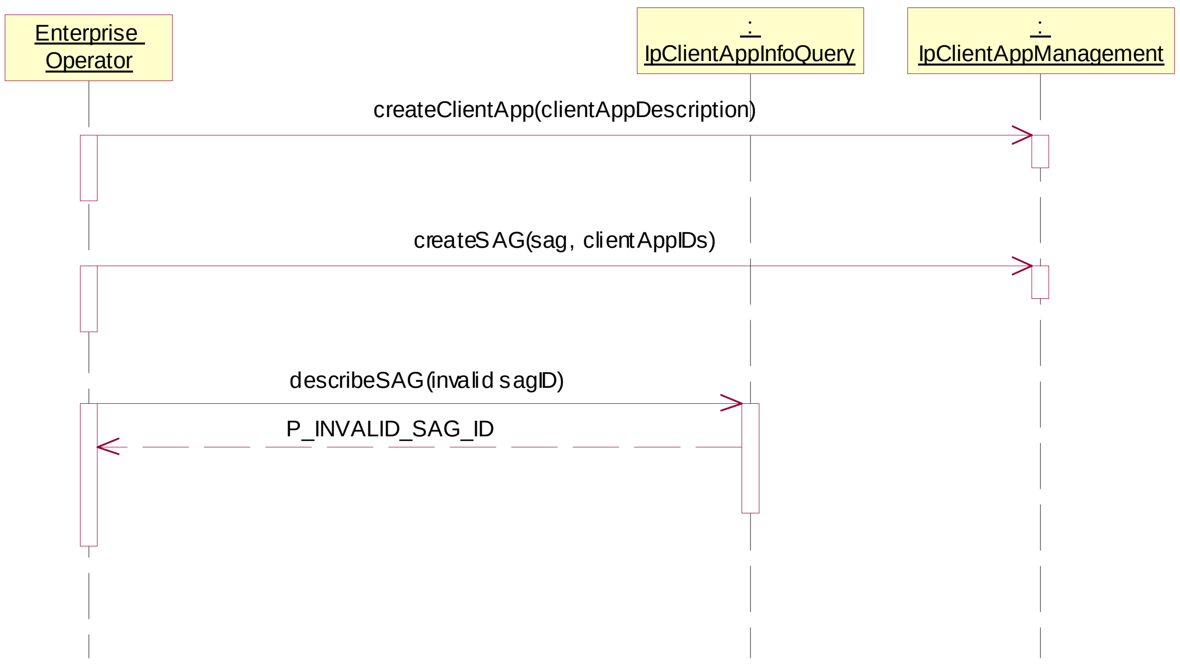
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **describeSAG()**
   Parameters: invalid sagID
   Check: P_INVALID_SAG_ID is returned.


Enterprise Operator
: IpClientAppInfoQuery
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
describeSAG(invalid sagID)
P_INVALID_SAG_ID

Test FW_FO_SS_16

Summary: **IpClientAppInfoQuery**, listSAGMembers, P_INVALID_SAG_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

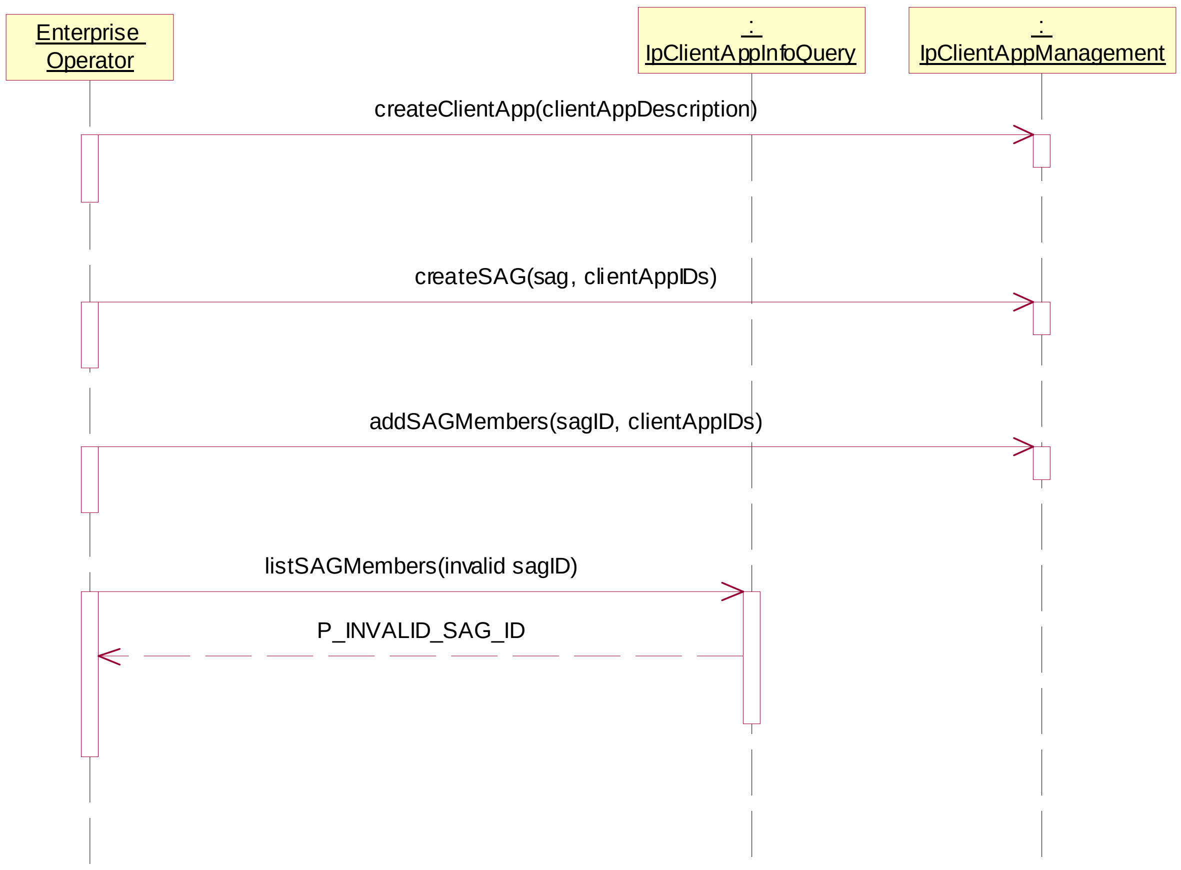
Test Sequence:

1. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **IpClientAppManagement.addSAGMembers()**
   Parameters: sagID given in 4, clientAppIDs
   Check: no exception is returned

4. Method call **listSAGMembers()**
   Parameters: invalid sagID
   Check: P_INVALID_SAG_ID is returned.


Enterprise Operator
: IpClientAppInfoQuery
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(sagID, clientAppIDs)
listSAGMembers(invalid sagID)
P_INVALID_SAG_ID

Test FW_FO_SS_17

Summary: **IpClientAppInfoQuery**, listClientAppMembership, P_INVALID_CLIENT_APP_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

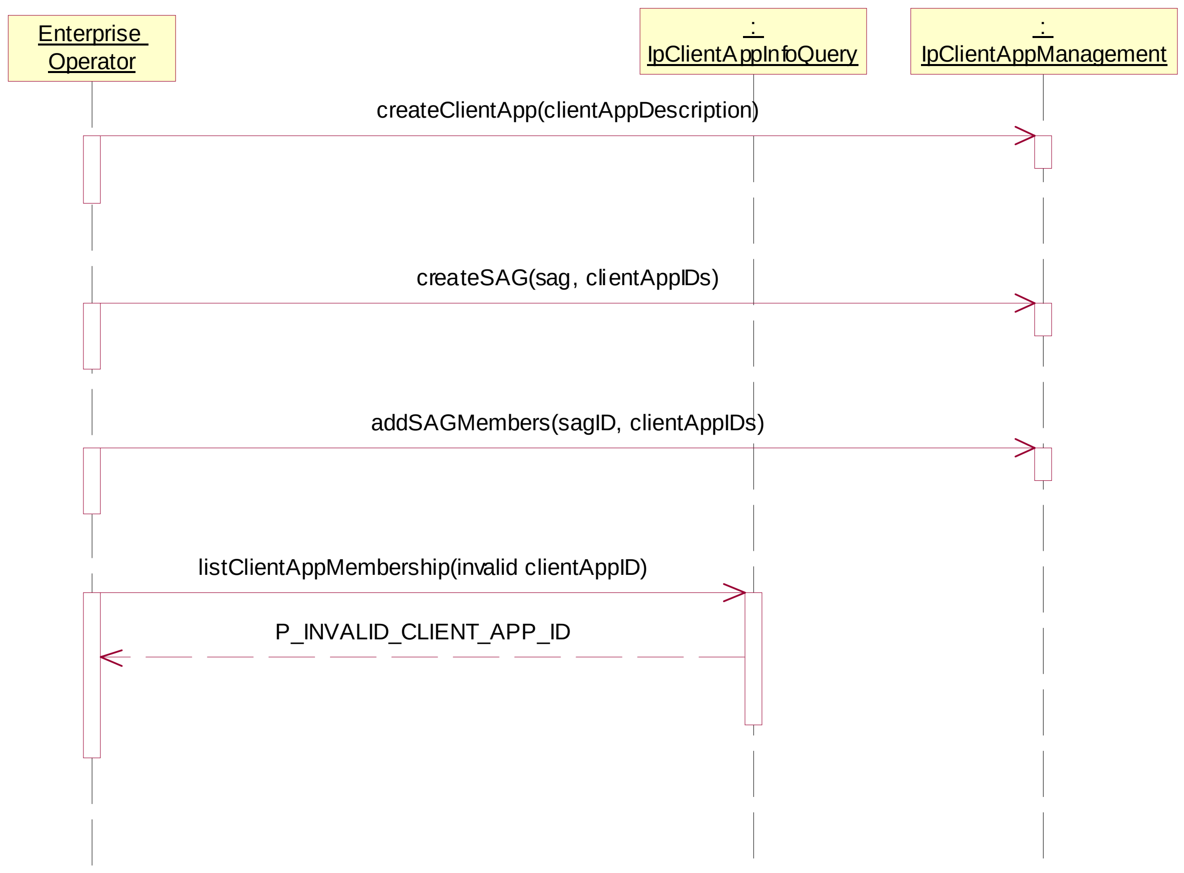
Test Sequence:

1. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

2. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

3. Method call **IpClientAppManagement.addSAGMembers()**
   Parameters: sagID given in 4, clientAppIDs
   Check: no exception is returned

4. Method call **listClientAppMembership()**
   Parameters: invalid clientAppID
   Check: P_INVALID_CLIENT_APP_ID is returned.


Enterprise Operator
: IpClientAppInfoQuery
: IpClientAppManagement
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
addSAGMembers(sagID, clientAppIDs)
listClientAppMembership(invalid clientAppID)
P_INVALID_CLIENT_APP_ID

Test FW_FO_SS_18

Summary: **IpServiceContractManagement**, all methods, successful.

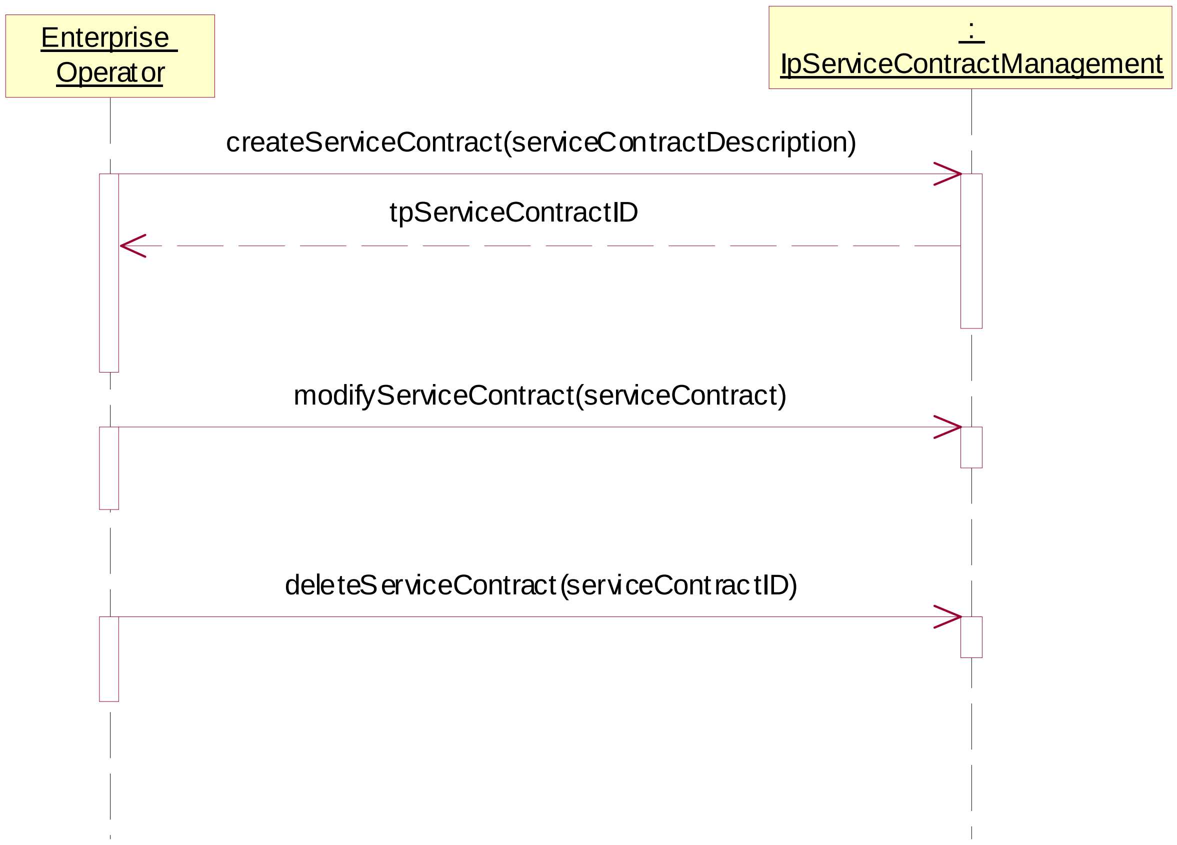
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

2. Method call **modifyServiceContract()**
   Parameters: serviceContract given in 1.
   Check: no exception is returned

3. Method call **deleteServiceContract()**
   Parameters: serviceContractID given in 1.
   Check: no exception is returned


Enterprise Operator
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
tpServiceContractID
modifyServiceContract(serviceContract)
deleteServiceContract(serviceContractID)

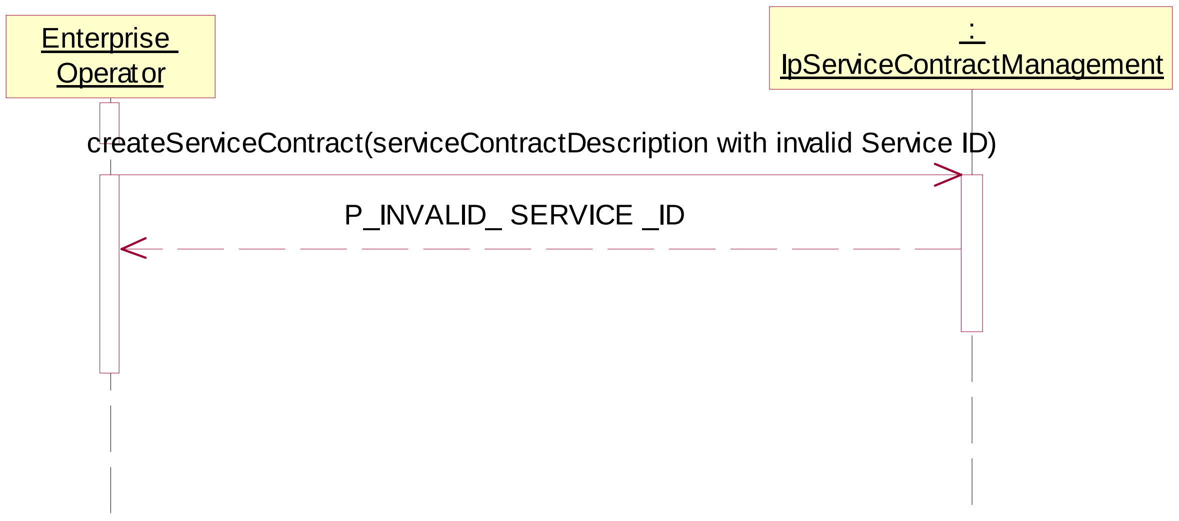
Test FW_FO_SS_19

Summary: **IpServiceContractManagement**, createServiceContract, P_INVALID_SERVICE _ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createServiceContract ()**
   Parameters: serviceContractDescription with invalid ID
   Check: P_INVALID_ SERVICE _ID is returned.


Enterprise Operator
: IpServiceContractManagement
createServiceContract(serviceContractDescription with invalid Service ID)
P_INVALID_ SERVICE _ID

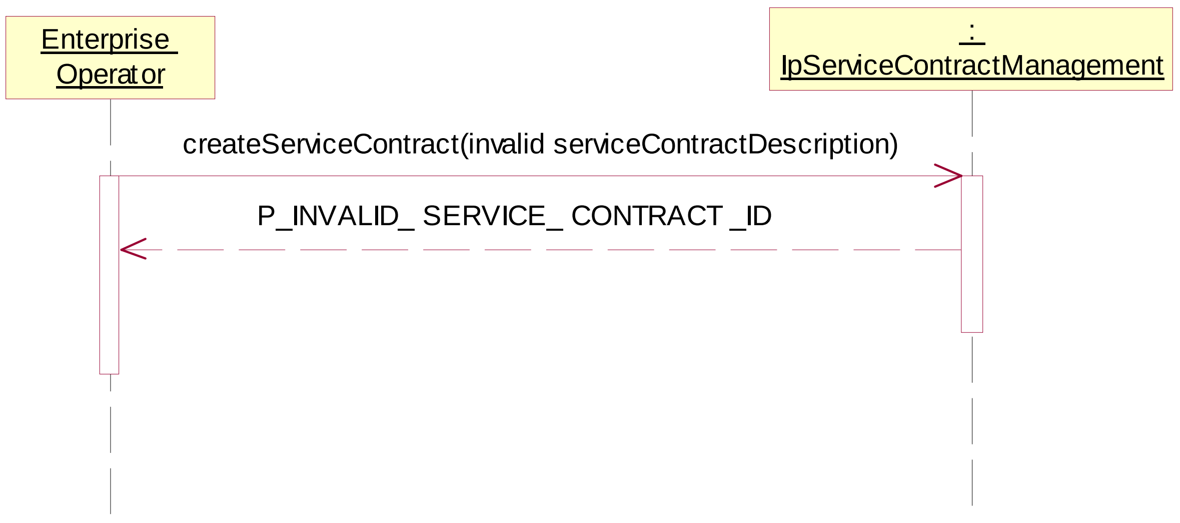
Test FW_FO_SS_20

Summary: **IpServiceContractManagement**, createServiceContract, P_INVALID_SERVICE_CONTRACT_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createServiceContract ()**
Parameters: invalid serviceContractDescription
Check: P_INVALID_ SERVICE_ CONTRACT _ID is returned.


Enterprise Operator
: IpServiceContractManagement
createServiceContract(invalid serviceContractDescription)
P_INVALID_ SERVICE_ CONTRACT _ID

Test FW_FO_SS_21

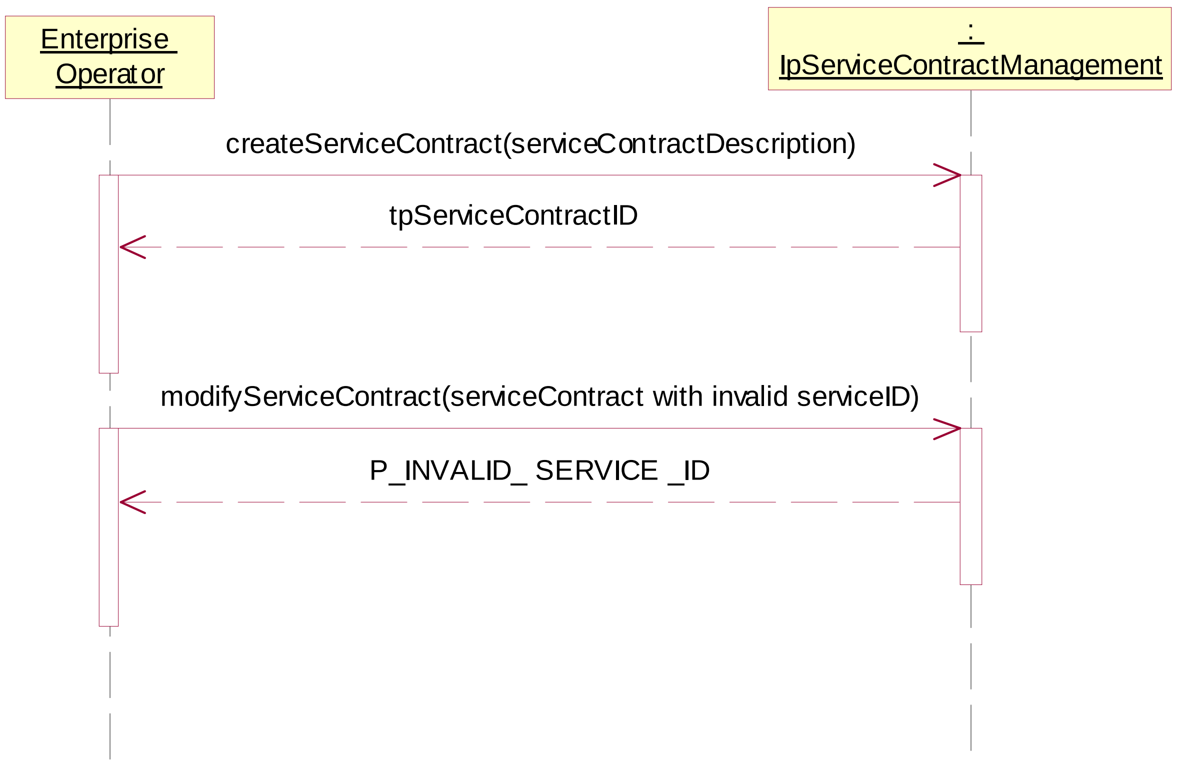
Summary: **IpServiceContractManagement**, modifyServiceContract, P_INVALID_SERVICE _ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createServiceContract()**
Parameters: serviceContractDescription
Check: valid TpServiceContractlID is returned

2. Method call **modifyServiceContract ()**
Parameters: serviceContract with invalid service ID
Check: P_INVALID_ SERVICE _ID is returned.


Enterprise Operator
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
tpServiceContractID
modifyServiceContract(serviceContract with invalid serviceID)
P_INVALID_ SERVICE _ID

Test FW_FO_SS_22

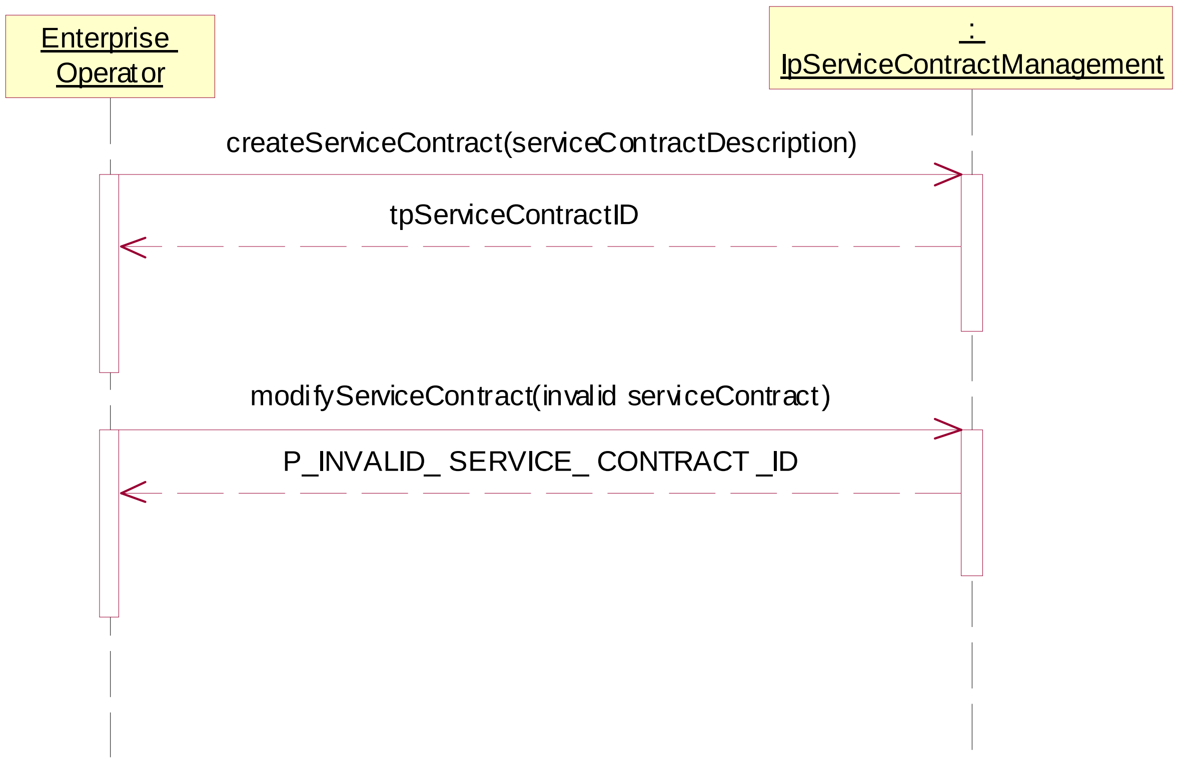
Summary: **IpServiceContractManagement**, modifyServiceContract, P_INVALID_SERVICE_CONTRACT_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

2. Method call **modifyServiceContract ()**
   Parameters: invalid serviceContract
   Check: P_INVALID_ SERVICE_ CONTRACT _ID is returned.


Enterprise Operator
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
tpServiceContractID
modifyServiceContract(invalid serviceContract)
P_INVALID_ SERVICE_ CONTRACT _ID

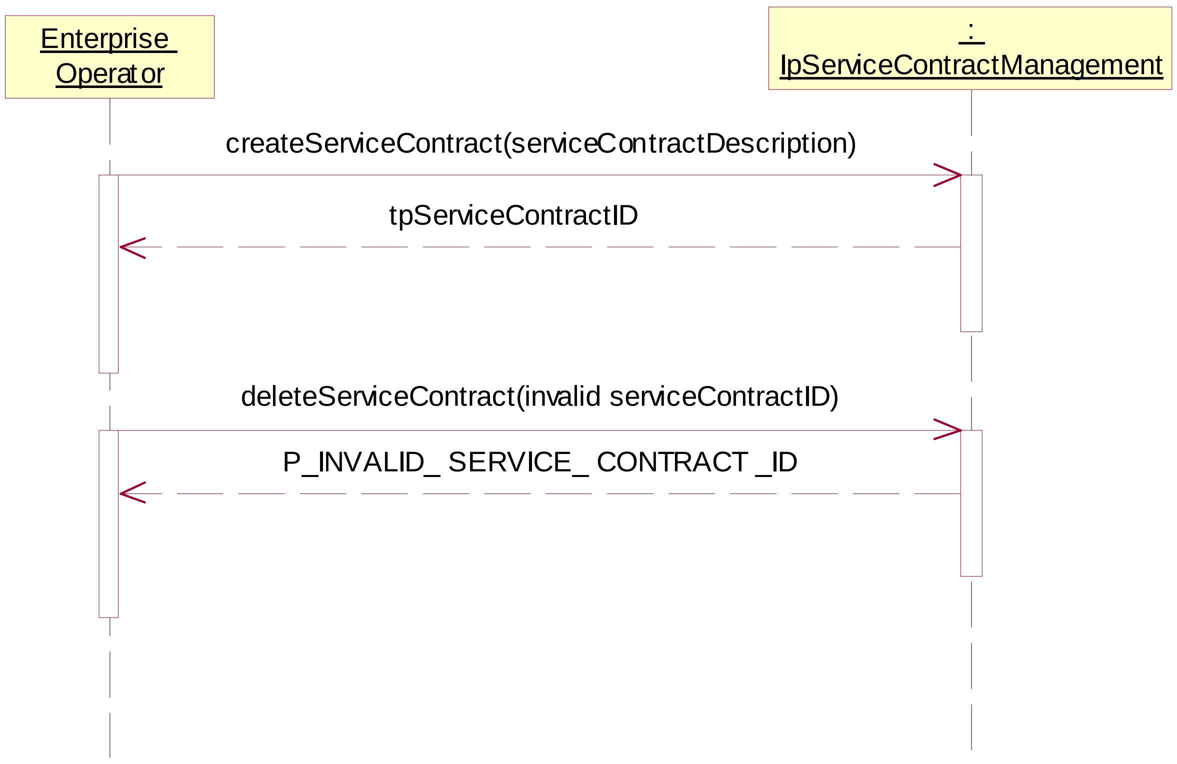
Test FW_FO_SS_23

Summary: **IpServiceContractManagement**, deleteServiceContract, P_INVALID_SERVICE_CONTRACT_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **deleteServiceContract ()**
   Parameters: invalid serviceContractID
   Check: P_INVALID_ SERVICE_ CONTRACT _ID is returned.


Enterprise Operator
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
tpServiceContractID
deleteServiceContract(invalid serviceContractID)
P_INVALID_ SERVICE_ CONTRACT _ID

Test FW_FO_SS_24

Summary: **IpServiceProfileManagement**, all methods, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **modifyServiceProfile()**
   Parameters: serviceProfile given in 1.
   Check: no exception is returned

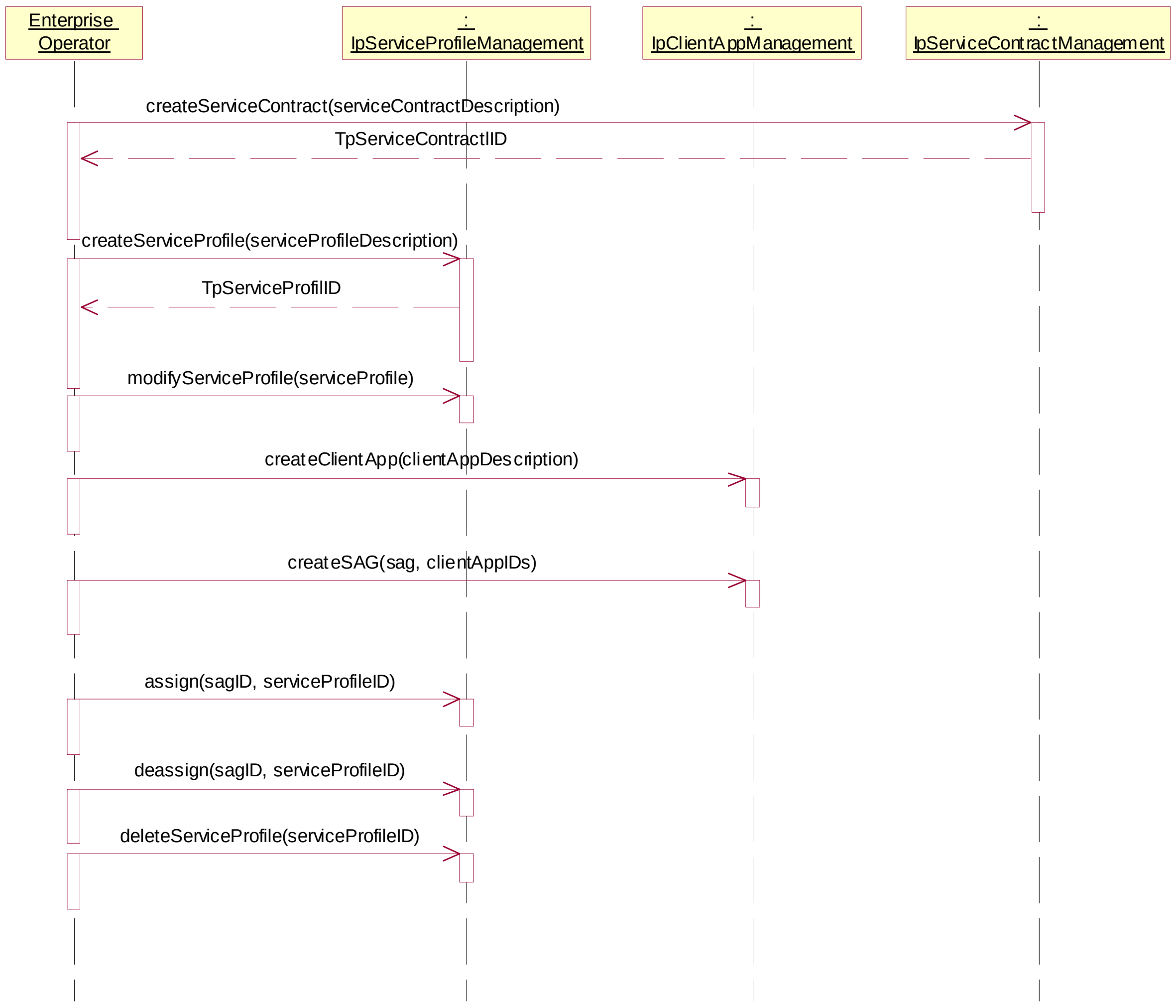
4. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

5. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

6. Method call **assign()**
   Parameters: sagID given in 3., serviceProfileID given in 1.
   Check: no exception is returned

7. Method call **desassign()**
   Parameters: sagID given in 3., serviceProfileID given in 1.
   Check: no exception is returned

8. Method call **deleteServiceProfile()**
   Parameters: serviceProfileID given in 1.
   Check: no exception is returned


Enterprise Operator
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractIID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
modifyServiceProfile(serviceProfile)
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(sagID, serviceProfileID)
deassign(sagID, serviceProfileID)
deleteServiceProfile(serviceProfileID)

Test FW_FO_SS_25

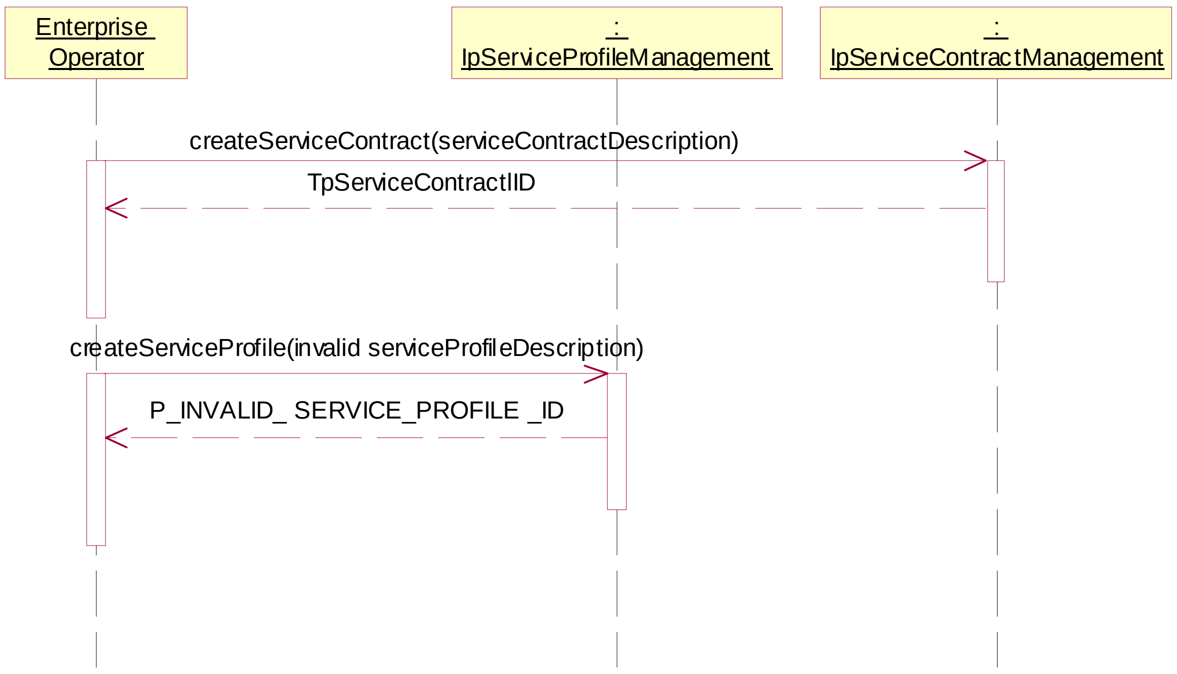
Summary: **IpServiceProfileManagement**, createServiceProfile, P_INVALID_SERVICE_PROFILE_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractIID is returned

2. Method call **createServiceProfile ()**
   Parameters: invalid serviceProfileDescription
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractIID
createServiceProfile(invalid serviceProfileDescription)
P_INVALID_ SERVICE_PROFILE _ID

Test FW_FO_SS_26

Summary: **IpServiceProfileManagement**, modifyServiceProfile, P_INVALID_SERVICE_PROFILE_ID.

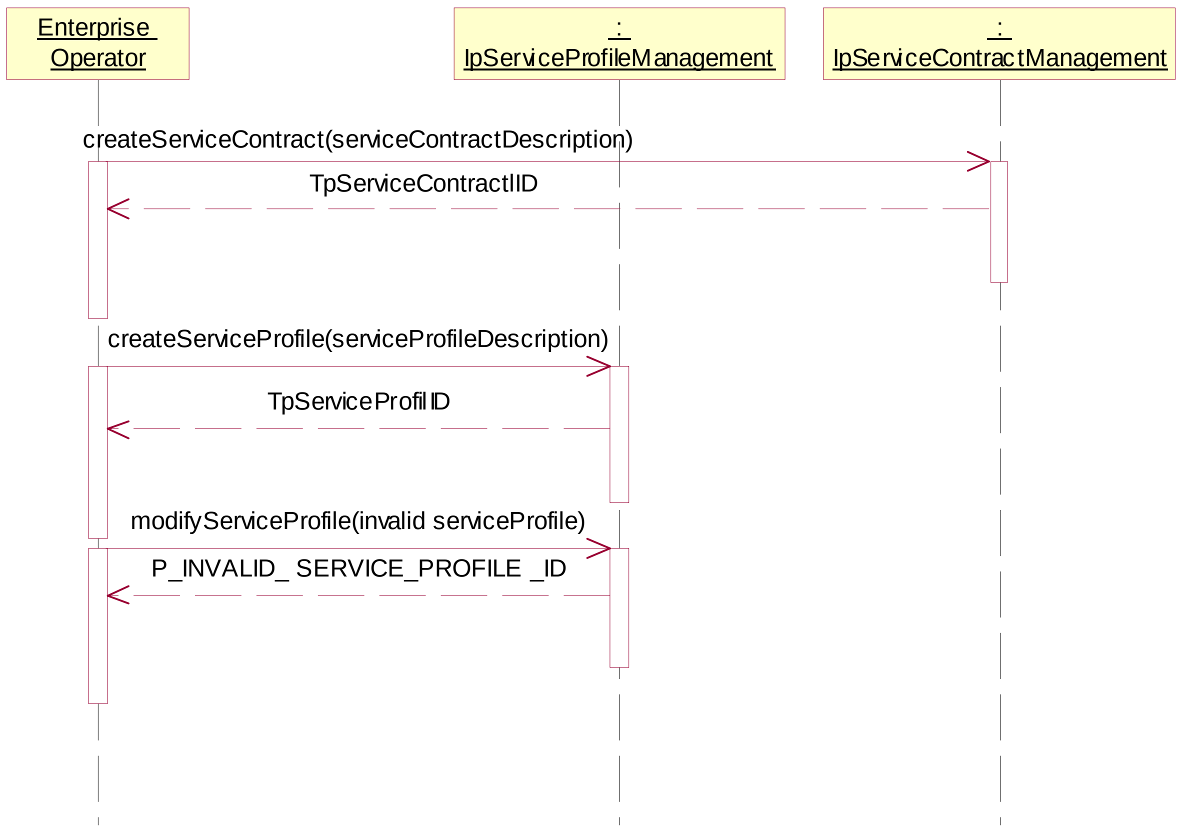
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **modifyServiceProfile ()**
   Parameters: invalid serviceProfile
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractIID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
modifyServiceProfile(invalid serviceProfile)
P_INVALID_ SERVICE_PROFILE _ID

Test FW_FO_SS_27

Summary: **IpServiceProfileManagement**, deleteServiceProfile, P_INVALID_SERVICE_PROFILE_ID.

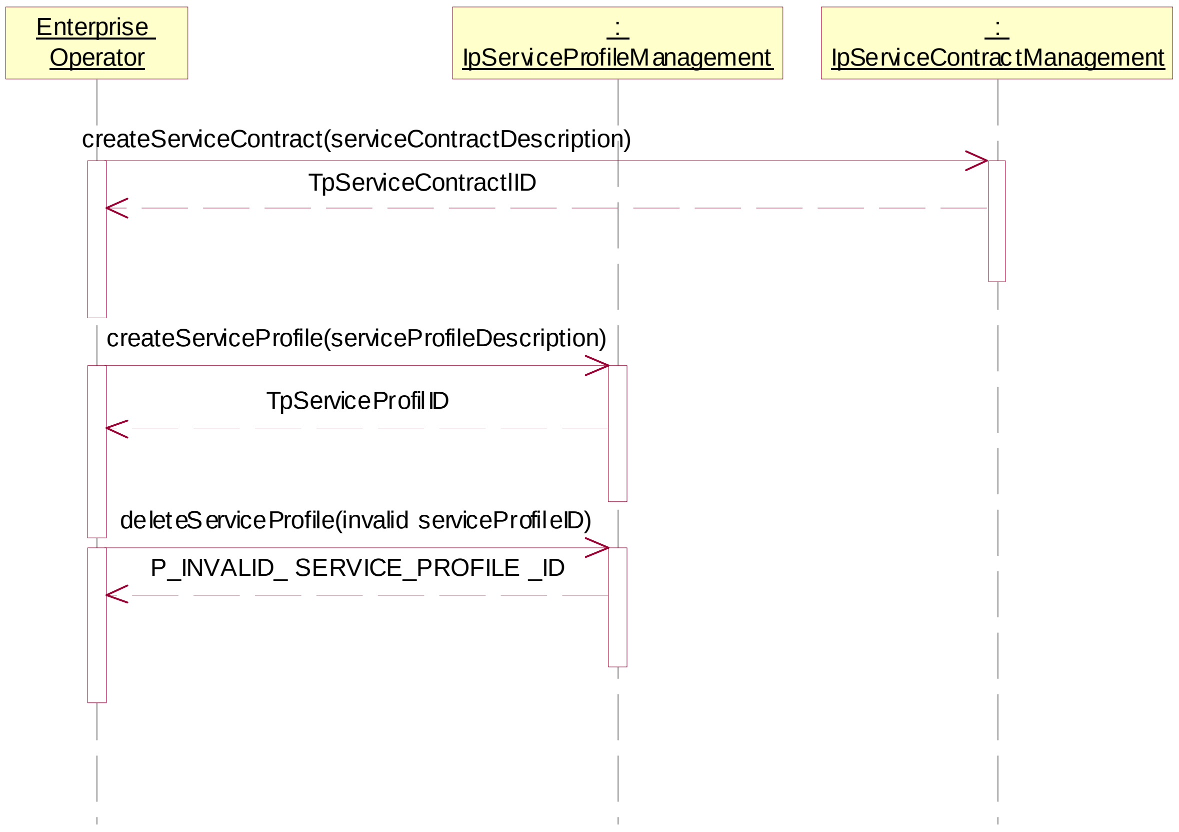
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractIID is returned

2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **deleteServiceProfile ()**
   Parameters: invalid serviceProfile
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractIID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
deleteServiceProfile(invalid serviceProfileID)
P_INVALID_ SERVICE_PROFILE _ID

Test FW_FO_SS_28

Summary: **IpServiceProfileManagement**, assign, P_INVALID_SAG_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

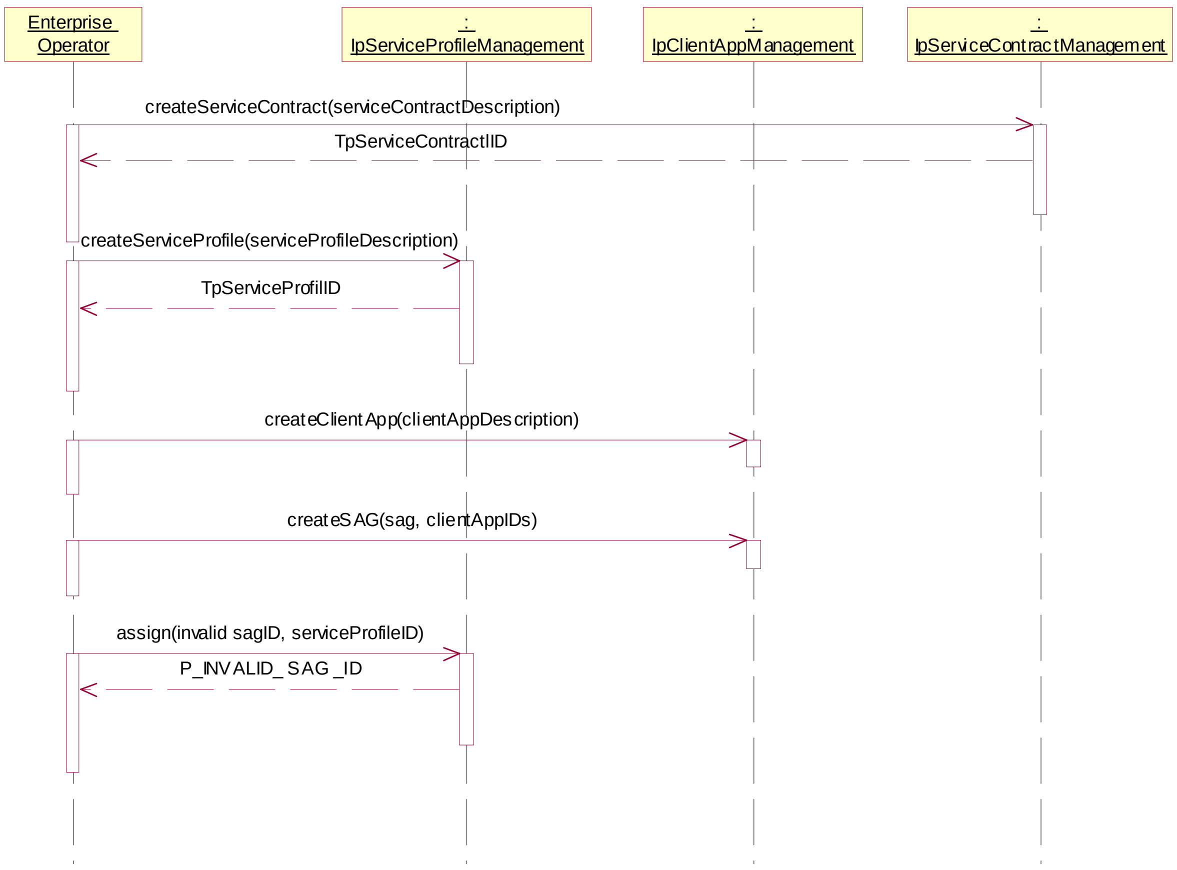
1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractIID is returned

2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

4. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

5. Method call **assign ()**
   Parameters: invalid sagID, serviceProfileID given in 1.
   Check: P_INVALID_ SAG _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractlID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(invalid sagID, serviceProfileID)
P_INVALID_SAG_ID

Test FW_FO_SS_29

Summary: **IpServiceProfileManagement**, assign, P_INVALID_SERVICE_PROFILE_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

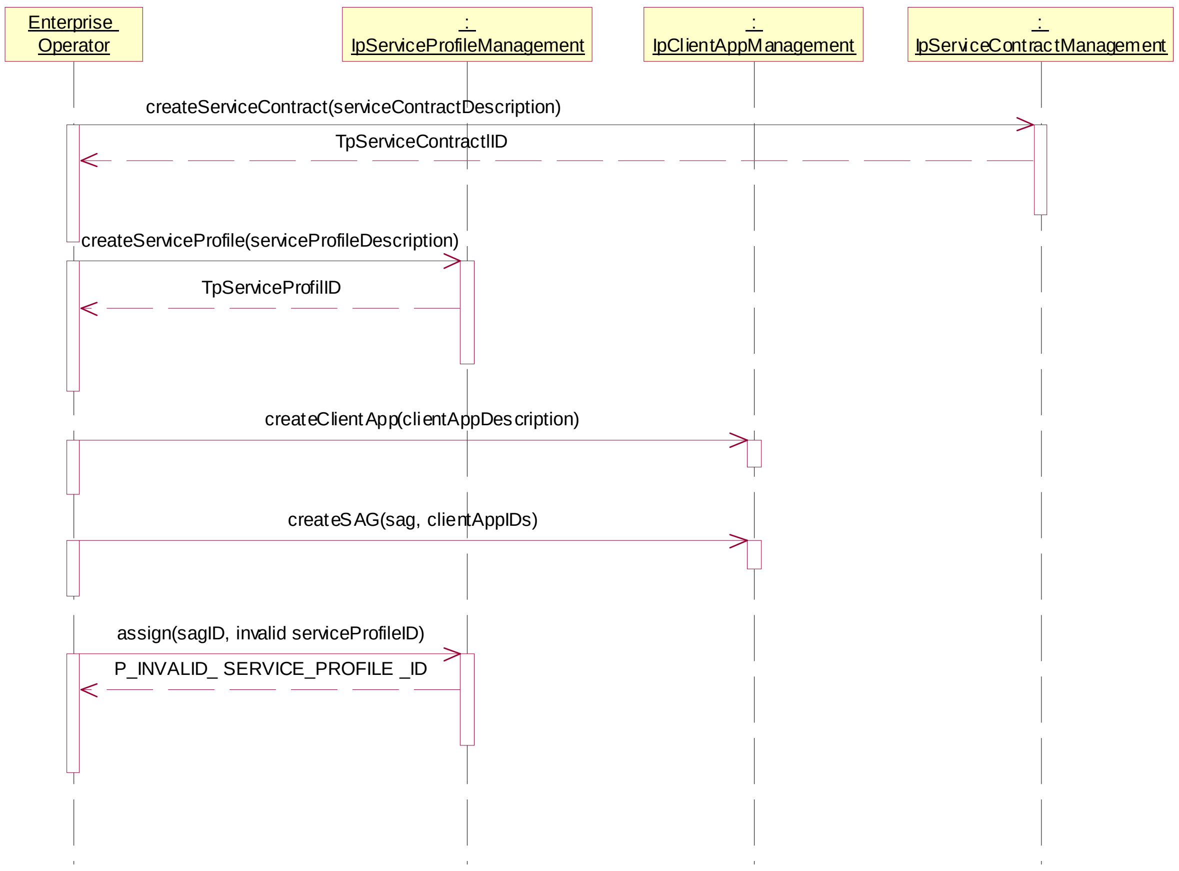
1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

4. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

5. Method call **assign ()**
   Parameters: sagID given in 3., invalid serviceProfileID
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractlID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(sagID, invalid serviceProfileID)
P_INVALID_ SERVICE_PROFILE _ID

Test FW_FO_SS_30

Summary: **IpServiceProfileManagement**, deassign, P_INVALID_SAG_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

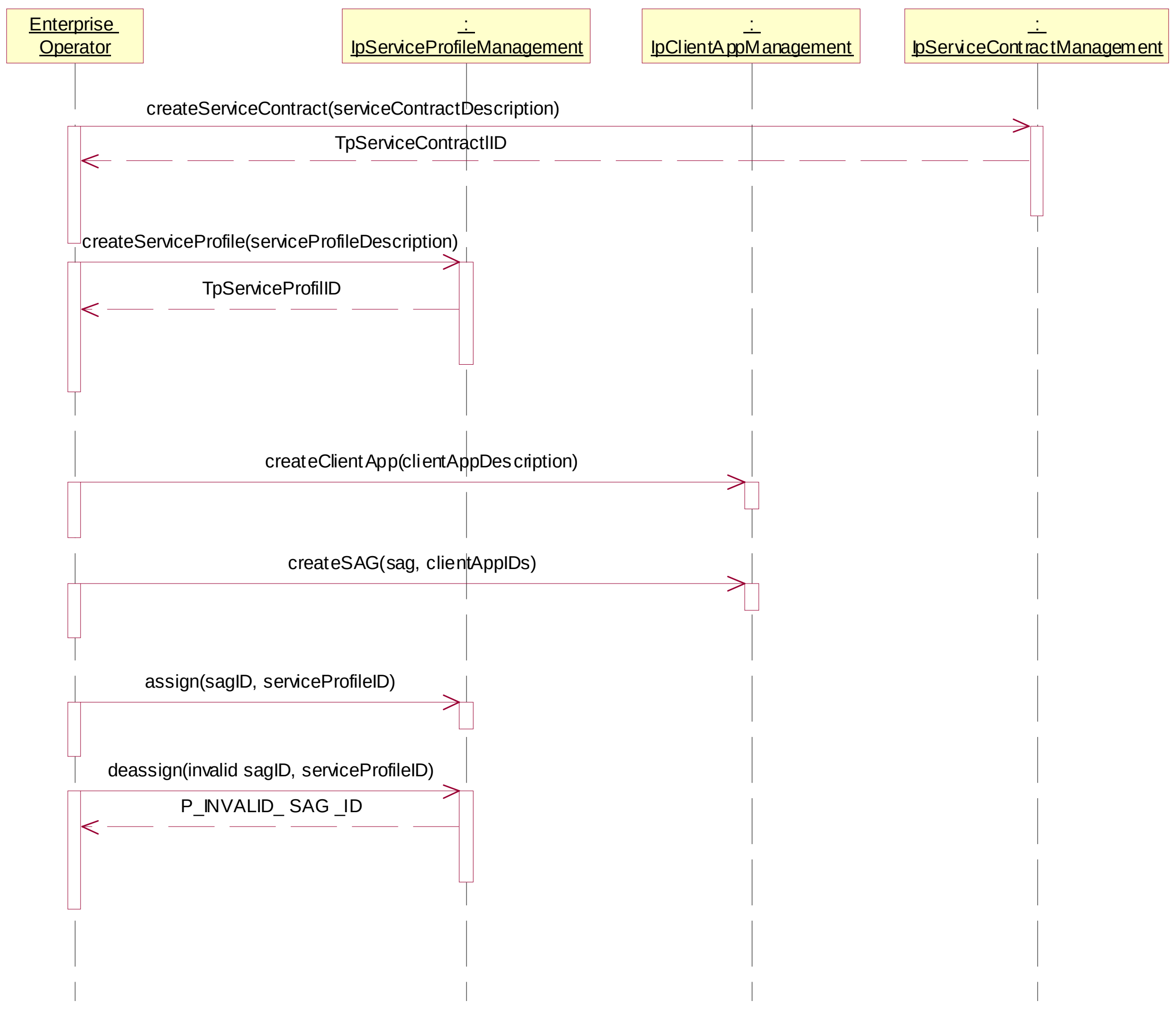
2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

4. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

5. Method call **assign()**
   Parameters: sagID given in 3., serviceProfileID given in 1.
   Check: no exception is returned

6. Method call **deassign ()**
   Parameters: invalid sagID, serviceProfileID
   Check: P_INVALID_ SAG _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractlID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(sagID, serviceProfileID)
deassign(invalid sagID, serviceProfileID)
P_INVALID_ SAG _ID

Test FW_FO_SS_31

Summary: **IpServiceProfileManagement**, deassign, P_INVALID_SERVICE_PROFILE_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

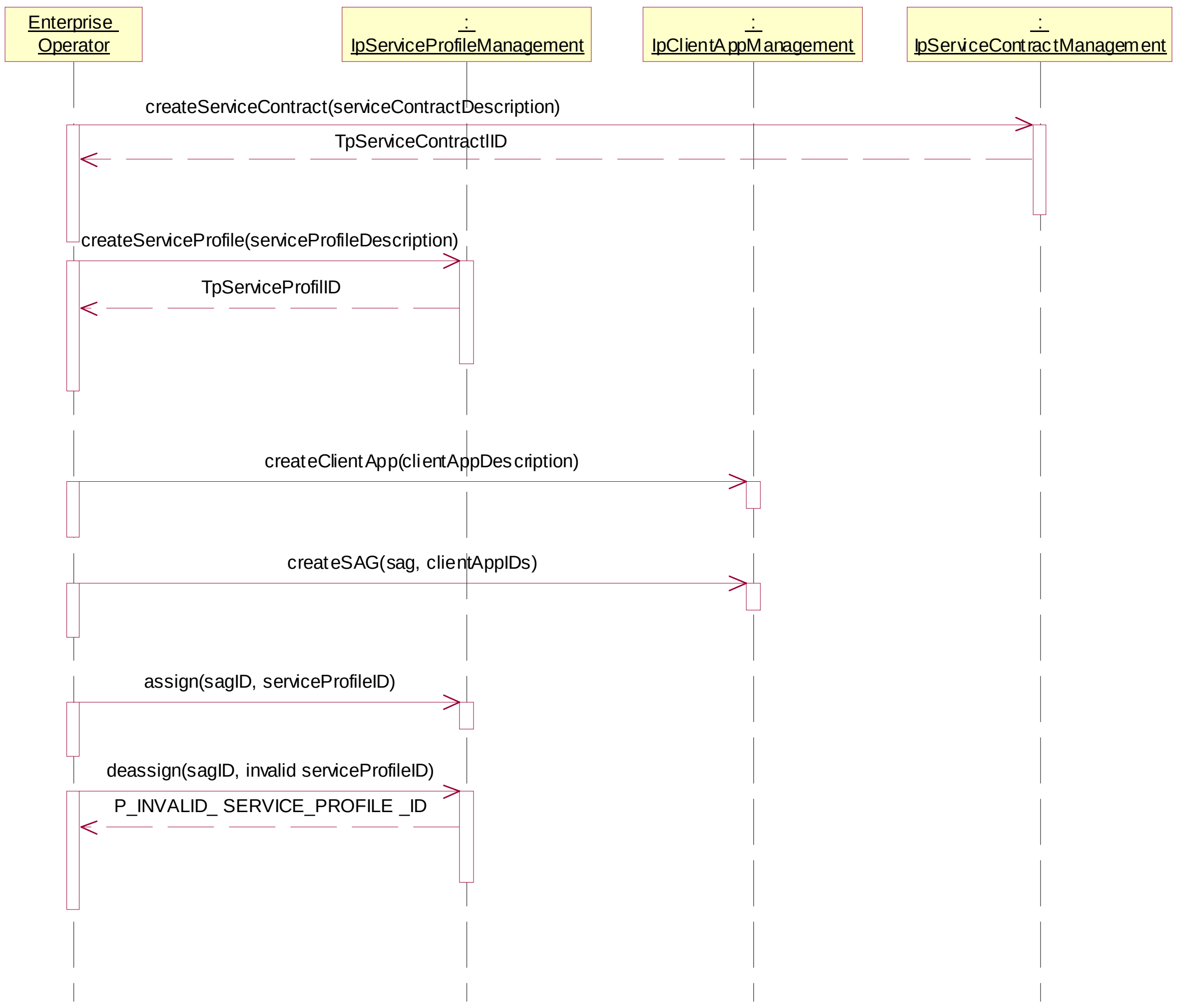
2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

4. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

5. Method call **assign()**
   Parameters: sagID given in 3., serviceProfileID given in 1.
   Check: no exception is returned

6. Method call **deassign ()**
   Parameters: sagID, invalid serviceProfileID
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractlID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(sagID, serviceProfileID)
deassign(sagID, invalid serviceProfileID)
P_INVALID_ SERVICE_PROFILE _ID

Test FW_FO_SS_42

Summary: **IpServiceProfileManagement**, requestConflictInfo, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractlID is returned

2. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **createServiceProfile()**
   Parameters: serviceProfileDescription different than previous one
   Check: valid TpServiceProfilID is returned

4. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

5. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription with parameters incompatible with other Client App in same SAG.
   Check: no exception is returned

6. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

7. Method call **assign()**
   Parameters: sagID given in 4., serviceProfileID given in 2.
   Check: no exception is returned

8. Method call **assign()**
   Parameters: sagID given in 5., serviceProfileID given in 3.
   Check: P_INVALID_SAG_TO_SERVICE_PROFILE_ASSIGNMENT is returned

9. Method call **requestConflictInfo ()**
   Parameters: None
   Check: valid TpAddSagMembersConflictList is returned

Test FW_FO_SS_32

Summary: **IpServiceContractInfoQuery**, all methods, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

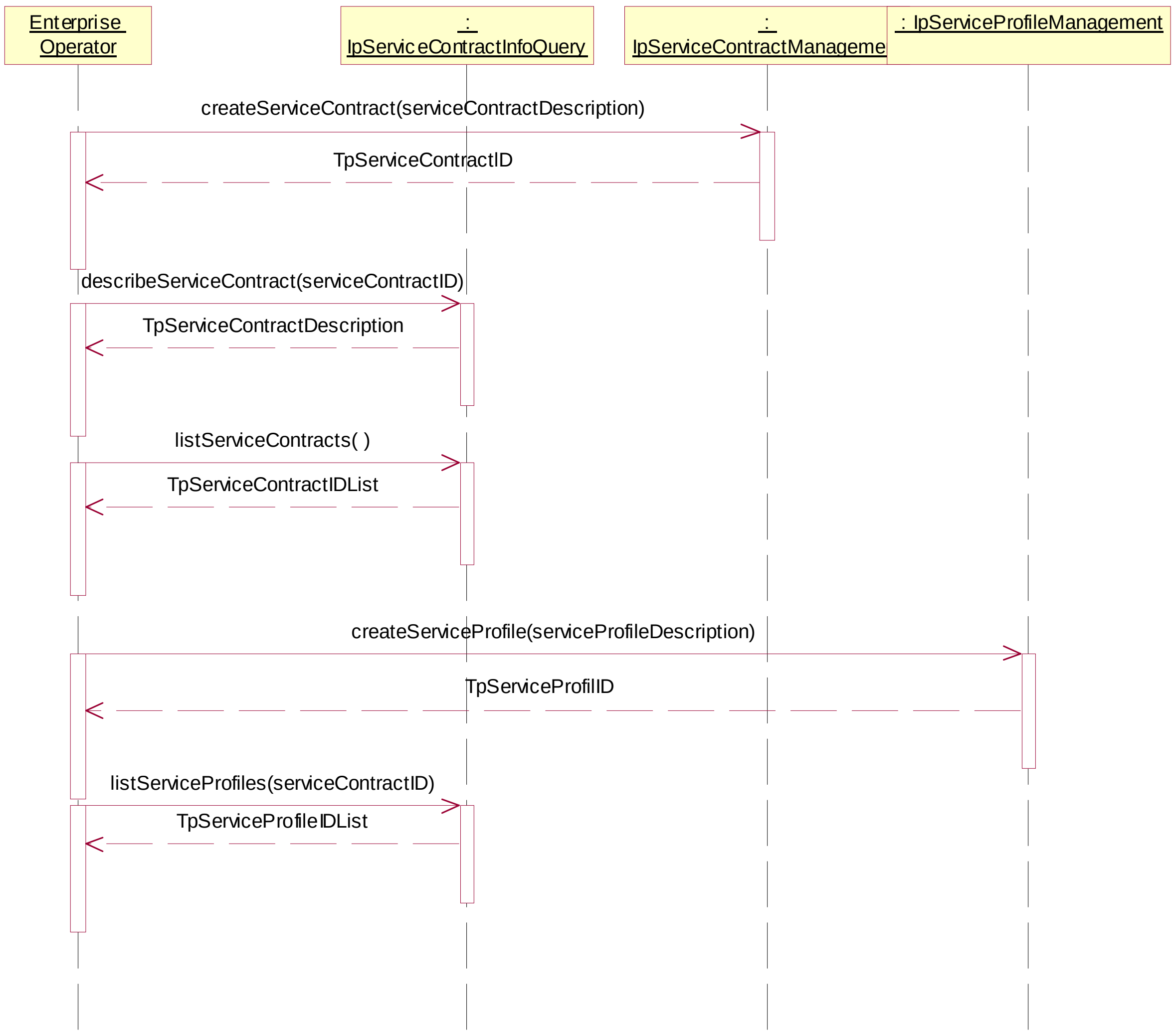
1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractID is returned

2. Method call **describeServiceContract()**
   Parameters: serviceContractID given in 1.
   Check: valid TpServiceContractDescription is returned

3. Method call **listServiceContracts()**
   Parameters: None
   Check: valid TpServiceContractIDList is returned

4. Method call **IpServiceProfileManagement.createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

5. Method call **listServiceProfiles()**
   Parameters: serviceContractID given in 1.
   Check: valid TpServiceProfileIDList is returned


Enterprise Operator
: IpServiceContractInfoQuery
: IpServiceContractManageme
: IpServiceProfileManagement
createServiceContract(serviceContractDescription)
TpServiceContractID
describeServiceContract(serviceContractID)
TpServiceContractDescription
listServiceContracts( )
TpServiceContractIDList
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
listServiceProfiles(serviceContractID)
TpServiceProfileIDList

Test FW_FO_SS_33

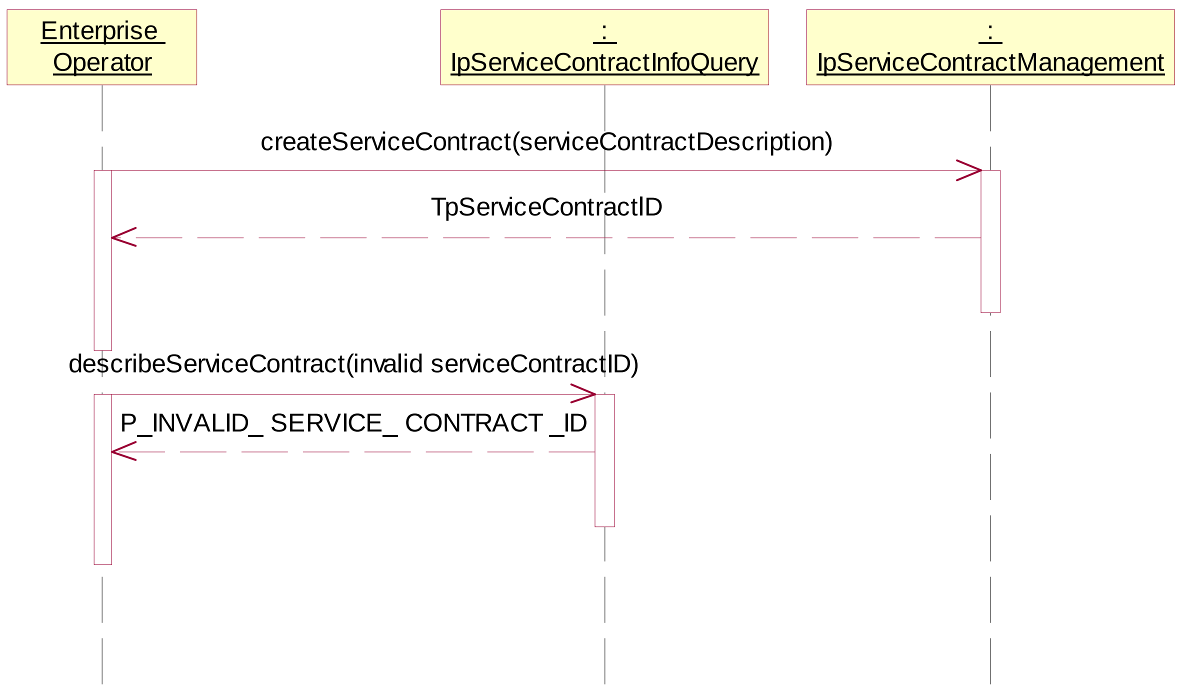
Summary: **IpServiceContractInfoQuery**, describeServiceContract, P_INVALID_SERVICE_CONTRACT_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractID is returned

2. Method call **describeServiceContract()**
   Parameters: invalid serviceContractID
   Check: P_INVALID_ SERVICE_ CONTRACT _ID is returned.


Enterprise Operator
: IpServiceContractInfoQuery
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractID
describeServiceContract(invalid serviceContractID)
P_INVALID_ SERVICE_ CONTRACT _ID

Test FW_FO_SS_34

Summary: **IpServiceContractInfoQuery**, listServiceProfiles, P_INVALID_SERVICE_CONTRACT_ID.

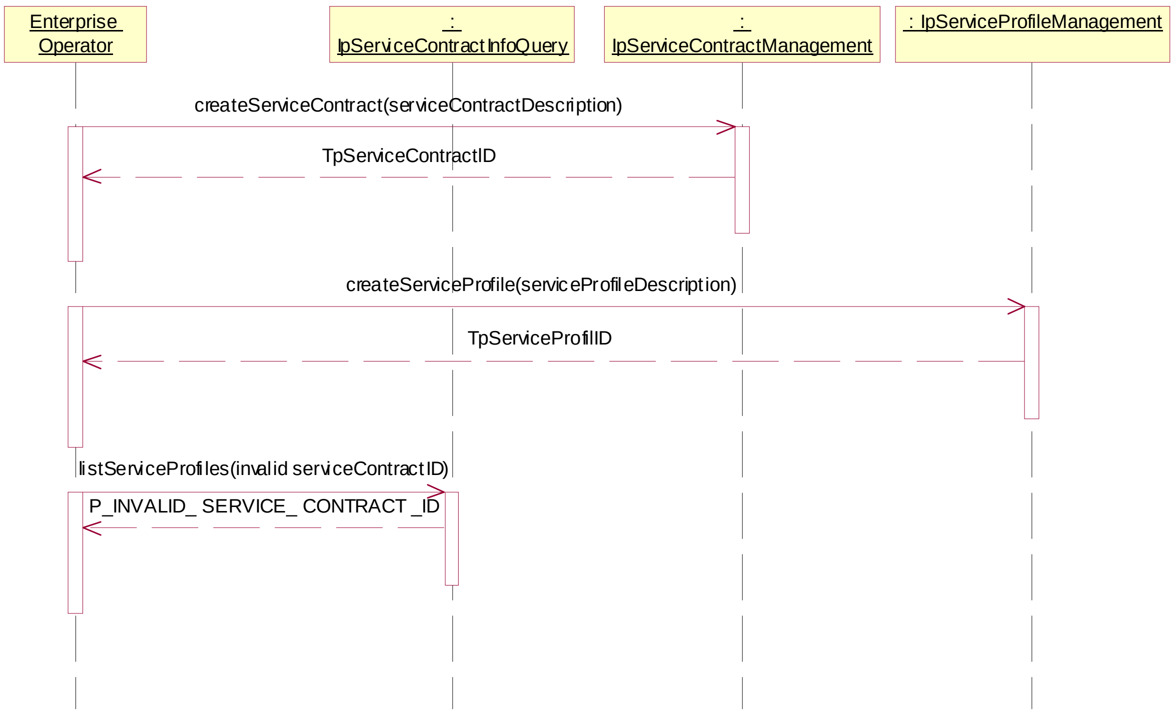
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractID is returned

2. Method call **IpServiceProfileManagement.createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: valid TpServiceProfilID is returned

3. Method call **listServiceProfiles ()**
   Parameters: invalid serviceContractID
   Check: P_INVALID_ SERVICE_ CONTRACT _ID is returned.


Enterprise Operator
: IpServiceContractInfoQuery
: IpServiceContractManagement
: IpServiceProfileManagement
createServiceContract(serviceContractDescription)
TpServiceContractID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
listServiceProfiles(invalid serviceContractID)
P_INVALID_ SERVICE_ CONTRACT _ID

Test FW_FO_SS_35

Summary: **IpServiceProfileInfoQuery**, all methods, successful.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractID is returned

2. Method call **IpServiceProfileManagement.createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: validTpServiceProfilID is returned

3. Method call **listServiceProfiles()**
   Parameters: None
   Check: valid TpServiceProfilIDList is returned including serviceProfileDescription given in 1.

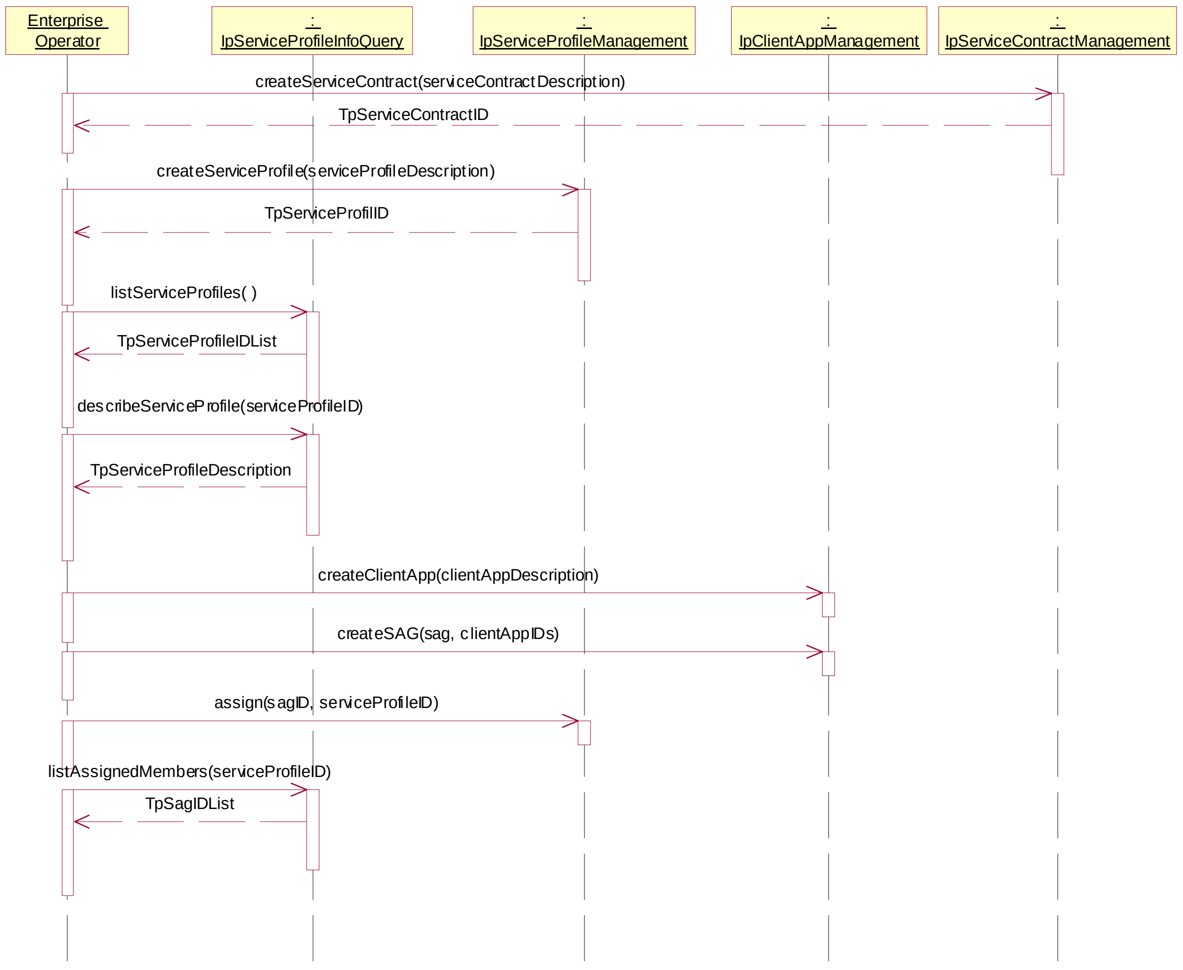
4. Method call **describeServiceProfile()**
   Parameters: serviceProfileID given in 1.
   Check: valid TpServiceProfilDescription is returned.

5. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

6. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

7. Method call **IpServiceProfileManagement.assign()**
   Parameters: sagID given in 6., serviceProfileID given in 1.
   Check: no exception is returned

8. Method call **listAssignedMembers()**
   Parameters: serviceProfileID given in 1.
   Check: valid TpSagIDList is returned including sagID given in 6


Enterprise Operator
: IpServiceProfileInfoQuery
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
listServiceProfiles( )
TpServiceProfileIDList
describeServiceProfile(serviceProfileID)
TpServiceProfileDescription
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(sagID, serviceProfileID)
listAssignedMembers(serviceProfileID)
TpSagIDList

Test FW_FO_SS_36

Summary: **IpServiceProfileInfoQuery**, describeServiceProfile, P_INVALID_SERVICE_PROFILE_ID.

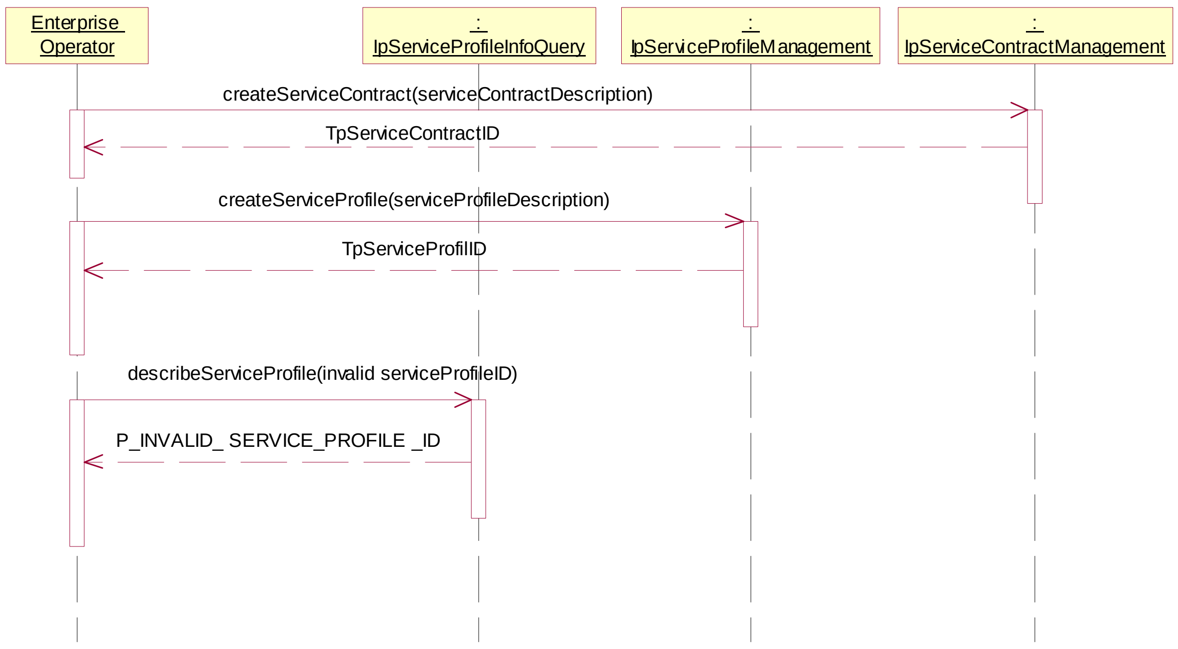
Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractID is returned

2. Method call **IpServiceProfileManagement.createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: validTpServiceProfilID is returned

3. Method call **describeServiceProfile ()**
   Parameters: invalid serviceProfileID
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileInfoQuery
: IpServiceProfileManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
describeServiceProfile(invalid serviceProfileID)
P_INVALID_ SERVICE_PROFILE _ID

Test FW_FO_SS_37

Summary: **IpServiceProfileInfoQuery**, listAssignedMembers, P_INVALID_SERVICE_PROFILE_ID.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **IpServiceContractManagement.createServiceContract()**
   Parameters: serviceContractDescription
   Check: valid TpServiceContractID is returned

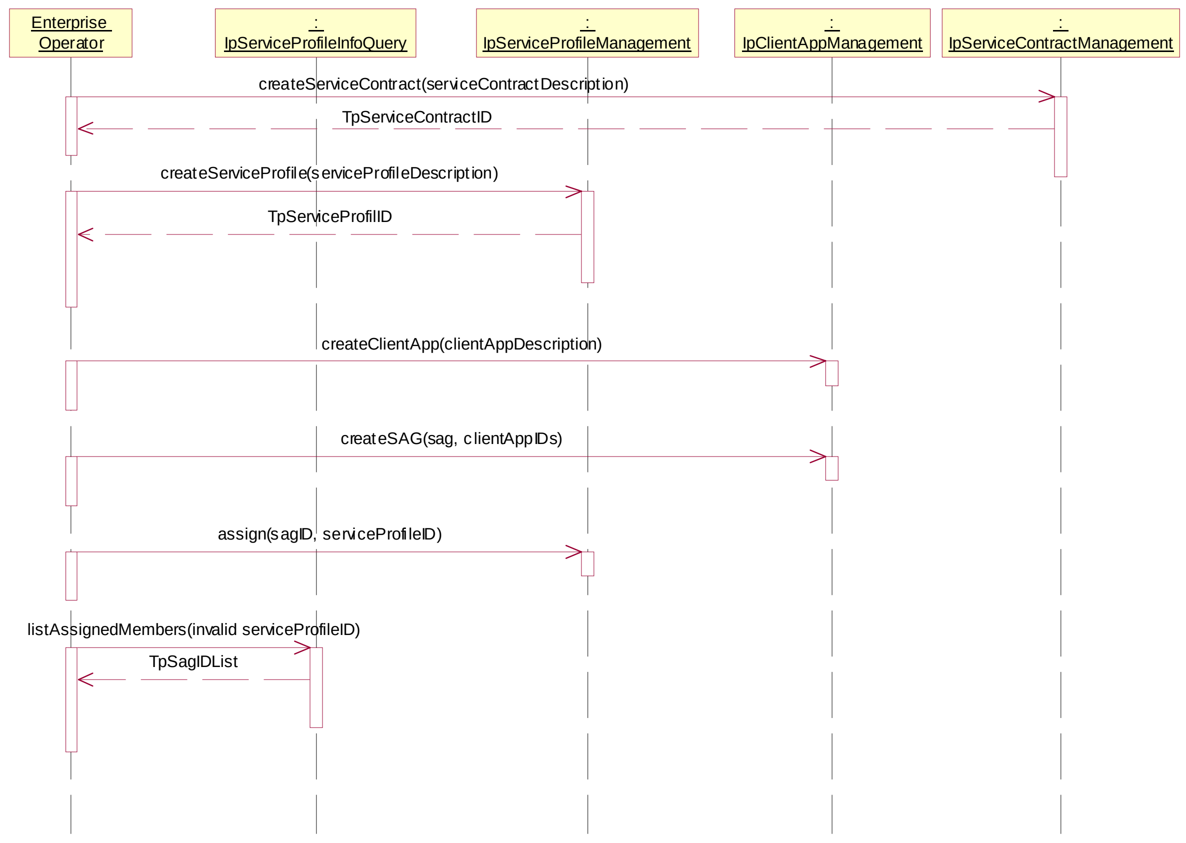
2. Method call **IpServiceProfileManagement.createServiceProfile()**
   Parameters: serviceProfileDescription
   Check: validTpServiceProfilID is returned

3. Method call **IpClientAppManagement.createClientApp()**
   Parameters: clientAppDescription
   Check: no exception is returned

4. Method call **IpClientAppManagement.createSAG()**
   Parameters: sag, clientAppIDs
   Check: no exception is returned

5. Method call **IpServiceProfileManagement.assign()**
   Parameters: sagID given in 4., serviceProfileID given in 1.
   Check: no exception is returned

6. Method call **listAssignedMembers ()**
   Parameters: invalid serviceProfileID
   Check: P_INVALID_ SERVICE_PROFILE _ID is returned.


Enterprise Operator
: IpServiceProfileInfoQuery
: IpServiceProfileManagement
: IpClientAppManagement
: IpServiceContractManagement
createServiceContract(serviceContractDescription)
TpServiceContractID
createServiceProfile(serviceProfileDescription)
TpServiceProfilID
createClientApp(clientAppDescription)
createSAG(sag, clientAppIDs)
assign(sagID, serviceProfileID)
listAssignedMembers(invalid serviceProfileID)
TpSagIDList

Test FW_FO_SS_38

Summary: **IpEntOpAccountManagement**, modifyEntOpAccount and deleteEntOpAccount, successful.

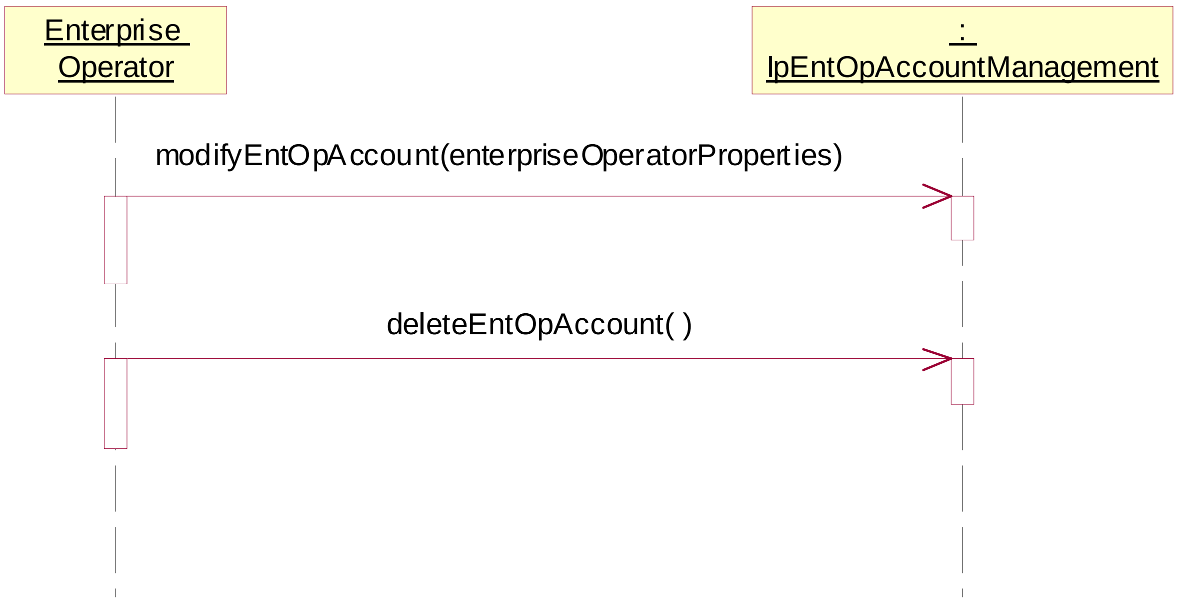
Preamble: The calling application must have a valid reference of an EntOpAccount interface.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **modifyEntOpAccount ()**
   Parameters: enterpriseOperatorProperties
   Check: no exception is returned

2. Method call **deleteEntOpAccount ()**
   Parameters: None
   Check: no exception is returned


Enterprise Operator
: IpEntOpAccountManagement
modifyEntOpAccount(enterpriseOperatorProperties)
deleteEntOpAccount( )

Test FW_FO_SS_39

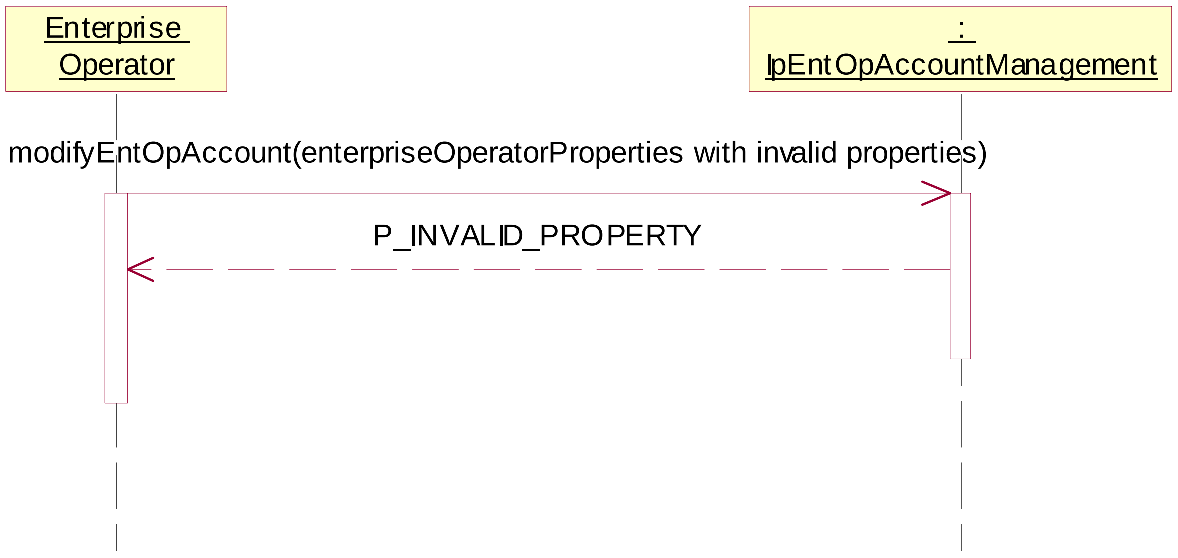
Summary: **IpEntOpAccountManagement**, modifyEntOpAccount, P_INVALID_PROPERTY.

Preamble: The calling application must have a valid reference of an EntOpAccount interface.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **modifyEntOpAccount ()**
   Parameters: enterpriseOperatorProperties with invalid properties
   Check: P_INVALID_PROPERTY is returned.


Enterprise Operator
: IpEntOpAccountManagement
modifyEntOpAccount(enterpriseOperatorProperties with invalid properties)
P_INVALID_PROPERTY

Test FW_FO_SS_40

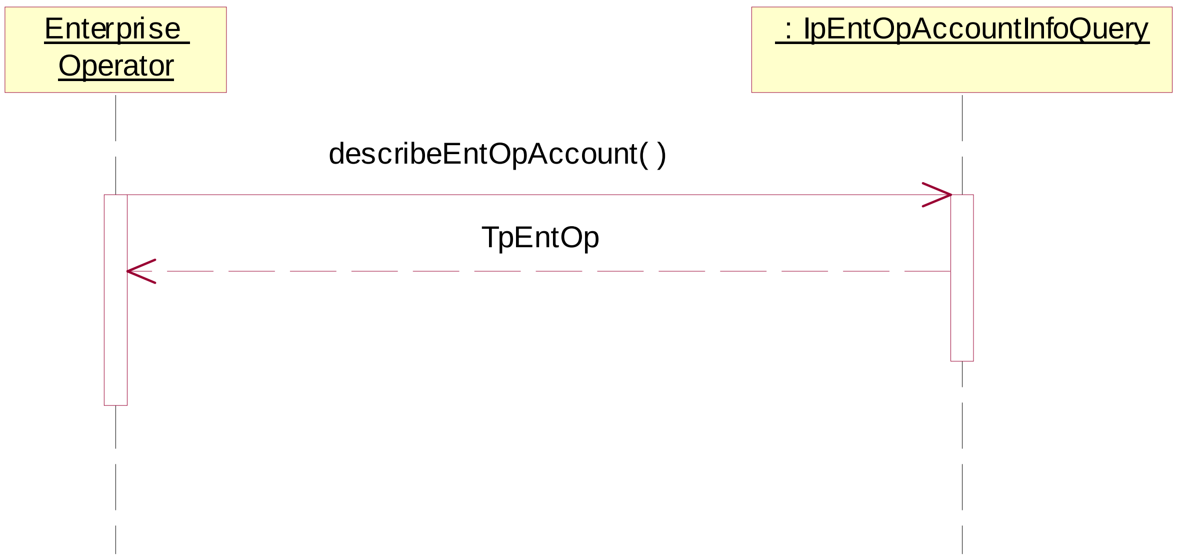
Summary: **IpEntOpAccountInfoQuery**, describeEntOpAccount, successful.

Preamble: The calling application must have a valid reference of an EntOpAccount interface.

Reference: ES 202 915-3 [1], clause 8.3.1.

Test Sequence:

1. Method call **describeEntOpAccount ()**
   Parameters: None
   Check: valid TpEnteOp is returned



5.4.4 Framework to Service API

5.4.4.1 Service Registration (SR)

Test FW_FS_SR_01

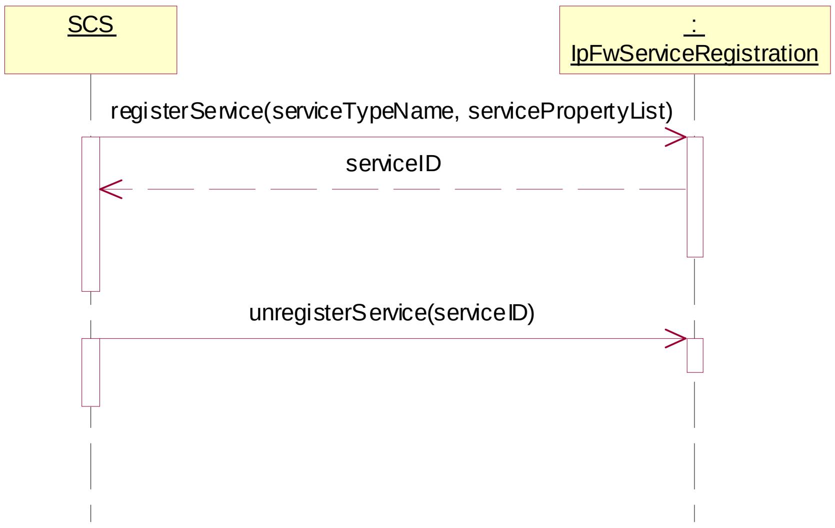
Summary: **IpFwServiceRegistration**, registerService and unregisterService methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList
   Check: valid value of TpServiceID is returned

2. Method call **unregisterService()**
   Parameters: serviceID given in 1.
   Check: no exception is returned


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList)
serviceID
unregisterService(serviceID)

Test FW_FS_SR_02

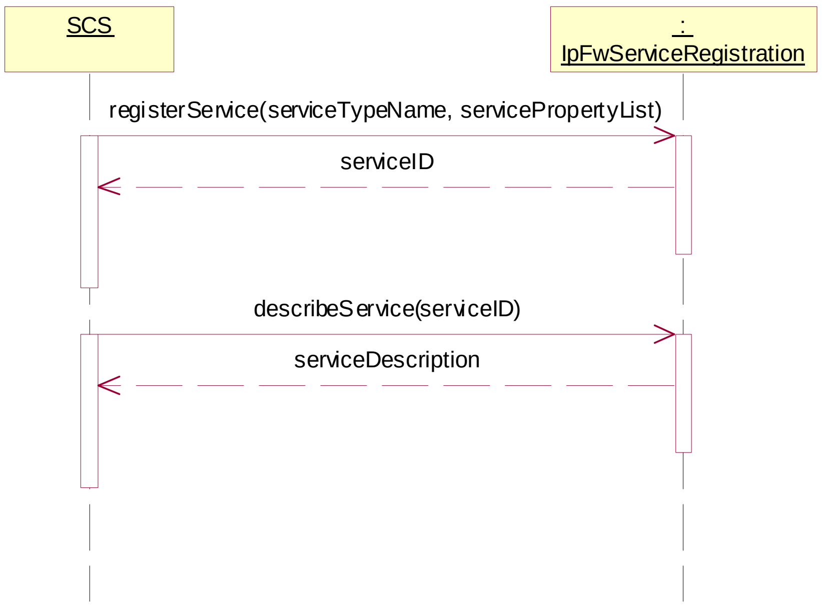
Summary: **IpFwServiceRegistration**, describeService method, successful.

Reference: ES 202 915-3 [1], clause 9.3.1.

Preamble: The service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **describeService()**
   Parameters: serviceID as returned by the registerService method
   Check: valid value of TpServiceDescription is returned


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList)
serviceID
describeService(serviceID)
serviceDescription

Test FW_FS_SR_03

Summary: **IpFwServiceRegistration**, announceServiceAvailability and unannounceService methods, successful.

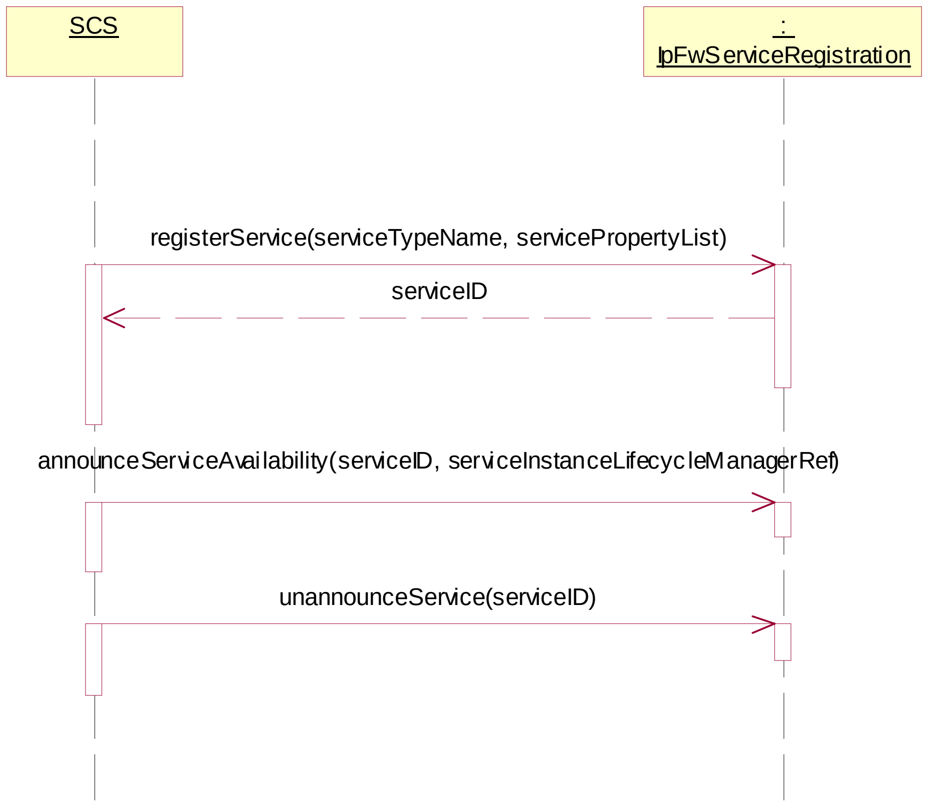
Reference: ES 202 915-3 [1], clause 9.3.1.

Preamble: The service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **announceServiceAvailability()**
   Parameters: serviceID as returned by the registerService method,
   serviceInstanceLifeCycleManagerRef
   Check: no exception is returned.

2. Method call **unannounceService()**
   Parameters: serviceID as returned by the registerService method,
   Check: no exception is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList)
serviceID
announceServiceAvailability(serviceID, serviceInstanceLifecycleManagerRef)
unannounceService(serviceID)

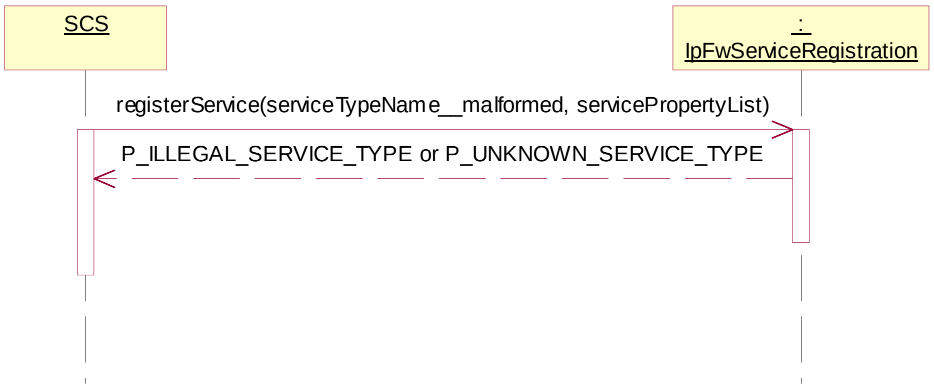
Test FW_FS_SR_04

Summary: **IpFwServiceRegistration**, registerService methods, P_ILLEGAL_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName malformed, servicePropertyList
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName__malformed, servicePropertyList)
P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE

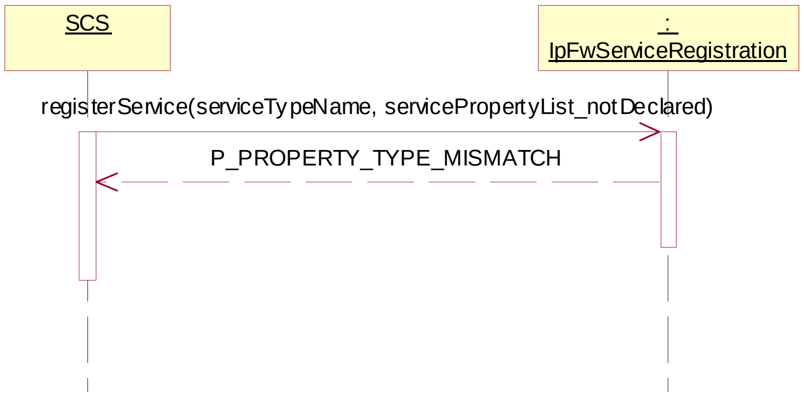
Test FW_FS_SR_05

Summary: **IpFwServiceRegistration**, registerService methods, P_PROPERTY_TYPE_MISMATCH.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList with a type of a property values not the same as the declared service type.
   Check: P_PROPERTY_TYPE_MISMATCH is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList_notDeclared)
P_PROPERTY_TYPE_MISMATCH

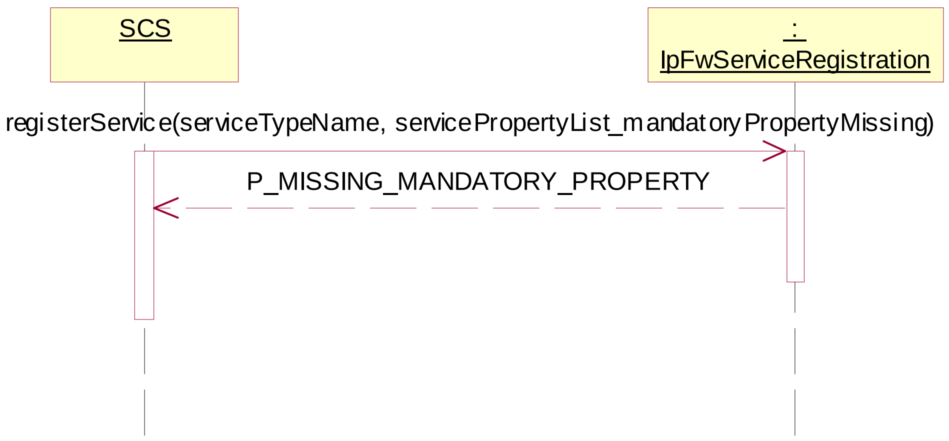
Test FW_FS_SR_06

Summary: **IpFwServiceRegistration**, registerService methods, P_MISSING_MANDATORY_PROPERTY.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList with a mandatory property missing.
   Check: P_MISSING_MANDATORY_PROPERTY is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList_mandatoryPropertyMissing)
P_MISSING_MANDATORY_PROPERTY

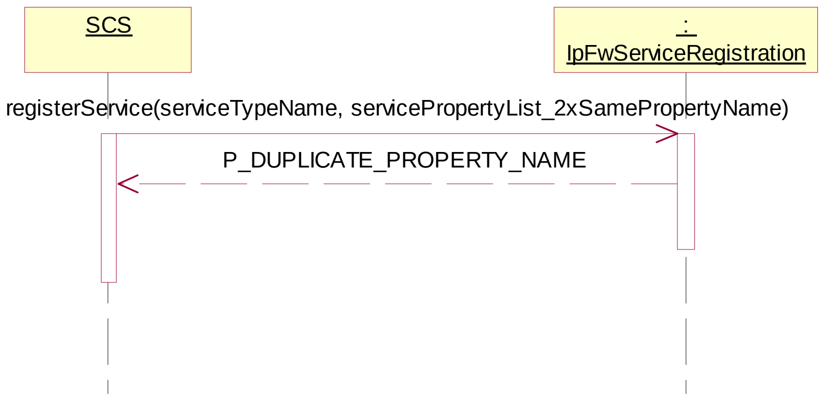
Test FW_FS_SR_07

Summary: **IpFwServiceRegistration**, registerService methods, P_DUPLICATE_PROPERTY_NAME.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList including two properties with the same property name.
   Check: P_DUPLICATE_PROPERTY_NAME is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList_2xSamePropertyName)
P_DUPLICATE_PROPERTY_NAME

Test FW_FS_SR_08

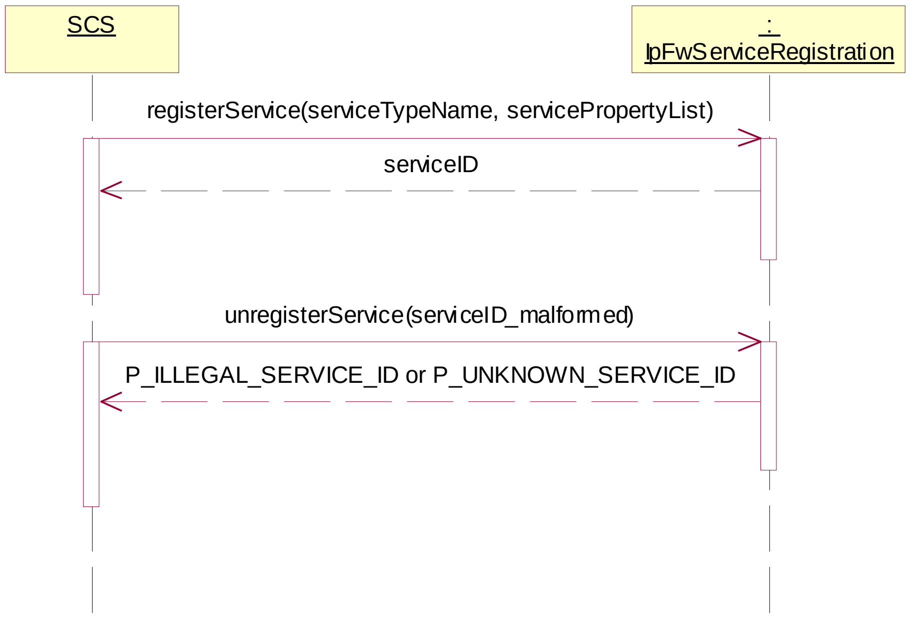
Summary: **IpFwServiceRegistration**, registerService and unregisterService methods, P_ILLEGAL_SERVICE_ID.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList
   Check: valid value of TpServiceID is returned

2. Method call **unregisterService()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList)
serviceID
unregisterService(serviceID_malformed)
P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID

Test FW_FS_SR_09

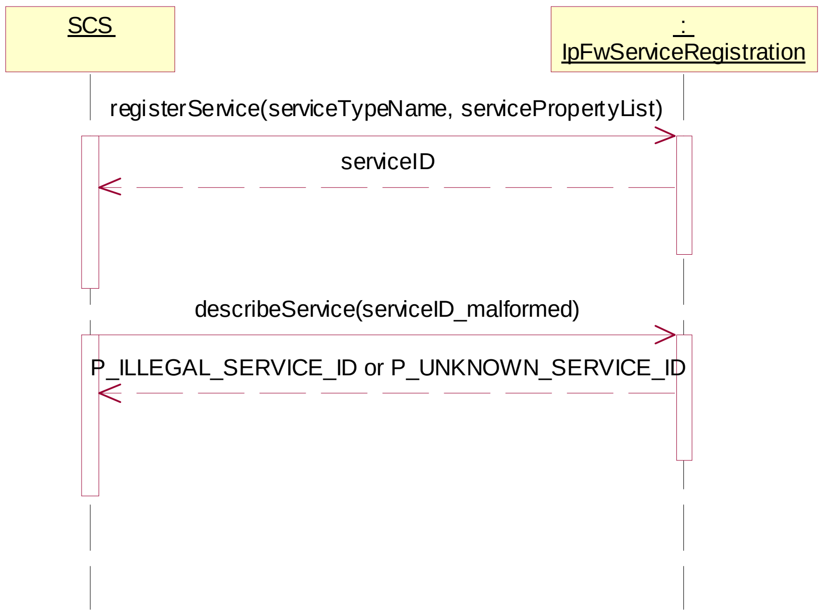
Summary: **IpFwServiceRegistration**, describeService method, P_ILLEGAL_SERVICE_ID.

Reference: ES 202 915-3 [1], clause 9.3.1.

Preamble: The service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **describeService()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList)
serviceID
describeService(serviceID_malformed)
P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID

Test FW_FS_SR_10

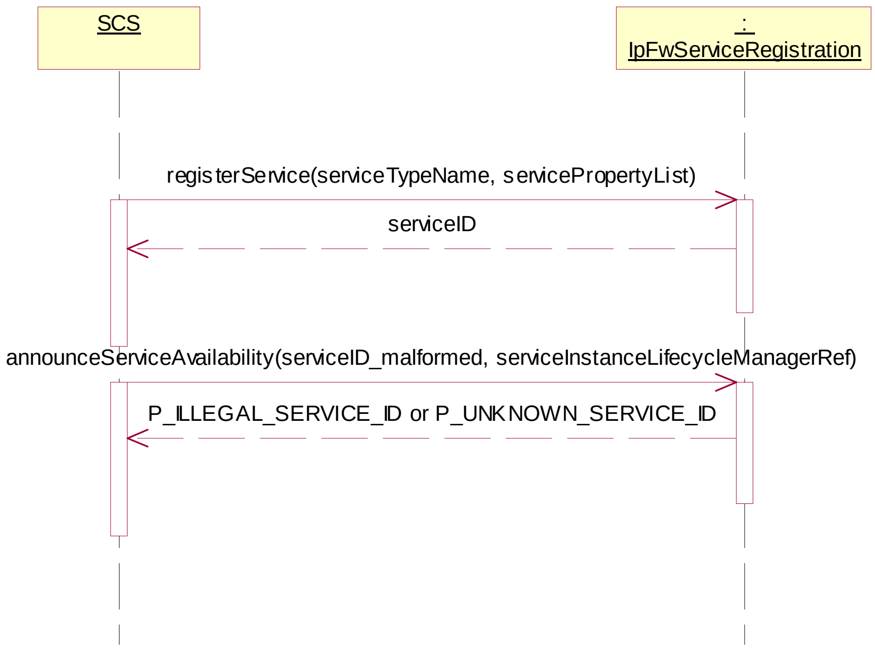
Summary: **IpFwServiceRegistration**, announceServiceAvailability methods, P_ILLEGAL_SERVICE_ID.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Method call **registerService()**
   Parameters: serviceTypeName, servicePropertyList
   Check: valid value of TpServiceID is returned

2. Method call **announceServiceAvailabilty()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID is returned.


SCS
: IpFwServiceRegistration
registerService(serviceTypeName, servicePropertyList)
serviceID
announceServiceAvailability(serviceID_malformed, serviceInstanceLifecycleManagerRef)
P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID

Test FW_FS_SR_11

Summary: **IpFwServiceRegistration**, unannounceService method, P_ILLEGAL_SERVICE_ID.

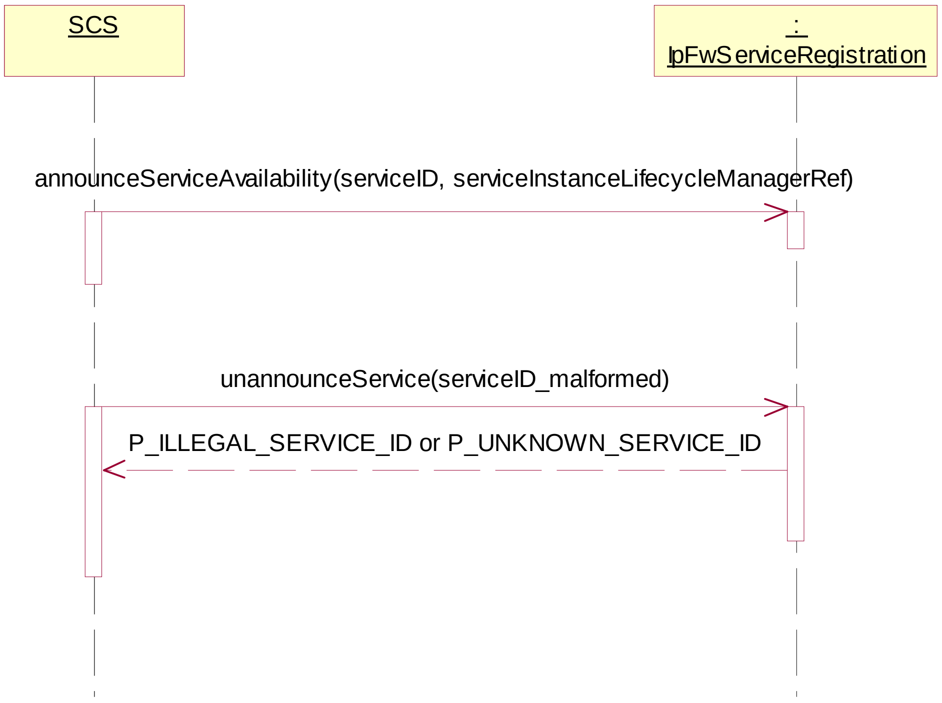
Reference: ES 202 915-3 [1], clause 9.3.1.

Preamble: The service has been previously registered (with the registerService method).

Test Sequence:

1. Method call **announceServiceAvailability()**
   Parameters: serviceID as returned by the registerService method, serviceInstanceLifeCycleManagerRef
   Check: no exception is returned.

2. Method call **unannounceService()**
   Parameters: serviceID not built according to the rules for service identifiers.
   Check: P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID is returned.


SCS
: IpFwServiceRegistration
announceServiceAvailability(serviceID, serviceInstanceLifecycleManagerRef)
unannounceService(serviceID_malformed)
P_ILLEGAL_SERVICE_ID or P_UNKNOWN_SERVICE_ID

5.4.4.2 Service Instance Lifecycle Management (SILM)

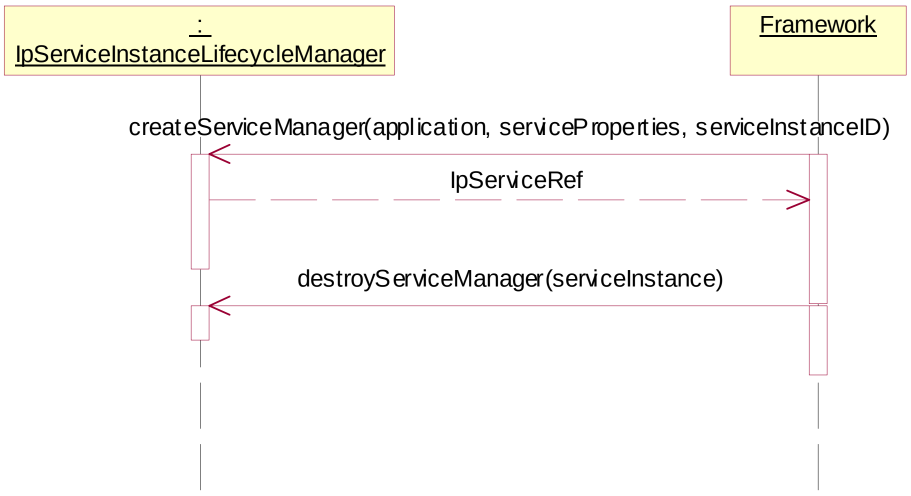
Test FW_FS_SILM_01

Summary:	**IpServiceInstanceLifecycleManager**, createServiceManager and destroyServiceManager methods, successful.
Reference:	ES 202 915-3 [1], clause 9.3.2.

Test Sequence:

1. Triggered action: cause IUT to call **createServiceManager()** method on the tester's (Service) **IpServiceInstanceLifecycleManager** interface.
   Parameters: application, serviceProperties, serviceInstanceID
   Check: valid value of IpServiceRef is returned

2. Triggered action: cause IUT to call **destroyServiceManager()** method on the tester's (Service) **IpServiceInstanceLifecycleManager** interface.
   Parameters: serviceInstanceID (same value as used in 1.).
   Check: no exception is returned


: IpServiceInstanceLifecycleManager
Framework
createServiceManager(application, serviceProperties, serviceInstanceID)
IpServiceRef
destroyServiceManager(serviceInstance)

5.4.4.3 Service Discovery (SD)

Test FW_FS_SD_01

Summary: **IpFwServiceDiscovery,** all methods, successful.

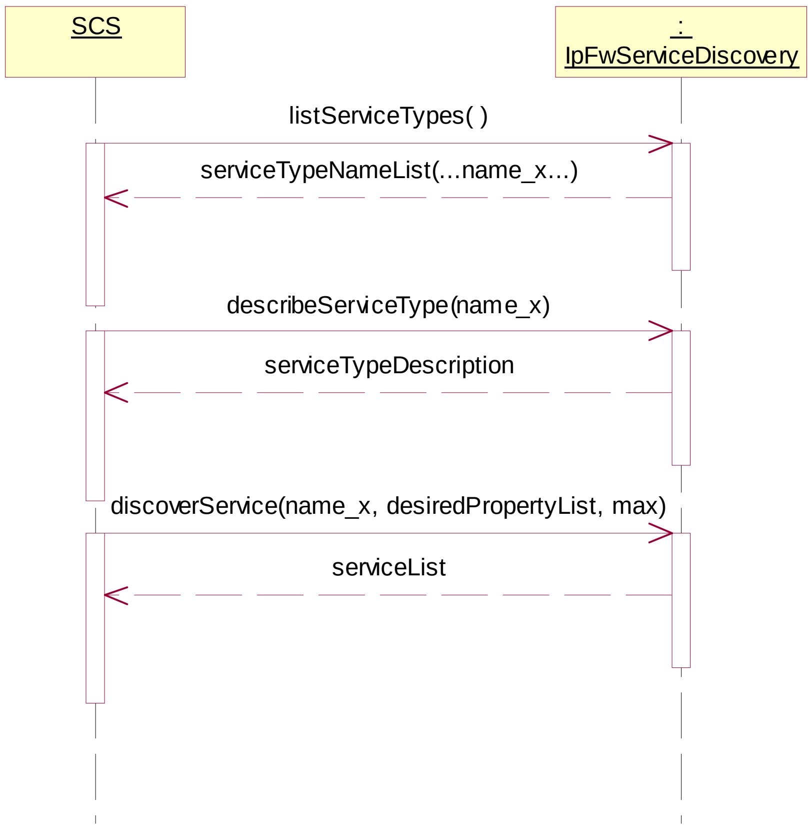
Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName from the list returned in 1.
   Check: valid value of TpServiceTypeDescription is returned

3. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., valid desiredPropertyList, valid max
   Check: valid value of TpServiceList is returned


SCS
: IpFwServiceDiscovery
listServiceTypes( )
serviceTypeNameList(...name_x...)
describeServiceType(name_x)
serviceTypeDescription
discoverService(name_x, desiredPropertyList, max)
serviceList

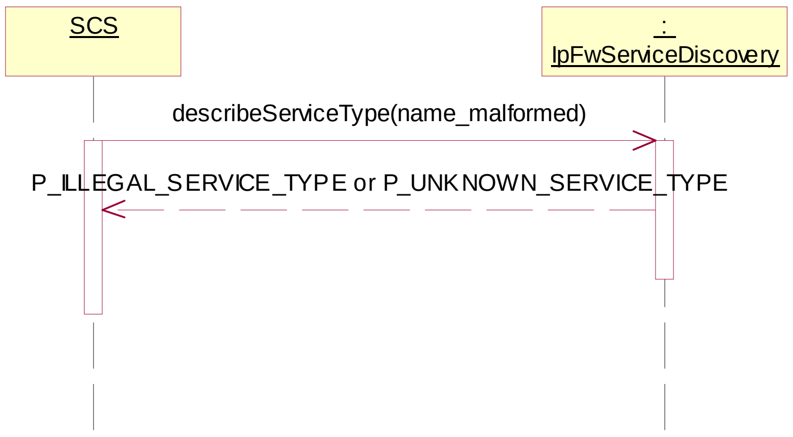
Test FW_FS_SD_02

Summary: **IpFwServiceDiscovery,** describeServiceType, P_ILLEGAL_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Method call **describeServiceType()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE is returned.


SCS
: IpFwServiceDiscovery
describeServiceType(name_malformed)
P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE

Test FW_FS_SD_03

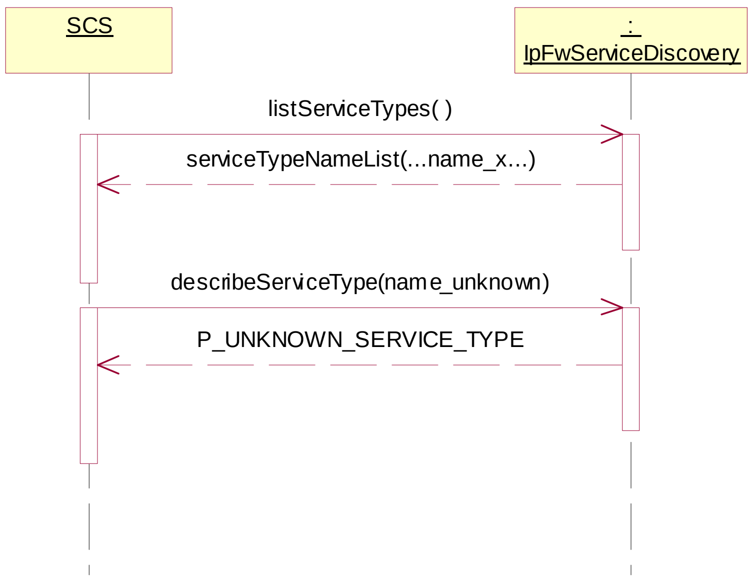
Summary: **IpFwServiceDiscovery,** describeServiceType, P_UNKNOWN_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **describeServiceType()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE is returned.


SCS
: IpFwServiceDiscovery
listServiceTypes( )
serviceTypeNameList(...name_x...)
describeServiceType(name_unknown)
P_UNKNOWN_SERVICE_TYPE

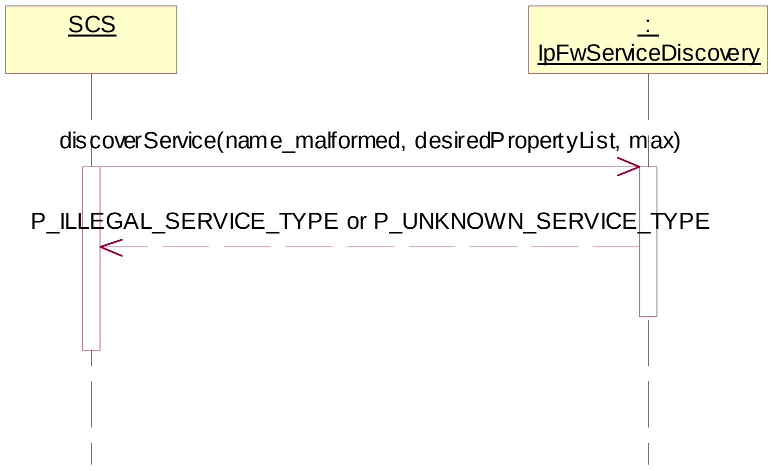
Test FW_FS_SD_04

Summary: **IpFwServiceDiscovery,** discoverService, P_ILLEGAL_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Method call **discoverService()**
   Parameters: serviceTypeName malformed
   Check: P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE is returned.


SCS
: IpFwServiceDiscovery
discoverService(name_malformed, desiredPropertyList, max)
P_ILLEGAL_SERVICE_TYPE or P_UNKNOWN_SERVICE_TYPE

Test FW_FS_SD_05

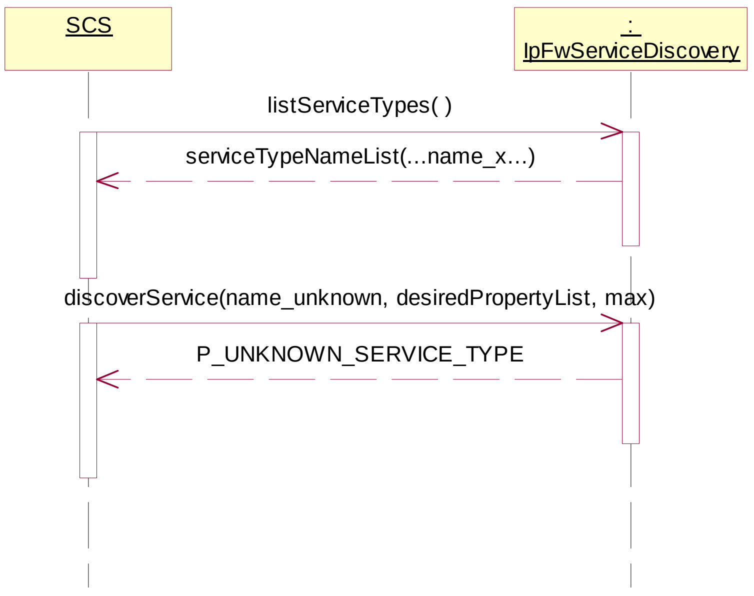
Summary: **IpFwServiceDiscovery,** discoverService, P_UNKNOWN_SERVICE_TYPE.

Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName well formed but not returned in 1.
   Check: P_UNKNOWN_SERVICE_TYPE is returned.


SCS
: IpFwServiceDiscovery
listServiceTypes( )
serviceTypeNameList(...name_x...)
discoverService(name_unknown, desiredPropertyList, max)
P_UNKNOWN_SERVICE_TYPE

Test FW_FS_SD_06

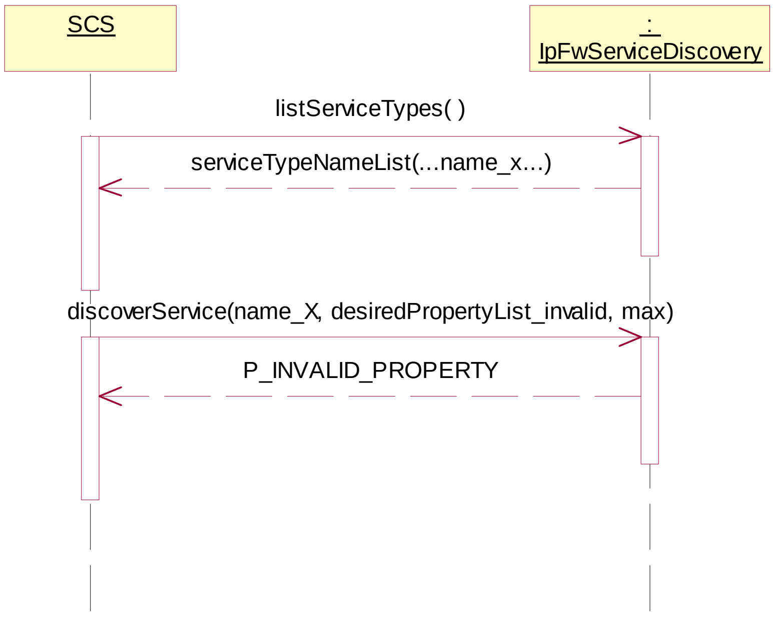
Summary: **IpFwServiceDiscovery,** discoverService, P_INVALID_PROPERTY.

Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Method call **listServicesTypes()**
   Parameters: none
   Check: valid value of TpServiceNameList is returned

2. Method call **discoverService()**
   Parameters: serviceTypeName from the list returned in 1., invalid desiredPropertyList, valid max
   Check: P_INVALID_PROPERTY is returned.


SCS
: IpFwServiceDiscovery
listServiceTypes( )
serviceTypeNameList(...name_x...)
discoverService(name_X, desiredPropertyList_invalid, max)
P_INVALID_PROPERTY

Test FW_FS_SD_07

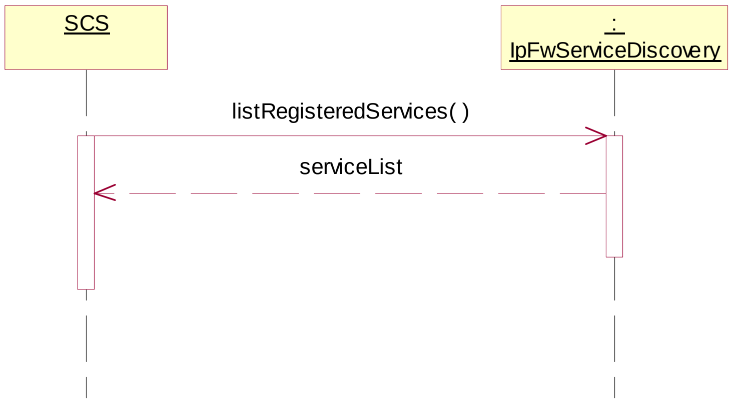
Summary: **IpFwServiceDiscovery,** listRegisteredServices, successful.

Reference: ES 202 915-3 [1], clause 9.3.3.

Precondition: listRegisteredServices supported.

Test Sequence:

1. Method call **listRegisteredServices()**
   Parameters: none
   Check: valid value of TpServiceList is returned



5.4.4.4 Integrity Management (IM)

Test FW_FS_IM_01

Summary: **IpFwHeartBeatMgmt**, all methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwHeartBeatMgt supported.

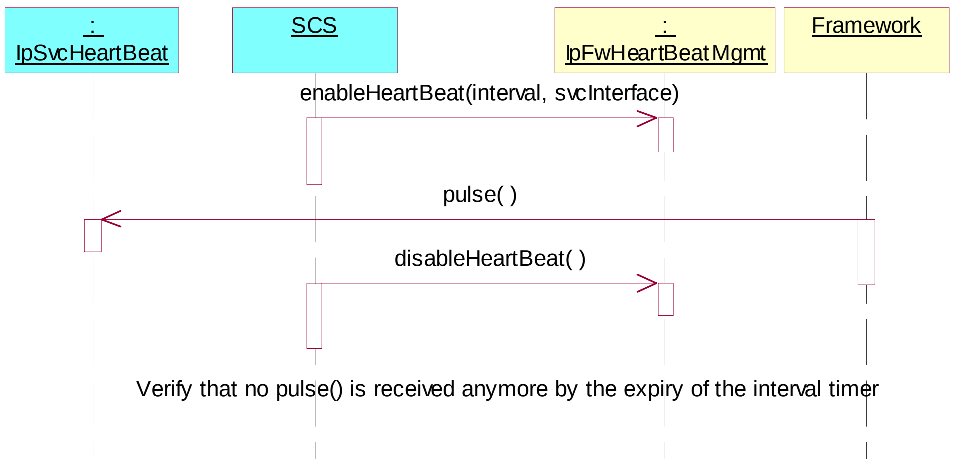
Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

1. Method call **enableHeartBeat()**
   Parameters: interval, svcInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to regularly call **pulse()** method on the tester's (Service) **IpSvcHeartBeat** interface.
   Parameters: none
   Check: no exception is returned. Check also that the pulse() method is invoked at the requested interval.

3. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no **pulse()** is received anymore by the expiry of the interval timer.



: IpSvcHeartBeat
SCS
: IpFwHeartBeatMgmt
Framework
enableHeartBeat(interval, svcInterface)
pulse( )
disableHeartBeat( )
Verify that no pulse() is received anymore by the expiry of the interval timer

Test FW_FS_IM_02

Summary: **IpFwHeartBeatMgmt**, all methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwHeartBeatMgmt, changeInterval supported.

Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

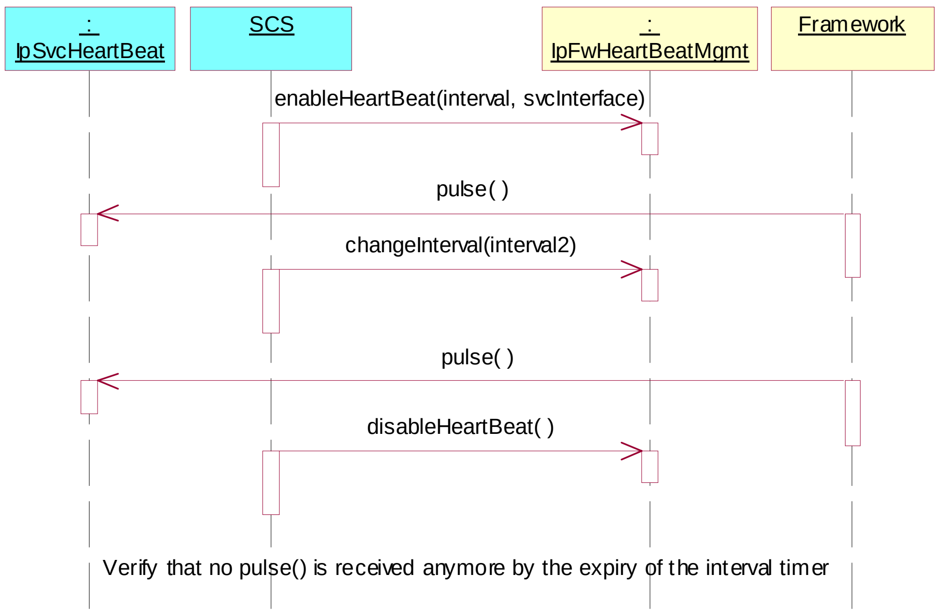
1. Method call **enableHeartBeat()**
   Parameters: interval, svcInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to call **pulse()** method regularly on the tester's (Service) **IpSvcHeartBeat** interface.
   Parameters: none
   Check: check that the pulse() method is invoked at the requested interval.

3. Method call **changeInterval()**
   Parameters: interval
   Check: no exception is returned.

4. Triggered action: cause IUT to call **pulse()** method regularly on the tester's (Service) **IpSvcHeartBeat** interface.
   Parameters: none
   Check: the pulse() method is invoked at the new requested interval.

5. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception Verify that no **pulse()** is received anymore by the expiry of the interval timer.



: IpSvcHeartBeat
SCS
: IpFwHeartBeatMgmt
Framework
enableHeartBeat(interval, svcInterface)
pulse( )
changeInterval(interval2)
pulse( )
disableHeartBeat( )
Verify that no pulse() is received anymore by the expiry of the interval timer

Test FW_FS_IM_03

Summary: **IpFwHeartBeat**, all methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

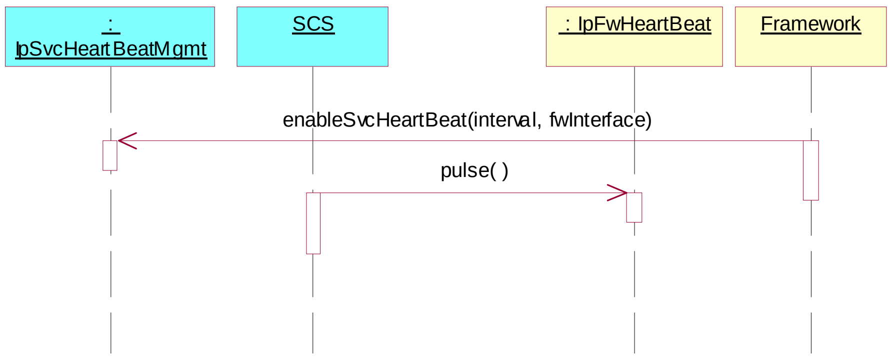
Precondition: IpFwHeartBeat is supported.

Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

1. Triggered action: cause IUT to call **enableSvcHeartBeat ()** method on the tester's (Service) **IpSvcHeartBeatMgmt** interface.
   Parameters: interval, fwInterface

2. Method call **pulse()**
   Parameters: none
   Check: no exception



Test FW_FS_IM_04

Summary: **IpFwFaultManager,** activityTestReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

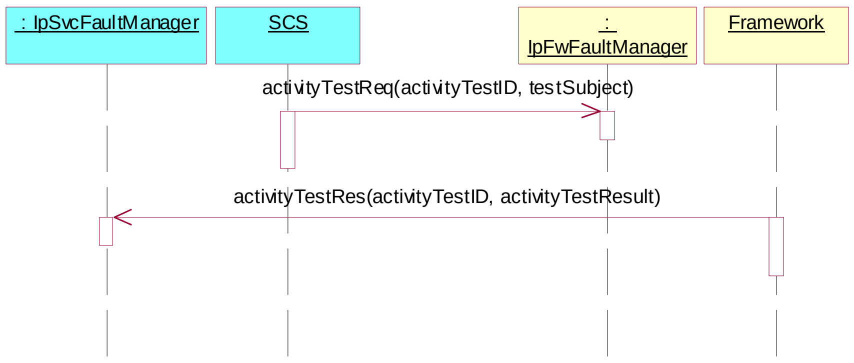
Precondition: IpFwFaultManager, activityTestReq supported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **activityTestReq()**
   Parameters: activityTestID, testSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **activityTestRes ()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID, activityTestResult


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
activityTestReq(activityTestID, testSubject)
activityTestRes(activityTestID, activityTestResult)

Test FW_FS_IM_05

Summary: **IpFwFaultManager,** generateFaultStatisticsRecordReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

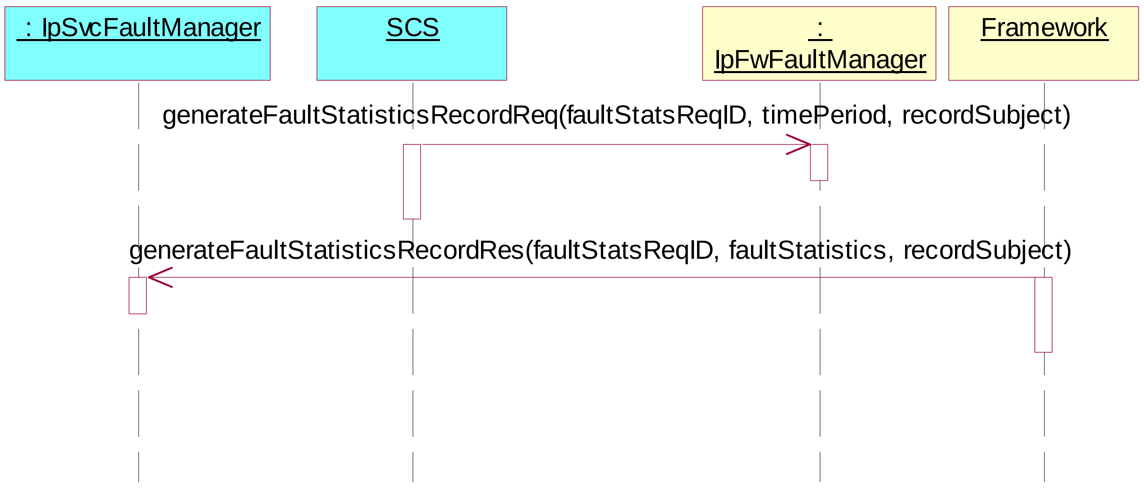
Precondition: IpFwFaultManager, generateFaultStatisticsRecordReqsupported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod, recordSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **generateFaultStatisticsRecordRes ()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: faultStatsReqID, faultStatistics, recordSubject


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, recordSubject)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics, recordSubject)

Test FW_FS_IM_06

Summary: **IpFwFaultManager,** generateFaultStatisticsRecordReq, unsuccessful.

Reference: ES 202 915-3 [1], clause 9.3.4.

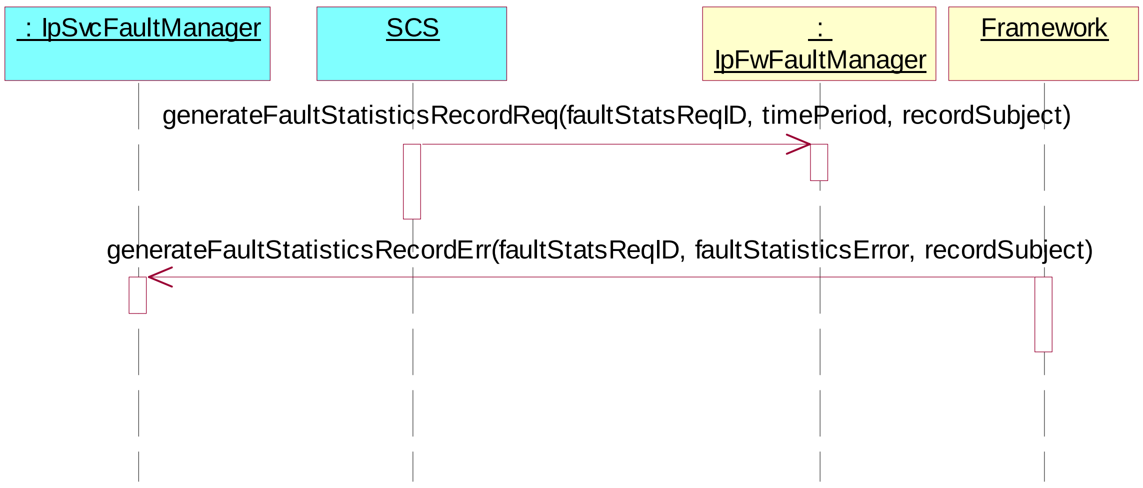
Precondition: IpFwFaultManager, and generateFaultStatisticsRecordErr supported
FW configured to respond with generateFaultStatisticsRecordErr (undefined or unavailable).

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod, recordSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call generateFaultStatisticsRecordErr **()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: faultStatsReqID, faultStatisticsError, recordSubject


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, recordSubject)
generateFaultStatisticsRecordErr(faultStatsReqID, faultStatisticsError, recordSubject)

Test FW_FS_IM_07

Summary: **IpFwFaultManager,** svcAvailStatusInd, successful.

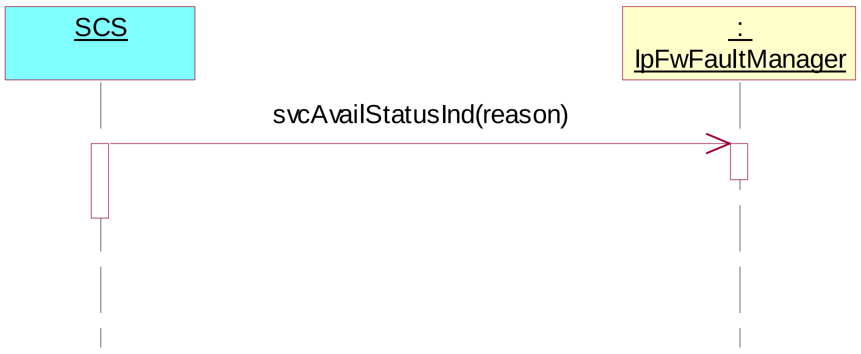
Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwFaultManager, svcAvailStatusInd supported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Method call **svcAvailStatusInd ()**
   Parameters: reason
   Check: no exception is returned



Test FW_FS_IM_08

Summary: **IpFwFaultManager,** svcActivityTestRes, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

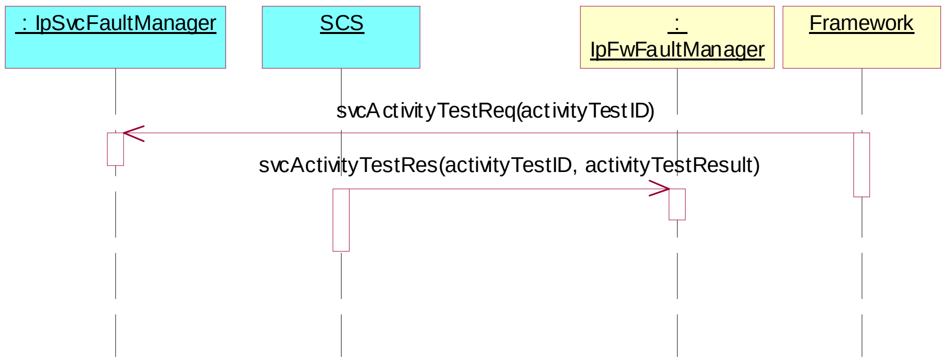
Precondition: IpFwFaultManager, svcActivityTestRes supported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestRes()**
   Parameters: activityTestID, activityTestResult
   Check: no exception is returned


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
svcActivityTestReq(activityTestID)
svcActivityTestRes(activityTestID, activityTestResult)

Test FW_FS_IM_09

Summary: **IpFwFaultManager,** svcActivityTestRes, P_INVALID_ACTIVITY_TEST_ID.

Reference: ES 202 915-3 [1], clause 9.3.4.

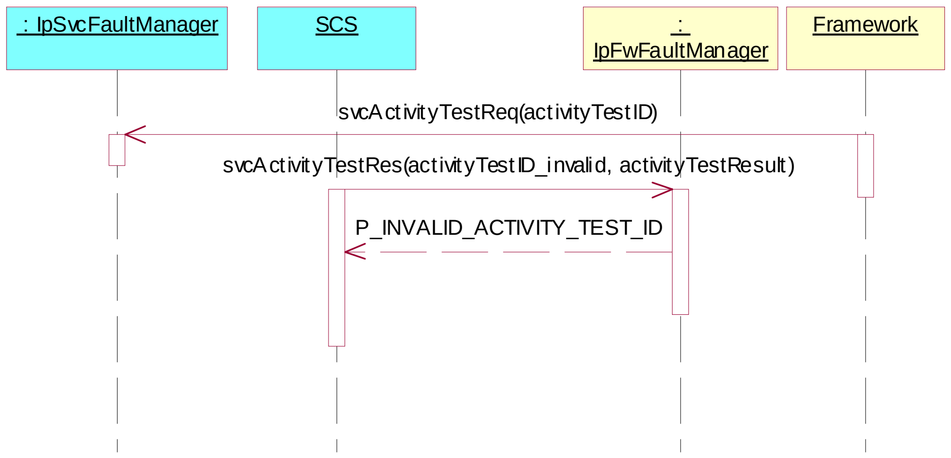
Precondition: IpFwFaultManager, svcActivityTestRes supported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestRes()**
   Parameters: invalid activityTestID, activityTestResult
   Check: P_INVALID_ACTIVITY_TEST_ID is returned.


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
svcActivityTestReq(activityTestID)
svcActivityTestRes(activityTestID_invalid, activityTestResult)
P_INVALID_ACTIVITY_TEST_ID

Test FW_FS_IM_10

Summary: **IpFwFaultManager,** svcActivityTestErr, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

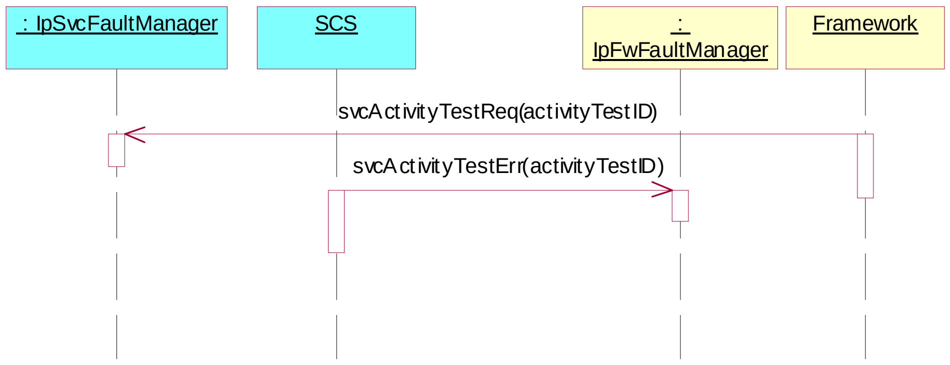
Precondition: IpFwFaultManager, svcActivityTestRes supported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestErr()**
   Parameters: activityTestID
   Check: no exception is returned


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
svcActivityTestReq(activityTestID)
svcActivityTestErr(activityTestID)

Test FW_FS_IM_11

Summary: **IpFwFaultManager,** svcActivityTestErr, P_INVALID_ACTIVITY_TEST_ID.

Reference: ES 202 915-3 [1], clause 9.3.4.

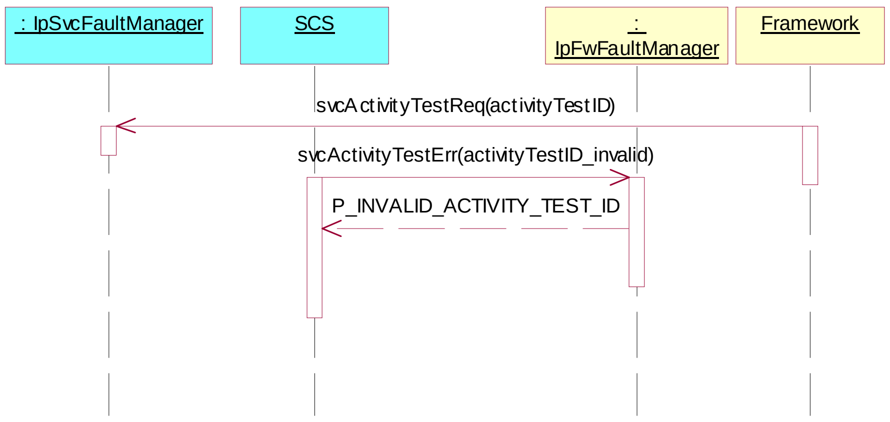
Precondition: IpFwFaultManager, svcActivityTestRes supported.

Preamble: There must be at least one service registered with the Framework.

Test Sequence:

1. Triggered action: cause IUT to call **svcActivityTestReq()** method on the tester's (Service) **IpSvcFaultManager** interface.
   Parameters: activityTestID

2. Method call **svcActivityTestErr()**
   Parameters: invalid activityTestID
   Check: P_INVALID_ACTIVITY_TEST_ID is returned.


: IpSvcFaultManager
SCS
: IpFwFaultManager
Framework
svcActivityTestReq(activityTestID)
svcActivityTestErr(activityTestID_invalid)
P_INVALID_ACTIVITY_TEST_ID

Test FW_FS_IM_12

Summary: **IpFwFaultManager,** appUnavailableInd, successful.

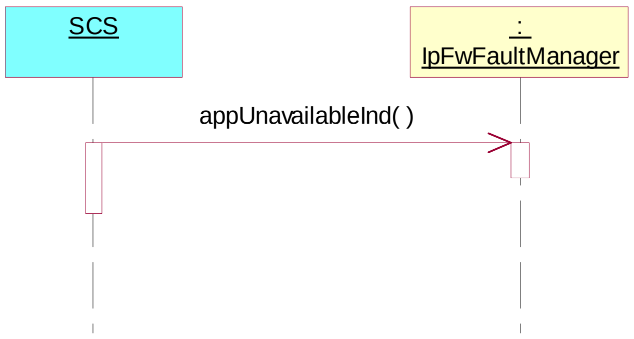
Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwFaultManager, appUnavailableInd supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **appUnavailableInd()**
   Parameters: none
   Check: no exception is returned



Test FW_FS_IM_13

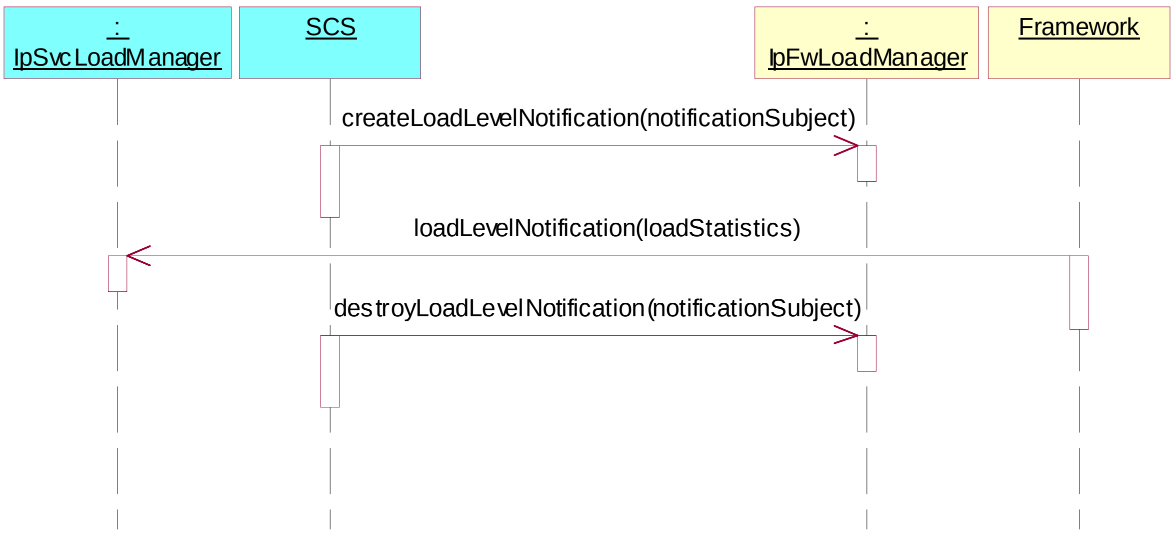
Summary:	**IpFwFaultManager,** createLoadLevelNotification and destroyLoadLevelNotification methods, successful.
Reference:	ES 202 915-3 [1], clause 9.3.4.
Precondition:	IpFwLoadManager, createLoadLevelNotification and destroyLoadLevelNotification notifications supported.
Preamble:	There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

2. Triggered action: cause IUT to call **loadLevelNotification()** method on the tester's (Application) **IpSvcLoadManager** interface.
   Parameters: loadStatistics

3. Method call **destroyLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned


: IpSvcLoadManager
SCS
: IpFwLoadManager
Framework
createLoadLevelNotification(notificationSubject)
loadLevelNotification(loadStatistics)
destroyLoadLevelNotification(notificationSubject)

Test FW_FS_IM_14

Summary: **IpFwLoadManager,** all methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwLoadManager, notifications with suspendNotification and resumeNotification supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

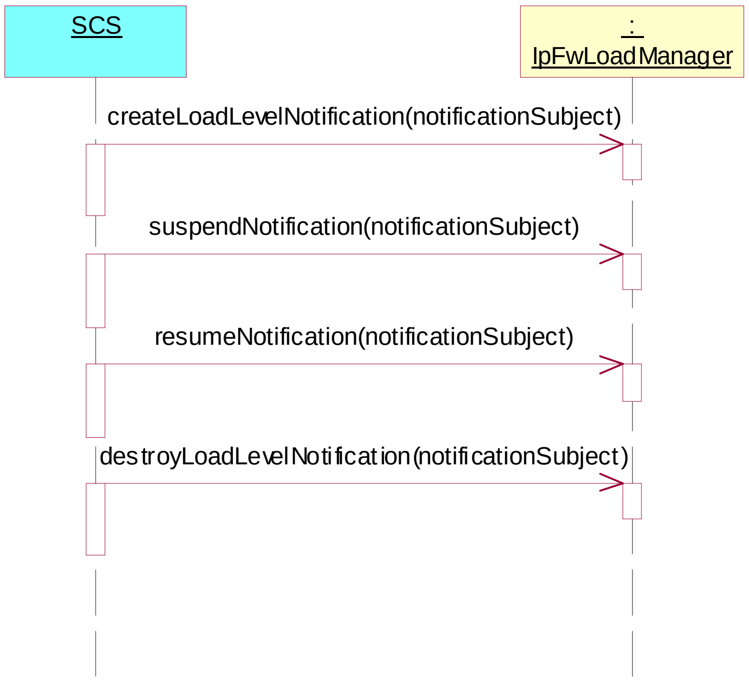
Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

2. Method call **suspendNotification()**
   Parameters: notificationSubject
   Check: no exception is returned, no load level notifications received until resumeNotification() is called.

3. Method call **resumeNotification()**
   Parameters: notificationSubject
   Check: no exception is returned

4. Method call **destroyLoadLevelNotification()**
   Parameters: notificationSubject
   Check: no exception is returned


SCS
: IpFwLoadManager
createLoadLevelNotification(notificationSubject)
suspendNotification(notificationSubject)
resumeNotification(notificationSubject)
destroyLoadLevelNotification(notificationSubject)

Test FW_FS_IM_15

Summary: **IpFwLoadManager,** reportLoad, successful.

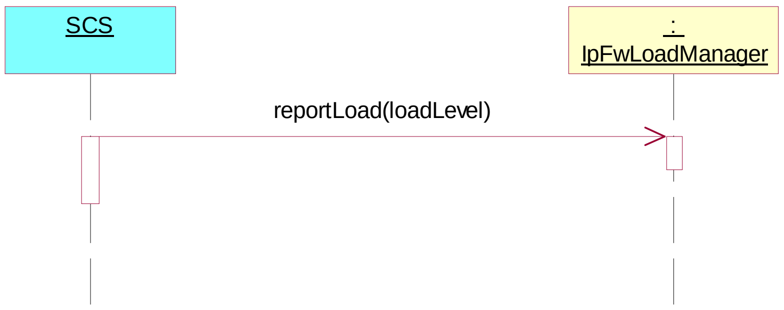
Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwLoadManager, reportLoad supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **reportLoad ()**
   Parameters: loadLevel
   Check: no exception is returned



Test FW_FS_IM_16

Summary: **IpFwLoadManager,** queryLoadStatsReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

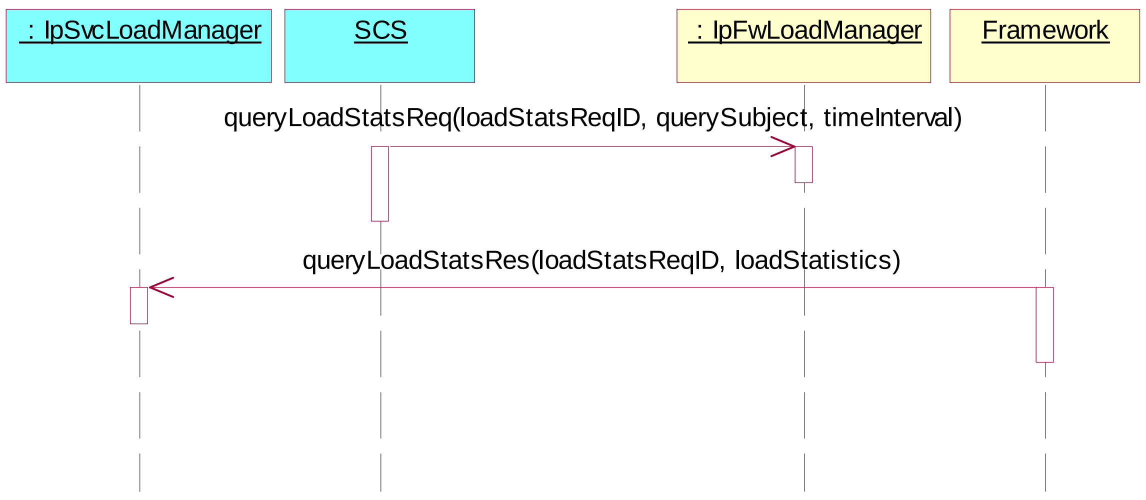
Precondition: IpFwLoadManager, queryLoadStatsReq supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Method call **queryLoadStatsReq()**
   Parameters: loadStatsReqID, querySubject, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryLoadStatsRes ()** method on the tester's (Service) **IpSvcLoadManager** interface.
   Parameters: loadStatsReqID, loadStatistics


: IpSvcLoadManager
SCS
: IpFwLoadManager
Framework
queryLoadStatsReq(loadStatsReqID, querySubject, timeInterval)
queryLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FS_IM_17

Summary: **IpFwLoadManager,** querySvcLoadStatsRes, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

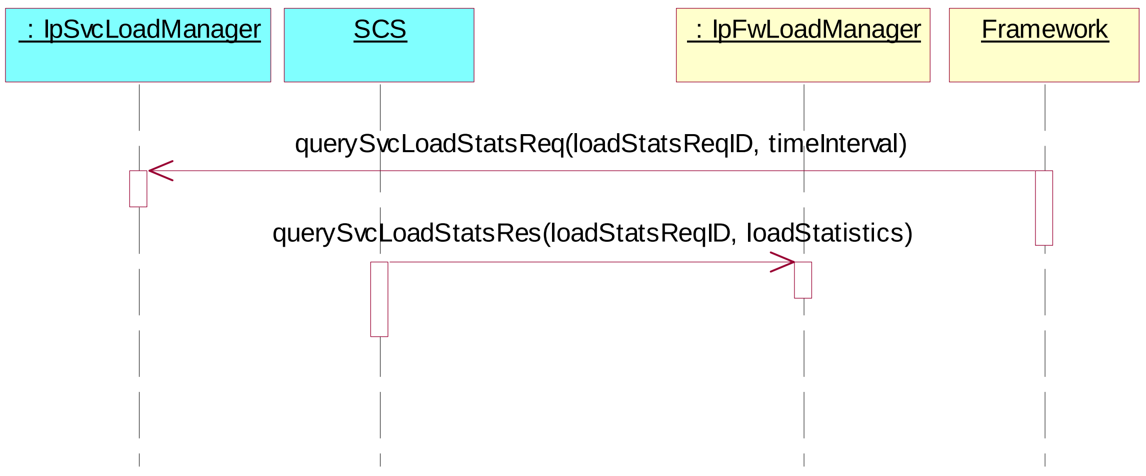
Precondition: IpFwLoadManager, querySvcLoadStatsRes supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Triggered action: cause IUT to call **querySvcLoadStatsReq ()** method on the tester's (Service) **IpSvcLoadManager** interface.
   Parameters: loadStatsReqID, timeInterval
   Check: no exception is returned.

2. Method call **querySvcLoadStatsRes ()**
   Parameters: loadStatsReqID, loadStatistics


: IpSvcLoadManager
SCS
: IpFwLoadManager
Framework
querySvcLoadStatsReq(loadStatsReqID, timeInterval)
querySvcLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FS_IM_18

Summary: **IpFwLoadManager,** querySvcLoadStatsErr, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

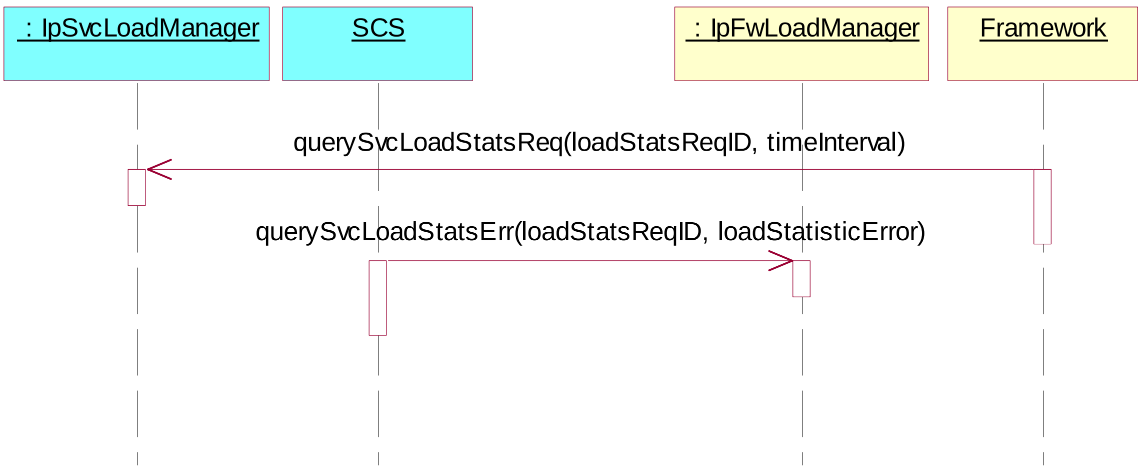
Precondition: IpFwLoadManager, querySvcLoadStatsRes supported.

Preamble: There must be at least one service registered with the Framework, and one application which has requested an instance of that service.

Test Sequence:

1. Triggered action: cause IUT to call **querySvcLoadStatsReq ()** method on the tester's (Service) **IpSvcLoadManager** interface.
   Parameters: loadStatsReqID, timeInterval
   Check: no exception is returned.

2. Method call **querySvcLoadStatsErr ()**
   Parameters: loadStatsReqID, loadStatisticsError
   Check: no exception is returned


: IpSvcLoadManager
SCS
: IpFwLoadManager
Framework
querySvcLoadStatsReq(loadStatsReqID, timeInterval)
querySvcLoadStatsErr(loadStatsReqID, loadStatisticError)

Test FW_FS_IM_19

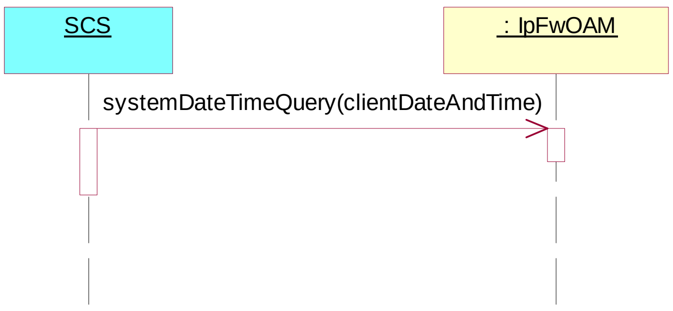
Summary: **IpFwOAM**, systemDateTimeQuery, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwOAM supported.

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: clientDateAndTime
   Check: valid value of TpDateAndTime is returned



Test FW_FS_IM_20

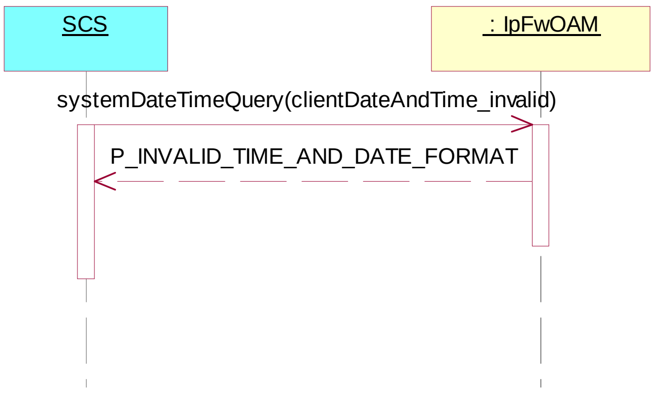
Summary: **IpFwOAM**, systemDateTimeQuery, P_INVALID_TIME_AND_DATE_FORMAT.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwOAM supported.

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: invalid clientDateAndTime
   Check: P_INVALID_TIME_AND_DATE_FORMAT is returned.


SCS
: IpFwOAM
systemDateTimeQuery(clientDateAndTime_invalid)
P_INVALID_TIME_AND_DATE_FORMAT

5.4.4.5 Event Notification (EN)

Test FW_FS_EN_01

Summary: **IpFwEventNotification**, createNotification and destroyNotification methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.5.

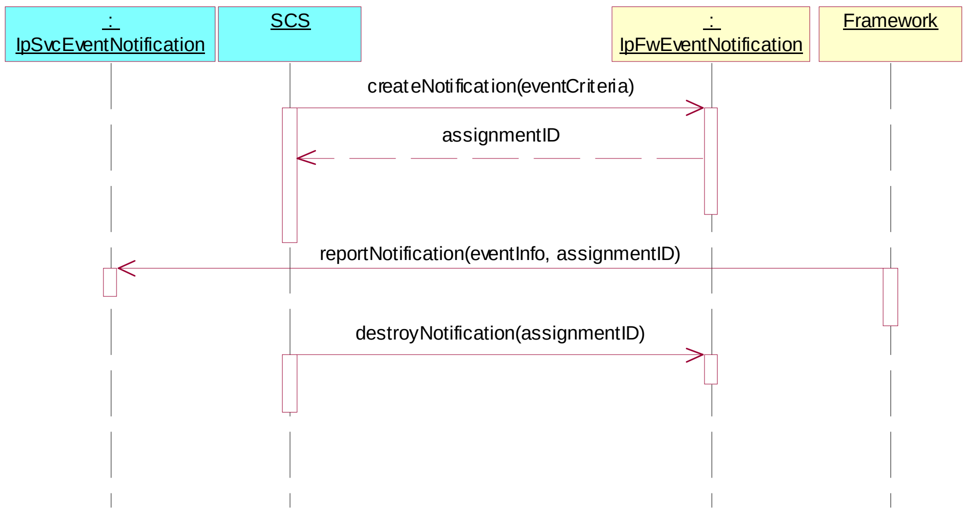
Precondition: IpFwEventNotification supported.

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Service) **IpSvcEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned


: IpSvcEventNotification
SCS
: IpFwEventNotification
Framework
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
destroyNotification(assignmentID)

Test FW_FS_EN_02

Summary: **IpFwEventNotification**, createNotification and destroyNotification methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.5.

Precondition: IpFwEventNotification supported.

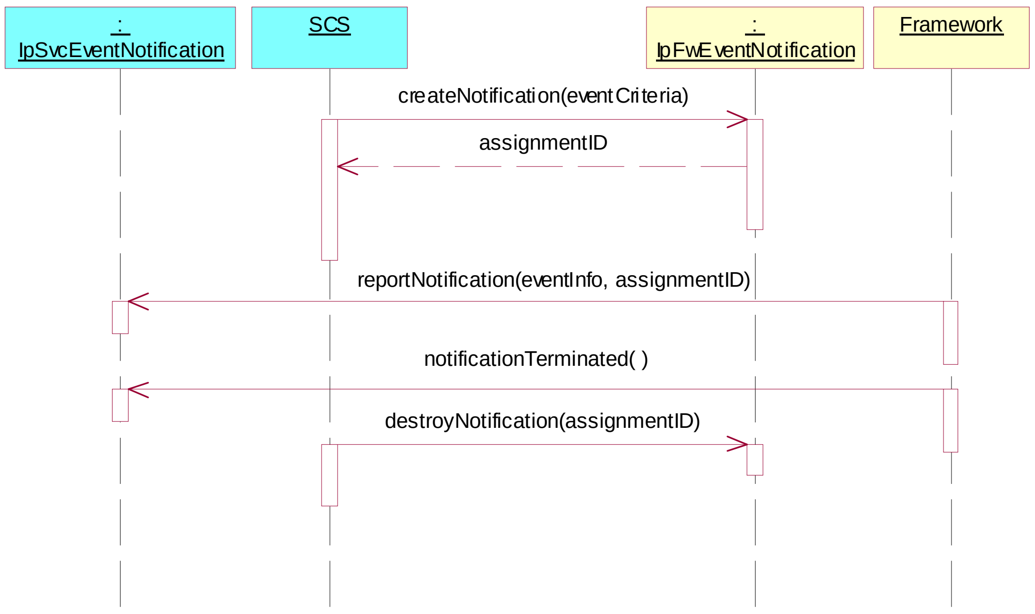
Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Triggered action: cause IUT to call **reportNotification()** method on the tester's (Service) **IpSvcEventNotification** interface.
   Parameters: eventInfo, assignmentID

3. Triggered action: cause IUT to call **notificationTerminated()** method on the tester's (Service) **IpSvcEventNotification** interface.
   Parameters: none.

4. Method call **destroyNotification()**
   Parameters: assignmentID give in 1.
   Check: no exception is returned


: IpSvcEventNotification
SCS
: IpFwEventNotification
Framework
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
notificationTerminated( )
destroyNotification(assignmentID)

Test FW_FS_EN_03

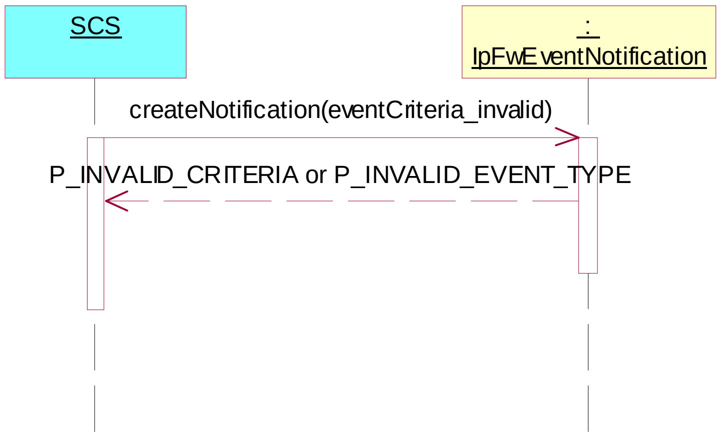
Summary: **IpFwEventNotification**, createNotification, P_INVALID_CRITERIA.

Reference: ES 202 915-3 [1], clause 9.3.5.

Precondition: IpFwEventNotification supported.

Test Sequence:

1. Method call **createNotification()**
   Parameters: invalid eventCriteria
   Check: P_INVALID_ CRITERIA or P_INVALID_EVENT_TYPE, is returned.


SCS
: IpFwEventNotification
createNotification(eventCriteria_invalid)
P_INVALID_CRITERIA or P_INVALID_EVENT_TYPE

Test FW_FS_EN_04

Summary: **IpFwEventNotification**, destroyNotification, P_INVALID_ ASSIGNMENT_ID.

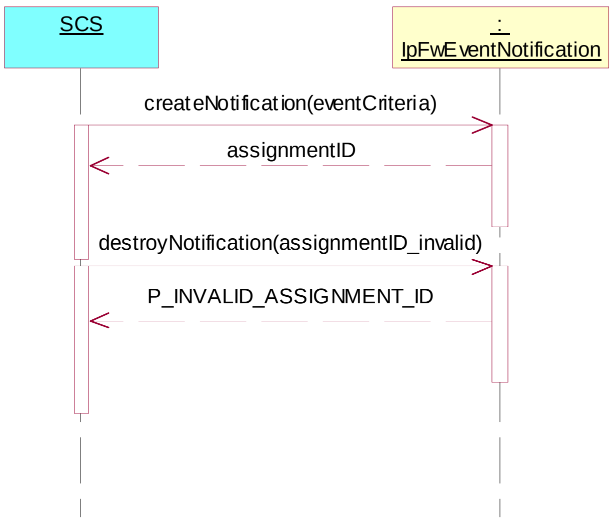
Reference: ES 202 915-3 [1], clause 9.3.5.

Precondition: IpFwEventNotification supported.

Test Sequence:

1. Method call **createNotification()**
   Parameters: eventCriteria
   Check: valid value of TpAssignmentID is returned

2. Method call **destroyNotification()**
   Parameters: invalid assignmentID
   Check: P_INVALID_ ASSIGNMENT_ID, is returned.


SCS
: IpFwEventNotification
createNotification(eventCriteria)
assignmentID
destroyNotification(assignmentID_invalid)
P_INVALID_ASSIGNMENT_ID

6 Test Suite Structure (TSS) for Access Client

Framework (FW)

- Framework Access Session (AS)
  - Trust and Security Management (TSM)
- Framework To Application (FTA)
  - Service discovery
  - Service agreement management
  - Integrity management
  - Event notification
- Framework To Service (FTS)
  - Service registration
  - Service instance lifecycle management
  - Service discovery
  - Integrity management
  - Event notification
- Framework To Enterprise operator (FTE)
  - Service subscription

7 Test Purposes (TP) for Access Client

For each test requirement a TP is defined.

7.1 TP naming convention

TPs are numbered, starting at 01, within each group. Groups are organized according to the TSS. Additional references are added to identify the actual test suite (see table 2).

Table 2: TP identifier naming convention scheme

Identifier: <suite_id>_<group>_<nnn>		
<suite_id>	= IUT name:	"FW" for **F**rame**W**ork SCF
<group>	= group number:	two character field representing the group reference according to TSS
<nn>	= sequential number:	(01-99)

7.2 Source of TP definition

The TPs are based on ES 202 915-3 [1].

7.3 Test strategy

As the base standard ES 202 915-3 [1] contains no explicit requirements for testing, the TPs were generated as a result of an analysis of the base standard and the ICS specification ES 202 363 [2].

The TPs are only based on conformance requirements related to the externally observable behaviour of the IUT and are limited to conceivable situations to which a real implementation is likely to be faced (see ETS 300 406 [5]).

7.4 TPs for the Framework Access Session API

All ICS items referred to in this clause are as specified in ES 202 363 [2] unless indicated otherwise by another numbered reference.

All parameters specified in method calls are valid unless specified.

The procedures to trigger the SCF to call methods in the framework are dependant on the SUT and are out of the scope of the present document. Those method calls are preceded by the words "Triggered action".

7.4.1 Trust and Security Management (TSM)

Test FW_AS_TSM_C01

Summary: Initial Access for Trusted Parties, no authentication is needed, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

Test Sequence:

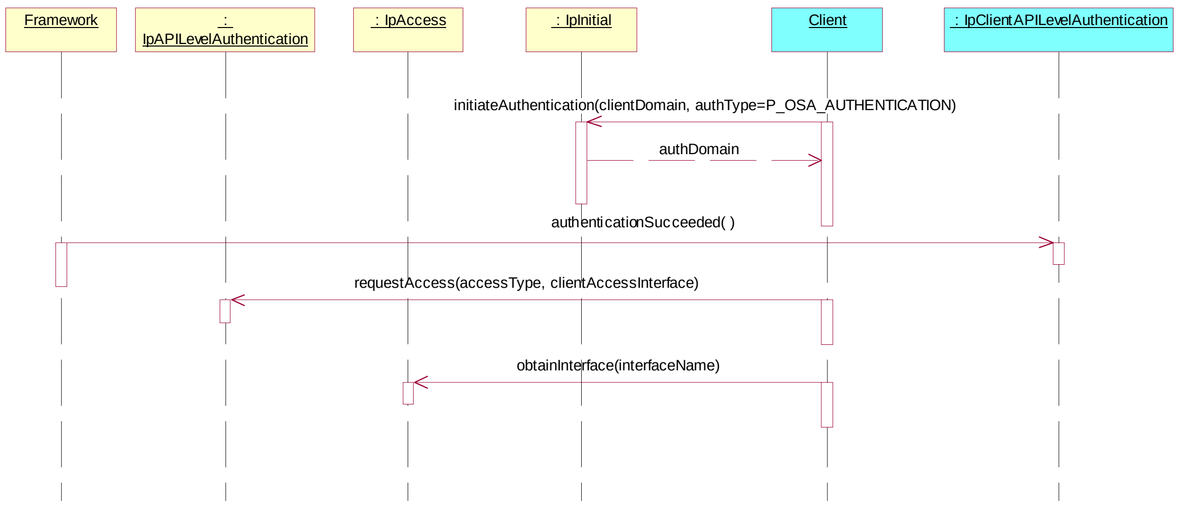
1. Triggered action: cause IUT to call **initiateAuthentication ()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION

2. Method call **authenticationSucceeded()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned

3. Triggered action: cause IUT to call **requestAccess**() method on the tester's (Framework) **IpAPILevelAuthentication** interface
   Parameters: accessType, clientAccessInterface

4. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)

The client may invoke selectEncryptionMethod() on the tester's (Framework) IpAPILevelAuthentication interface at any time following the invocation of initiateAuthentication().


Framework
: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
: IpClientAPILevelAuthentication
initiateAuthentication(clientDomain, authType=P_OSA_AUTHENTICATION)
authDomain
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C02

Summary: API level authentication, FW authenticates the client only, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

Test Sequence:

1. Triggered action: cause IUT to call **initiateAuthentication ()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION

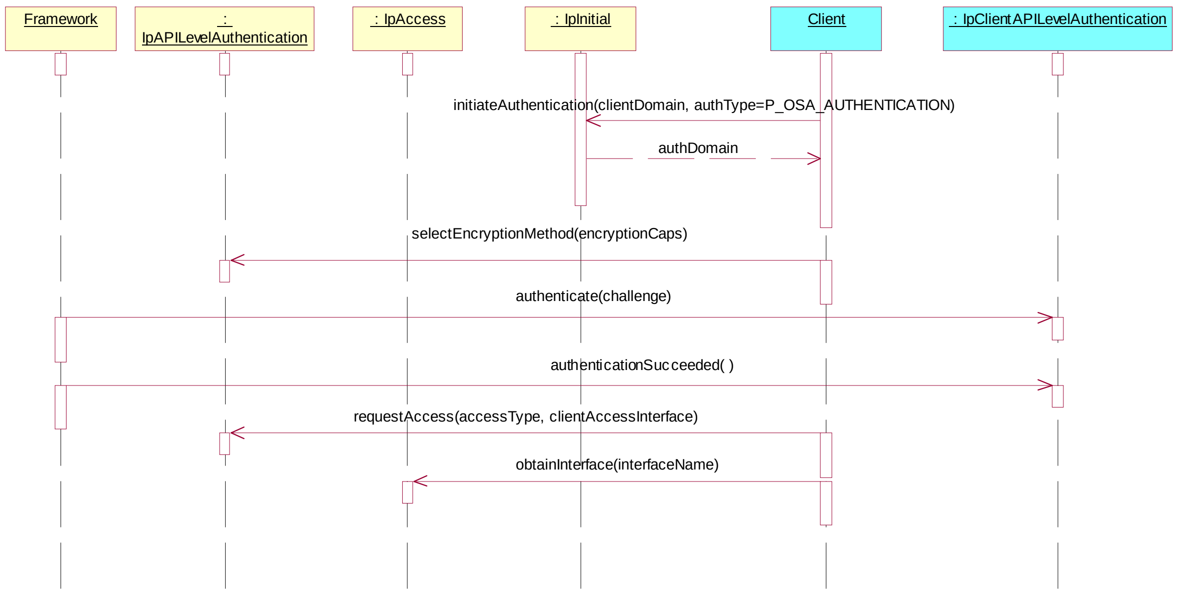
2. Triggered action: cause IUT to call **selectEncryptionMethod ()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps

3. Method call **authenticate()** on **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

4. Method call **authenticationSucceeded()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned

5. Triggered action: cause IUT to call **requestAccess**() method on the tester's (Framework) **IpAPILevelAuthentication** interface
   Parameters: accessType, clientAccessInterface

6. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)


Framework
: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
: IpClientAPILevelAuthentication
initiateAuthentication(clientDomain, authType=P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
authenticate(challenge)
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C03

Summary: API level authentication, FW and client authenticate mutually, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

Test Sequence:

1. Triggered action: cause IUT to call **initiateAuthentication ()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION

2. Triggered action: cause IUT to call **selectEncryptionMethod ()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps

3. Method call **authenticate()** on **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

4. Method call **authenticationSucceeded()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned

5. Triggered action: cause IUT to call authenticate() on IpAPILevelAuthentication interface.
   Framework returns valid value of TpOctetSet

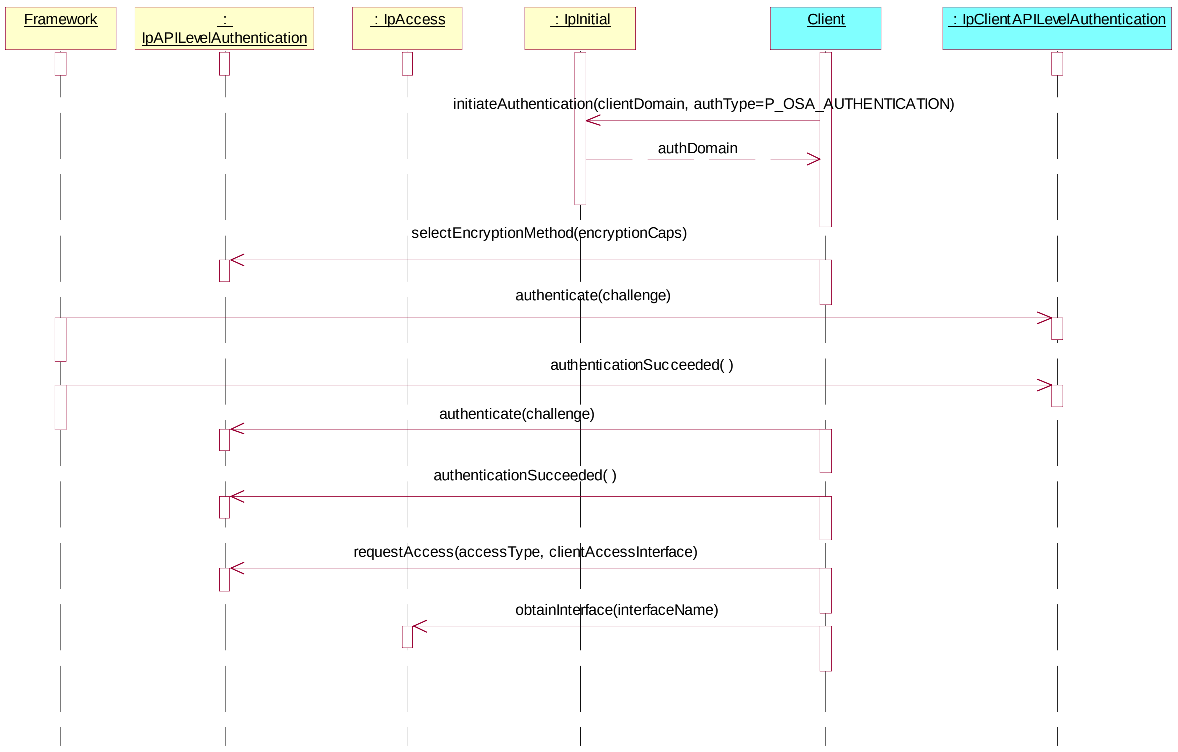
6. Triggered action: cause IUT to call authenticationSucceeded() on IpAPILevelAuthentication interface

NOTE 1: Methods 5 and 6 may be invoked in this order, but interleaved before, during or after methods 3 or 4.

7. Triggered action: cause IUT to call requestAccess() method on the tester's (Framework) IpAPILevelAuthentication interface
   Parameters: accessType, clientAccessInterface

NOTE 2: This method may be invoked any time following the IUT's calling of authenticationSucceeded() on the IpClientAPILevelAuthentication interface.

8. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)


Framework
: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
: IpClientAPILevelAuthentication
initiateAuthentication(clientDomain, authType=P_OSA_AUTHENTICATION)
authDomain
selectEncryptionMethod(encryptionCaps)
authenticate(challenge)
authenticationSucceeded( )
authenticate(challenge)
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C04

Summary: Authentication, using Underlying Distribution Technology Mechanism, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.1.1.3.

Precondition: Underlying authentication supported.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

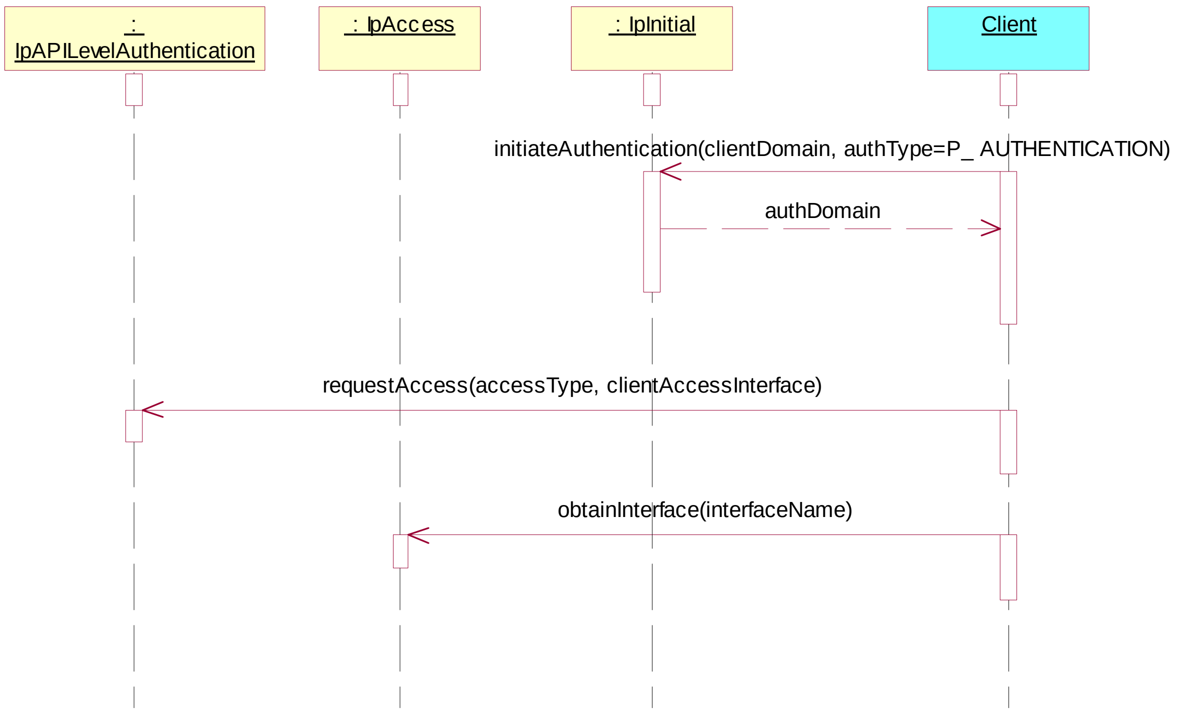
Test Sequence:

1. Perform underlying authentication between tester and IUT.

2. Triggered action: cause IUT to call **initiateAuthentication** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_ AUTHENTICATION

3. Triggered action: cause IUT to call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface

4. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)


: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
initiateAuthentication(clientDomain, authType=P_ AUTHENTICATION)
authDomain
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C05

Summary: API level authentication, client authenticates the FW, FW aborts authentication.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

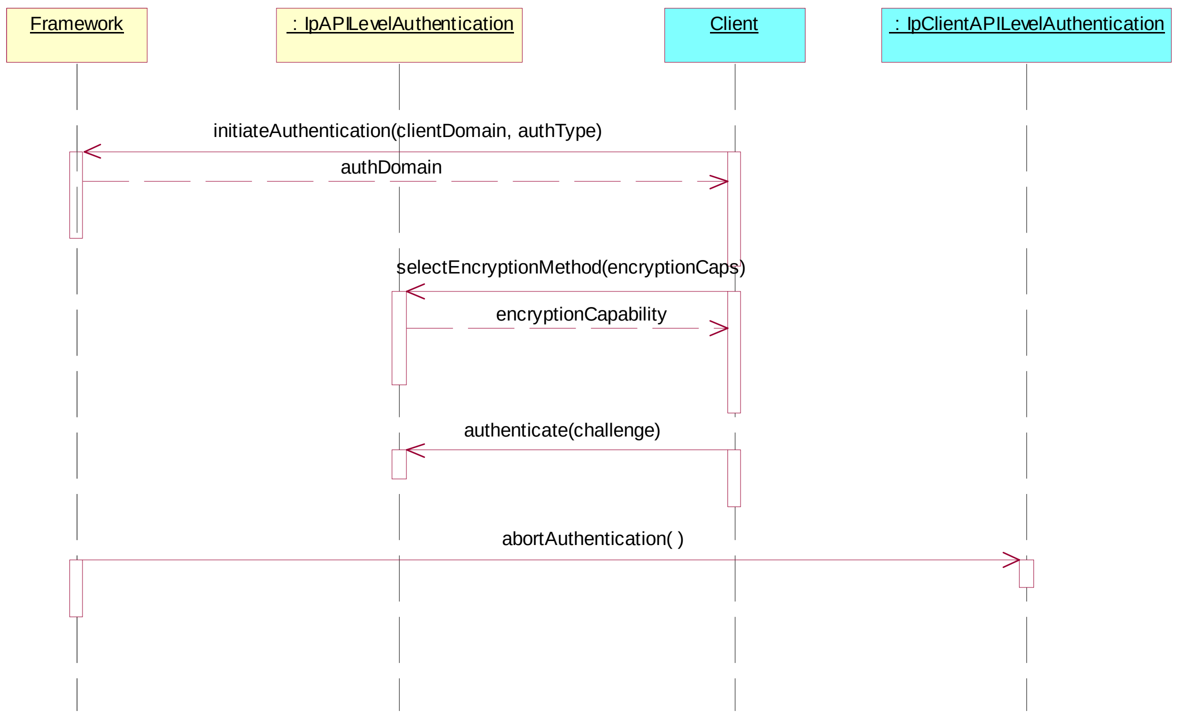
Test Sequence:

1. Triggered action: cause IUT to call **initiateAuthentication ()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION

2. Triggered action: cause IUT to call **selectEncryptionMethod ()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps

3. Triggered action: cause IUT to call **authenticate ()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: encryptionCaps

4. Method call **abortAuthentication()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned


Framework
: IpAPILevelAuthentication
Client
: IpClientAPILevelAuthentication
initiateAuthentication(clientDomain, authType)
authDomain
selectEncryptionMethod(encryptionCaps)
encryptionCapability
authenticate(challenge)
abortAuthentication( )

Test FW_AS_TSM_C06

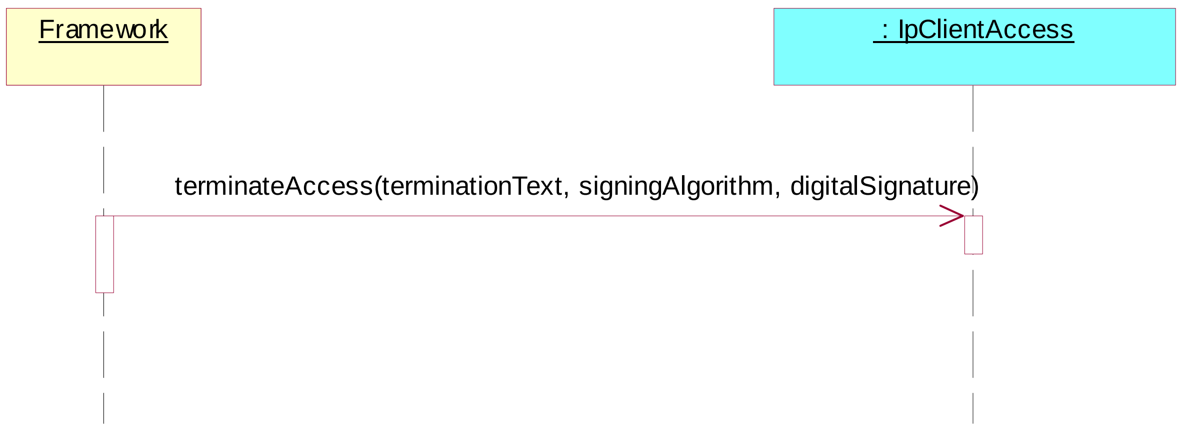
Summary: **IpClientAccess**, terminateAccess, successful.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement, and an access session has been established between the Framework and the Client.

Test Sequence:

1. Method call **terminateAccess** on **IpClientAccess** interface.
   Parameters: terminationText, signingAlgorithm, digitalSignature
   Check: no exception is returned.


Framework
: IpClientAccess
terminateAccess(terminationText, signingAlgorithm, digitalSignature)

Test FW_AS_TSM_C07

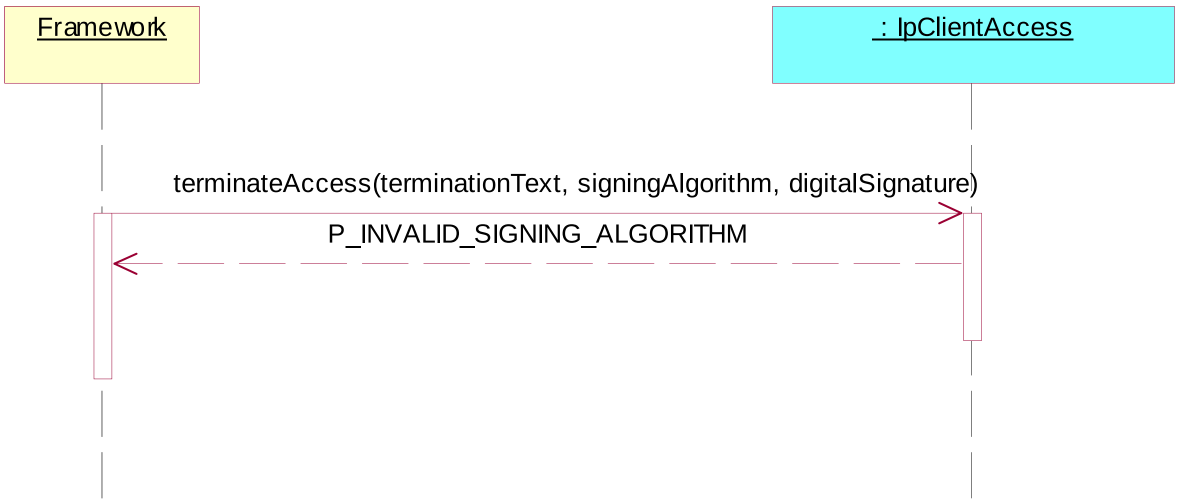
Summary: **IpClientAccess**, terminateAccess, P_INVALID_SIGNING_ALGORITHM.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Preamble: Registration of the IUT (Application) to the tester (framework) by an off-line service agreement, and an access session has been established between the Framework and the Client.

Test Sequence:

1. Method call **terminateAccess** on **IpClientAccess** interface.
   Parameters: terminationText, invalid signingAlgorithm, digitalSignature
   Check: P_INVALID_SIGNING_ALGORITHM is returned.


Framework
: IpClientAccess
terminateAccess(terminationText, signingAlgorithm, digitalSignature)
P_INVALID_SIGNING_ALGORITHM

Test FW_AS_TSM_C08

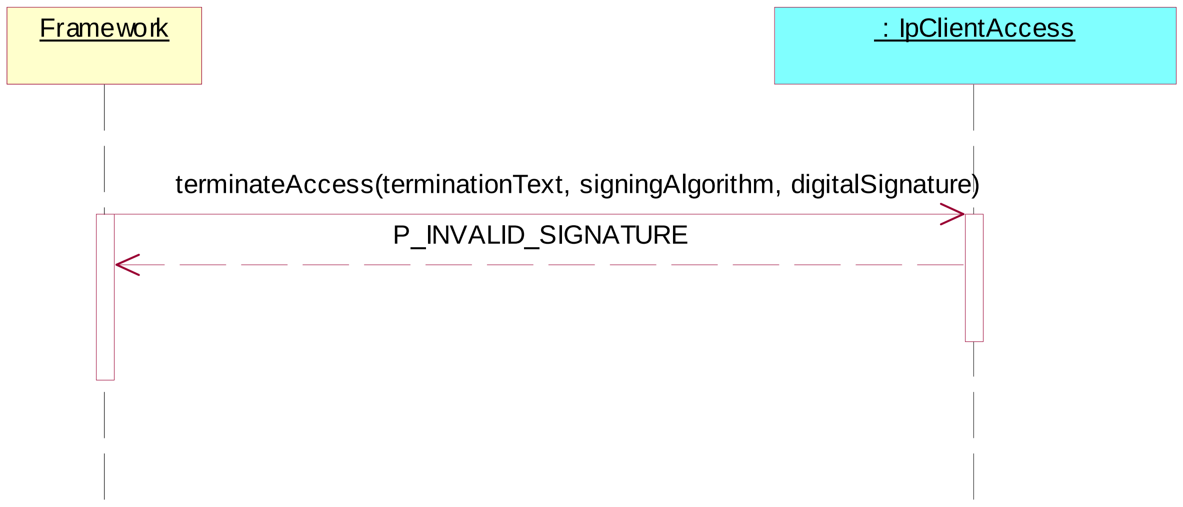
Summary: **IpClientAccess**, terminateAccess, P_INVALID_SIGNATURE.

Reference: ES 202 915-3 [1], clauses 6.1.1.2 and 6.1.1.4.

Preamble: Registration of the IUT (Application) to the tester (framework) by an off-line service agreement, and an access session has been established between the Framework and the Client.

Test Sequence:

1. Method call **terminateAccess** on **IpClientAccess** interface.
   Parameters: terminationText, signingAlgorithm, invalid digitalSignature
   Check: P_INVALID_SIGNATURE is returned.


Framework
: IpClientAccess
terminateAccess(terminationText, signingAlgorithm, digitalSignature)
P_INVALID_SIGNATURE

Test FW_AS_TSM_C09

Summary: Initial Access for Trusted Parties, no authentication is needed, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

Test Sequence:

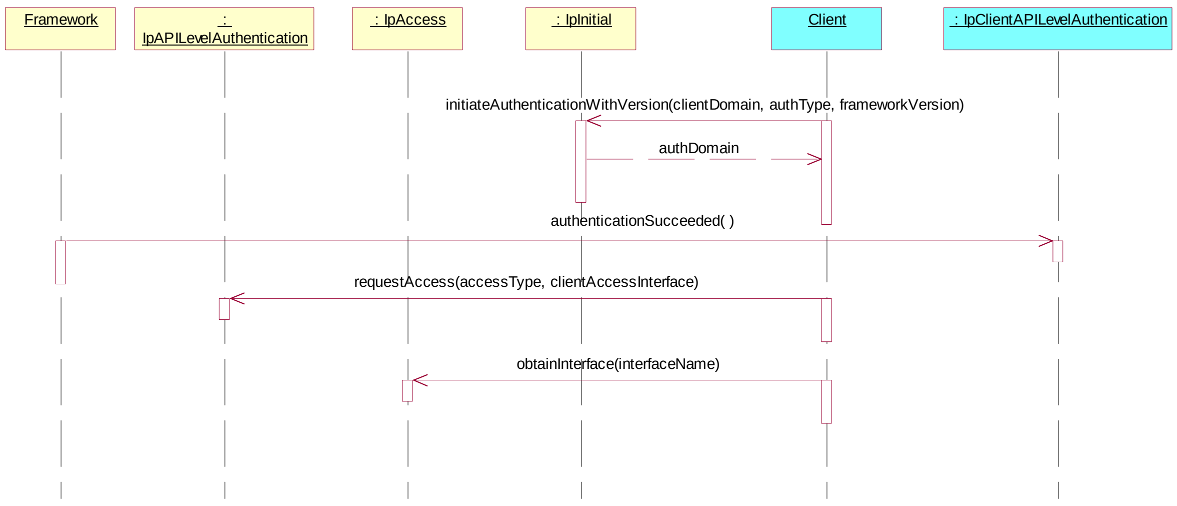
1. Triggered action: cause IUT to call **initiateAuthenticationWithVersion()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion

2. Method call **authenticationSucceeded()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned

3. Triggered action: cause IUT to call **requestAccess**() method on the tester's (Framework) **IpAPILevelAuthentication** interface
   Parameters: accessType, clientAccessInterface

4. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)

The client may invoke selectEncryptionMethod() on the tester's (Framework) IpAPILevelAuthentication interface at any time following the invocation of initiateAuthentication().


Framework
: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
: IpClientAPILevelAuthentication
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C10

Summary: API level authentication, FW authenticates the client only, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

Test Sequence:

1. Triggered action: cause IUT to call **initiateAuthenticationWithVersion()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion

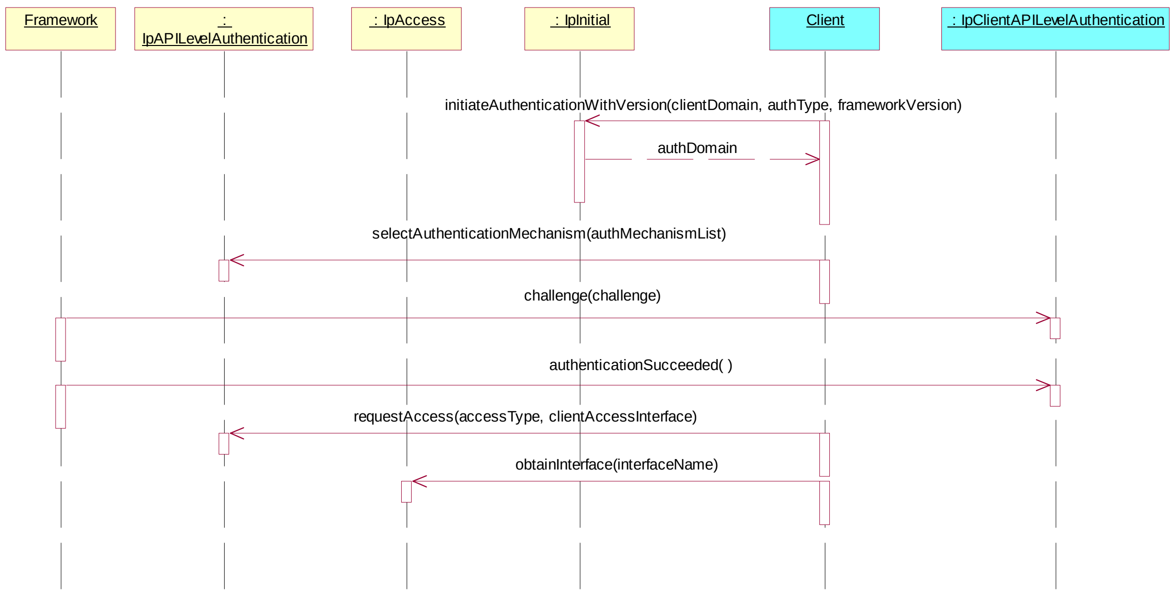
2. Triggered action: cause IUT to call **selectAuthenticationMechanism()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList

3. Method call **challenge()** on **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

4. Method call **authenticationSucceeded()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned

5. Triggered action: cause IUT to call **requestAccess**() method on the tester's (Framework) **IpAPILevelAuthentication** interface
   Parameters: accessType, clientAccessInterface

6. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)


Framework
: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
: IpClientAPILevelAuthentication
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
challenge(challenge)
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C11

Summary: API level authentication, FW and client authenticate mutually, all methods, successful.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

Test Sequence:

1. Triggered action: cause IUT to call **initiateAuthenticationWithVersion()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion

2. Triggered action: cause IUT to call **selectAuthenticationMechanism()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList

3. Method call **challenge()** on **IpClientAPILevelAuthentication** interface.
   Parameters: challenge
   Check: valid value of TpOctetSet is returned

4. Method call **authenticationSucceeded()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned

5. Triggered action: cause IUT to call challenge() on IpAPILevelAuthentication interface.
   Framework returns valid value of TpOctetSet

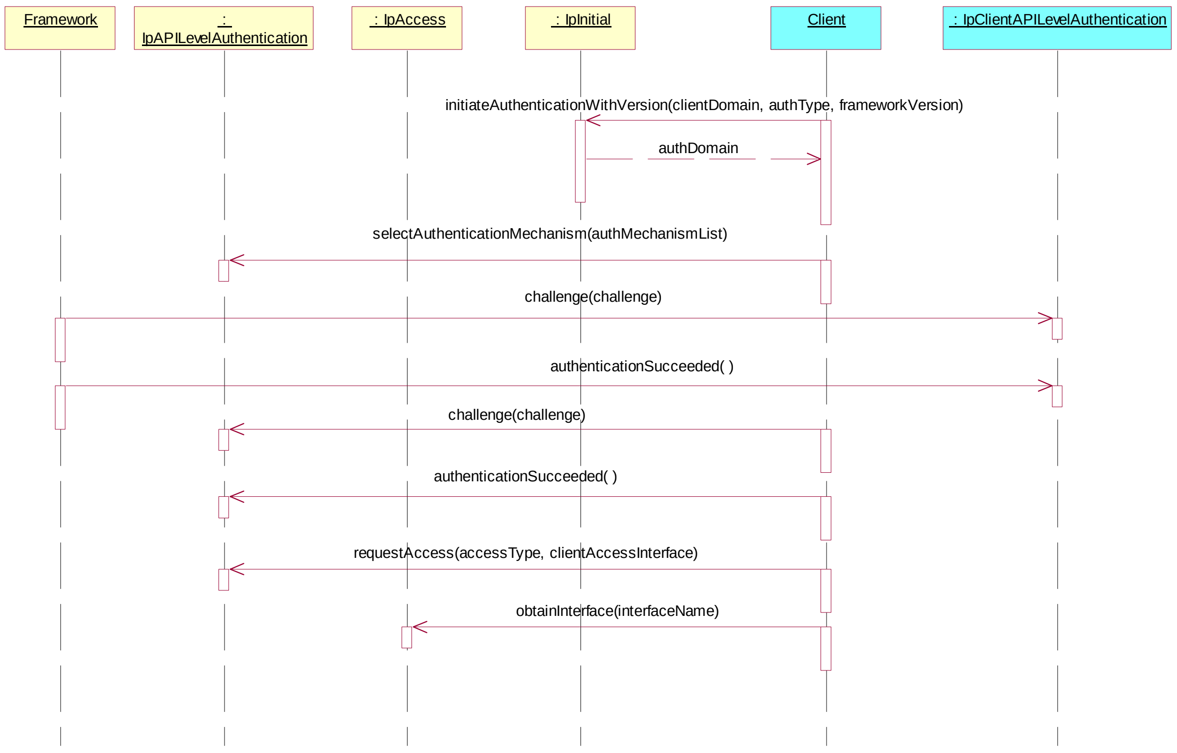
6. Triggered action: cause IUT to call authenticationSucceeded() on IpAPILevelAuthentication interface

NOTE 1: Methods 5 and 6 may be invoked in this order, but interleaved before, during or after methods 3 or 4.

7. Triggered action: cause IUT to call requestAccess() method on the tester's (Framework) IpAPILevelAuthentication interface
   Parameters: accessType, clientAccessInterface

NOTE 2: This method may be invoked any time following the IUT's calling of authenticationSucceeded() on the IpClientAPILevelAuthentication interface.

8. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)


Framework
: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
: IpClientAPILevelAuthentication
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
challenge(challenge)
authenticationSucceeded( )
challenge(challenge)
authenticationSucceeded( )
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C12

Summary: Authentication, using Underlying Distribution Technology Mechanism, all methods, successful.

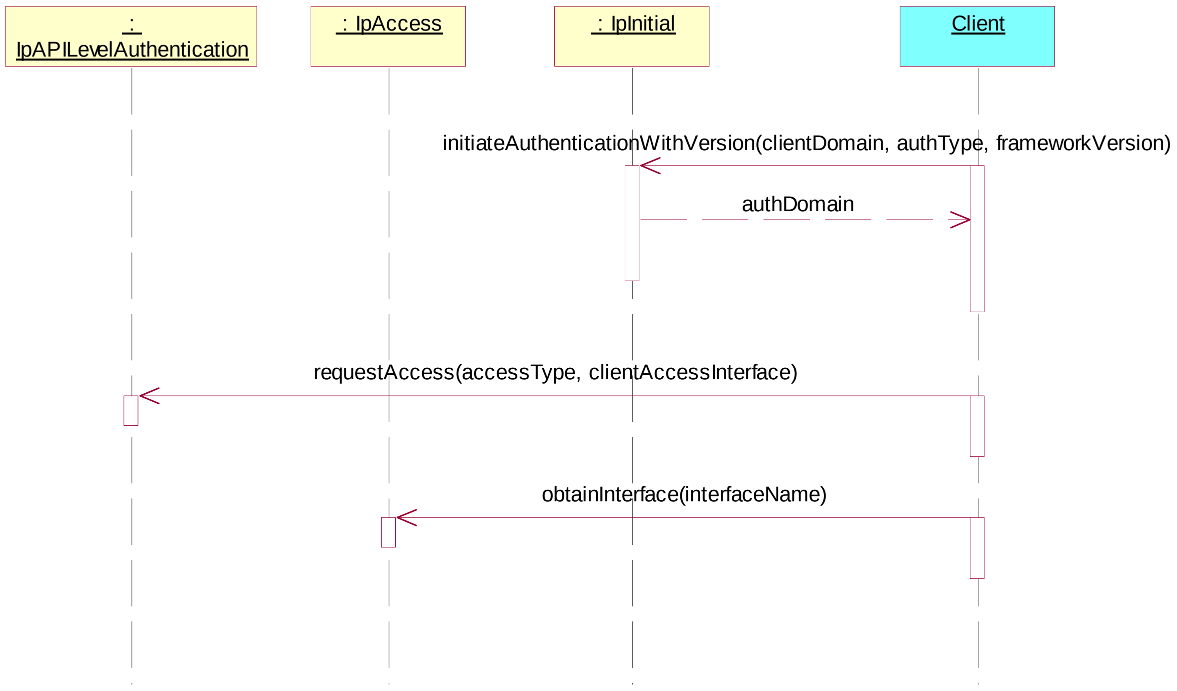
Reference: ES 202 915-3 [1], clause 6.1.1.3.

Precondition: Underlying authentication supported.

Preamble: Registration of the tester (Application) to the IUT (framework) by an off-line service agreement.

Test Sequence:

1. Perform underlying authentication between tester and IUT.
2. Triggered action: cause IUT to call **initiateAuthenticationWithVersion** on **IpInitial** interface.
   Parameters: clientDomain, authType=P_ AUTHENTICATION, frameworkVersion
3. Triggered action: cause IUT to call **requestAccess** on **IpAPILevelAuthentication** interface.
   Parameters: accessType, clientAccessInterface
4. Triggered action: cause IUT to call **obtainInterface** on **IpAccess** interface.
   Parameters: interfaceName
   (IUT may call **obtainInterfaceWithCallback** instead)


: IpAPILevelAuthentication
: IpAccess
: IpInitial
Client
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
requestAccess(accessType, clientAccessInterface)
obtainInterface(interfaceName)

Test FW_AS_TSM_C13

Summary: API level authentication, client authenticates the FW, FW aborts authentication.

Reference: ES 202 915-3 [1], clause 6.3.1.1.

Preamble: Registration of the IUT (Application) to the tester (Framework) by an off-line service agreement.

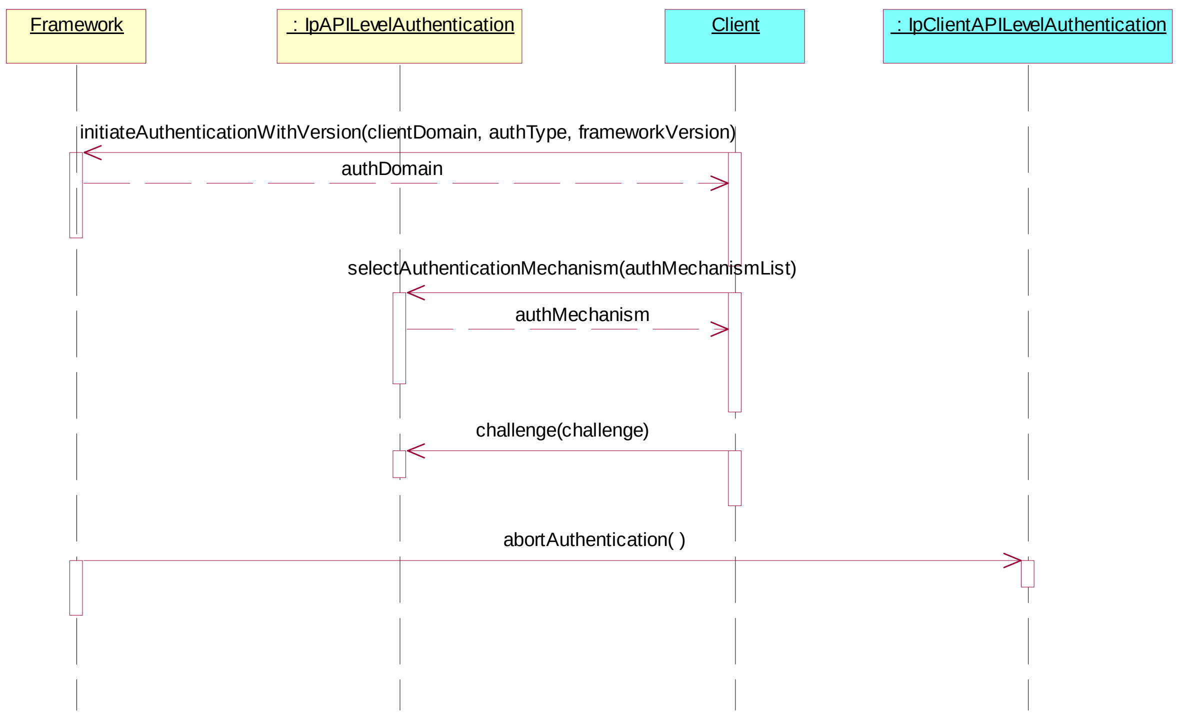
Test Sequence:

1. Triggered action: cause IUT to call **initiateAuthenticationWithVersion()** method on the tester's (Framework) **IpInitial** interface.
   Parameters: clientDomain, authType=P_OSA_AUTHENTICATION, frameworkVersion

2. Triggered action: cause IUT to call **selectAuthenticationMechanism()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: authMechanismList

3. Triggered action: cause IUT to call **challenge()** method on the tester's (Framework) **IpAPILevelAuthentication** interface.
   Parameters: challenge

4. Method call **abortAuthentication()** on **IpClientAPILevelAuthentication** interface.
   Parameters: none
   Check: no exception is returned


Framework
: IpAPILevelAuthentication
Client
: IpClientAPILevelAuthentication
initiateAuthenticationWithVersion(clientDomain, authType, frameworkVersion)
authDomain
selectAuthenticationMechanism(authMechanismList)
authMechanism
challenge(challenge)
abortAuthentication( )

8 Test Suite Structure (TSS) for Application

Framework (FW)

- Framework Access Session (AS)
    - Trust and Security Management (TSM)
- Framework To Application (FTA)
    - Service discovery
    - Service agreement management
    - Integrity management
    - Event notification
- Framework To Service (FTS)
    - Service registration
    - Service instance lifecycle management
    - Service discovery
    - Integrity management
    - Event notification
- Framework To Enterprise operator (FTE)
    - Service subscription

9 Test Purposes (TP) for Application

For each test requirement a TP is defined.

9.1 TP naming convention

TPs are numbered, starting at 01, within each group. Groups are organized according to the TSS. Additional references are added to identify the actual test suite (see table 3).

Table 3: TP identifier naming convention scheme

Identifier: <suite_id>_<group>_<nnn>		
<suite_id>	= IUT name:	"FW" for **F**rame**W**ork SCF
<group>	= group number:	two character field representing the group reference according to TSS
<nn>	= sequential number:	(01-99)

9.2 Source of TP definition

The TPs are based on ES 202 915-3 [1].

9.3 Test strategy

As the base standard ES 202 915-3 [1] contains no explicit requirements for testing, the TPs were generated as a result of an analysis of the base standard and the ICS specification ES 202 363 [2].

The TPs are only based on conformance requirements related to the externally observable behaviour of the IUT and are limited to conceivable situations to which a real implementation is likely to be faced (see ETS 300 406 [5]).

9.4 TPs for the Framework to Application API

All ICS items referred to in this clause are as specified in ES 202 363 [2] unless indicated otherwise by another numbered reference.

All parameters specified in method calls are valid unless specified.

The procedures to trigger the application to call methods in the framework are dependant on the SUT and are out of the scope of the present document. Those method calls are preceded by the words "Triggered action".

9.4.1 Service Discovery (SD)

Test FW_FA_SD_A01

Summary: **IpServiceDiscovery,** discoverService, successful.

Reference: ES 202 915-3 [1], clause 7.3.1.

Test Sequence:

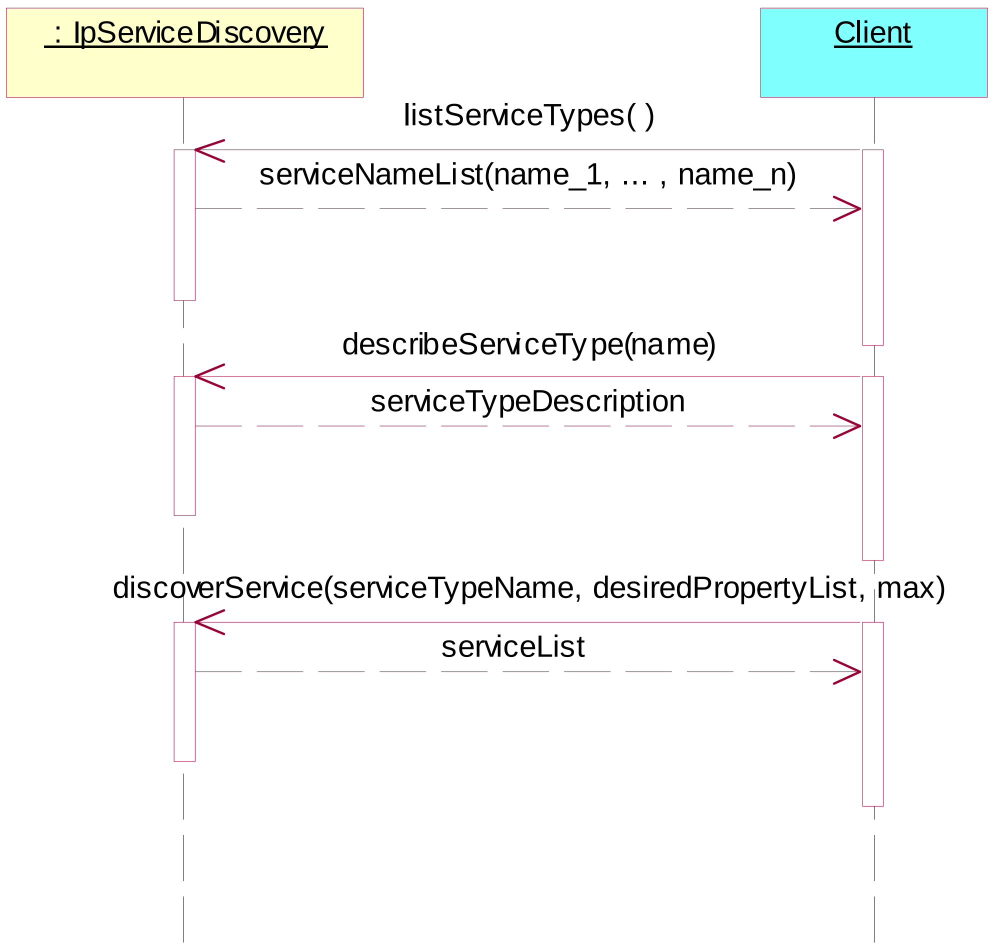
1. Triggered action: cause IUT to call **listServicesTypes ()** method on the tester's (Framework) **IpServiceDiscovery** interface
   Parameters: none

NOTE 1: This method need not be invoked by the application for this test to succeed.

2. Triggered action: cause IUT to call **describeServiceType ()** method on the tester's (Framework) **IpServiceDiscovery** interface
   Parameters: serviceTypeName from the list returned in 1.

NOTE 2: This method need not be invoked by the application for this test to succeed.

3. Triggered action: cause IUT to call **discoverService ()** method on the tester's (Framework) **IpServiceDiscovery** interface
   Parameters: serviceTypeName from the list returned in 1., valid desiredPropertyList, valid max


: IpServiceDiscovery
Client
listServiceTypes( )
serviceNameList(name_1, ... , name_n)
describeServiceType(name)
serviceTypeDescription
discoverService(serviceTypeName, desiredPropertyList, max)
serviceList

Test FW_FA_SD_A02

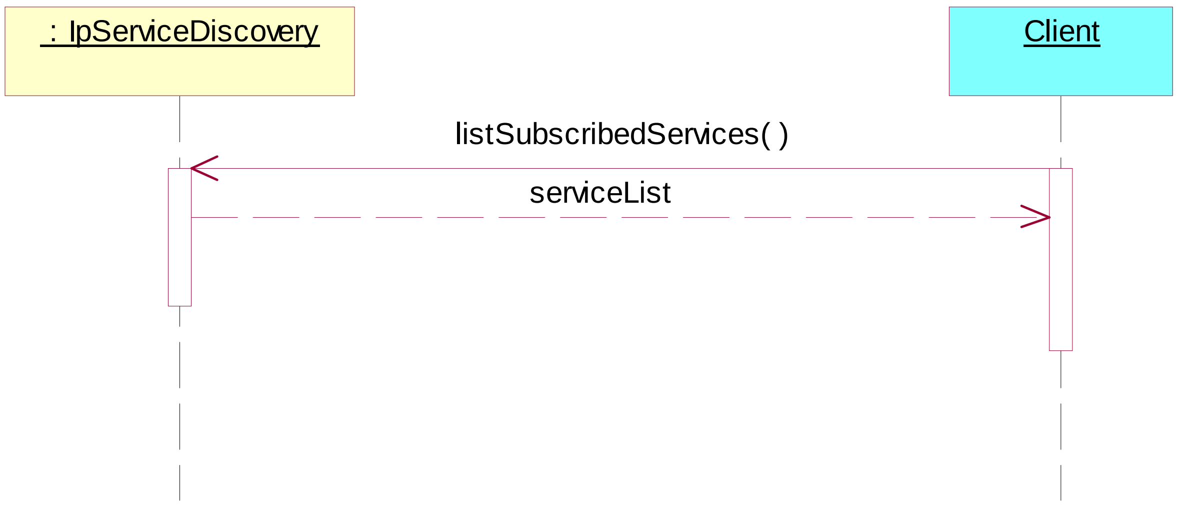
Summary: **IpServiceDiscovery,** listSubscribedService, successful.

Reference: ES 202 915-3 [1], clause 7.3.1.

Precondition: listSubscribedServices supported.

Test Sequence:

1. Triggered action: cause IUT to call **listSubscribedServices()** method on the tester's (Framework) **IpServiceDiscovery** interface
   Parameters: none



9.4.2 Service Agreement Management (SA)

Test FW_FA_SA_A01

Summary: **IpAppServiceAgreementManagement**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

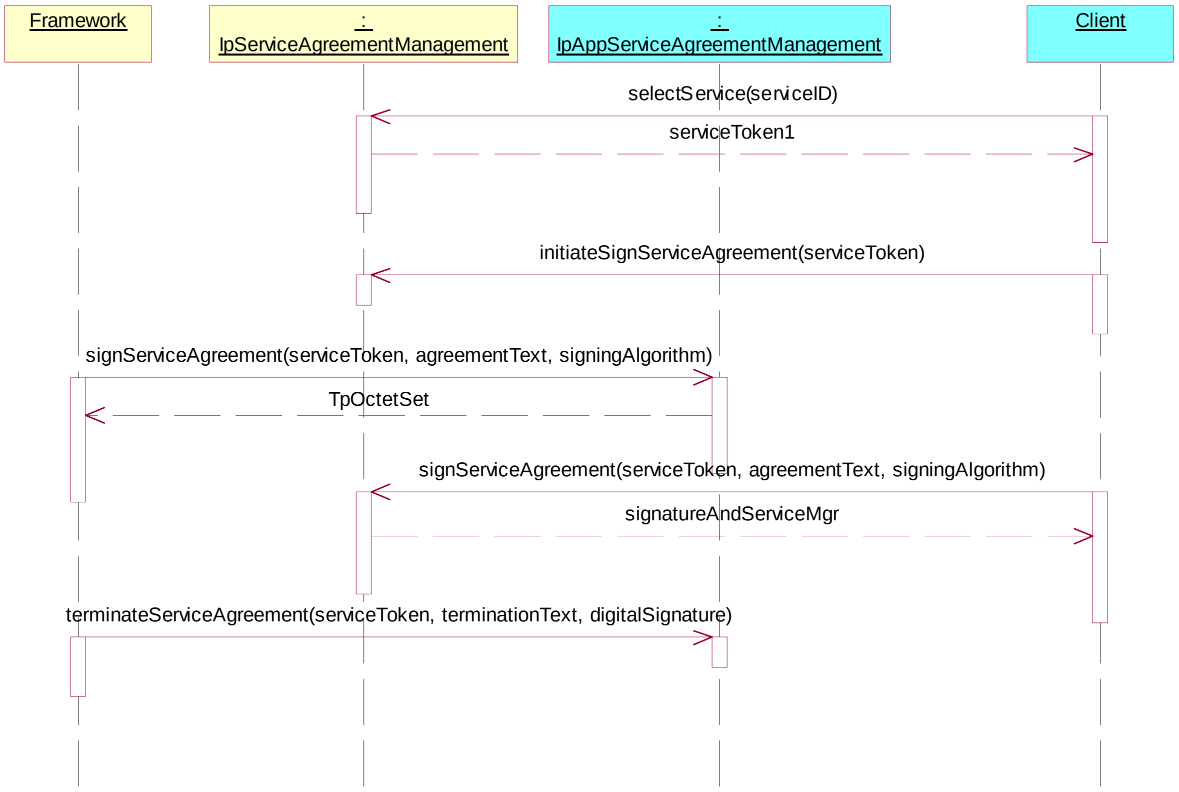
1. Triggered action: cause IUT to call **selectService()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceID

2. Triggered action: cause IUT to call **initiateSignServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1.

3. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpOctetSet is returned

4. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm

5. Method call **terminateServiceAgreement()**
   Parameters: serviceToken returned in 1., terminationText, digitalSignature
   Check: no exception is returned


Framework
: IpServiceAgreementManagement
: IpAppServiceAgreementManagement
Client
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
TpOctetSet
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signatureAndServiceMgr
terminateServiceAgreement(serviceToken, terminationText, digitalSignature)

Test FW_FA_SA_A02

Summary: **IpAppServiceAgreementManagement**, signServiceAgreement, P_INVALID_AGREEMENT_TEXT.

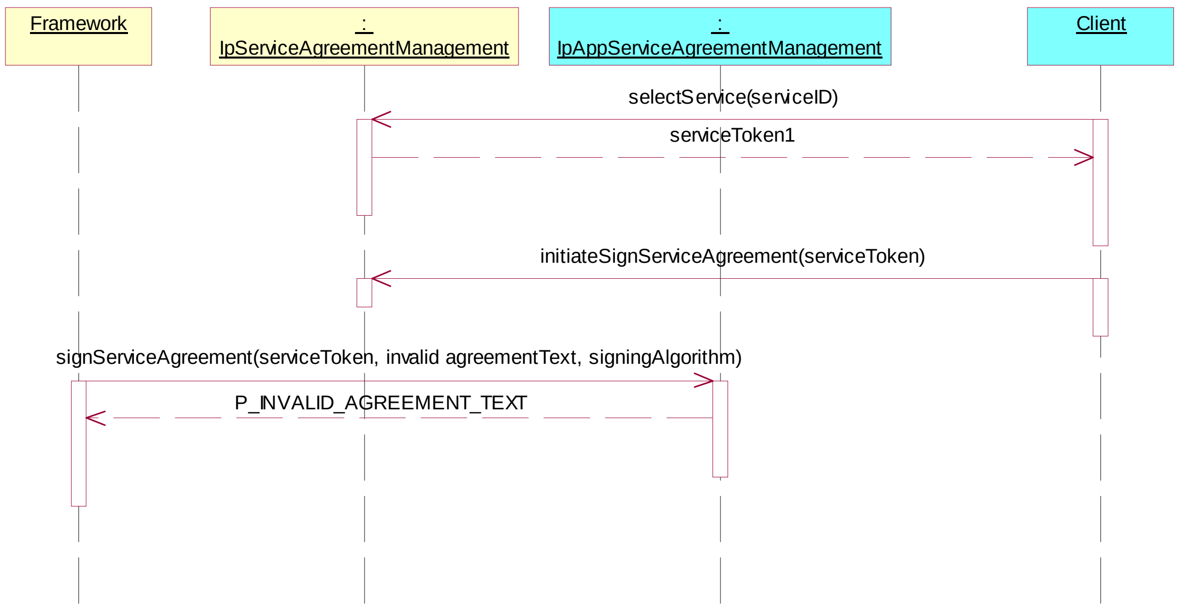
Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

1. Triggered action: cause IUT to call **selectService()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Triggered action: cause IUT to call **initiateSignServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1.
   Check: no exception is returned.

3. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., invalid agreememtText, signingAlgorithm
   Check: P_INVALID_AGREEMENT_TEXT is returned.


Framework
: IpServiceAgreementManagement
: IpAppServiceAgreementManagement
Client
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken)
signServiceAgreement(serviceToken, invalid agreementText, signingAlgorithm)
P_INVALID_AGREEMENT_TEXT

Test FW_FA_SA_A03

Summary: **IpAppServiceAgreementManagement**, signServiceAgreement, P_INVALID_SERVICE_TOKEN.

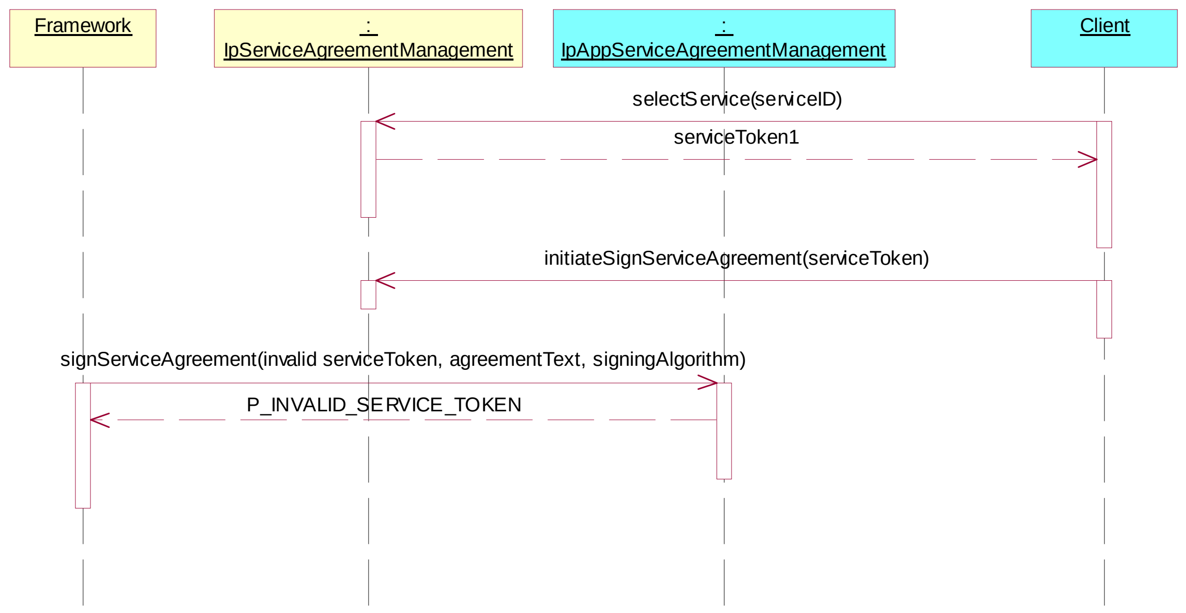
Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

1. Triggered action: cause IUT to call **selectService()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceID
   Check: valid value of TpServiceToken is returned

2. Triggered action: cause IUT to call **initiateSignServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1.
   Check: no exception is returned.

3. Method call **signServiceAgreement**()
   Parameters: invalid serviceToken, agreememtText, signingAlgorithm
   Check: P_INVALID_SERVICE_TOKEN is returned.


Framework
: IpServiceAgreementManagement
: IpAppServiceAgreementManagement
Client
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken)
signServiceAgreement(invalid serviceToken, agreementText, signingAlgorithm)
P_INVALID_SERVICE_TOKEN

Test FW_FA_SA_A04

Summary: **IpAppServiceAgreementManagement**, signServiceAgreement**,** P_INVALID_SIGNING_ALGORITHM.

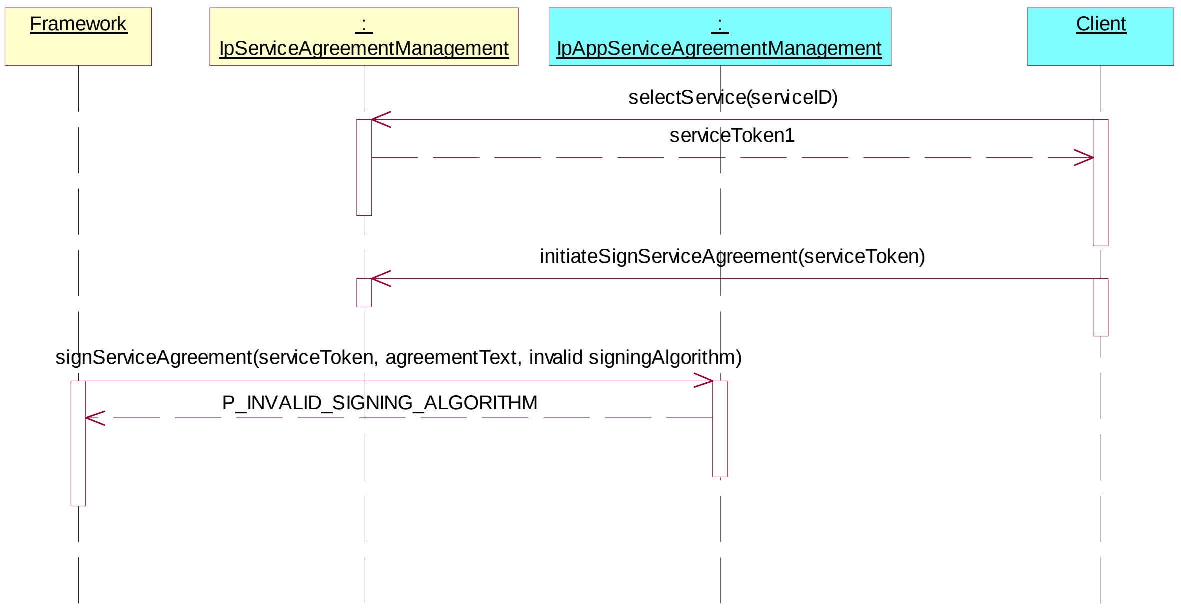
Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

1. Triggered action: cause IUT to call **selectService()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceID

2. Triggered action: cause IUT to call **initiateSignServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1.

3. Method call **signServiceAgreement**()
   Parameters: serviceToken, agreememtText, invalid signingAlgorithm
   Check: P_INVALID_SIGNING_ALGORITHM is returned.


Framework
: IpServiceAgreementManagement
: IpAppServiceAgreementManagement
Client
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken)
signServiceAgreement(serviceToken, agreementText, invalid signingAlgorithm)
P_INVALID_SIGNING_ALGORITHM

Test FW_FA_SA_A05

Summary: **IpAppServiceAgreementManagement**, terminateServiceAgreement**,** P_INVALID_SIGNATURE.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

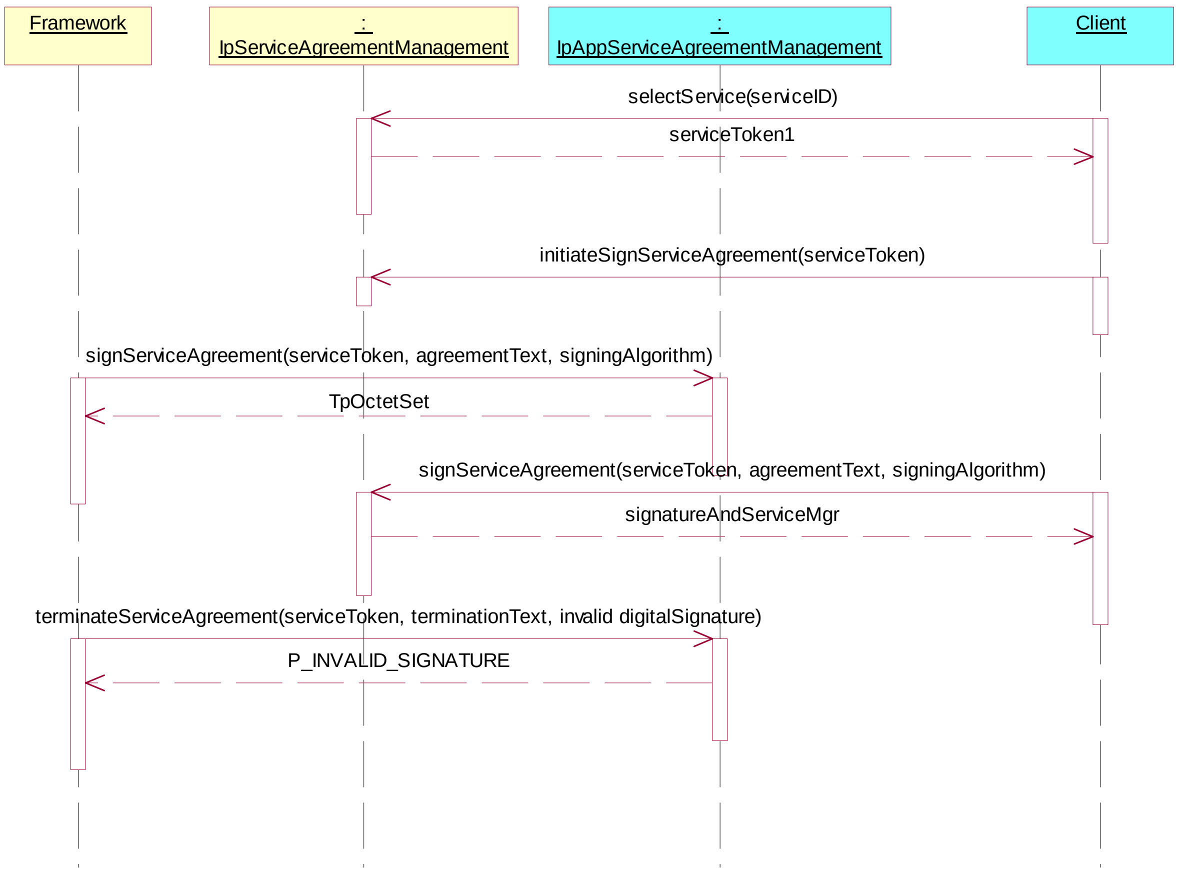
1. Triggered action: cause IUT to call **selectService()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceID

2. Triggered action: cause IUT to call **initiateSignServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1.

3. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpOctetSet is returned

4. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm

5. Method call **terminateServiceAgreement()**
   Parameters: serviceToken returned in 1., terminationText, invalid digitalSignature
   Check: P_INVALID_SIGNATURE is returned.


Framework
: IpServiceAgreementManagement
: IpAppServiceAgreementManagement
Client
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
TpOctetSet
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signatureAndServiceMgr
terminateServiceAgreement(serviceToken, terminationText, invalid digitalSignature)
P_INVALID_SIGNATURE

Test FW_FA_SA_A06

Summary: **IpAppServiceAgreementManagement**, terminateServiceAgreement**,** P_INVALID_SERVICE_TOKEN.

Reference: ES 202 915-3 [1], clause 7.3.2.

Test Sequence:

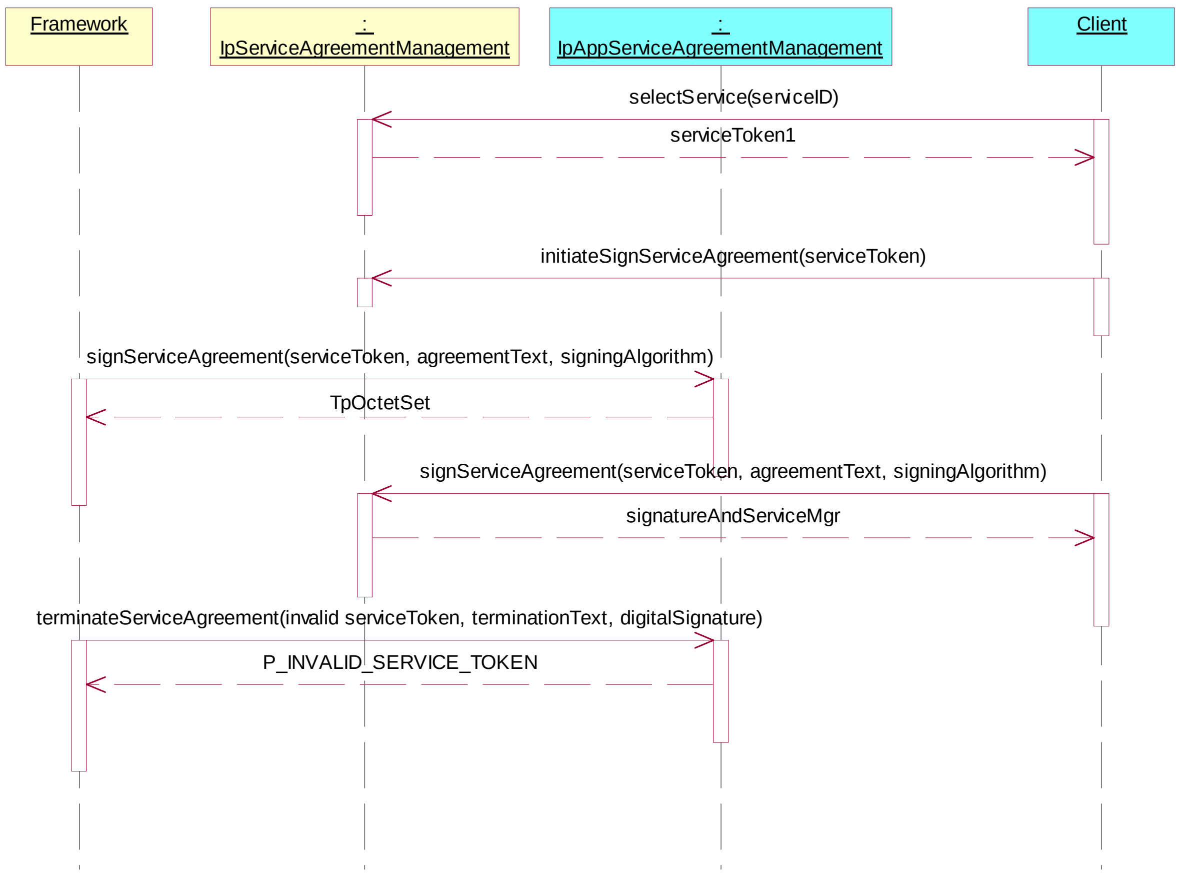
1. Triggered action: cause IUT to call **selectService()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceID

2. Triggered action: cause IUT to call **initiateSignServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1.

3. Method call **signServiceAgreement()**
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm
   Check: valid value of TpOctetSet is returned

4. Triggered action: cause IUT to call **signServiceAgreement ()** method on the tester's (Framework) **IpServiceAgreementManagement** interface.
   Parameters: serviceToken returned in 1., agreememtText, signingAlgorithm

5. Method call **terminateServiceAgreement()**
   Parameters: invalid serviceToken, terminationText, digitalSignature
   Check: P_INVALID_SERVICE_TOKEN is returned.


Framework
: IpServiceAgreementManagement
: IpAppServiceAgreementManagement
Client
selectService(serviceID)
serviceToken1
initiateSignServiceAgreement(serviceToken)
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
TpOctetSet
signServiceAgreement(serviceToken, agreementText, signingAlgorithm)
signatureAndServiceMgr
terminateServiceAgreement(invalid serviceToken, terminationText, digitalSignature)
P_INVALID_SERVICE_TOKEN

9.4.3 Integrity Management (IM)

Test FW_FA_IM_A01

Summary: **IpAppHeartBeatMgmt**, all methods, successful.

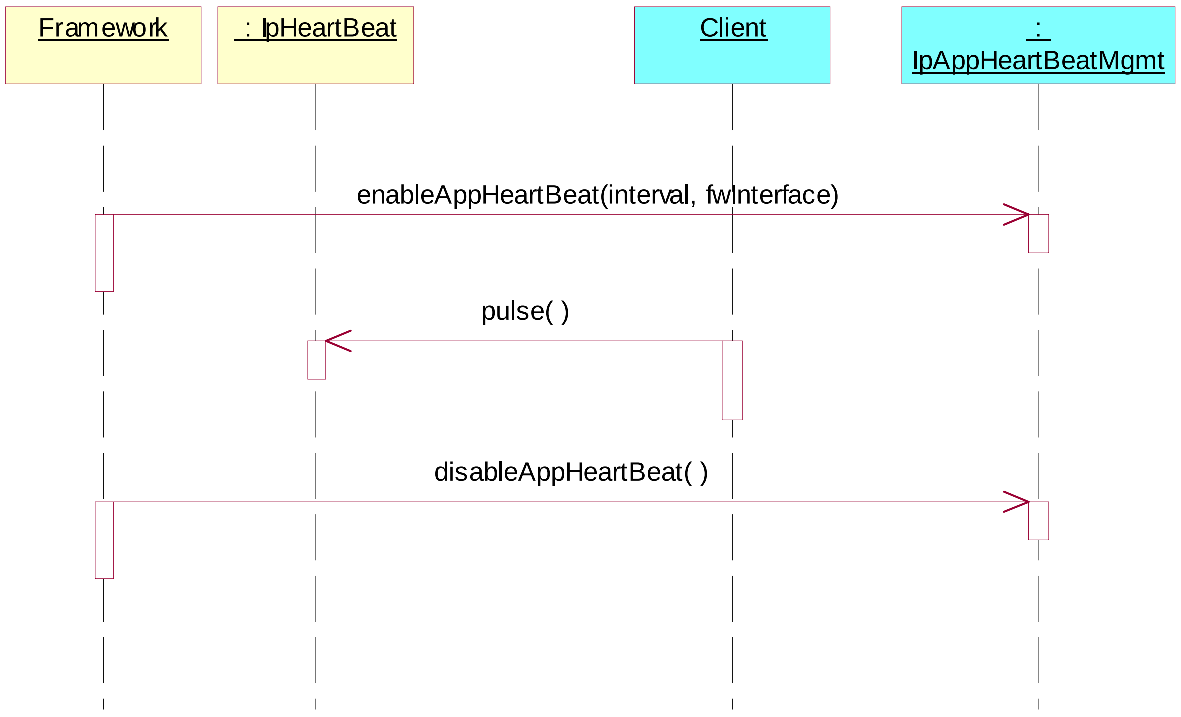
Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **enableAppHeartBeat()**
   Parameters: interval, fwInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to regularly call **pulse**() method on the tester's (Framework) **IpHeartBeat** interface.
   Parameters: none
   Check: Check that the **pulse**() method is invoked at the requested interval.

3. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.



Test FW_FA_IM_A02

Summary: **IpAppHeartBeatMgmt**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

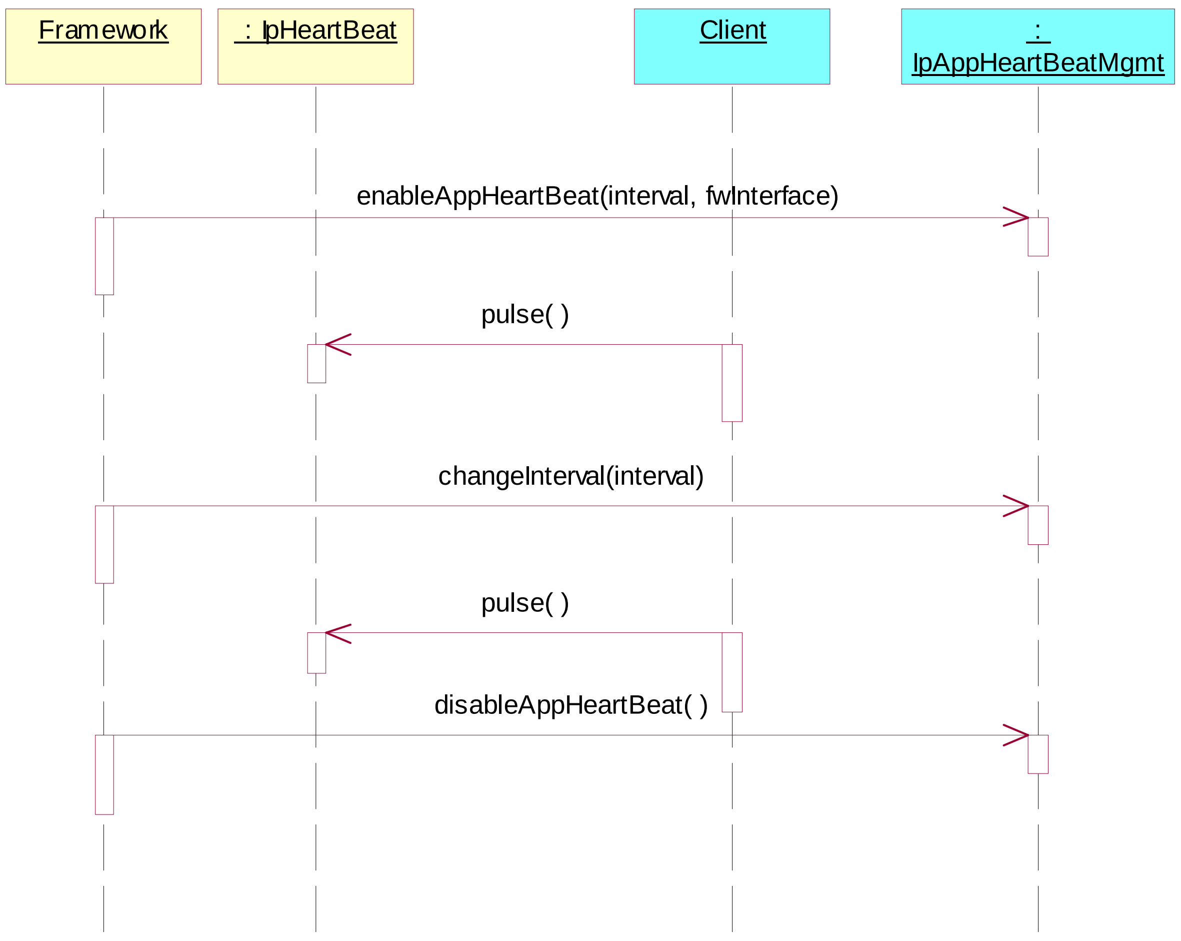
1. Method call **enableAppHeartBeat()**
   Parameters: interval, fwInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to call **pulse**() method regularly on the tester's (Application) **IpHeartBeat** interface.
   Parameters: none
   Check: Check that the **pulse**() method is invoked at the requested interval.

3. Method call **changeInterval()**
   Parameters: interval
   Check: no exception is returned.

4. Triggered action: cause IUT to call **pulse**() method regularly on the tester's (Application) **IpHeartBeat** interface.
   Parameters: none
   Check: the **pulse**() method is invoked at the new requested interval.

5. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.



Test FW_FA_IM_A03

Summary: **IpAppHeartBeat**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

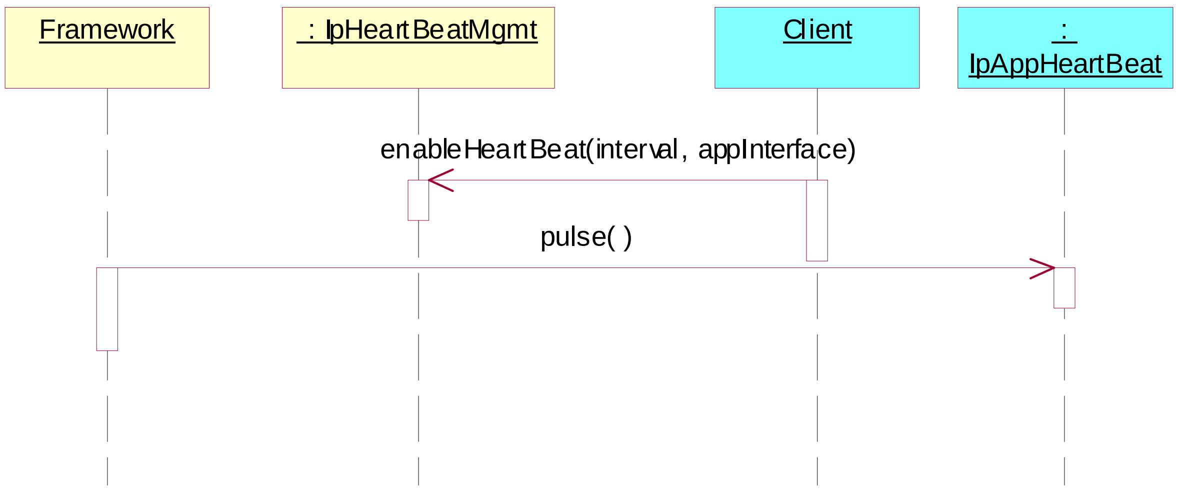
Precondition: IpHeartBeat is supported.

Preamble: The calling application must have a callback interface and a reference to this interface.

Test Sequence:

1. Triggered action: cause IUT to call **enableHeartBeat()** method on the tester's (Application) **IpHeartBeatMgmt** interface.
   Parameters: interval, appInterface

2. Method call **pulse()**
   Parameters: none
   Check: no exception



Test FW_FA_IM_A04

Summary: **IpAppFaultManager,** activityTestRes, successful.

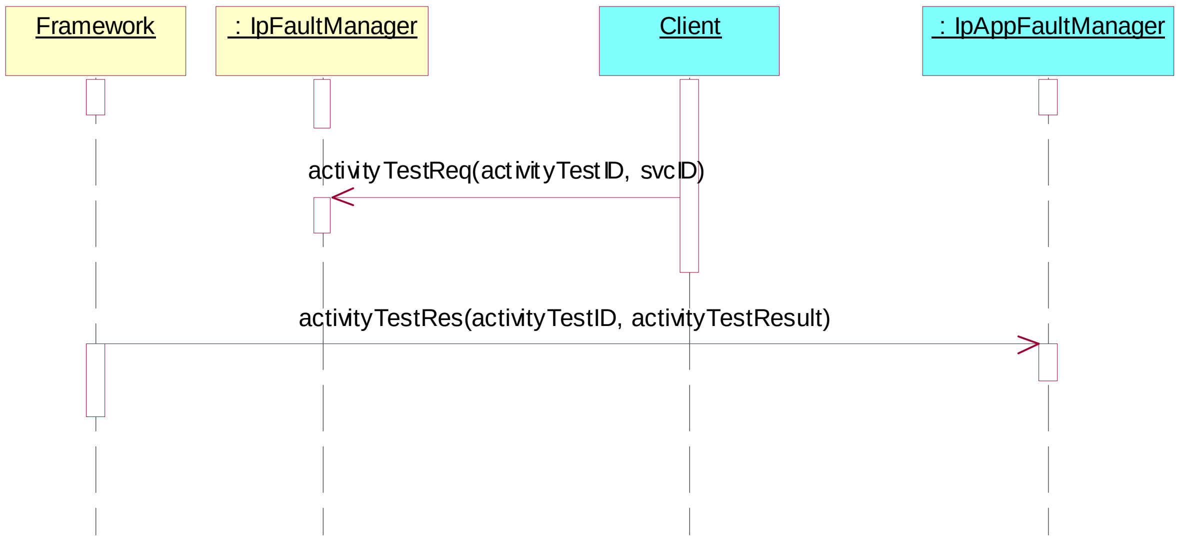
Reference: ES 202 915-3 [1], clause 7.3.3.

Preamble: The IUT (application) must have subscribed to at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **activityTestReq()** method on the tester's (Framework) **IpFaultManager** interface.
   Parameters: activityTestID, svcID

2. Method call **activityTestRes ()**
   Parameters: activityTestID, activityTestResult
   Check: no exception is returned


Framework
: IpFaultManager
Client
: IpAppFaultManager
activityTestReq(activityTestID, svcID)
activityTestRes(activityTestID, activityTestResult)

Test FW_FA_IM_A05

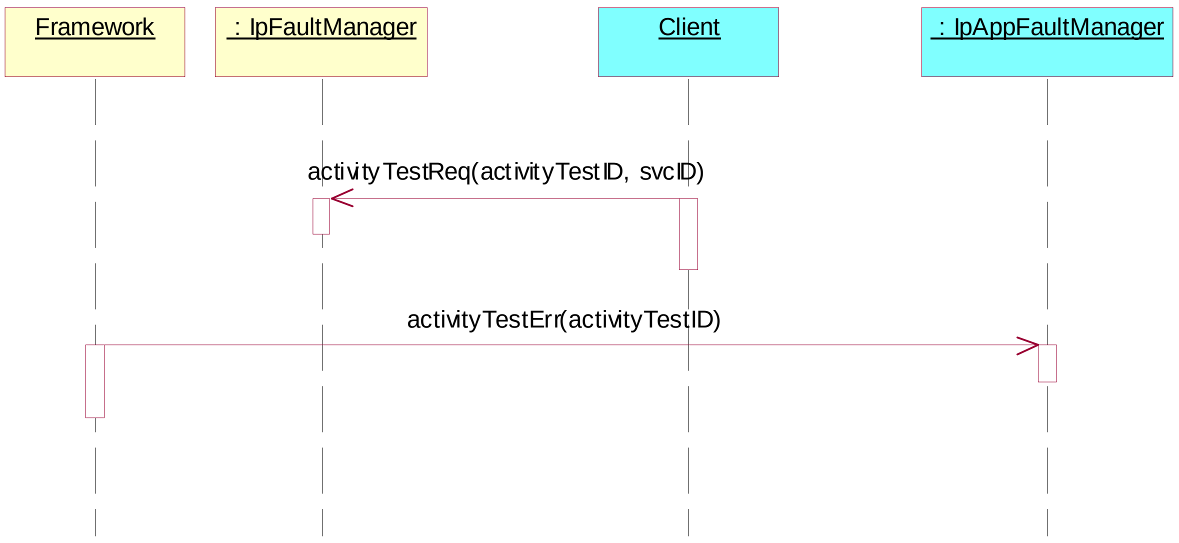
Summary: **IpAppFaultManager,** activityTestErr, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **activityTestReq()** method on the tester's (Framework) **IpFaultManager** interface.
   Parameters: activityTestID, svcID

2. Method call **activityTestErr()**
   Parameters: activityTestID
   Check: no exception is returned


Framework
: IpFaultManager
Client
: IpAppFaultManager
activityTestReq(activityTestID, svcID)
activityTestErr(activityTestID)

Test FW_FA_IM_A06

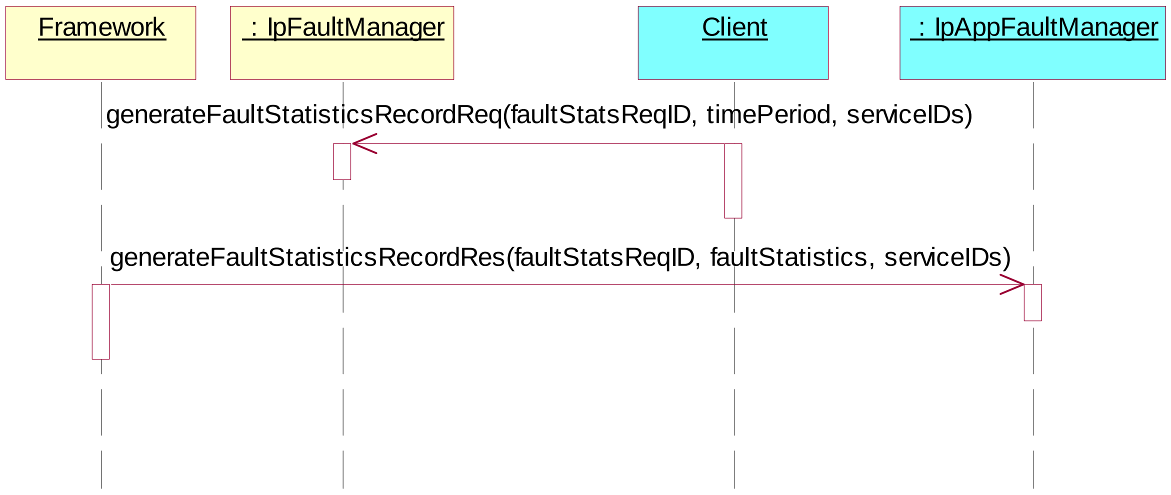
Summary: **IpAppFaultManager,** generateFaultStatisticsRecordRes, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **generateFaultStatisticsRecordReq ()** method on the tester's (Framework) **IpFaultManager** interface.
   Parameters: faultStatsReqID, timePeriod, serviceIDs

2. Method call **generateFaultStatisticsRecordRes ()**
   Parameters: faultStatsReqID, faultStatistics, ServiceIDs
   Check: no exception is returned


Framework
: IpFaultManager
Client
: IpAppFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, serviceIDs)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics, serviceIDs)

Test FW_FA_IM_A07

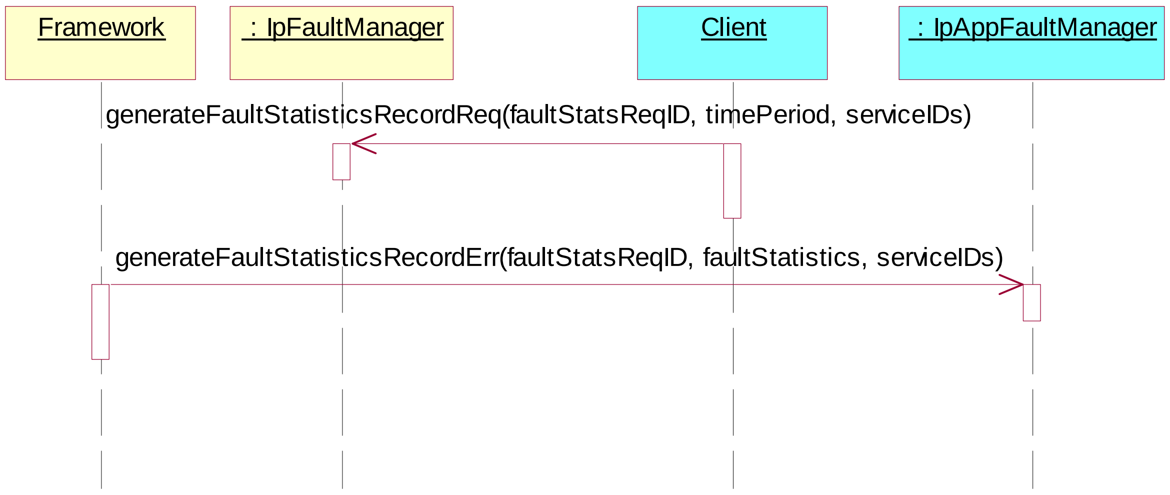
Summary: **IpAppFaultManager,** generateFaultStatisticsRecordErr, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **generateFaultStatisticsRecordReq ()** method on the tester's (Framework) **IpFaultManager** interface.
   Parameters: faultStatsReqID, timePeriod, serviceIDs

2. Method call **generateFaultStatisticsRecordErr ()**
   Parameters: faultStatsReqID, faultStatisticsError, ServiceIDs
   Check: no exception is returned


Framework
: IpFaultManager
Client
: IpAppFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, serviceIDs)
generateFaultStatisticsRecordErr(faultStatsReqID, faultStatistics, serviceIDs)

Test FW_FA_IM_A08

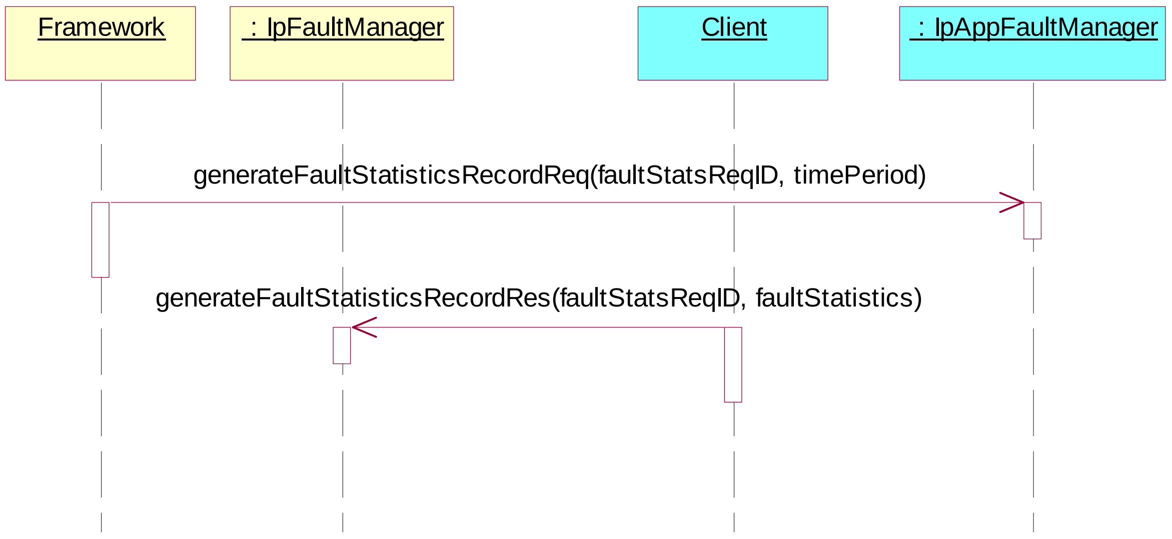
Summary: **IpAppFaultManager,** generateFaultStatisticsRecordReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod
   Check: no exception is returned

2. Triggered action: cause IUT to call **generateFaultStatisticsRecordRes ()** method on the tester's (Framework) **IpFaultManager** interface.
   Parameters: faultStatsReqID, faultStatistics


Framework
: IpFaultManager
Client
: IpAppFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics)

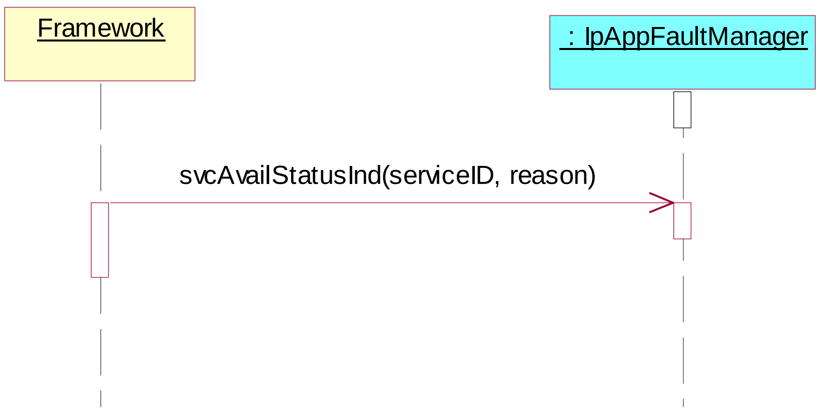
Test FW_FA_IM_A09

Summary: **IpAppFaultManager,** svcAvailStatusInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **svcAvailStatusInd ()**
   Parameters: serviceID, reason
   Check: no exception is returned



Test FW_FA_IM_A10

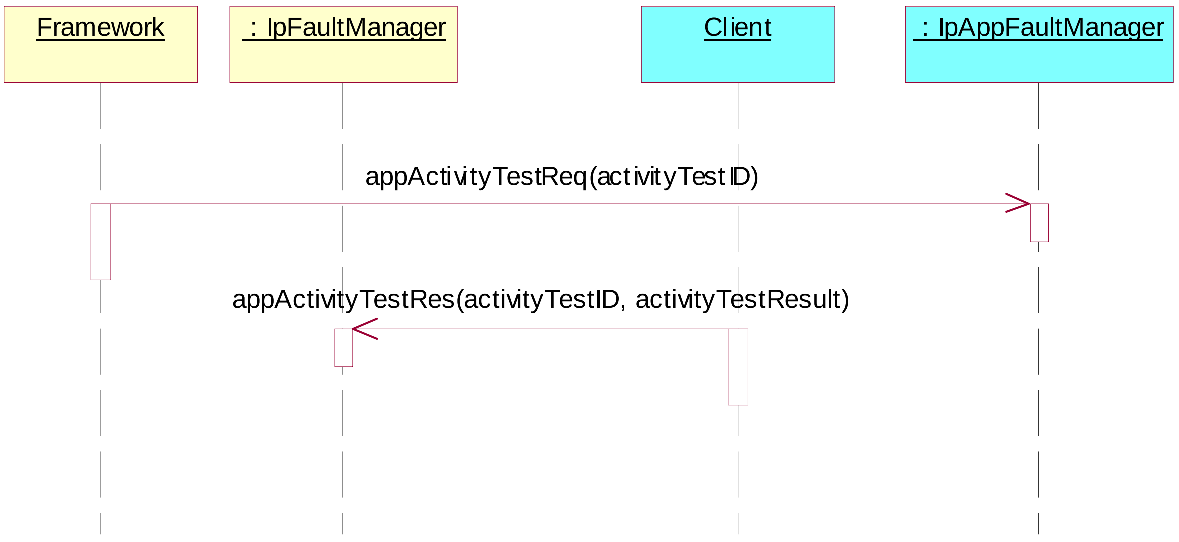
Summary: **IpAppFaultManager,** appActivityTestReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **appActivityTestReq ()**
   Parameters: activityTestID
   Check: no exception is returned

2. Triggered action: cause IUT to call **appActivityTestRes ()** method on the tester's (Framework) **IpFaultManager** interface.
   Parameters: activityTestID, activityTestResult


Framework
: IpFaultManager
Client
: IpAppFaultManager
appActivityTestReq(activityTestID)
appActivityTestRes(activityTestID, activityTestResult)

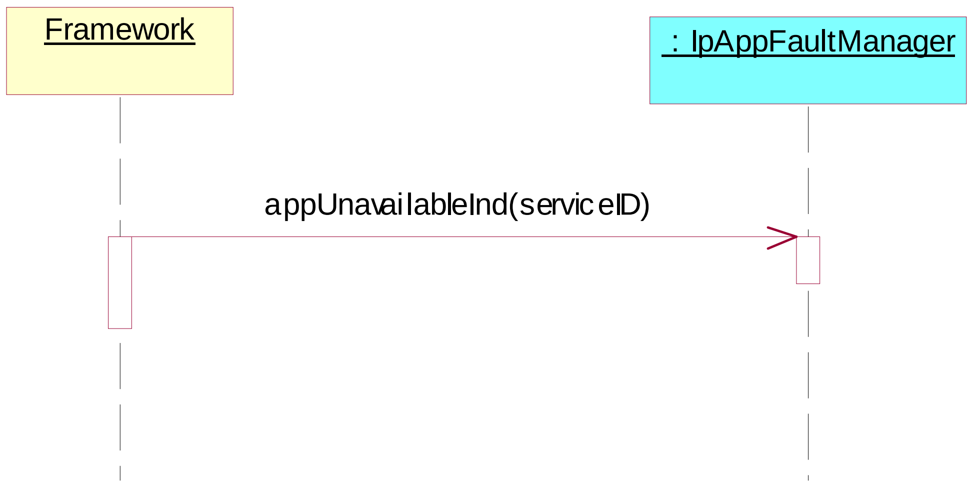
Test FW_FA_IM_A11

Summary: **IpAppFaultManager,** appUnavailableInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **appUnavailableInd()**
   Parameters: serviceID
   Check: no exception is returned



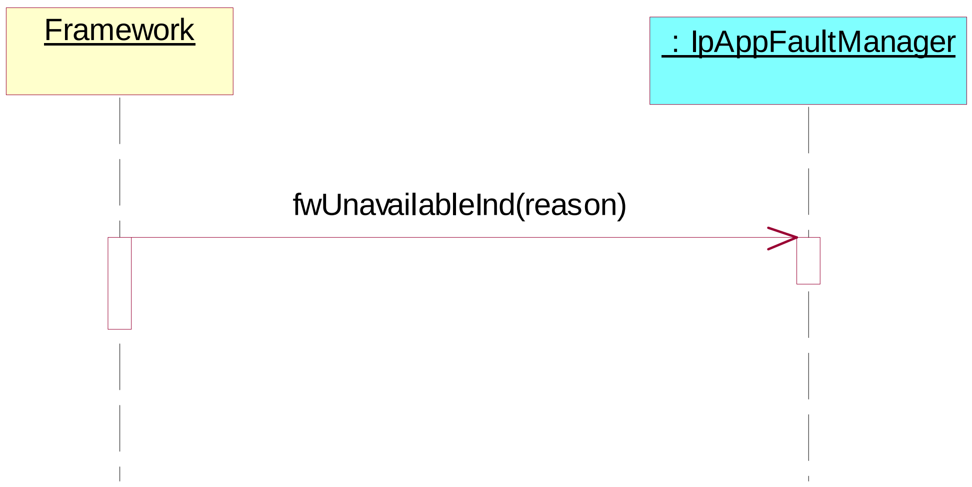
Test FW_FA_IM_A12

Summary: **IpAppFaultManager,** fwUnavailableInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **fwUnavailableInd()**
   Parameters: reason
   Check: no exception is returned



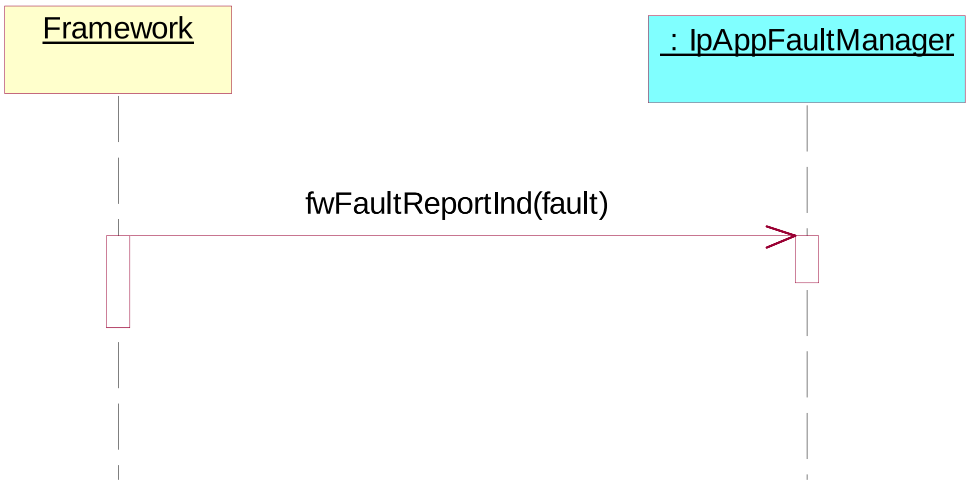
Test FW_FA_IM_A13

Summary: **IpAppFaultManager,** fwFaultReportInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **fwFaultReportInd()**
   Parameters: fault
   Check: no exception is returned, client application no longer use the Framework.



Test FW_FA_IM_A14

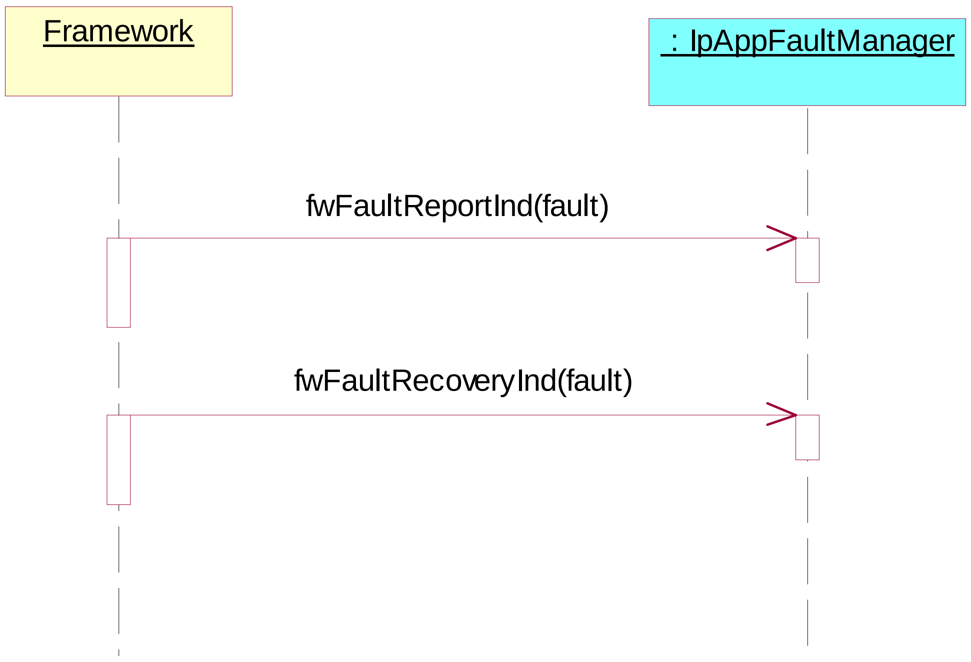
Summary: **IpAppFaultManager,** fwFaultReportInd and fwFaultRecoveryInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **fwFaultReportInd()**
   Parameters: fault
   Check: no exception is returned, client application no longer use the Framework.

2. Method call **fwFaultRecoveryInd()**
   Parameters: fault
   Check: no exception is returned, client application resume using the Framework.



Test FW_FA_IM_A15

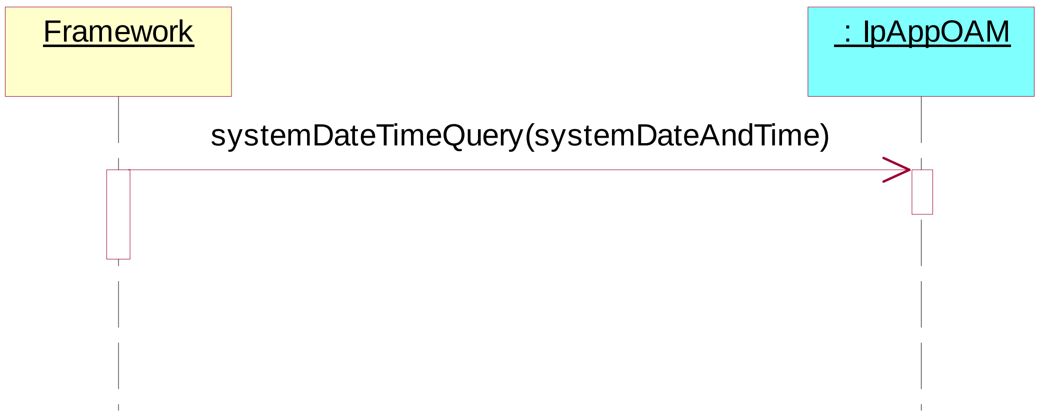
Summary: **IpAppOAM**, systemDateTimeQuery, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpAppFwOAM supported.

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: clientDateAndTime
   Check: valid value of TpDateAndTime is returned



Test FW_FA_IM_A16

Summary: **IpAppLoadManager** All methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: none
   Check: no exception is returned

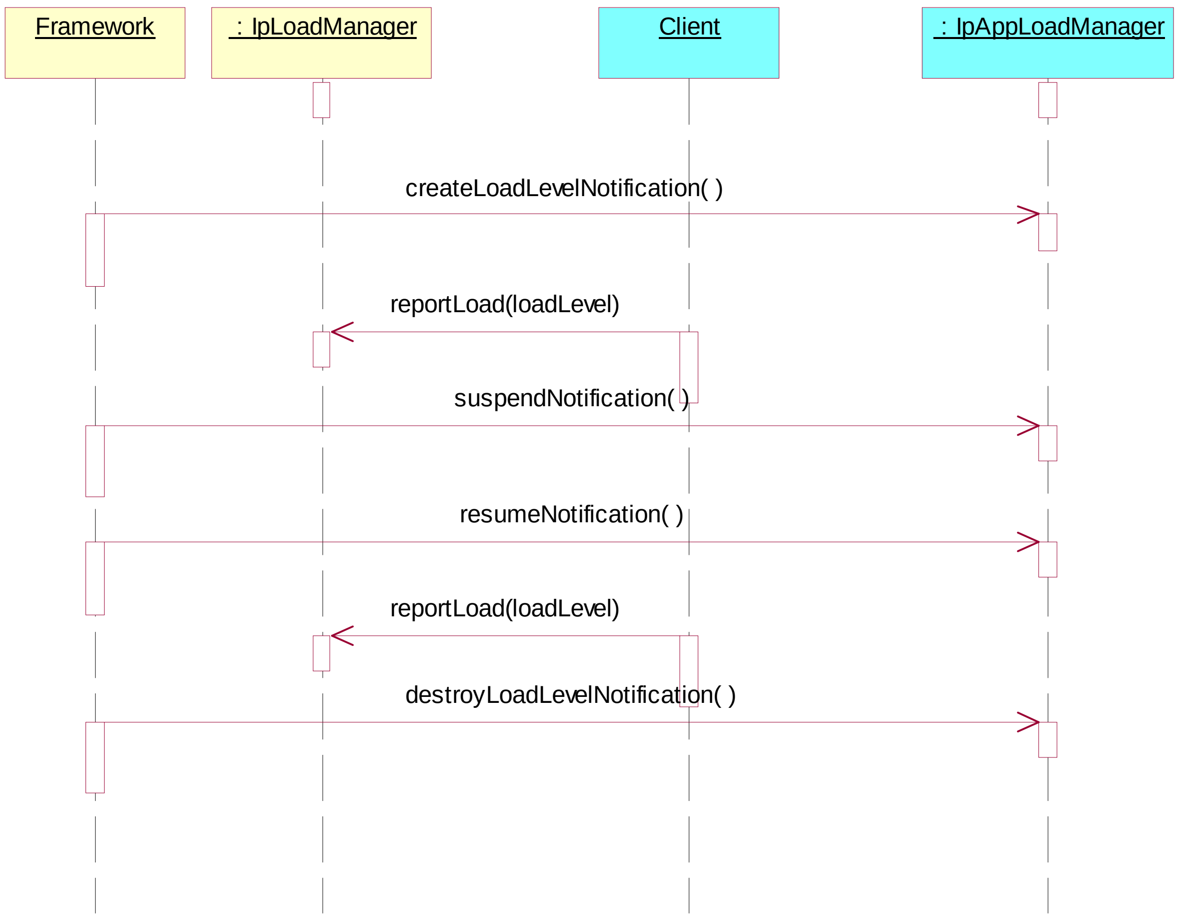
2. Triggered action: cause IUT to call **reportLoad()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: loadLevel

3. Method call **suspendNotification()**
   Parameters: none
   Check: no exception is returned, no load level notifications received until resumeNotification() is called.

4. Method call **resumeNotification()**
   Parameters: none
   Check: no exception is returned

5. Triggered action: cause IUT to call **reportLoad()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: loadLevel

6. Method call **destroyLoadLevelNotification()**
   Parameters: none
   Check: no exception is returned



Test FW_FA_IM_A17

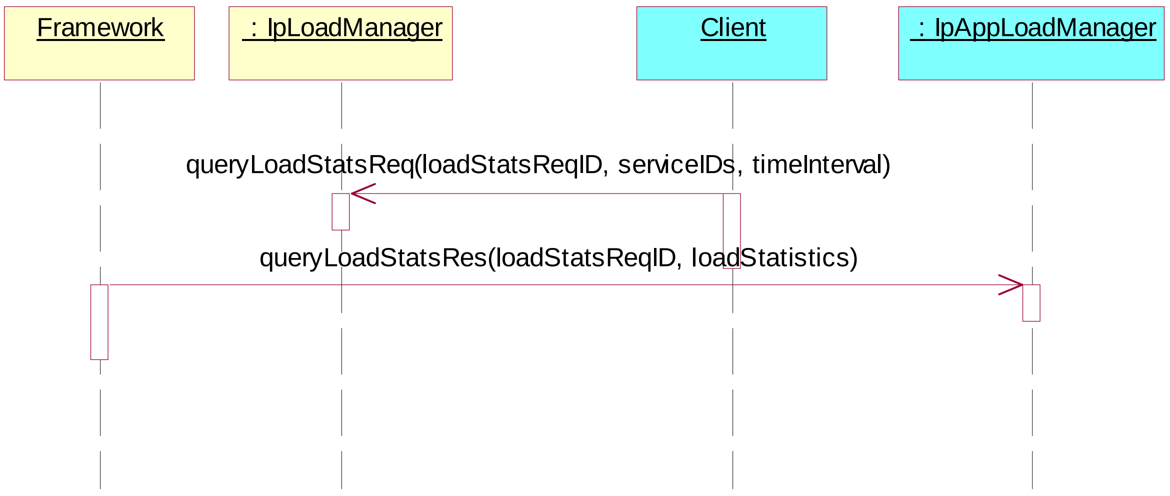
Summary: **IpAppLoadManager,** queryLoadStatsReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **queryLoadStatsReq()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: loadStatsReqID, querySubject, timeInterval

2. Method call **queryLoadStatsRes ()**
   Parameters: loadStatsReqID, loadStatistics
   Check: no exception is returned


Framework
: IpLoadManager
Client
: IpAppLoadManager
queryLoadStatsReq(loadStatsReqID, serviceIDs, timeInterval)
queryLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FA_IM_A18

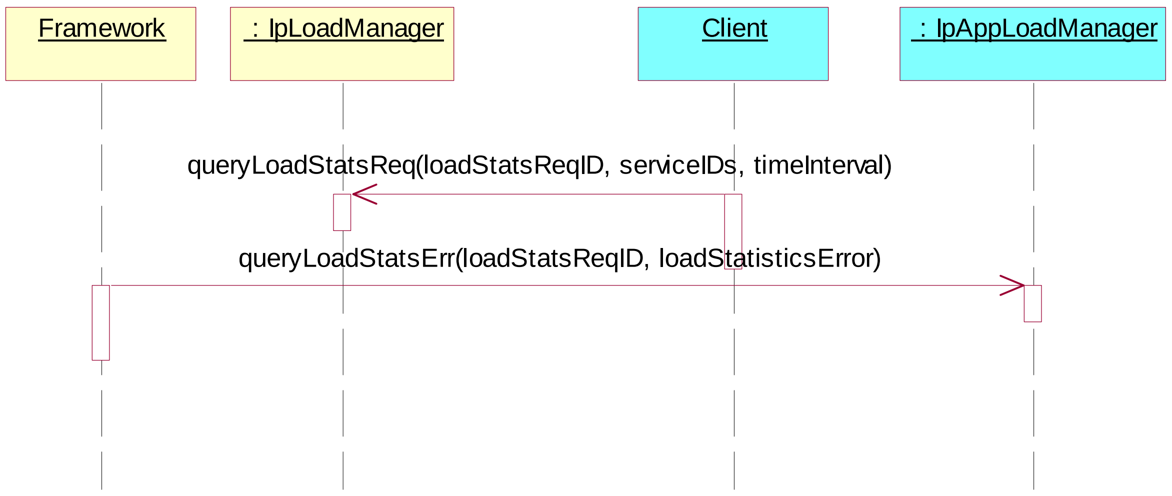
Summary: **IpAppLoadManager,** queryStatsLoadReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **queryLoadStatsReq()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: loadStatsReqID, querySubject, timeInterval

2. Method call **queryLoadStatsErr()**
   Parameters: loadStatsReqID, loadStatisticsError
   Check: no exception is returned


Framework
: IpLoadManager
Client
: IpAppLoadManager
queryLoadStatsReq(loadStatsReqID, serviceIDs, timeInterval)
queryLoadStatsErr(loadStatsReqID, loadStatisticsError)

Test FW_FA_IM_A19

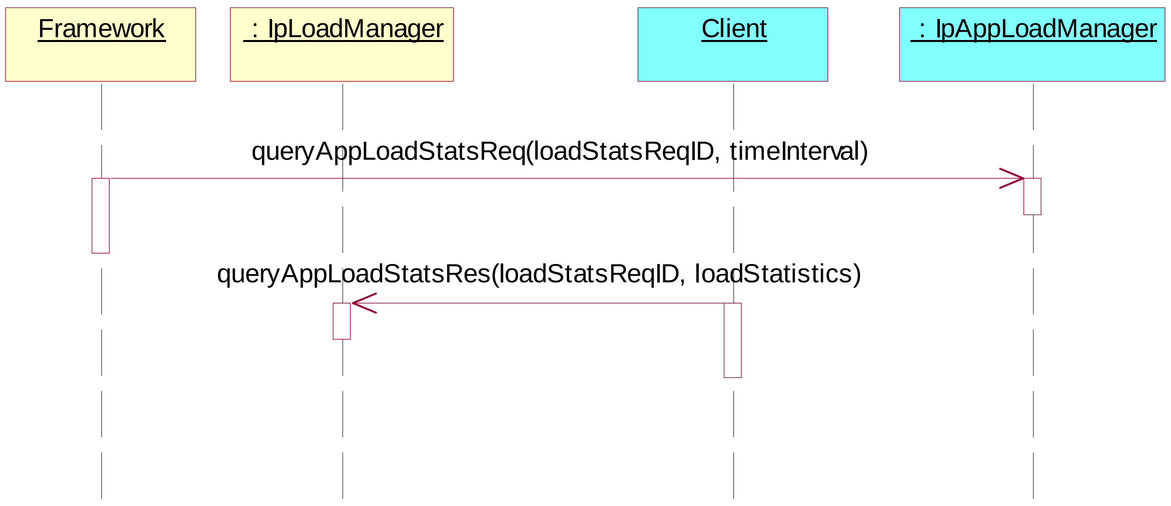
Summary: **IpAppLoadManager,** queryLoadStatsReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Method call **queryAppLoadStatsReq()**
   Parameters: loadStatsReqID, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **queryAppLoadStatsRes()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: loadStatsReqID, loadStatistics


Framework
: IpLoadManager
Client
: IpAppLoadManager
queryAppLoadStatsReq(loadStatsReqID, timeInterval)
queryAppLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FA_IM_A20

Summary: **IpAppLoadManager,** LoadLevelNotification, successful.

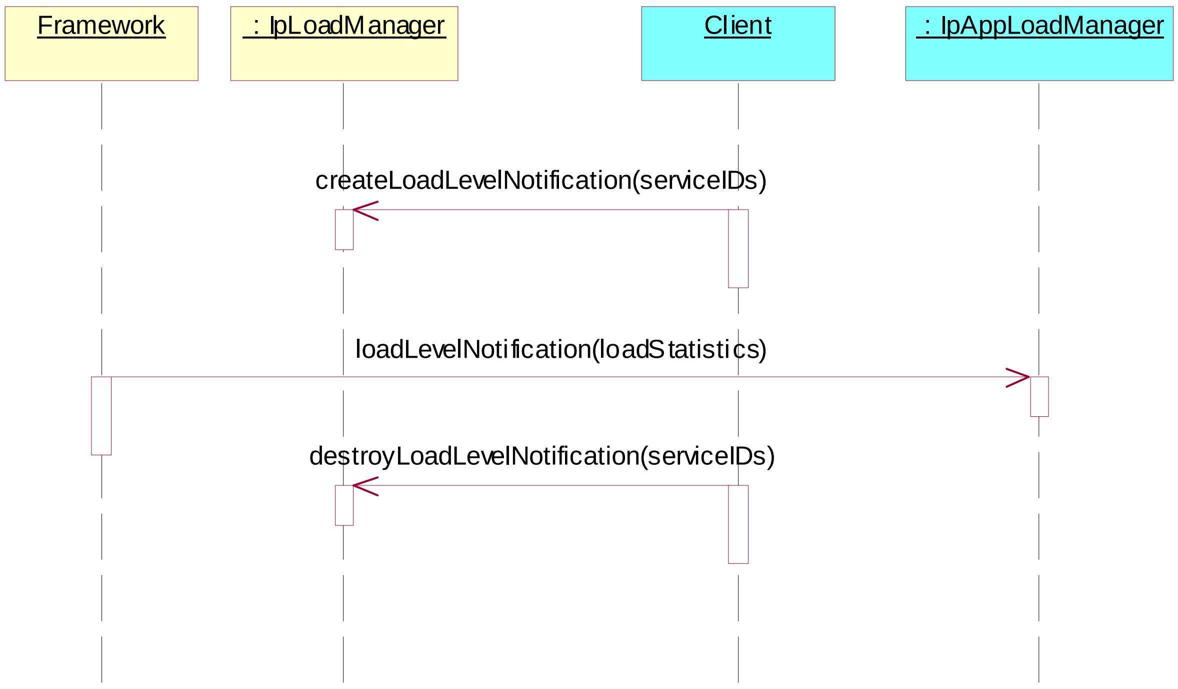
Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **createLoadLevelNotification()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: serviceIDs

2. Method call **LoadLevelNotification ()**
   Parameters: loadStatistics
   Check: no exception is returned

3. Triggered action: cause IUT to call **destroyLoadLevelNotification()** method on the tester's (Service) **IpLoadManager** interface.
   Parameters: serviceIDs


Framework
: IpLoadManager
Client
: IpAppLoadManager
createLoadLevelNotification(serviceIDs)
loadLevelNotification(loadStatistics)
destroyLoadLevelNotification(serviceIDs)

9.4.4 Event Notification (EN)

Test FW_FA_EN_A01

Summary: **IpAppEventNotification**, reportNotification method, successful.

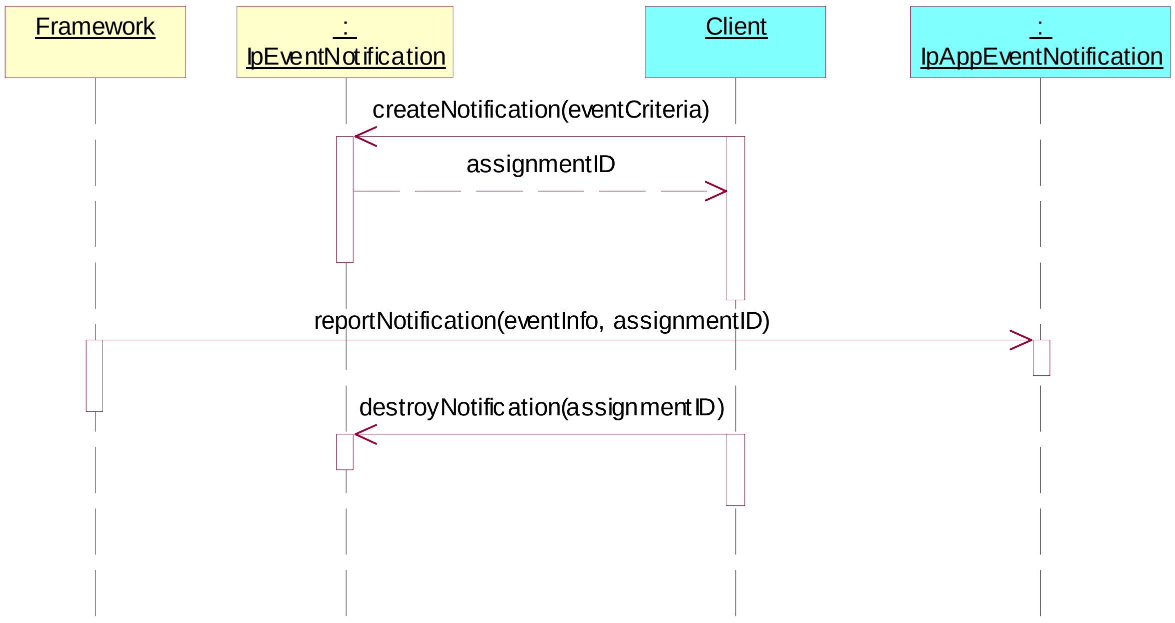
Reference: ES 202 915-3 [1], clause 7.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **createNotification ()** method on the tester's (Framework) **IpEventNotification** interface.
   Parameters: eventCriteria

2. Method call **reportNotification()**
   Parameters: eventInfo, assignmentID
   Check: no exception is returned

3. Triggered action: cause IUT to call **destroyNotification ()** method on the tester's (Framework) **IpEventNotification** interface.
   Parameters: assignmentID given in 1.


Framework
: IpEventNotification
Client
: IpAppEventNotification
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
destroyNotification(assignmentID)

Test FW_FA_EN_A02

Summary: **IpEventNotification**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.4.

Precondition: IpEventNotification supported.

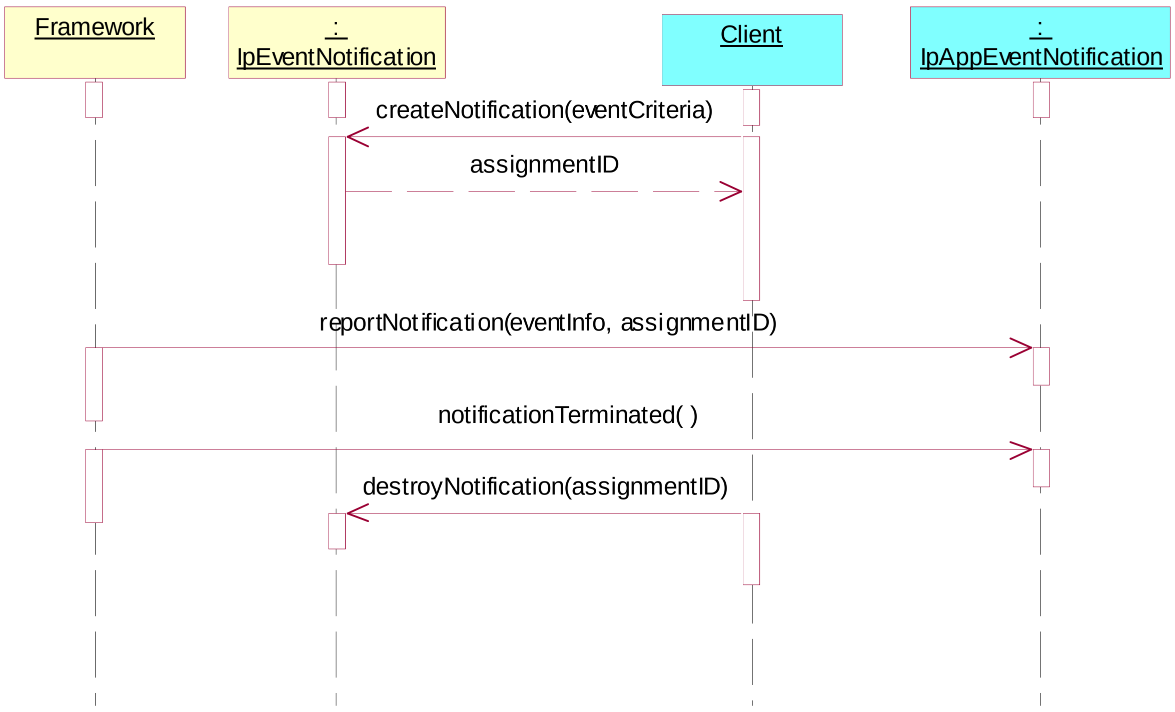
Test Sequence:

1. Triggered action: cause IUT to call **createNotification ()** method on the tester's (Framework) **IpEventNotification** interface.
   Parameters: eventCriteria

2. Method call **reportNotification()**
   Parameters: eventInfo, assignmentID
   Check: no exception is returned

3. Method call **notificationTerminated()**
   Parameters: none.
   Check: no exception is returned

4. Triggered action: cause IUT to call **destroyNotification ()** method on the tester's (Framework) **IpEventNotification** interface.
   Parameters: assignmentID given in 1.


Framework
: IpEventNotification
Client
: IpAppEventNotification
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
notificationTerminated( )
destroyNotification(assignmentID)

10 Test Suite Structure (TSS) for Service

Framework (FW)

- Framework Access Session (AS)
    - Trust and Security Management (TSM)
- Framework To Application (FTA)
    - Service discovery
    - Service agreement management
    - Integrity management
    - Event notification
- Framework To Service (FTS)
    - Service registration
    - Service instance lifecycle management
    - Service discovery
    - Integrity management
    - Event notification
- Framework To Enterprise operator (FTE)
    - Service subscription

11 Test Purposes (TP) for Service

For each test requirement a TP is defined.

11.1 TP naming convention

TPs are numbered, starting at 01, within each group. Groups are organized according to the TSS. Additional references are added to identify the actual test suite (see table 4).

Table 4: TP identifier naming convention scheme

Identifier: <suite_id>_<group>_<nnn>		
<suite_id>	= IUT name:	"FW" for **F**rame**W**ork SCF
<group>	= group number:	two character field representing the group reference according to TSS
<nn>	= sequential number:	(01-99)

11.2 Source of TP definition

The TPs are based on ES 202 915-3 [1].

11.3 Test strategy

As the base standard ES 202 915-3 [1] contains no explicit requirements for testing, the TPs were generated as a result of an analysis of the base standard and the ICS specification ES 202 363 [2].

The TPs are only based on conformance requirements related to the externally observable behaviour of the IUT and are limited to conceivable situations to which a real implementation is likely to be faced (see ETS 300 406 [5]).

11.4 TPs for the Framework Access Session API

All ICS items referred to in this clause are as specified in ES 202 363 [2] unless indicated otherwise by another numbered reference.

All parameters specified in method calls are valid unless specified.

The procedures to trigger the SCF to call methods in the framework are dependant on the SUT and are out of the scope of the present document. Those method calls are preceded by the words "Triggered action".

11.4.1 Service Registration (SR)

Test FW_FS_SR_S01

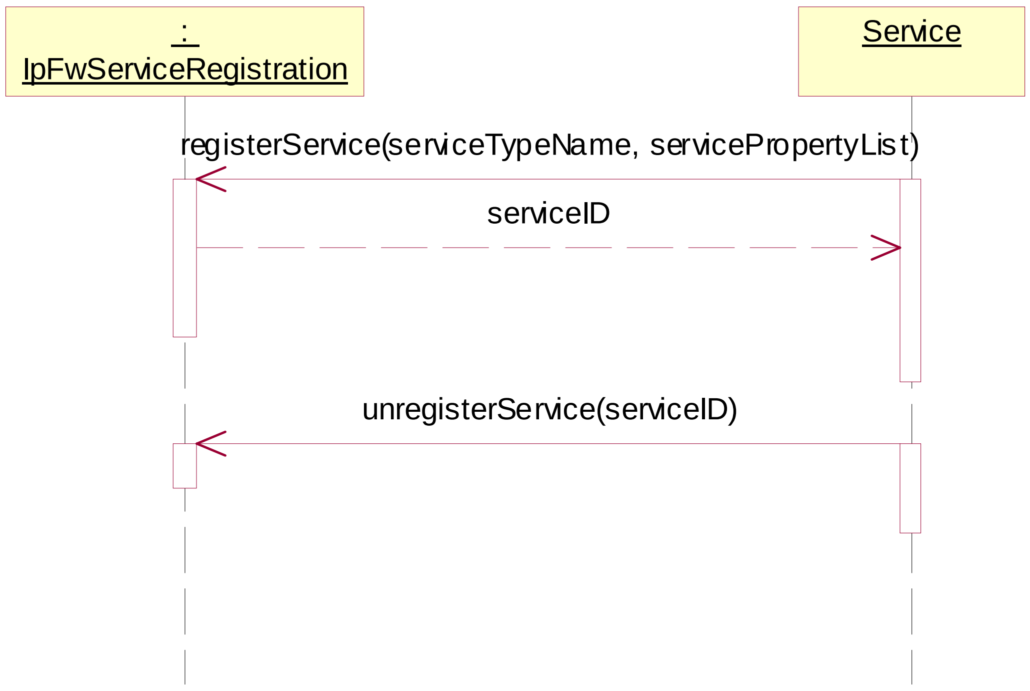
Summary: **IpFwServiceRegistration**, registerService and unregisterService methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Triggered action: cause IUT to call **registerService ()** method on the tester's (Framework) **IpFwServiceRegistration** interface.
   Parameters: serviceTypeName, servicePropertyList

2. Triggered action: cause IUT to call **unregisterService ()** method on the tester's (Framework) **IpFwServiceRegistration** interface.
   Parameters: serviceID


: IpFwServiceRegistration
Service
registerService(serviceTypeName, servicePropertyList)
serviceID
unregisterService(serviceID)

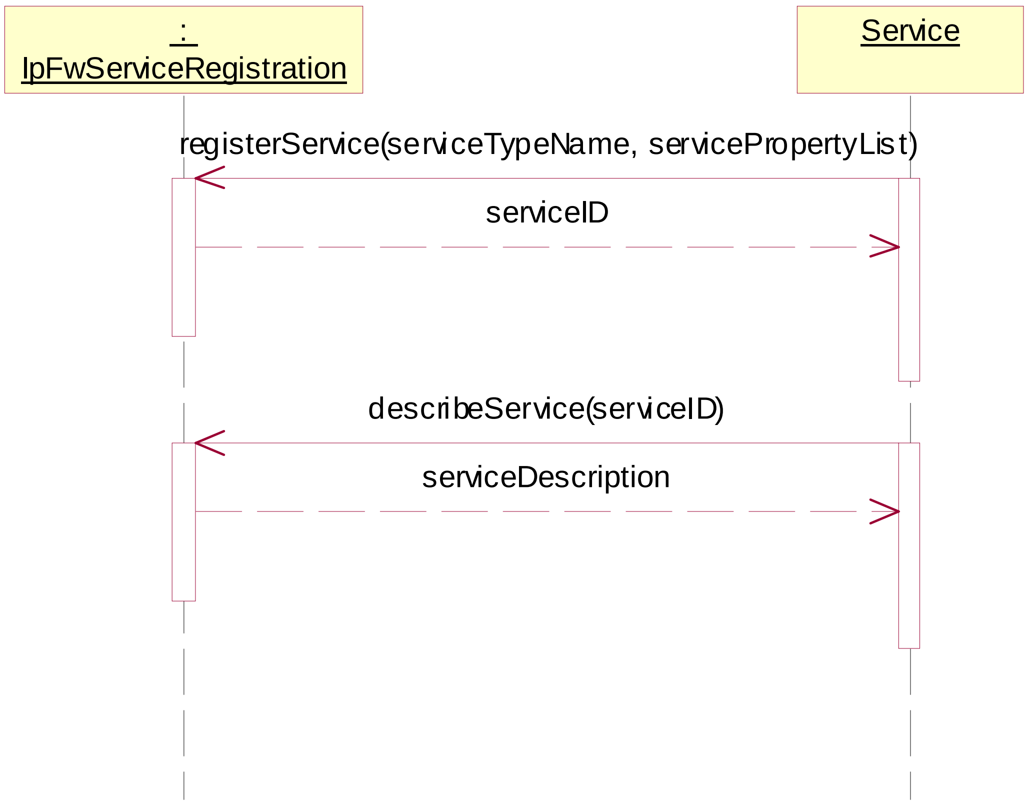
Test FW_FS_SR_S02

Summary: **IpFwServiceRegistration**, describeService method, successful.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Triggered action: cause IUT to call **describeService()** method on the tester's (Framework) **IpFwServiceRegistration** interface.
   Parameters: serviceID as returned by the registerService method


: IpFwServiceRegistration
Service
registerService(serviceTypeName, servicePropertyList)
serviceID
describeService(serviceID)
serviceDescription

Test FW_FS_SR_S03

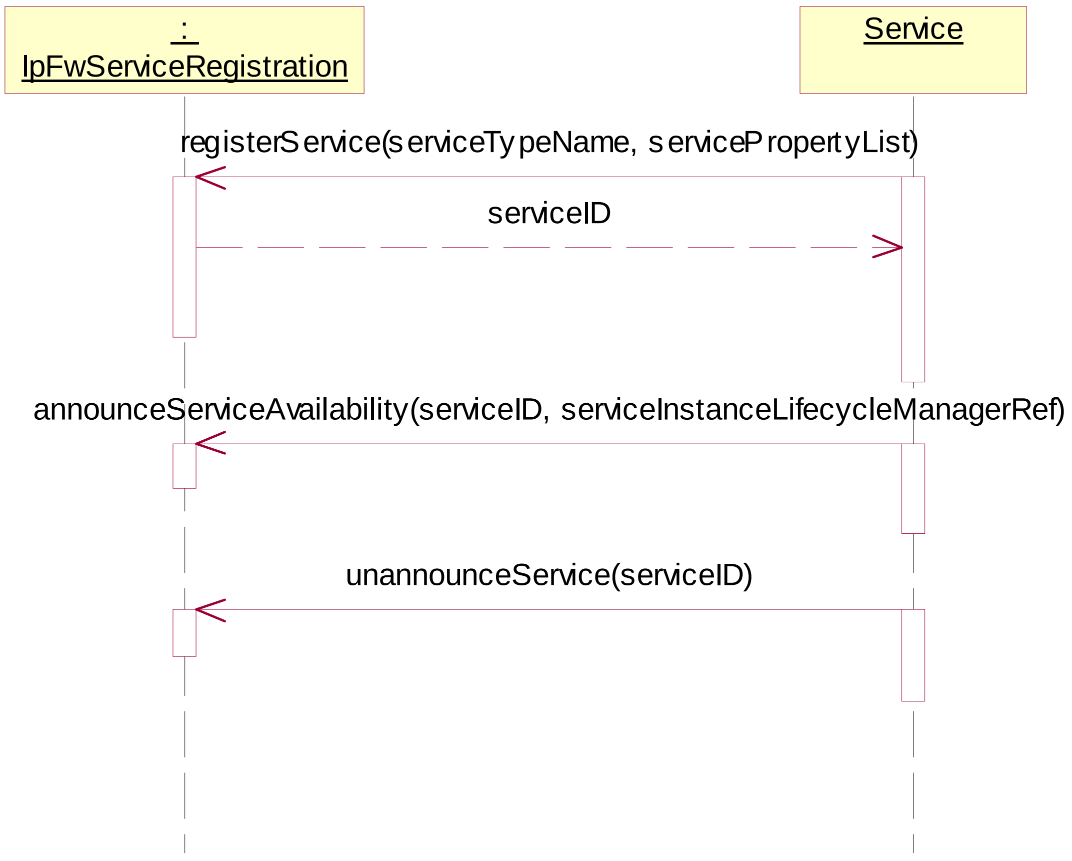
Summary: **IpFwServiceRegistration**, announceServiceAvailability and unannounceService methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.1.

Test Sequence:

1. Triggered action: cause IUT to call **announceServiceAvailability()** method on the tester's (Framework) **IpFwServiceRegistration** interface.
   Parameters: serviceID as returned by the registerService method,
   serviceInstanceLifeCycleManagerRef

2. Triggered action: cause IUT to call **unannounceService()** method on the tester's (Framework) **IpFwServiceRegistration** interface.
   Parameters: serviceID as returned by the registerService method


: IpFwServiceRegistration
Service
registerService(serviceTypeName, servicePropertyList)
serviceID
announceServiceAvailability(serviceID, serviceInstanceLifecycleManagerRef)
unannounceService(serviceID)

11.4.2 Service Instance Lifecycle Management (SILM)

Test FW_FS_SILM_S01

Summary: **IpServiceInstanceLifecycleManager**, createServiceManager and destroyServiceManager methods, successful.

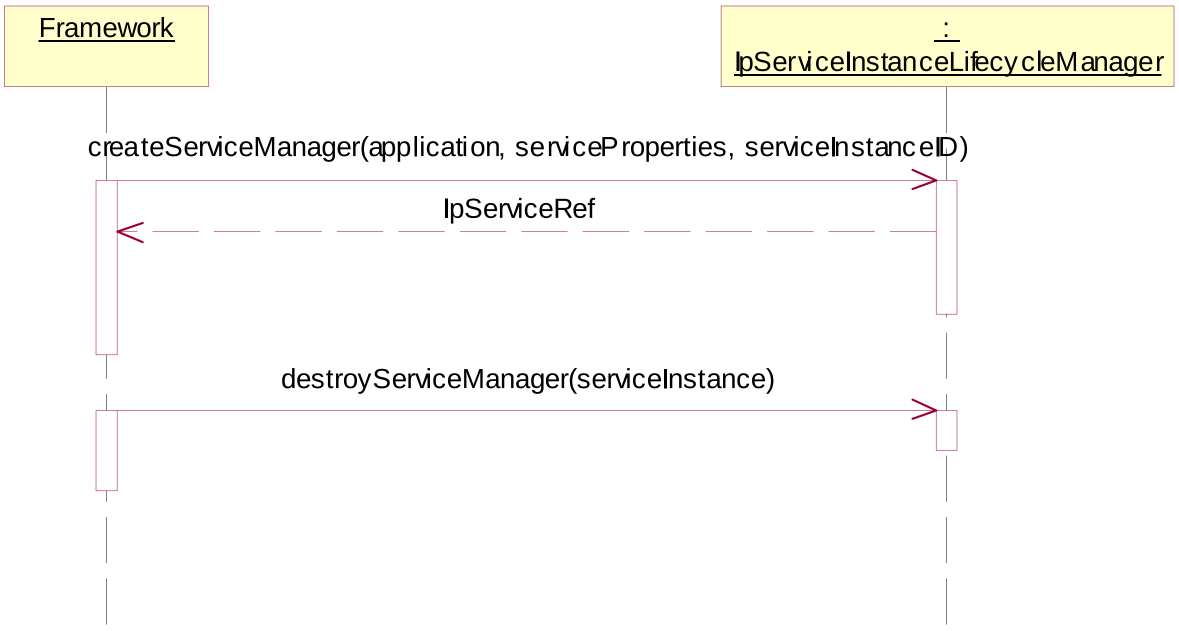
Reference: ES 202 915-3 [1], clause 9.3.2.

Preamble: The service has been previously registered and announced (with the registerService and announceServiceAvailability methods).

Test Sequence:

1. Method call **createServiceManager()**
   Parameters: application, serviceProperties, serviceInstanceID
   Check: valid value of IpServiceRef is returned

2. Method call **destroyServiceManager()**
   Parameters: serviceInstanceID (same value as used in 1.).
   Check: no exception is returned


Framework
: IpServiceInstanceLifecycleManager
createServiceManager(application, serviceProperties, serviceInstanceID)
IpServiceRef
destroyServiceManager(serviceInstance)

Test FW_FS_SILM_S02

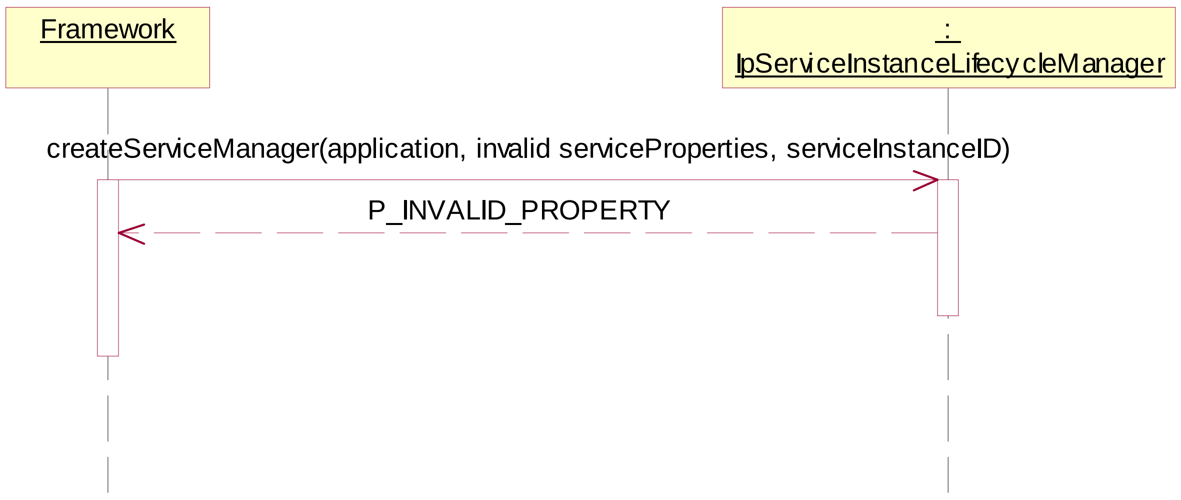
Summary: **IpServiceInstanceLifecycleManager**, createServiceManager method, P_INVALID_PROPERTY.

Reference: ES 202 915-3 [1], clause 9.3.2.

Preamble: The service has been previously registered and announced (with the registerService and announceServiceAvailability methods).

Test Sequence:

1. Method call **createServiceManager()**
   Parameters: application, invalid serviceProperties, serviceInstanceID
   Check: P_INVALID_PROPERTY is returned.


Framework
: IpServiceInstanceLifecycleManager
createServiceManager(application, invalid serviceProperties, serviceInstanceID)
P_INVALID_PROPERTY

11.4.3 Service Discovery (SD)

Test FW_FS_SD_S01

Summary: **IpFwServiceDiscovery** all methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **listServicesTypes()** method on the tester's (Framework) **IpFwServiceDiscovery** interface.
   Parameters: none

2. Triggered action: cause IUT to call **listRegisteredServices()** method on the tester's (Framework) **IpFwServiceDiscovery** interface.
   Parameters: none

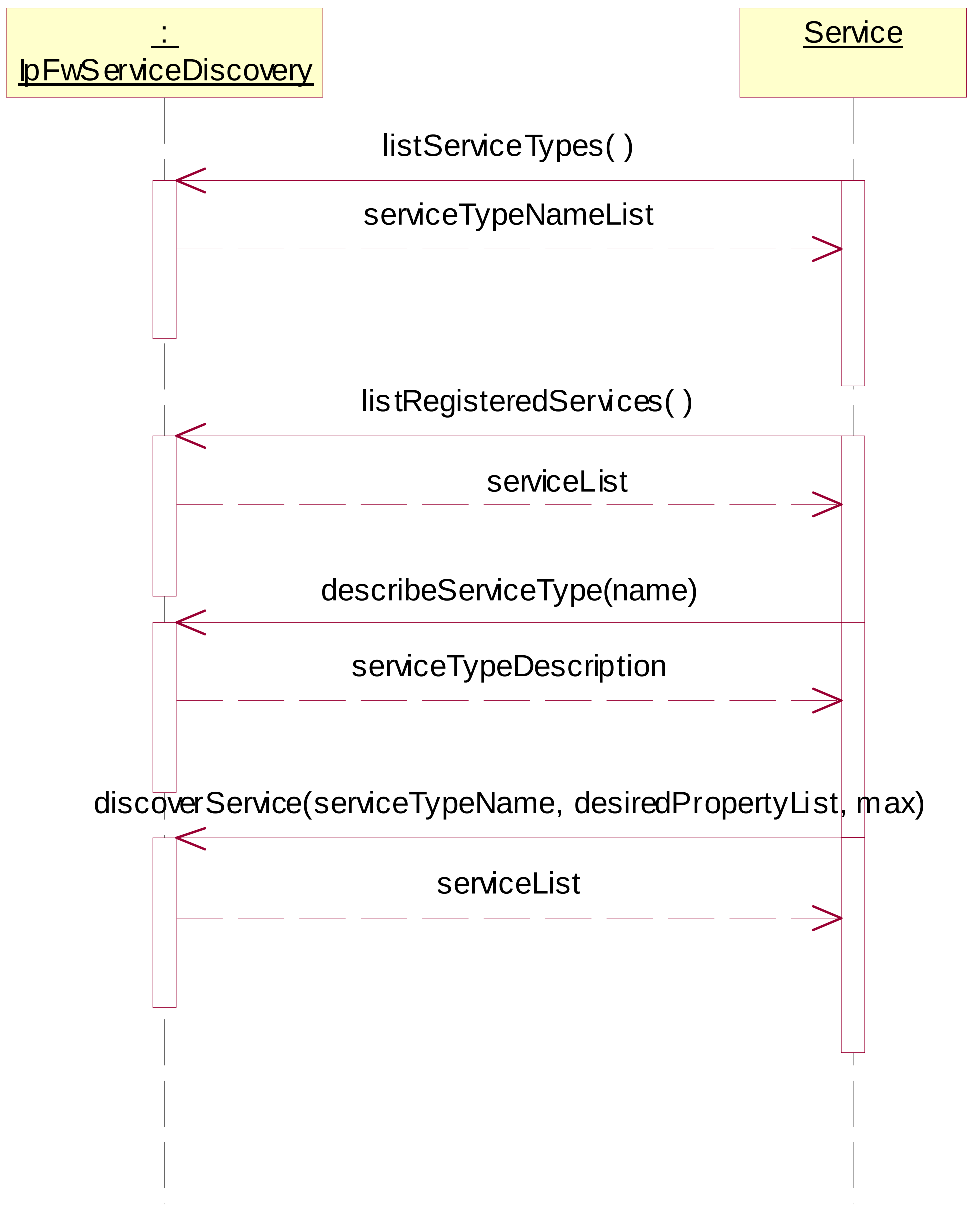
NOTE 1: This method need not be invoked by the application for this test to succeed.

3. Triggered action: cause IUT to call **describeServiceType()** method on the tester's (Framework) **IpFwServiceDiscovery** interface.
   Parameters: serviceTypeName

NOTE 2: This method need not be invoked by the application for this test to succeed.

4. Triggered action: cause IUT to call **discoverService()** method on the tester's (Framework) **IpFwServiceDiscovery** interface.
   Parameters: serviceTypeName, desiredPropertyList, max

NOTE 3: This method need not be invoked by the application for this test to succeed.



11.4.4 Integrity Management (IM)

Test FW_FS_IM_S01

Summary: **IpSvcHeartBeatMgmt**, enableSvcHeartBeat and disableHeartBeat methods, successful.

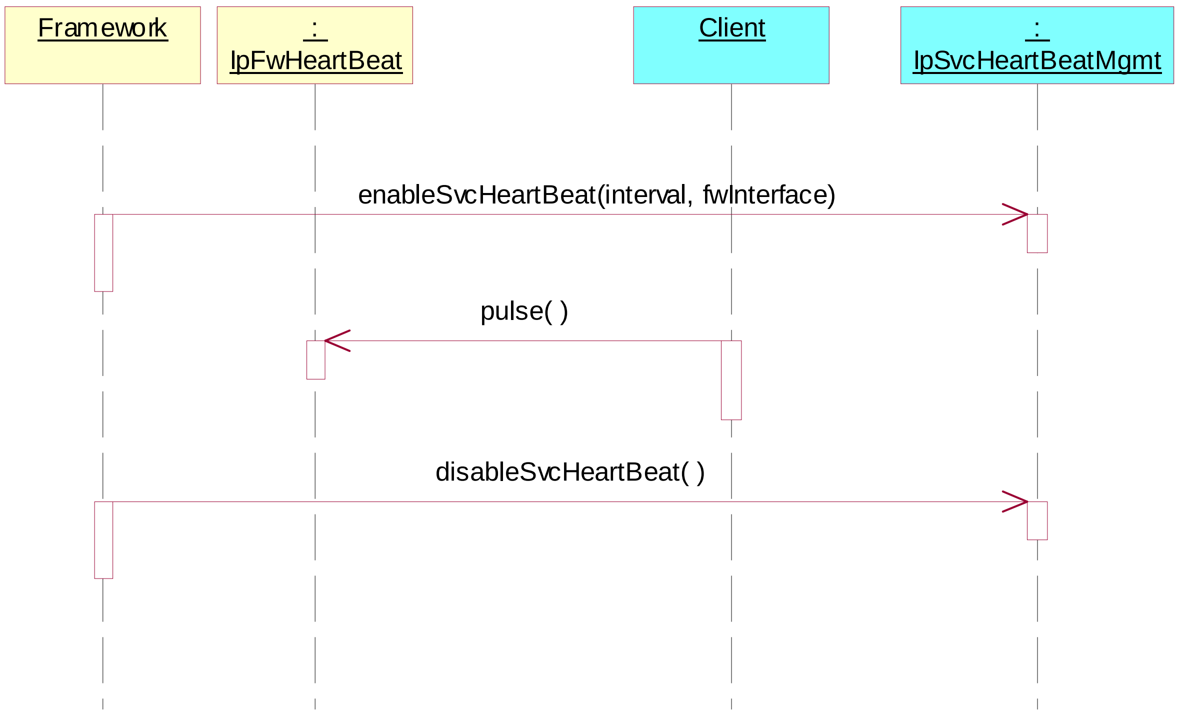
Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **enableSvcHeartBeat()**
   Parameters: interval, fwInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to regularly call **pulse**() method on the tester's (Framework) **IpFwHeartBeat** interface.
   Parameters: none
   Check: Check that the **pulse**() method is invoked at the requested interval.

3. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.



Test FW_FS_IM_S02

Summary: **IpSvcHeartBeatMgmt**, all methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

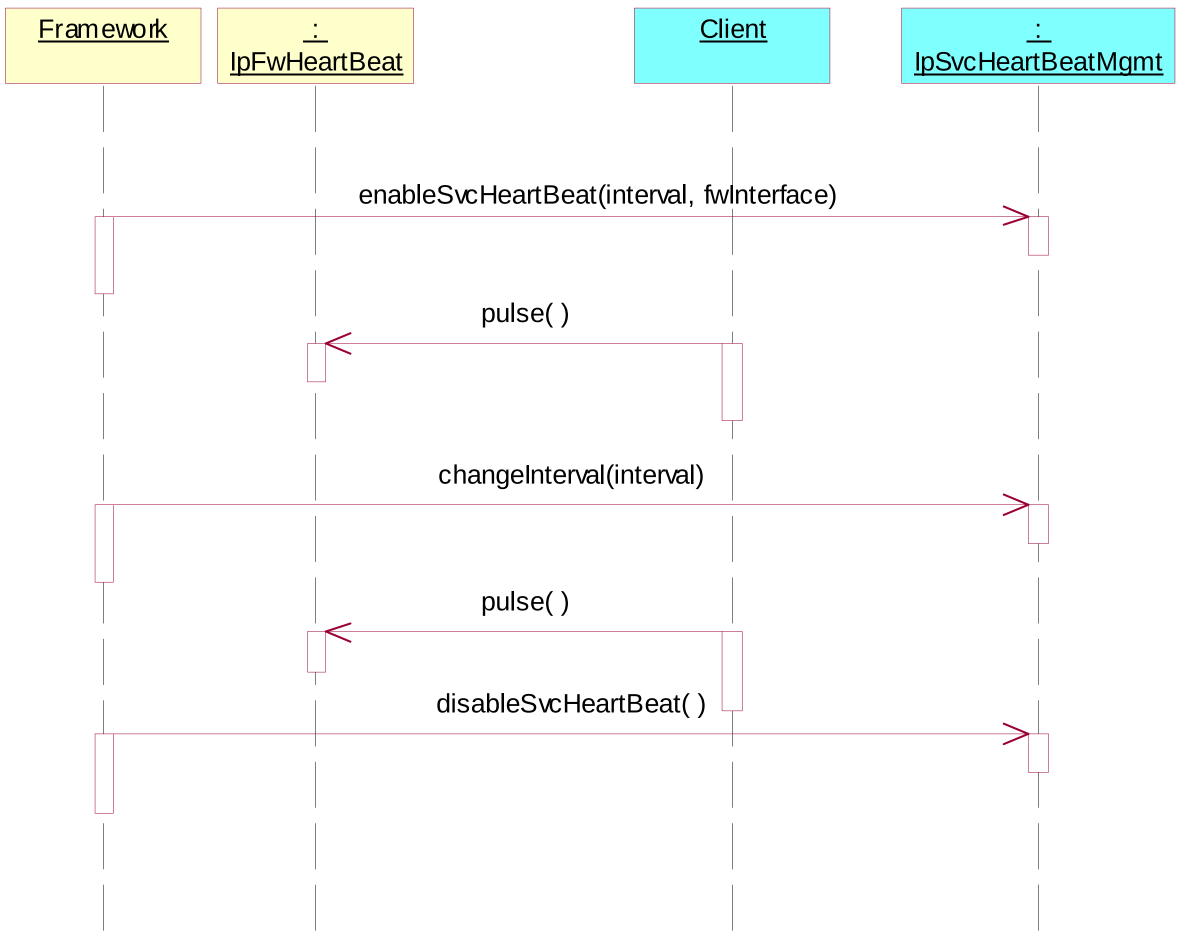
1. Method call **enableSvcHeartBeat()**
   Parameters: interval, fwInterface
   Check: no exception is returned.

2. Triggered action: cause IUT to call **pulse**() method regularly on the tester's (Framework) **IpFwHeartBeat** interface.
   Parameters: none
   Check: Check that the **pulse**() method is invoked at the requested interval.

3. Method call **changeInterval()**
   Parameters: interval
   Check: no exception is returned.

4. Triggered action: cause IUT to call **pulse**() method regularly on the tester's (Framework) **IpFwHeartBeat** interface.
   Parameters: none
   Check: the **pulse**() method is invoked at the new requested interval.

5. Method call **disableHeartBeat()**
   Parameters: none
   Check: no exception. Verify that no pulse() is received anymore by the expiry of the interval timer.



Test FW_FS_IM_S03

Summary: **IpSvcHeartBeat**, pulse, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

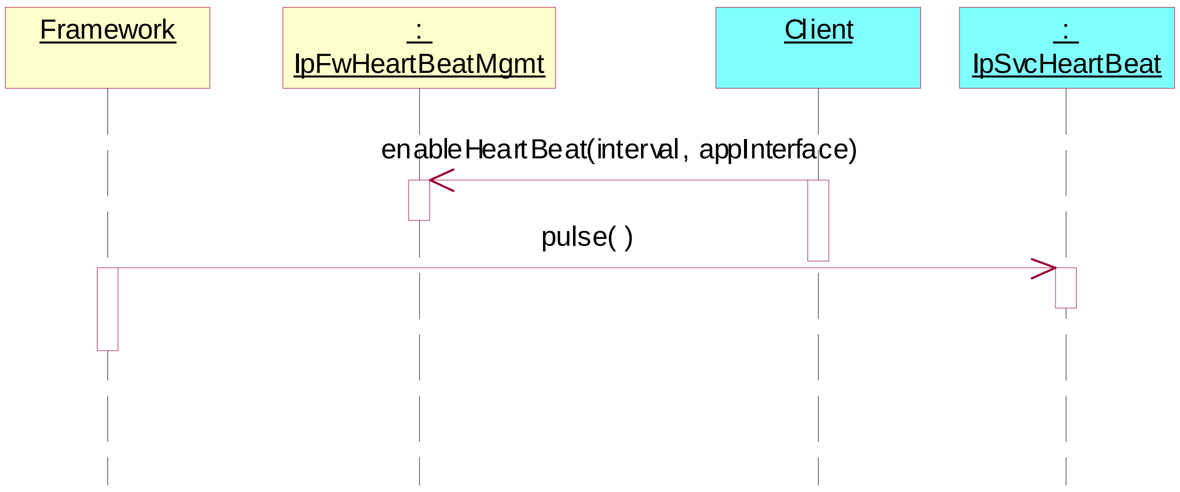
Precondition: IpHeartBeat is supported.

Preamble: The calling service must have a callback interface and a reference to this interface.

Test Sequence:

1. Triggered action: cause IUT to call **enableHeartBeat()** method on the tester's (Framework) **IpFwHeartBeatMgmt** interface.
   Parameters: interval, svcInterface

2. Method call **pulse()**
   Parameters: none
   Check: no exception



Test FW_FS_IM_S04

Summary: **IpSvcFaultManager,** activityTestRes, successful.

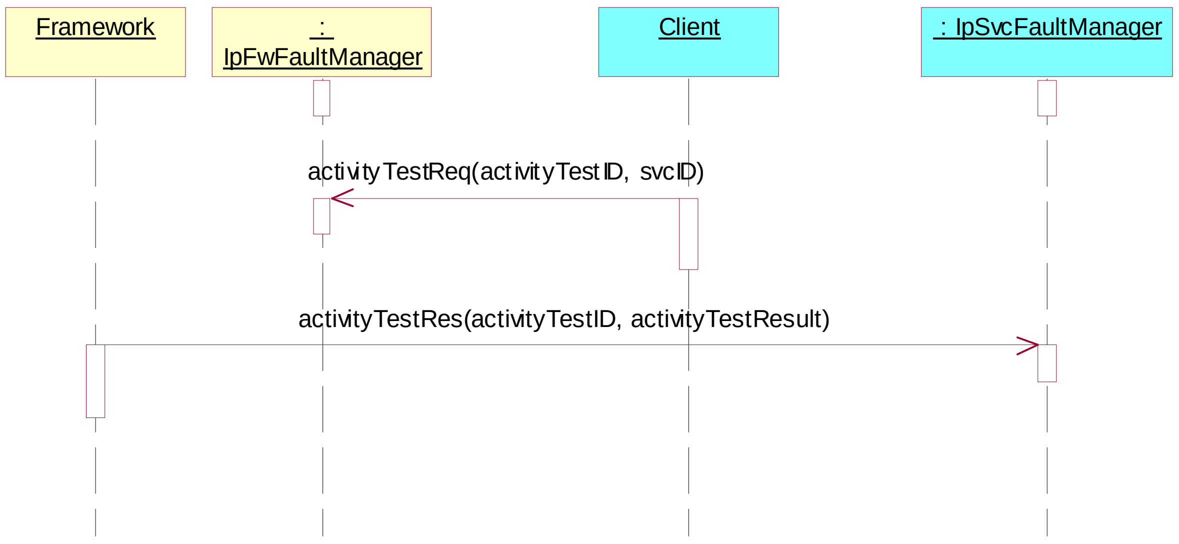
Reference: ES 202 915-3 [1], clause 7.3.3.

Preamble: The IUT (Service) must have subscribed to at least one service.

Test Sequence:

1. Triggered action: cause IUT to call **activityTestReq()** method on the tester's (Framework) **IpFwFaultManager** interface.
   Parameters: activityTestID, svcID

2. Method call **activityTestRes ()**
   Parameters: activityTestID, activityTestResult
   Check: no exception is returned


Framework
: IpFwFaultManager
Client
: IpSvcFaultManager
activityTestReq(activityTestID, svcID)
activityTestRes(activityTestID, activityTestResult)

Test FW_FS_IM_S05

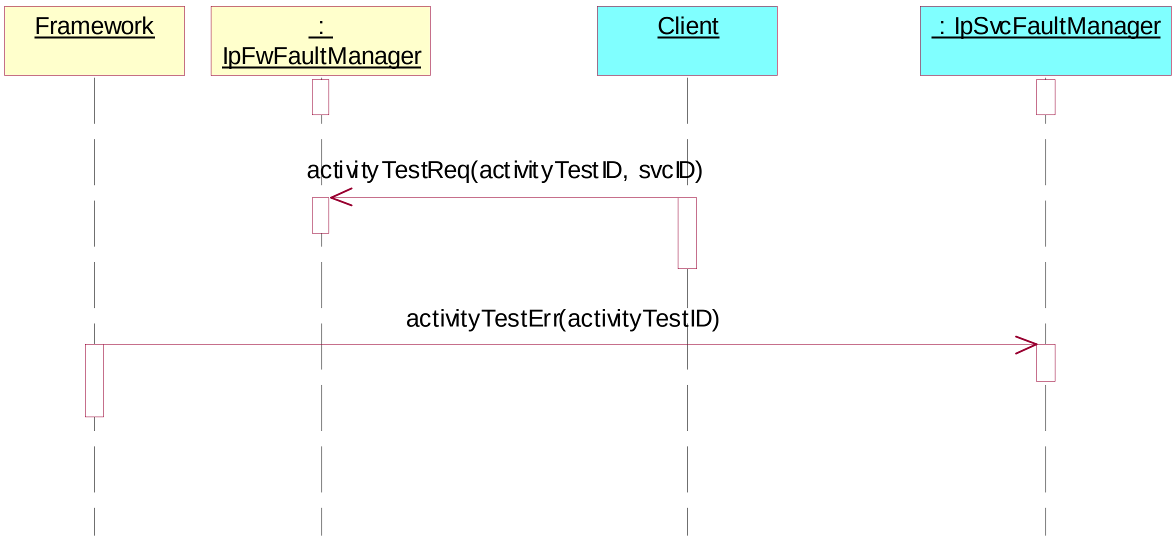
Summary: **IpSvcFaultManager,** activityTestErr, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **activityTestReq()** method on the tester's (Framework) **IpFwFaultManager** interface.
   Parameters: activityTestID, svcID

2. Method call **activityTestErr()**
   Parameters: activityTestID
   Check: no exception is returned


Framework
: IpFwFaultManager
Client
: IpSvcFaultManager
activityTestReq(activityTestID, svcID)
activityTestErr(activityTestID)

Test FW_FS_IM_S06

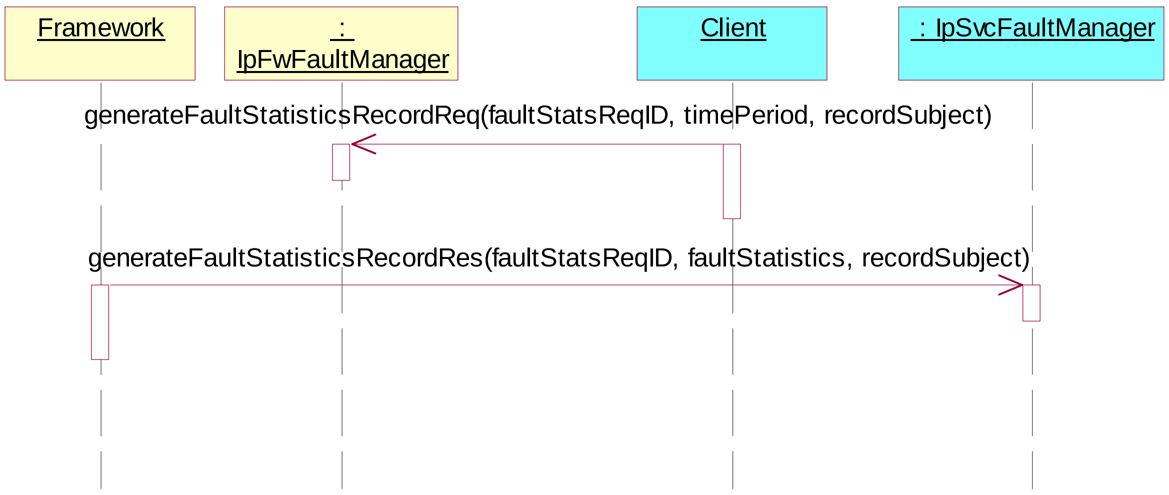
Summary: **IpSvcFaultManager,** generateFaultStatisticsRecordRes, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **generateFaultStatisticsRecordReq ()** method on the tester's (Framework) **IpFwFaultManager** interface.
   Parameters: faultStatsReqID, timePeriod, serviceIDs

2. Method call **generateFaultStatisticsRecordRes ()**
   Parameters: faultStatsReqID, faultStatistics, ServiceIDs
   Check: no exception is returned


Framework
: IpFwFaultManager
Client
: IpSvcFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, recordSubject)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics, recordSubject)

Test FW_FS_IM_S07

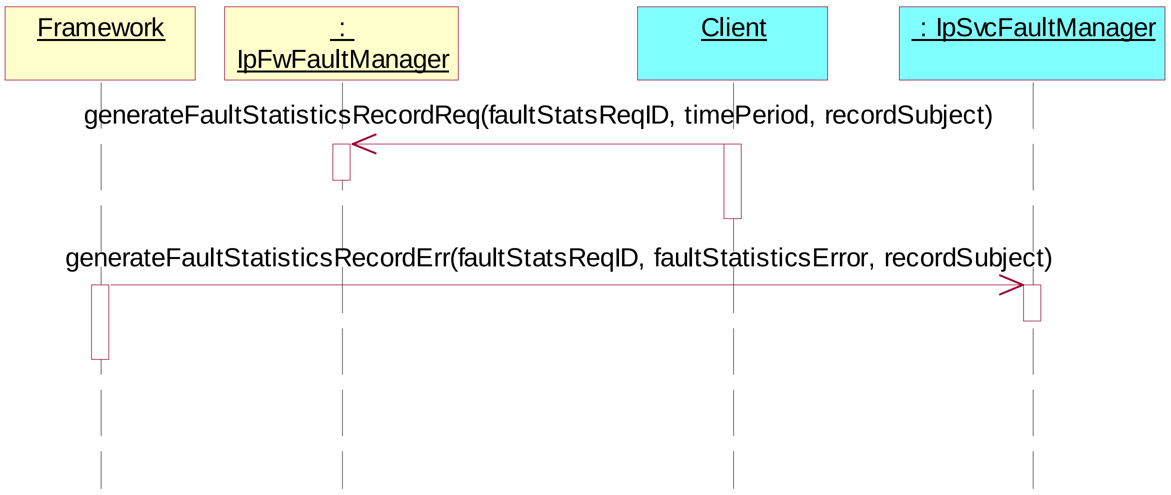
Summary: **IpSvcFaultManager,** generateFaultStatisticsRecordErr, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Triggered action: cause IUT to call **generateFaultStatisticsRecordReq ()** method on the tester's (Framework) **IpFwFaultManager** interface.
   Parameters: timePeriod, serviceIDs

2. Method call **generateFaultStatisticsRecordErr ()**
   Parameters: faultStatsReqID, faultStatisticsError, ServiceIDs
   Check: no exception is returned


Framework
: IpFwFaultManager
Client
: IpSvcFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod, recordSubject)
generateFaultStatisticsRecordErr(faultStatsReqID, faultStatisticsError, recordSubject)

Test FW_FS_IM_S08

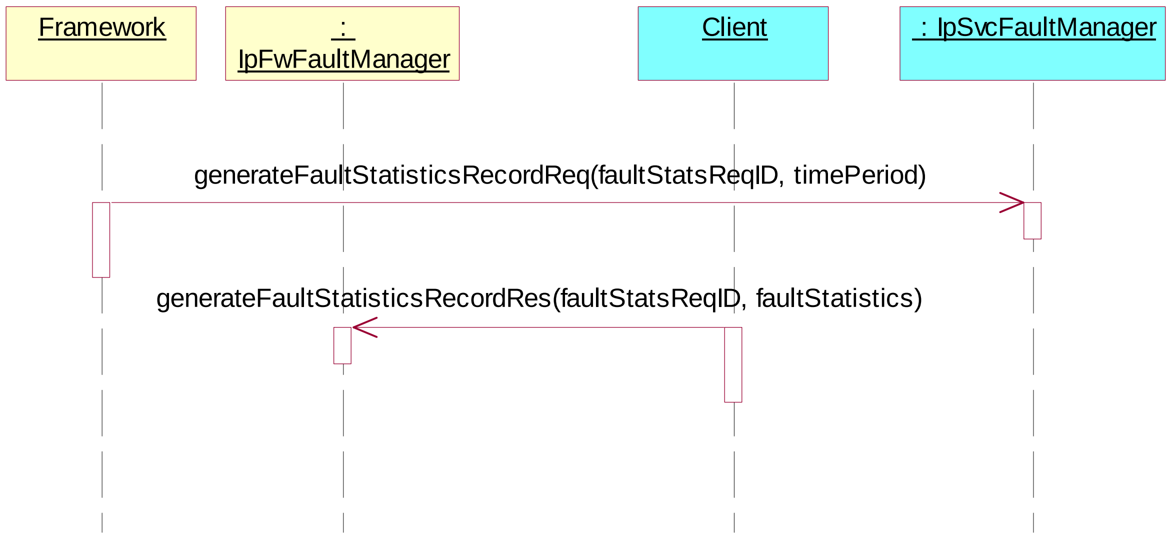
Summary: **IpSvcFaultManager,** generateFaultStatisticsRecordReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **generateFaultStatisticsRecordReq ()**
   Parameters: faultStatsReqID, timePeriod
   Check: no exception is returned

2. Triggered action: cause IUT to call **generateFaultStatisticsRecordRes ()** method on the tester's (Framework) **IpFwFaultManager** interface.
   Parameters: faultStatsReqID, faultStatistics


Framework
: IpFwFaultManager
Client
: IpSvcFaultManager
generateFaultStatisticsRecordReq(faultStatsReqID, timePeriod)
generateFaultStatisticsRecordRes(faultStatsReqID, faultStatistics)

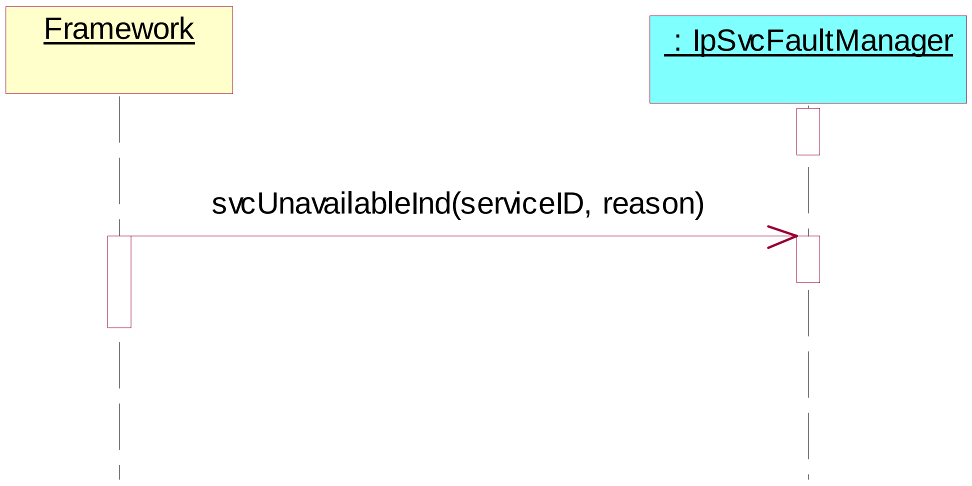
Test FW_FS_IM_S09

Summary: **IpSvcFaultManager,** svcUnavailableInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: serviceID, reason
   Check: no exception is returned



Test FW_FS_IM_S10

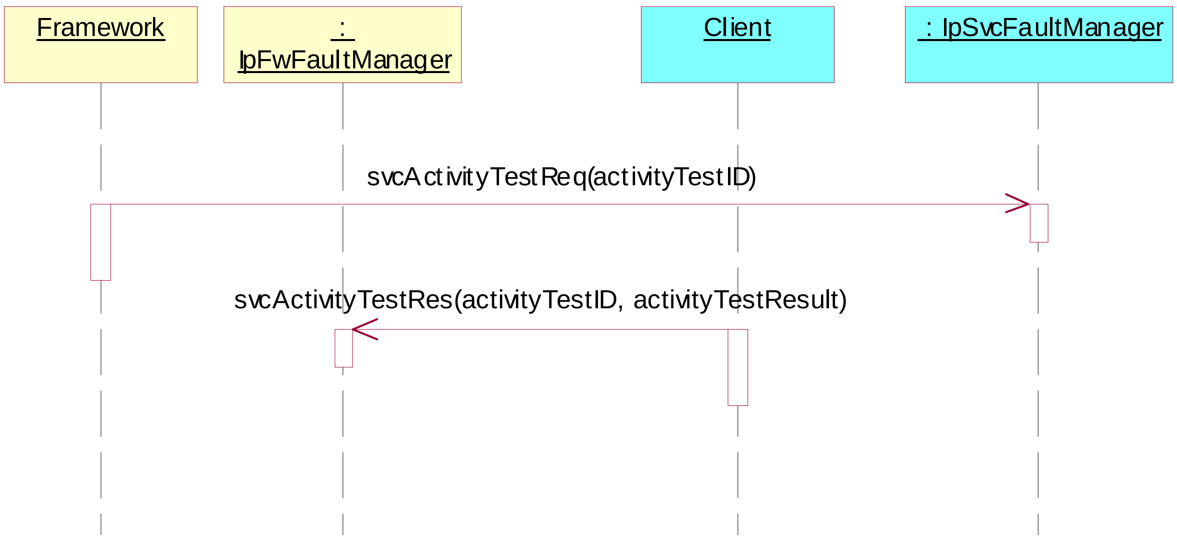
Summary: **IpSvcFaultManager,** svcActivityTestReq, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **svcActivityTestReq ()**
   Parameters: activityTestID
   Check: no exception is returned

2. Triggered action: cause IUT to call **svcActivityTestRes ()** method on the tester's (Framework) **IpFwFaultManager** interface.
   Parameters: activityTestID, activityTestResult


Framework
: IpFwFaultManager
Client
: IpSvcFaultManager
svcActivityTestReq(activityTestID)
svcActivityTestRes(activityTestID, activityTestResult)

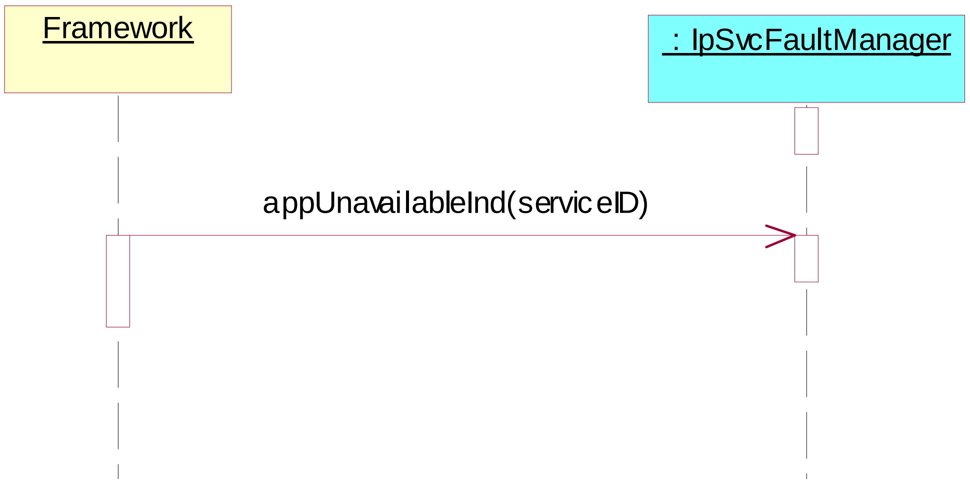
Test FW_FS_IM_S11

Summary: **IpSvcFaultManager,** svcUnavailableInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **svcUnavailableInd()**
   Parameters: serviceID
   Check: no exception is returned



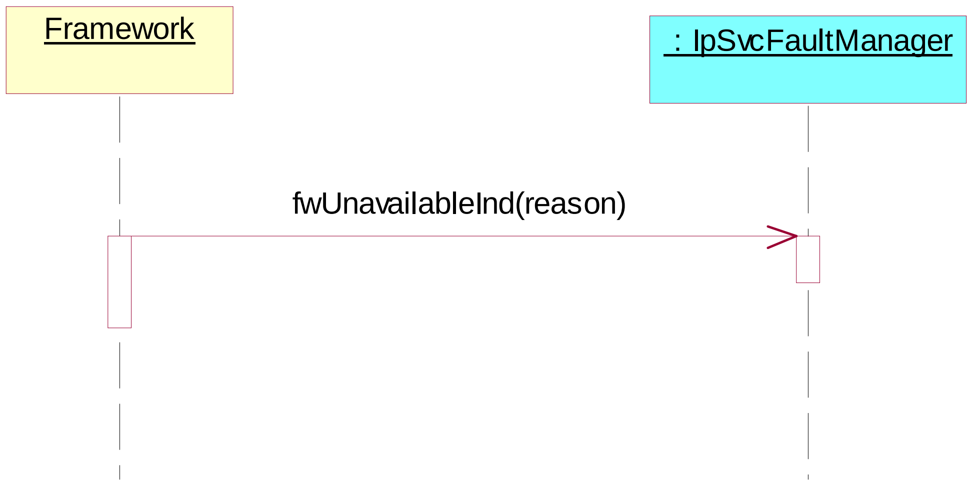
Test FW_FS_IM_S12

Summary: **IpSvcFaultManager,** fwUnavailableInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **fwUnavailableInd()**
   Parameters: reason
   Check: no exception is returned



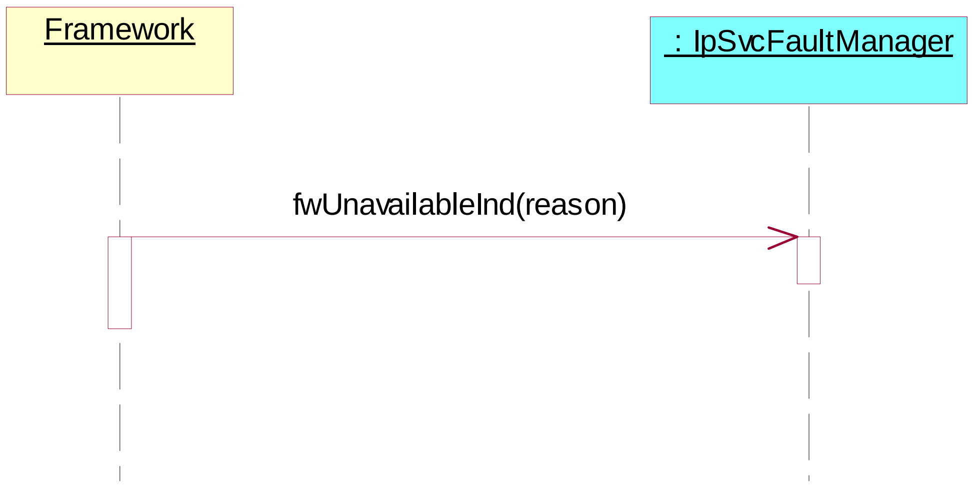
Test FW_FS_IM_S13

Summary: **IpSvcFaultManager,** fwFaultReportInd, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **fwFaultReportInd()**
   Parameters: fault
   Check: no exception is returned, client service no longer use the Framework.



Test FW_FS_IM_S14

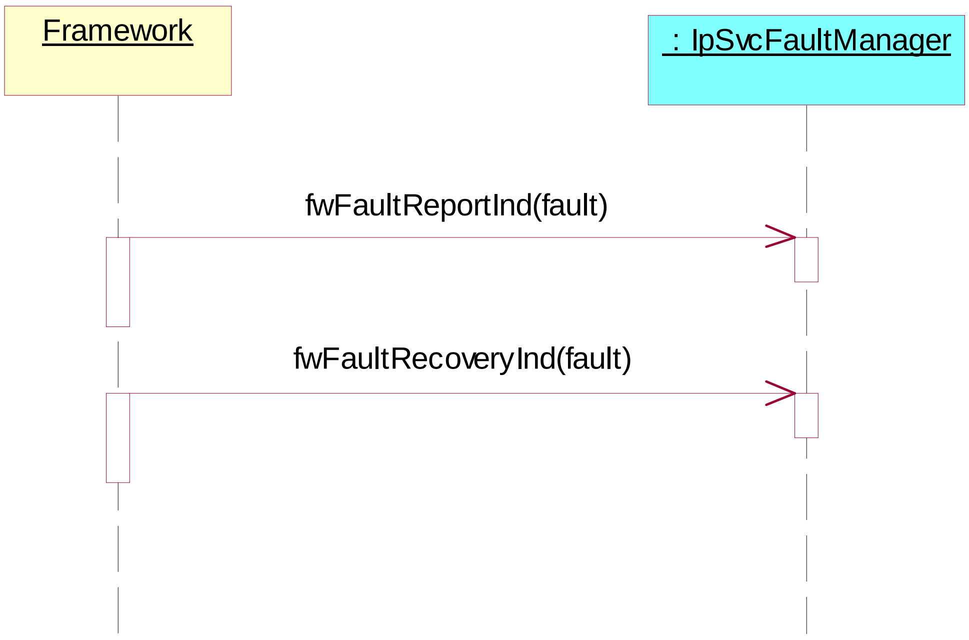
Summary: **IpSvcFaultManager,** fwFaultReportInd and fwFaultRecoveryInd methods, successful.

Reference: ES 202 915-3 [1], clause 7.3.3.

Test Sequence:

1. Method call **fwFaultReportInd()**
   Parameters: fault
   Check: no exception is returned, client service no longer use the Framework.

2. Method call **fwFaultRecoveryInd()**
   Parameters: fault
   Check: no exception is returned, client service resume using the Framework.



Test FW_FS_IM_S15

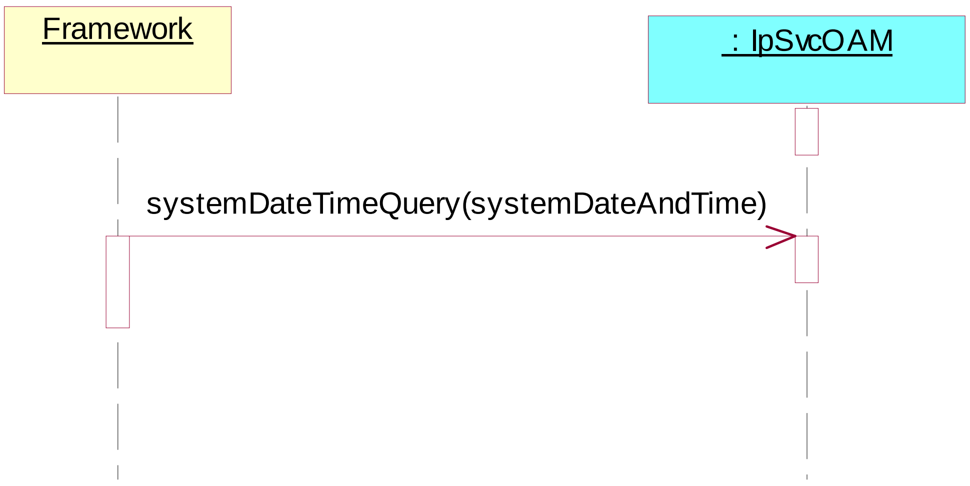
Summary: **IpSvcOAM**, systemDateTimeQuery, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Precondition: IpFwOAM supported.

Test Sequence:

1. Method call **systemDateTimeQuery()**
   Parameters: clientDateAndTime
   Check: valid value of TpDateAndTime is returned



Test FW_FS_IM_S16

Summary: **IpSvcLoadManager,** all methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Method call **createLoadLevelNotification()**
   Parameters: none
   Check: no exception is returned

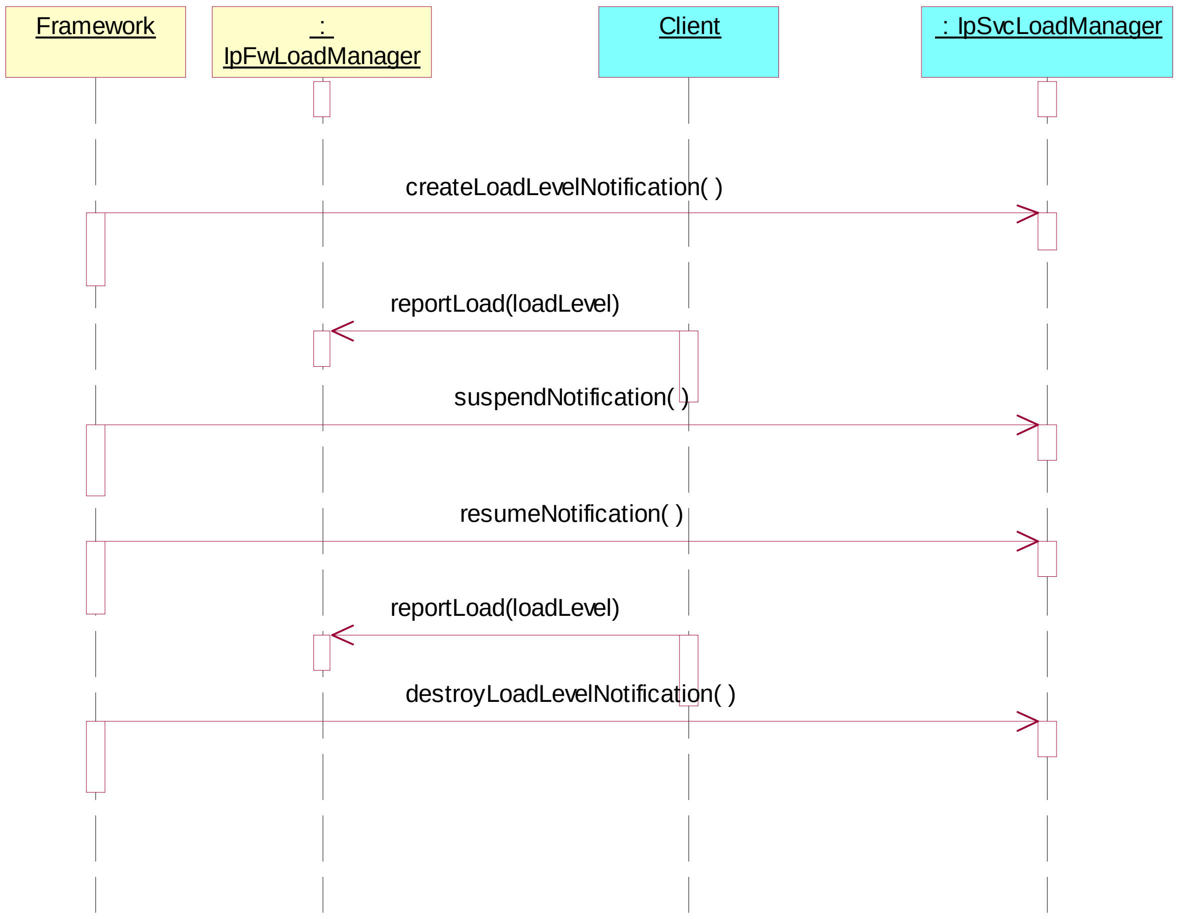
2. Triggered action: cause IUT to call **reportLoad()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: loadLevel

3. Method call **suspendNotification()**
   Parameters: none
   Check: no exception is returned, no load level notifications received until resumeNotification() is called.

4. Method call **resumeNotification()**
   Parameters: none
   Check: no exception is returned

5. Triggered action: cause IUT to call **reportLoad()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: loadLevel

6. Method call **destroyLoadLevelNotification()**
   Parameters: none
   Check: no exception is returned



Test FW_FS_IM_S17

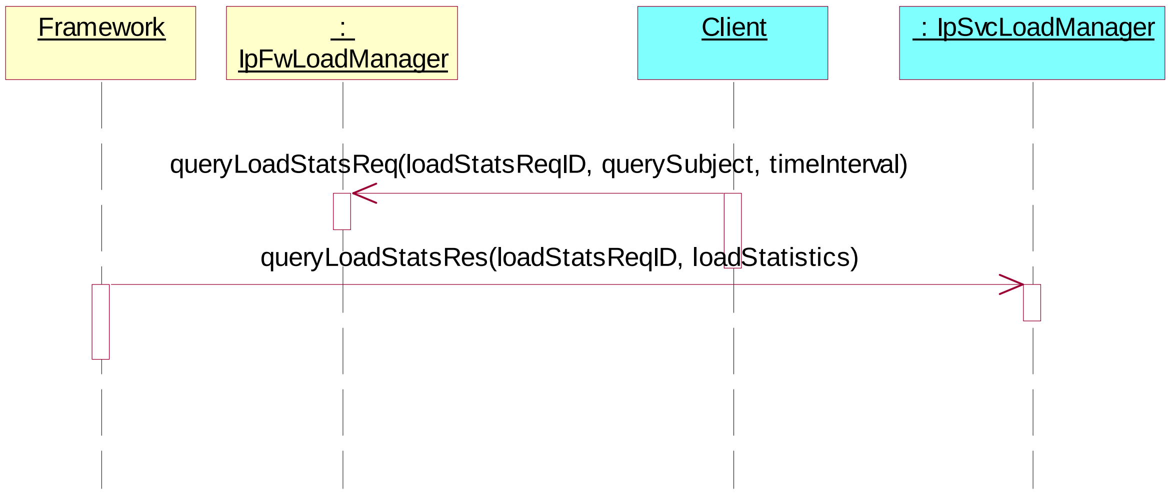
Summary: **IpSvcLoadManager,** queryLoad Stats Res, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **queryLoadStatsReq()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: loadStatsReqID, querySubject, timeInterval

2. Method call **queryLoadStatsRes ()**
   Parameters: loadStatsReqID, loadStatistics
   Check: no exception is returned


Framework
: IpFwLoadManager
Client
: IpSvcLoadManager
queryLoadStatsReq(loadStatsReqID, querySubject, timeInterval)
queryLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FS_IM_S18

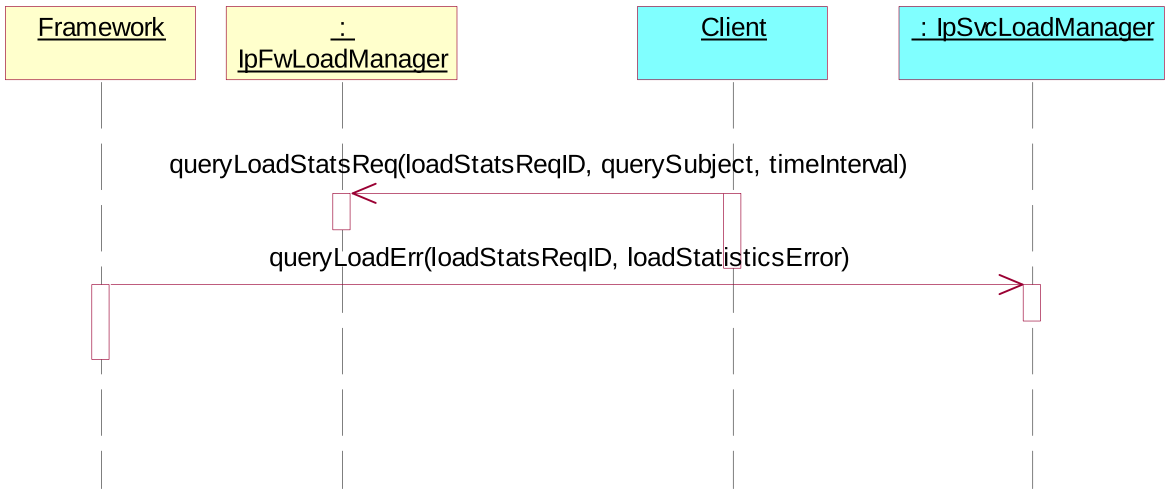
Summary: **IpSvcLoadManager,** queryLoadStatsErr, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **queryLoadStatsReq()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: loadStatsReqID, querySubject, timeInterval

2. Method call **queryLoadStatsErr()**
   Parameters: loadStatsReqID, loadStatisticsError
   Check: no exception is returned


Framework
: IpFwLoadManager
Client
: IpSvcLoadManager
queryLoadStatsReq(loadStatsReqID, querySubject, timeInterval)
queryLoadErr(loadStatsReqID, loadStatisticsError)

Test FW_FS_IM_S19

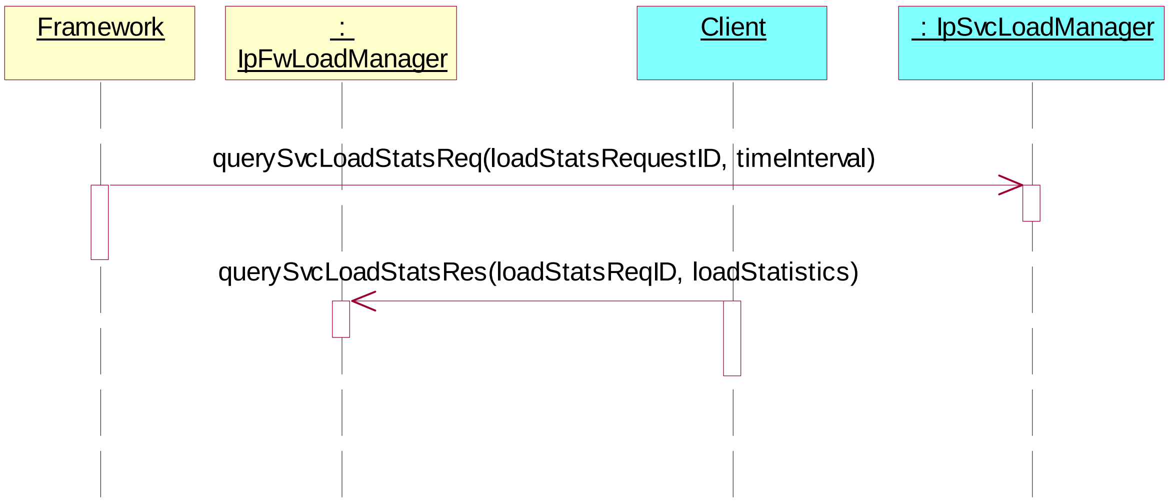
Summary: **IpSvcLoadManager,** querySvcLoadStatsReq, successful.

Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Method call **querySvcLoadStatsReq()**
   Parameters: loadStatsReqID, timeInterval
   Check: no exception is returned

2. Triggered action: cause IUT to call **querySvcLoadStatsRes()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: loadStatsReqID, loadStatistics


Framework
: IpFwLoadManager
Client
: IpSvcLoadManager
querySvcLoadStatsReq(loadStatsRequestID, timeInterval)
querySvcLoadStatsRes(loadStatsReqID, loadStatistics)

Test FW_FS_IM_S20

Summary: **IpSvcLoadManager,** LoadLevelNotification, successful.

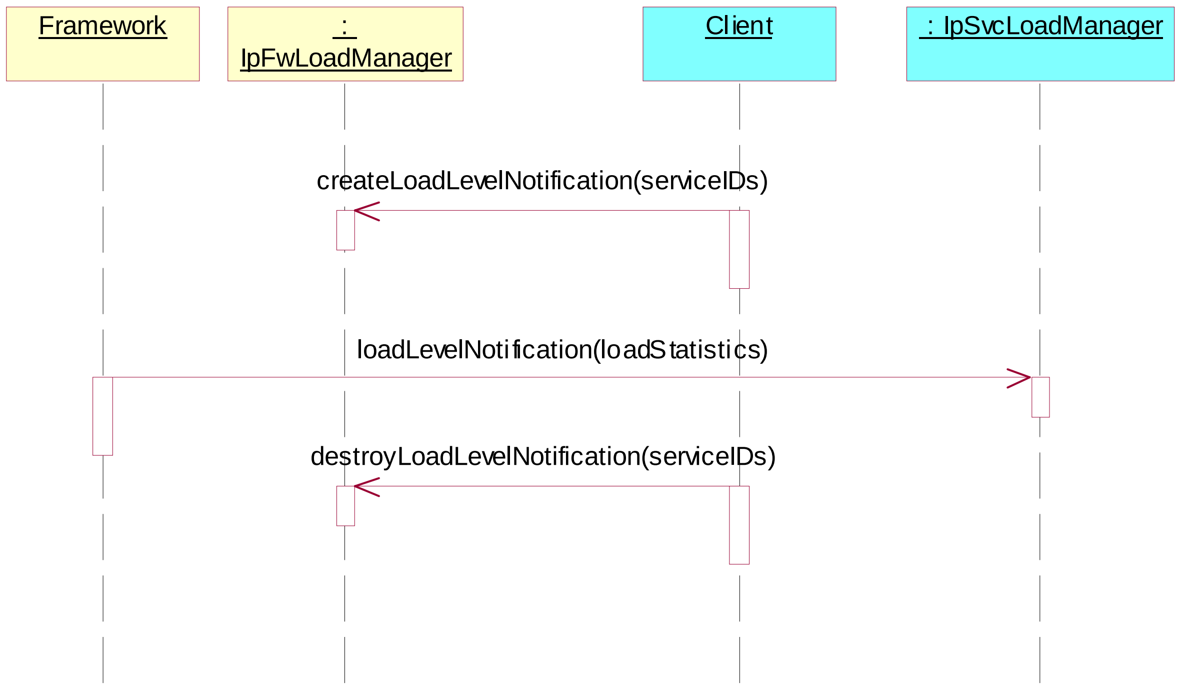
Reference: ES 202 915-3 [1], clause 9.3.4.

Test Sequence:

1. Triggered action: cause IUT to call **createLoadLevelNotification()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: serviceIDs

2. Method call **LoadLevelNotification()**
   Parameters: loadStatistics
   Check: no exception is returned

3. Triggered action: cause IUT to call **destroyLoadLevelNotification()** method on the tester's (Service) **IpFwLoadManager** interface.
   Parameters: serviceIDs


Framework
: IpFwLoadManager
Client
: IpSvcLoadManager
createLoadLevelNotification(serviceIDs)
loadLevelNotification(loadStatistics)
destroyLoadLevelNotification(serviceIDs)

11.4.5 Event Notification (EN)

Test FW_FS_EN_S01

Summary: **IpSvcEventNotification,** reportNotification method, successful.

Reference: ES 202 915-3 [1], clause 9.3.5.

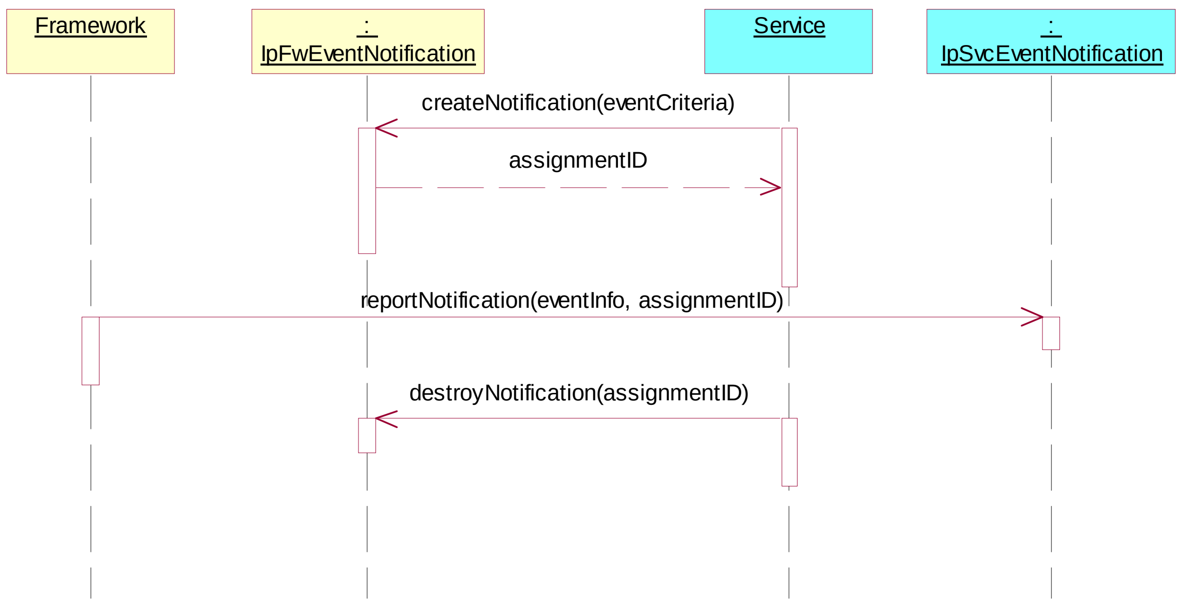
Precondition: IpSvcEventNotification supported.

Test Sequence:

1. Triggered action: cause IUT to call **createNotification()** method on the tester's (Framework) **IpFwEventNotification** interface.
   Parameters: eventCriteria

2. Method call **reportNotification ()**
   Parameters: eventInfo, assignmentID
   Check: no exception is returned

3. Triggered action: cause IUT to call **destroyNotification()** method on the tester's (Framework) **IpFwEventNotification** interface.
   Parameters: assignmentID.


Framework
: IpFwEventNotification
Service
: IpSvcEventNotification
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
destroyNotification(assignmentID)

Test FW_FS_EN_S02

Summary: **IpSvcEventNotification**, reportNotification and notificationTerminated methods, successful.

Reference: ES 202 915-3 [1], clause 9.3.5.

Precondition: IpSvcEventNotification supported.

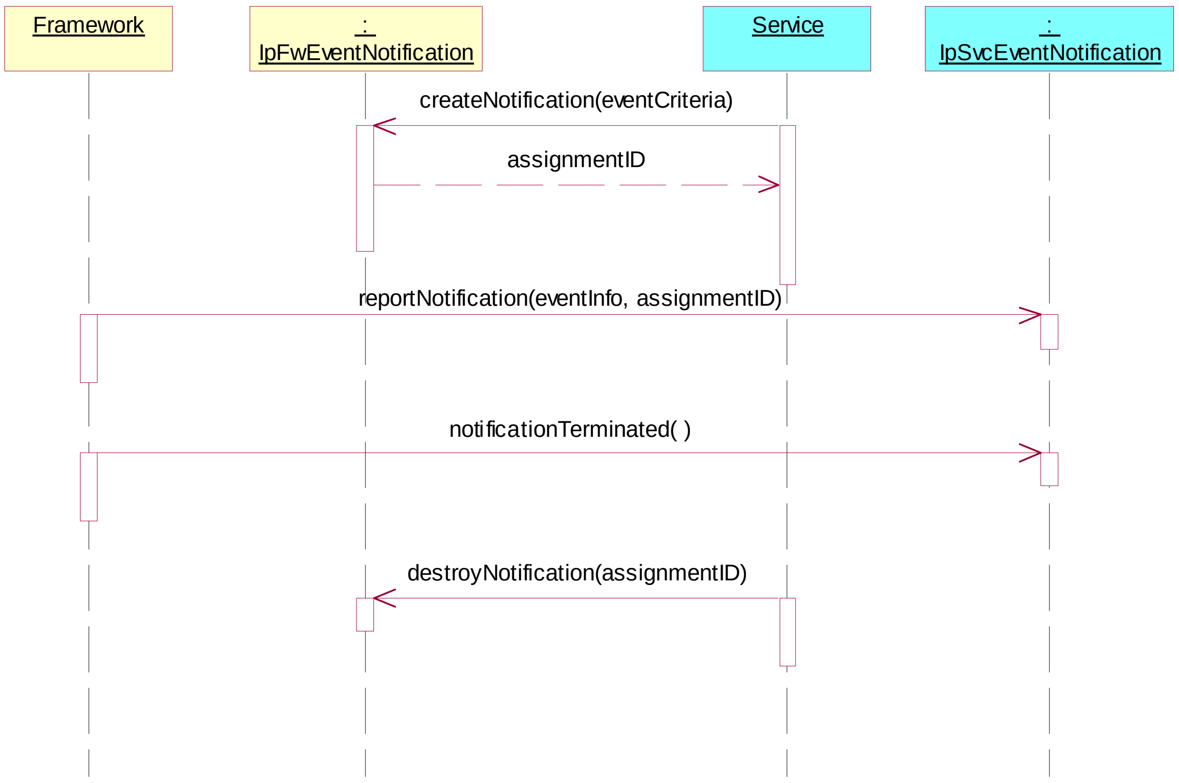
Test Sequence:

1. Triggered action: cause IUT to call **createNotification()** method on the tester's (Framework) **IpFwEventNotification** interface.
   Parameters: eventCriteria

2. Method call **reportNotification ()**
   Parameters: eventInfo, assignmentID
   Check: no exception is returned

3. Method call **notificationTerminated()**
   Parameters: none
   Check: no exception is returned

4. Triggered action: cause IUT to call **destroyNotification()** method on the tester's (Framework) **IpFwEventNotification** interface.
   Parameters: assignmentID.


Framework
: IpFwEventNotification
Service
: IpSvcEventNotification
createNotification(eventCriteria)
assignmentID
reportNotification(eventInfo, assignmentID)
notificationTerminated( )
destroyNotification(assignmentID)

History

Document history			
V1.1.1	January 2005	Membership Approval Procedure	MV 20050311: 2005-01-11 to 2005-03-11
V1.1.1	March 2005	Publication	